Bamboo Data Center and Server 9.4

Contents

Bamboo documentation

Bamboo is a continuous integration and delivery tool that ties automated builds, tests, and releases into a single workflow.

Get started

New to using Bamboo? Get started with some introductory information.

Let's start

What’s new

Read all about the latest changes in Bamboo.

Have a look

Getting started with Bamboo

This page describes how to install, set up, and start using Bamboo.

If you're upgrading Bamboo, read the Bamboo upgrade guide instead of this page.

For production installs we recommend that you read Bamboo best practice - system requirements.

Atlassian Bamboo is a continuous integration (CI) and deployment server. Bamboo assists software development teams by providing:

- automated building and testing of software source-code status.
- updates on successful and failed builds.
- reporting tools for statistical analysis.

Please see the following pages for information about getting started with Bamboo:

- Bamboo best practice - system requirements
- Understanding the Bamboo CI Server - a conceptual overview of using Bamboo for continuous integration (CI).

1. Install and start Bamboo

See one of:

- Installing Bamboo on Linux
- Installing Bamboo on Mac OS X
- Installing Bamboo on Windows

Once it's started, you can access Bamboo in your browser at `http://localhost:8085/`.

2. Set up notifications

Bamboo can send build result notifications using:

- Email - see Configuring Bamboo to send SMTP Email
- Other services - see Notifications

3. Get building with Bamboo

Bamboo has the concept of a 'plan' to look after the configuration for a build. So, to run your first build, you create and run a plan:

- Getting started with Java and Bamboo - a guide to setting up a simple CI workflow for Java code.
- Getting started with .NET and Bamboo - a guide to setting up a simple CI workflow on Windows.

Understanding the Bamboo CI Server

Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a continuous delivery pipeline.

What does this mean?

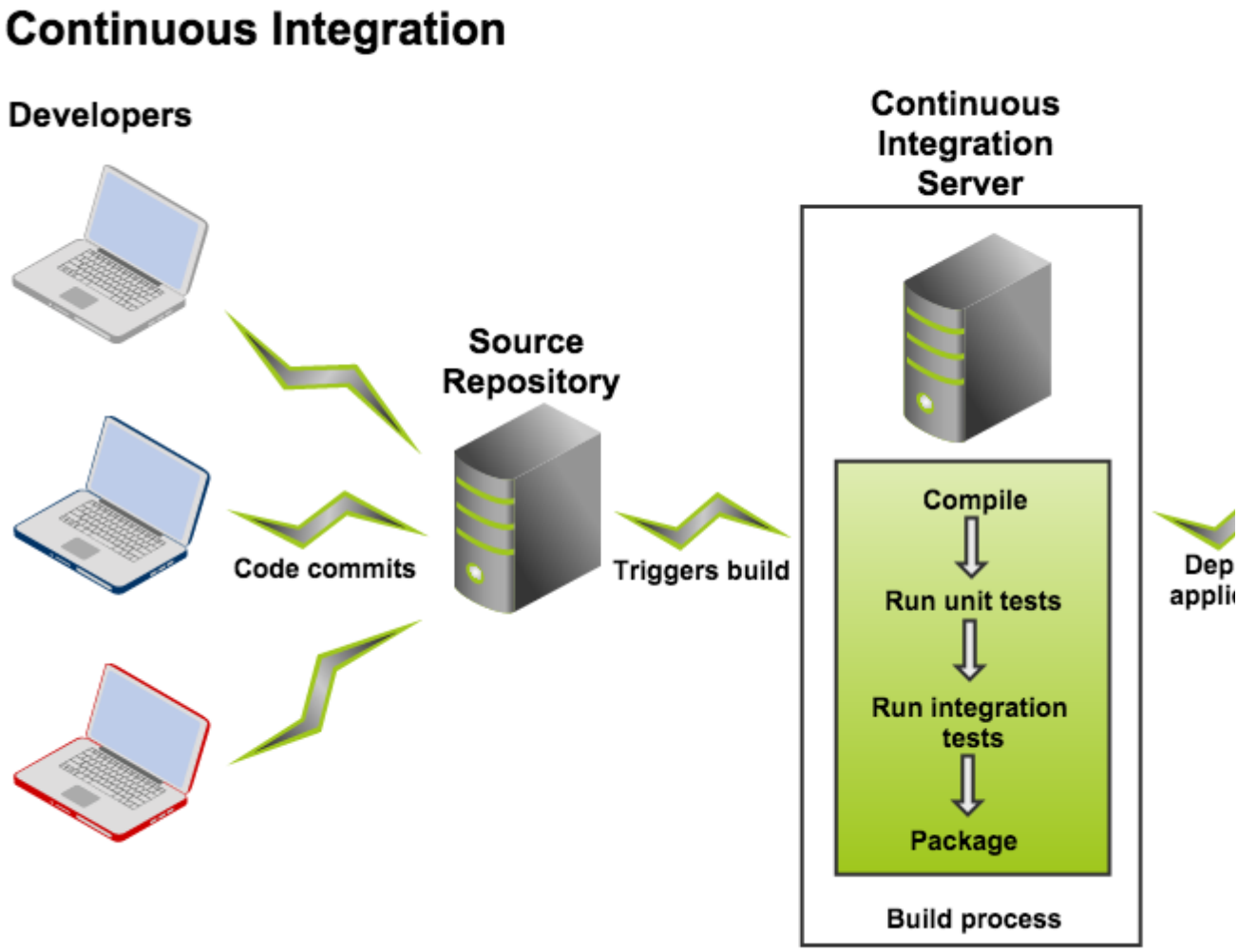
CI is a software development methodology in which a build, unit tests and integration tests are performed, or triggered, whenever code is committed to the repository, to ensure that new changes integrate well into the existing code base. Integration builds provide early 'fail fast' feedback on the quality of new changes.

Release management describes the steps that are typically performed to release a software application, including building and functional testing, tagging releases, assigning versions, and deploying and activating the new version in production.

On this page:

Related Pages:

- Getting started with Java and Bamboo
- Getting started with .NET and Bamboo
- Using Bamboo
- Installing and upgrading



What problem does Bamboo solve?

If you are a solo developer, then using Bamboo gives you:

- an automated, and therefore reliable, build and test process, leaving you free to code more.
- a way to manage builds that have different requirements or targets.
- automatic deployment to a server, such as the App Store or Google Play.

If you work in a team, then as well as the above advantages, using Bamboo also means that:

- your build and test process is not dependent on a specific local environment.
- builds and integration tests are triggered automatically as soon as a developer commits code (continuous integration).

If you work on a large, complex application, then, in addition to all the above advantages, using Bamboo means that:

- you can optimize build performance through parallelism.
- you can leverage elastic resources.
- you can deploy continuously, for example to user acceptance testing (UAT).
- you can implement release management.

How does Bamboo do this?

- Bamboo is the central management server which schedules and coordinates all work.
- Bamboo itself has interfaces and plugins for lots of types of work.
- Bamboo first gets your source from a source repository (lots of plugins here for a variety of systems).

- Then Bamboo starts the build - that can be done by calling something like MSBuild to build your Visual Studio solution, or Maven to call your compiler and linker - whatever you use.
- Once your solution or project is built, you have "artifacts" (build results, for example, an executable app, config files, etc.).
- You can do additional things with the build artifacts:
  - zip them up into a ZIP file and copy them somewhere.
  - run an install builder on them and create an MSI.
  - install them on a test server to make sure everything installs just fine.
- Bamboo provides a web front-end for configuration and for reporting the status of builds.

What does Bamboo need?

Bamboo schedules and coordinates the work involved in building and testing your application. Therefore, to use Bamboo, you will need to already have the following set up:

- a code repository that contains the complete source code for the project.
- build scripts
- test suites

It is generally assumed that the person who commits a change to the code is responsible for fixing any resulting build errors immediately.

How is a Bamboo workflow organized?

Bamboo uses the concept of a 'plan' with 'jobs' and 'tasks' to configure and order the actions in the workflow.

Project

- Has none, one, or more, plans.
- Provides reporting (using the wallboard, for example) across all plans in the project.
- Provides links to other applications.
- Allows setting up permissions for all the plans it contains

Plan

- Has a single stage, by default, but can be used to group jobs into multiple stages.
- Processes a series of one or more stages that are run **sequentially** using the **same** repository.
- Specifies the default repository.
- Specifies how the build is triggered, and the triggering dependencies between the plan and other plans in the project.
- Specifies notifications of build results.
- Specifies who has permission to view and configure the plan and its jobs.
- Provides for the definition of plan variables.

Stage

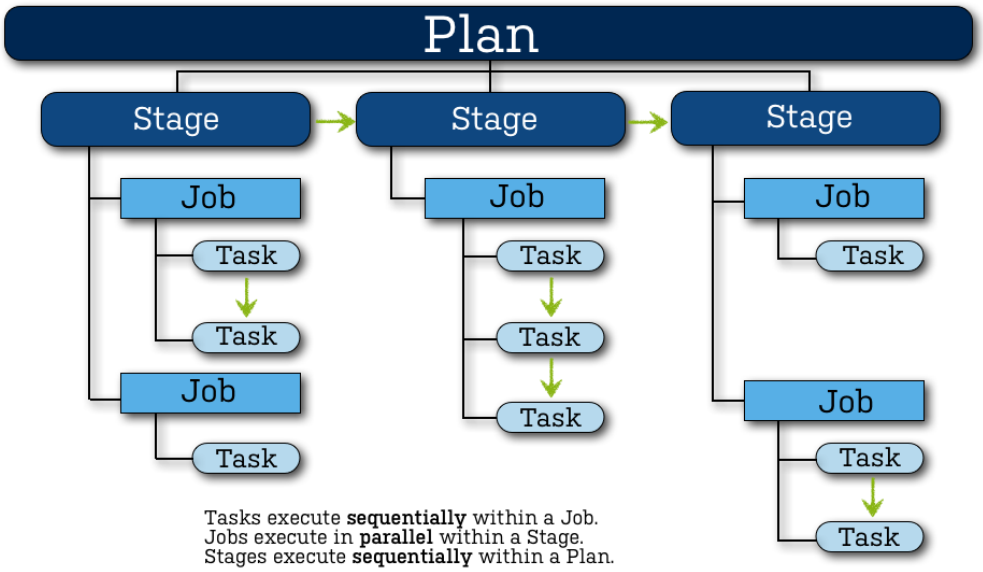
- By default has a single job but can be used to group multiple jobs.
- Processes its jobs in **parallel**, on **multiple** agents (where available).
- Must successfully complete all its jobs before the next stage in the plan can be processed.
- May produce artifacts that can be made available for use by a subsequent stage.

Job

- Processes a series of one or more tasks that are run **sequentially** on the **same** agent.
- Controls the order in which tasks are performed.
- Collects the requirements of individual tasks in the job, so that these requirements can be matched with agent capabilities.
- Defines the artifacts that the build will produce.
- Can only use artifacts produced in a previous stage.
- Specifies any labels with which the build result or build artifacts will be tagged.

Task

- Is a small discrete unit of work, such as source code checkout, executing a Maven goal, running a script, or parsing test results.
- Is run sequentially within a job on a Bamboo working directory.



AWS account for Bamboo

Create and configure your AWS (Amazon Web Services) account for smooth Elastic Bamboo setup and maintenance.

Bamboo	AWS account required	Comment
Cloud	no	Runs builds on local agents and/or in cloud (with Elastic Bamboo)
Server	no	Runs builds on local agents and/or in cloud (with Elastic Bamboo)

Creating AWS accounts

You can create an AWS root account on http://aws.amazon.com.

Cost management

The cost of all Amazon Web Services usage is billed to your AWS account, separately from your Atlassian subscription. It means that you are responsible for all AWS usage costs incurred on your AWS account.

You can check the current AWS cost in **AWS Billing & Cost Management** in the AWS management console. For more information, see What is AWS Billing and Cost Management?

> The AWS account billing doesn't distinguish between your Bamboo EC2 usage and your other (non-Bamboo) EC2 usage.

Recommendations

We respect different ways in which you might want to structure your working environment. However, we thought we'd let you know what is important from our perspective.

IAM (AWS Identity and Access Management)

For security reasons, Atlassian recommends using IAM for user and access key management. For more information, see IAM Best Practices and IAM Users and Groups.

Tips:

- Bamboo uses access keys for authorization
- Lost access keys? You must generate a new key set in the AWS account console. For more information, see How Do I Get Security Credentials?

Getting started with Java and Bamboo

This tutorial outlines how to use Bamboo to run, and get rapid feedback on, builds for your Java project. Bamboo has the concept of a 'plan' to look after the configuration for your continuous integration workflow. So, to run your first build, you'll create and run a Bamboo plan.

Information you need before you begin

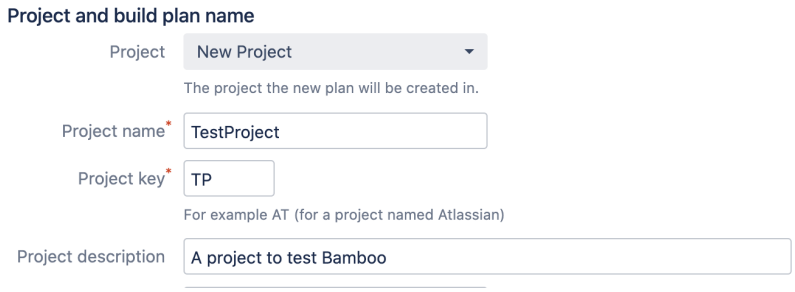
This tutorial assumes you have an individual Bitbucket account. If you don't, it only takes minutes to create one, and you can always delete it after you're done.

1. Create a project and plan

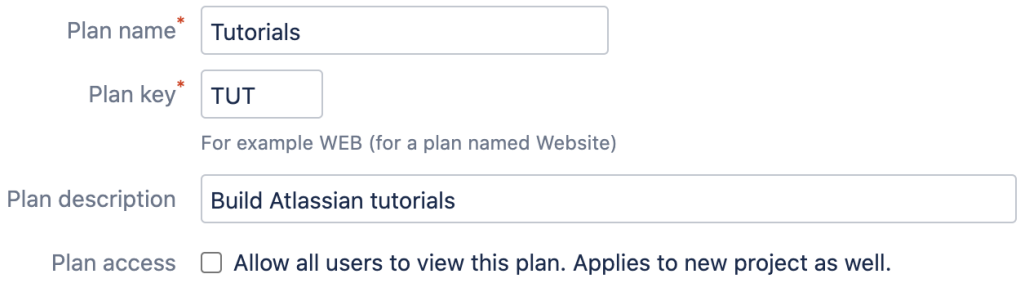
A Bamboo plan specifies the source code repository, the tasks to run in your build, and when to trigger a build. We start by creating a new plan:

1. Log into your Bamboo instance as a user with permissions to create plans.
2. Select **Create** > **Create plan** from the menu bar.

Every plan belongs to a project. We don't have a project yet, so select **Project** > **New Project**, and enter details for both the project and plan.



Bamboo needs to know the plan name, plan key and a brief description of what the plan is for.

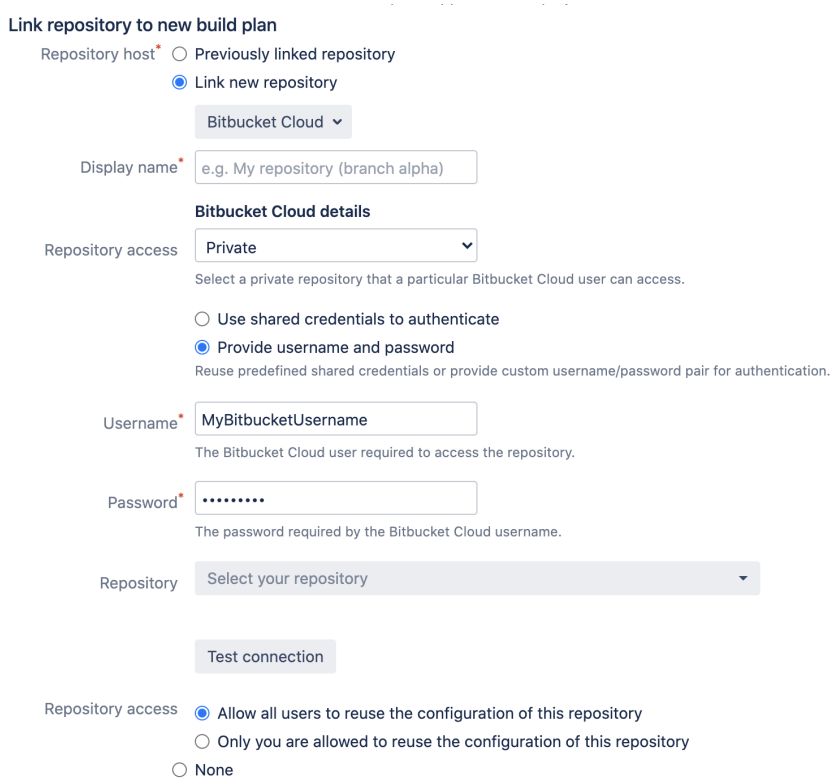


See Configuring plans for more details.

2. Connect to a source repository

Bamboo needs to know where the source code repository is located, and needs permissions to access the repo, so that it can check out the code when it runs a build. Enter your Bitbucket credentials, and select your repository.

Connect to the demonstration `atlassian_tutorial/helloworld` repo on Bitbucket for this tutorial, if you like.



See Linking to source code repositories for more details.

3. Configure tasks

Each plan needs to have at least one task specified. Tasks do the real work of the plan.

The source code checkout task

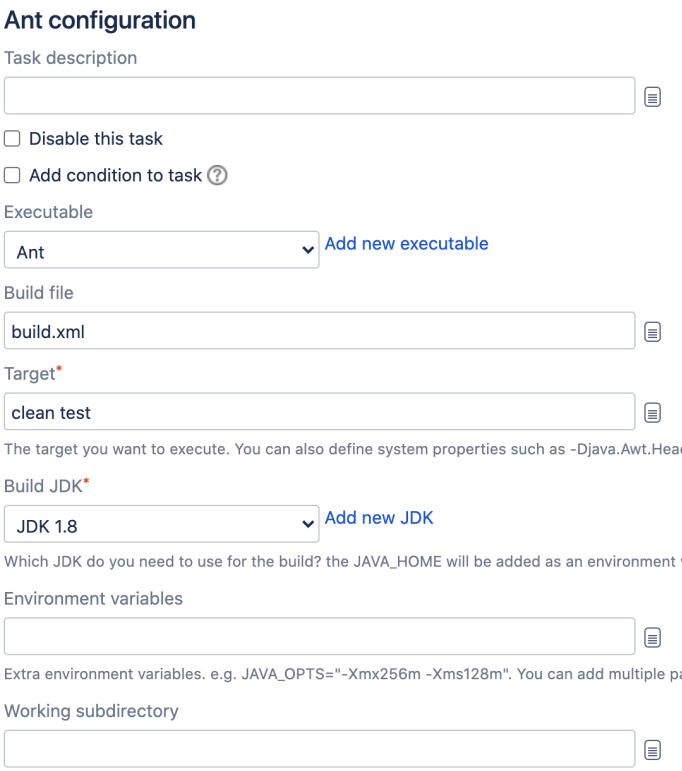
A newly created plan has a default Source Code Checkout task that gets the source code from the source repository specified earlier.

See Checking out code for details.

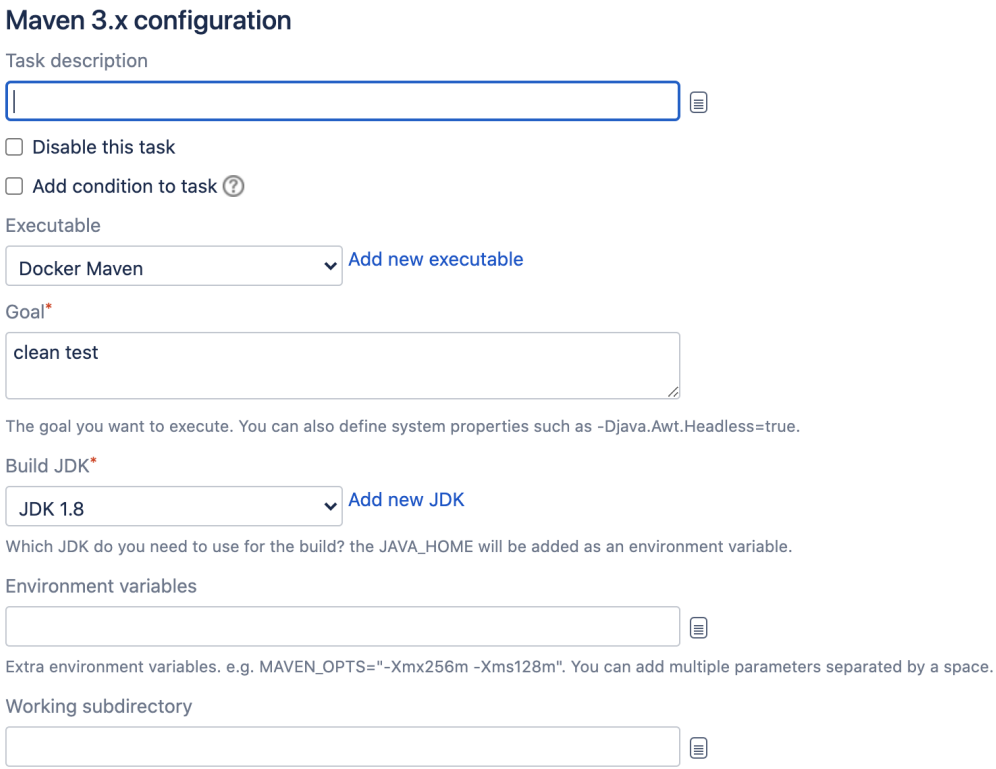
The builder task

We also want to compile the code, and run the unit and integration tests. We'll add a builder task to the Bamboo plan to do that. We assume that your project already has a build process set up that Bamboo can call.

Select **Add Task**, then **Builder** and choose the task that matches the build tool for your project. Expand one of the following sections to see configuration details specific to that builder task:

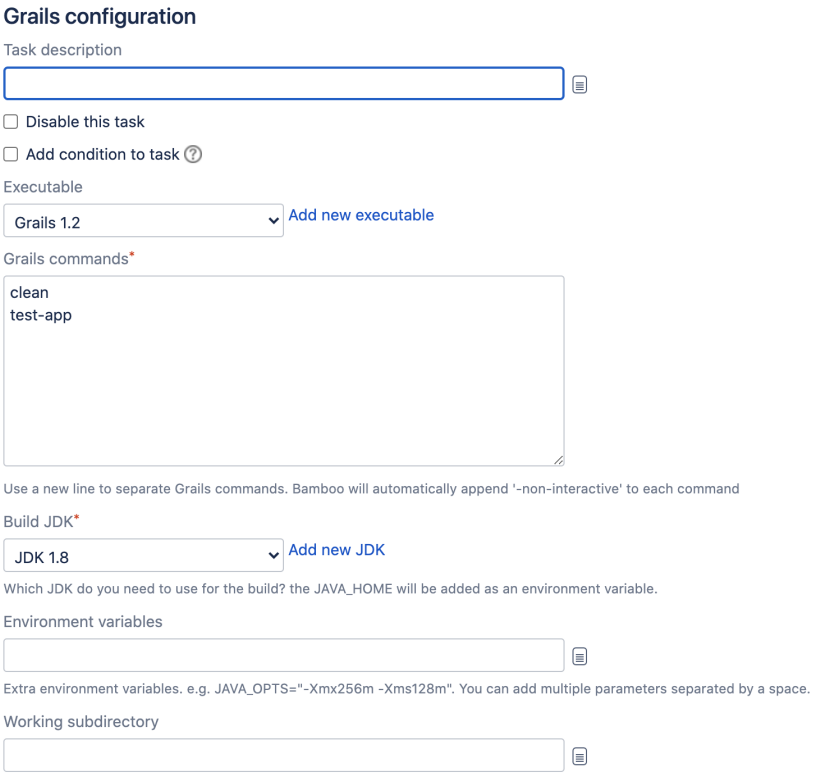


See http://ant.apache.org/manual/index.html for information about Ant.



Bamboo also supports Maven 1.0 and Maven 2.0.

See http://maven.apache.org/ for information about Maven.



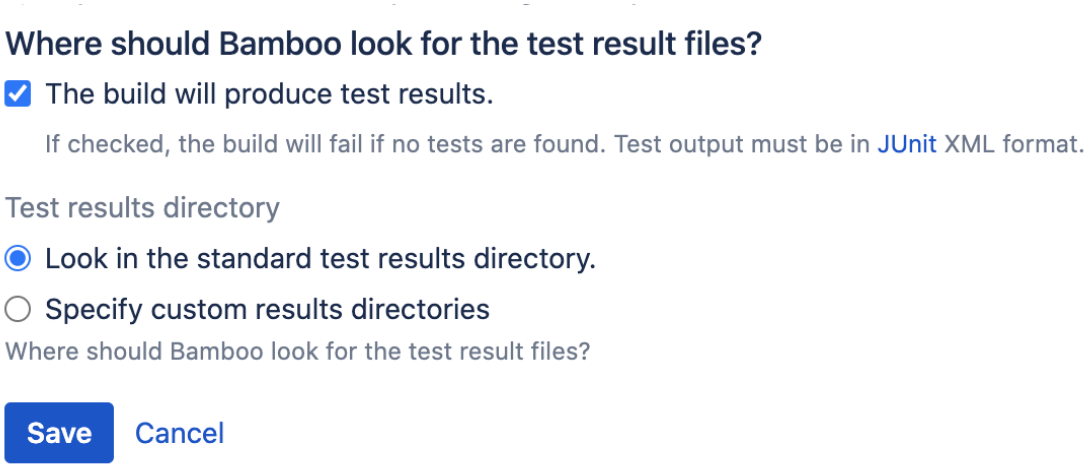
See http://grails.org/doc/latest/guide/index.html for information about Grails.

Note that:

- A build tool needs to be installed on the Bamboo server machine before you can use the Bamboo task.
- There are plugins available for Bamboo that add build tasks for other tools, such as Gant and Gradle. See the Atlassian Marketplace for details.

Getting the test results

Your tests will be run when the builder task compiles the code. Each of the builder tasks above has a section to tell Bamboo to expect test results and where to look for them. You can specify a custom results location if your project directory doesn't use the conventional structure.



See Configuring jobs and Configuring tasks for details.

5. Run!

Enable the plan, and click **Create**.

You should see the plan run. Bamboo will:

- Connect to the code repository
- Check out the source code
- Compile the code
- Run unit and integration tests
- Report back the test results

The 'Plan summary' tab will report whether the build succeeded or not.

Tests in the appropriate directory in the source code repository will be run automatically as part of the build, and the test results will be displayed in Bamboo.

Now, whenever you commit a change to the repository, Bamboo will build your source code and report on your test results.

6. Get feedback

Bamboo displays a summary of the results of the build on the dashboard.

You can get further information about the build in the following ways:

- Build results for one or more plans can be displayed on a wallboard.
- You can get notifications about build results sent to you by email, IM and RSS feed.
- You can get build statistics about plans, and about developers contributing code to the build.
- You can drill down into the results to see the code changes that triggered the build, and the tests that were run for that build.

See Getting feedback for details.

Getting started with .NET and Bamboo

This page describes how your development team can start using the Bamboo continuous integration server to get rapid feedback on your .NET project.

You may want to read Understanding the Bamboo CI Server first.

We assume that you already have:

- Bamboo installed and running. See Installing and upgrading for details. You'll want user accounts in Bamboo for each member of your team.
- Source code under version control. Each team member will have access to the repository.
- Tests, as part of the source code for the project.
- A command that builds the code and executes the tests.

The continuous integration workflow we want is:

1. A developer commits code.
2. Bamboo builds the project:
    a. Connects to the repository and checks out the source code.
    b. Compiles the code.
    c. Runs the unit and integration tests.
3. Bamboo provides feedback on the test results.

How do we achieve this with Bamboo?

Well, we'll create a new Bamboo plan that knows how to check out and build our source code, and then report on our test results.

On this page:

Related pages:

- Getting started with Java and Bamboo

Create a Bamboo plan

A Bamboo plan is where you define the details of your continuous integration workflow.

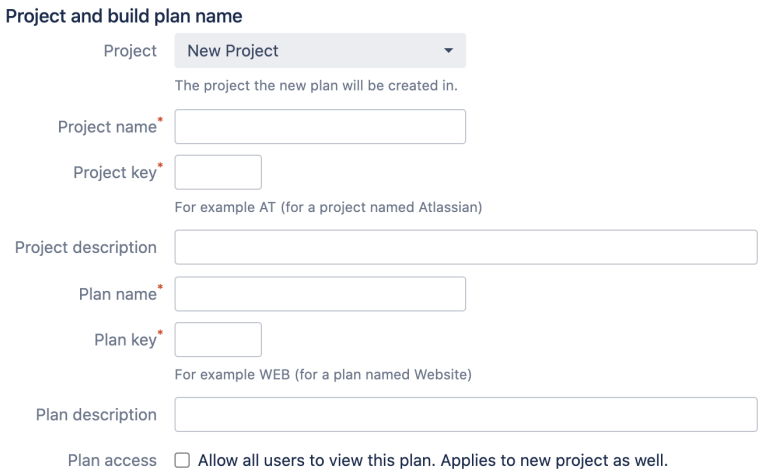
A plan allows us to specify a source code repository, when Bamboo gets triggered to run the build, and how Bamboo should provide feedback on the test results.

1. Plan details

Select **Create** in the menu bar, and then **Create plan**.

Every plan belongs to a project. We don't have a project yet, so select **Project** > **New Project**, and enter details for both the project and plan.

See Configuring plans for details.

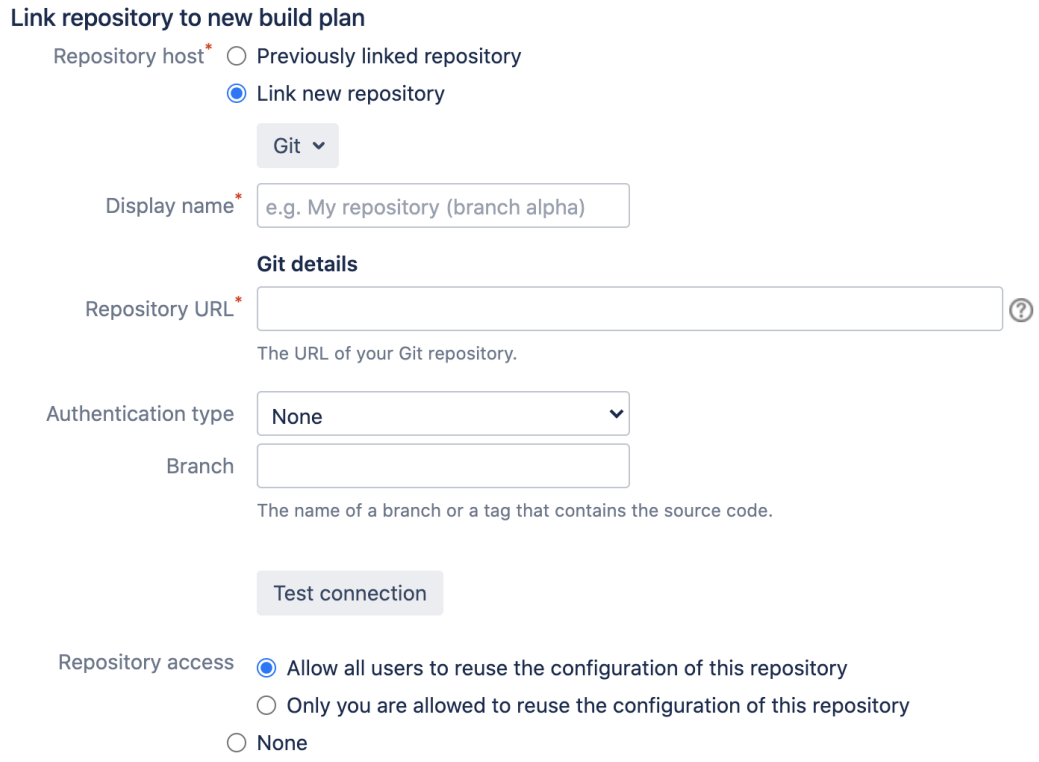


2. Choose a source repository

Bamboo needs to know where the source code repository is located, and needs access to the repo so that it can check out the code when it runs a build.

Select the repository host, and provide access details such as username and password.

See Linking to source code repositories for details.



3. Configure tasks

Each plan needs to have one or more tasks specified. Tasks do the real work of the plan.

The source code checkout task

A newly created plan has a default Source Code Checkout task that gets the source code from the source repository specified earlier.

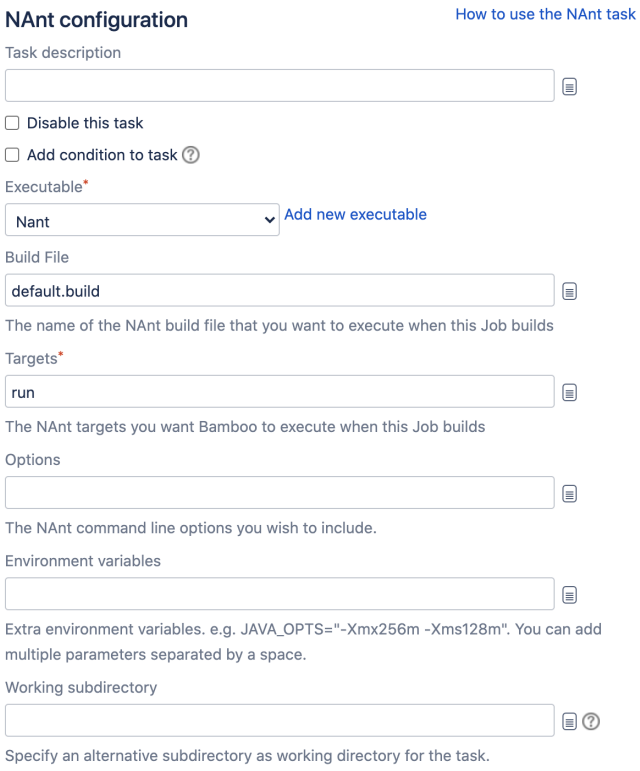
See Checking out code for details.

The builder task

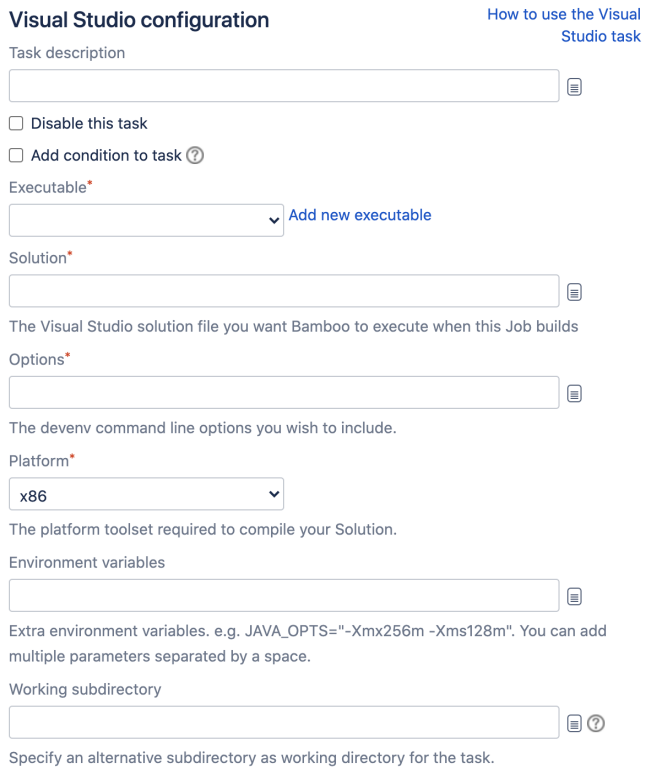
We also want to compile the code. We'll add a builder task to the Bamboo plan to do that. We assume that your project already has a build process set up that Bamboo can call upon.

Select **Add task**, then **Builder** and choose the task that matches the build tool for your project. Expand one of the following sections to see configuration details specific to that builder task:

See http://msdn.microsoft.com/en-us/library/ms171452%28v=vs.90%29.aspx for information about MSBuild.



See http://nant.sourceforge.net/ for information about NAnt.



See http://www.microsoft.com/visualstudio for information about Visual Studio.

Note that a build tool needs to be installed on the Bamboo server machine before you can use the Bamboo task.

See Configuring a builder task for details.

Getting the test results

Now we want to run the unit and integration tests, and display the results from those. You need to set up one of the MSTest, NUnit or MBUnit tasks so Bamboo can get and display the test results. You can specify a custom results location if your project directory doesn't use the conventional structure.

See Configuring a test task for details.

5. Go!

Enable the plan, and click **Create**.

You should see the plan run. The 'Plan summary' tab will report whether the build succeeded or not.

Tests in the appropriate directory in the source code repository will be run automatically as part of the build, and the test results will be displayed in Bamboo.

Now, whenever you commit a change to the repository, Bamboo will build your source code and report on your test results.

Get feedback

Bamboo displays a summary of the results of the build on the dashboard.

You can get further information about the build in the following ways:

- Build results for one or more plans can be displayed on a wallboard.
- You can get notifications about build results sent to you by email, IM and RSS feed.
- You can get build statistics about plans, and about developers contributing code to the build.

- You can drill down into the results to see the code changes that triggered the build, and the tests that were run for that build.

See Getting feedback for details.

Getting started with PHP and Bamboo

This page describes how to use Bamboo to get rapid feedback on your PHP project. The worked example builds a Bamboo plan where a developer commits code and Bamboo responds by:

- Connecting to the code repository
- Checking out the source code
- Compiling the code
- Running unit and integration tests
- Reporting back test results

On this page:

Related pages:

- Getting started with .NET and Bamboo

Information you need before you begin

This introduction assumes you are using Bamboo Server installed on your local network. You need to make sure you or your company administrator have properly installed and configured Bamboo for running plans.

You will also need to install:

- The PHP framework
- PHPUnit testing framework

Step 1: Install the PHP base code framework

In order to get full functionality from Bamboo and PHP, you will need to install the PHP base code framework. If you are using Ubuntu, then use the following command to install PHP.

```
$ sudo apt-get install php5-cli
```

See also:

- Installing PHP on MacOS
- Installing PHP on Windows

Step 2: Install PHPUnit

PHPUnit.de provides an excellent PHP archive resource called PHAR.

```
$ wget https://phar.phpunit.de/phpunit.phar  // download the PHPUnit packages
$ chmod +x phpunit.phar  // make PHPUnit executable
$ mv phpunit.phar /usr/local/bin/phpunit  // copy PHPUnit into your path
$ phpunit --version  // double check it's installed completely
```

If you prefer, you may use Composer or PEAR to download and install PHPUnit along with its dependencies, however these approaches are beyond the scope of this introduction.

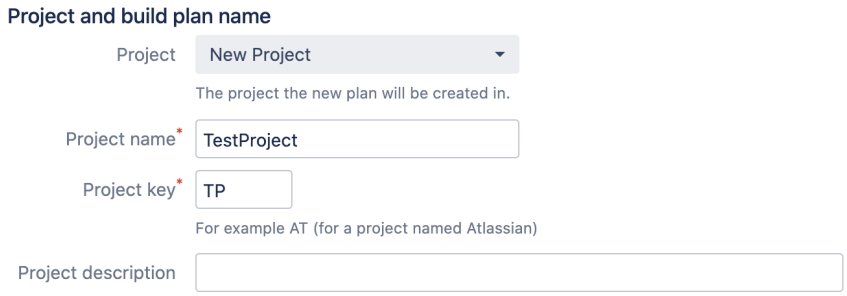
Step 3. Create a project and plan

1. Create a new project

A Bamboo plan defines the details of your continuous integration workflow. You use a plan to identify the source code repository, specify the tasks to run in your build, and when to trigger a build. Each plan belongs to a project. You can add a plan to an existing project or create a new project. In this example, you create both a new project and a new plan in that project.

1. Log into your Bamboo instance as a user with permissions to create plans.
2. Select **Create** > **Create plan** from the menu bar.

Every plan belongs to a project. We don't have a project yet, so select **Project** > **New Project**, and enter details for both the project and plan.



Project

New Project

Project Name

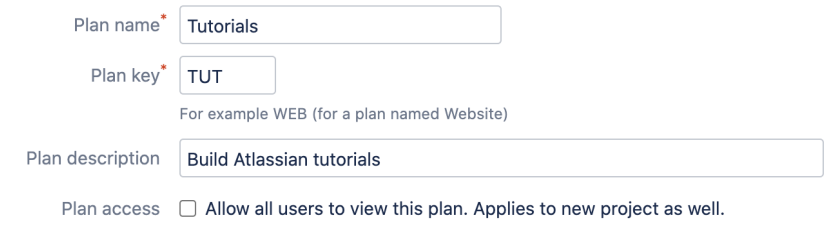
`TestProject`

Project Key

TP

2. Configure the plan details

Bamboo needs to know the Plan name, Plan key and a brief description of what the plan is for. See Configuring plans for more details.



Plan name

Tutorials

Plan key

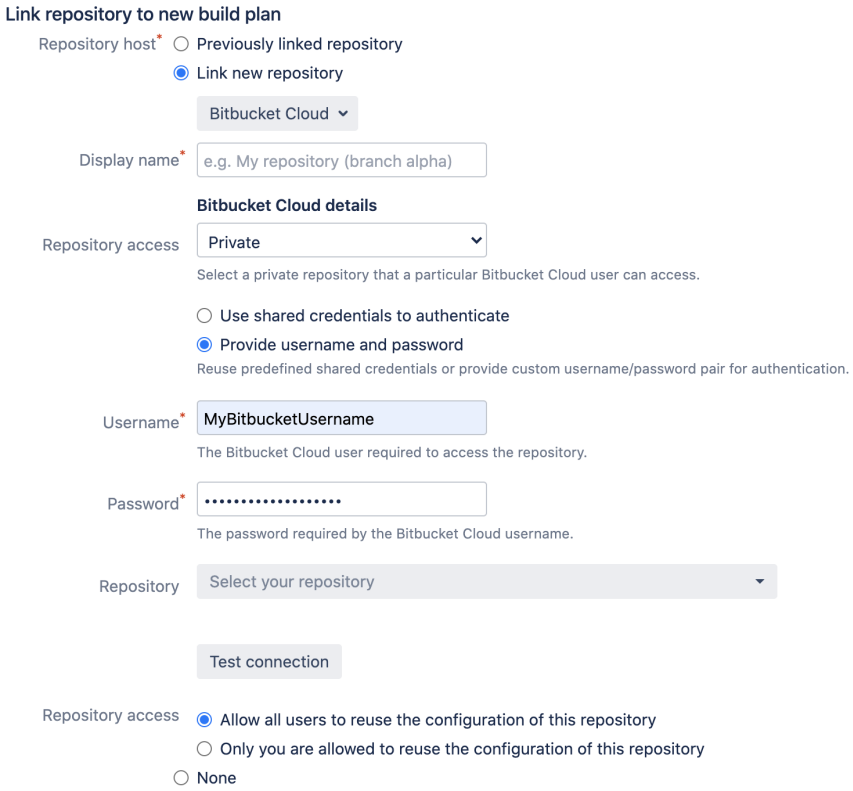
TUT

Description

Build Atlassian tutorials

3. Choose a source repository

Bamboo needs to know where the source code repository is located, and needs access to the repo so that it can check out the code when it runs a build. See Linking to source code repositories for more details.



Source Repository

Bitbucket

Username

Your Bitbucket username

Password

Your Bitbucket password

Repository

`atlassian_tutorial/hellworld (git)`

Branch

`master`

Step 4. Configure tasks

Each plan needs to have at least one task specified. Tasks do the real work of the plan.

The source code checkout task

A newly created plan has a default Source Code Checkout task that gets the source code from the source repository specified earlier.

See Checking out code for details.

Unit testing

Unit testing for PHP is completed using the PHPUnit testing framework. This is a port of the popular Java JUnit testing framework to PHP. PHPUnit provides also produces test results in the JUnit XML format required by Bamboo.

You will need to add a server executable capability to run PHPUnit:

1. Go to **Overview** > **Server capabilities**.
2. Select **Add capability** and complete the configuration using the following:

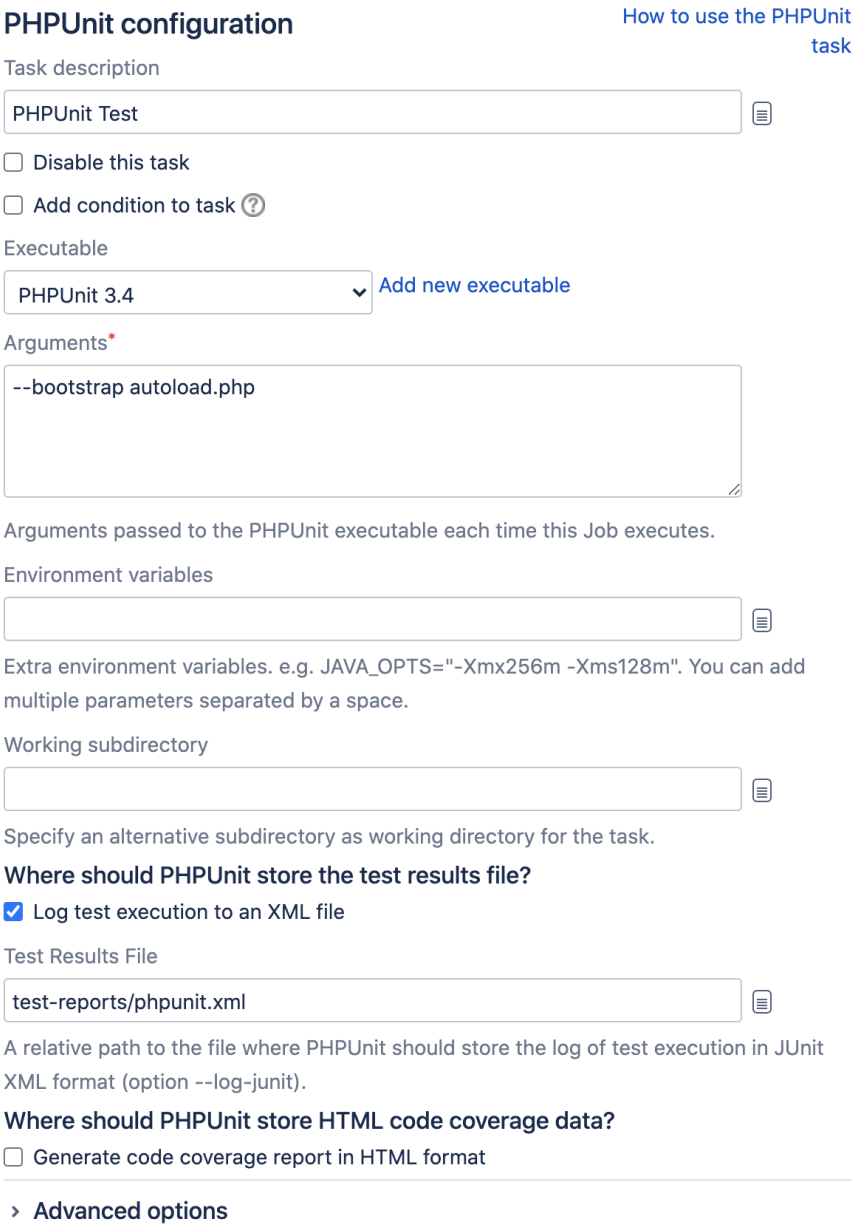
   Capability type
   Executable

   Type
   PHPUnit

   Executable label
   PHPUnit x.x

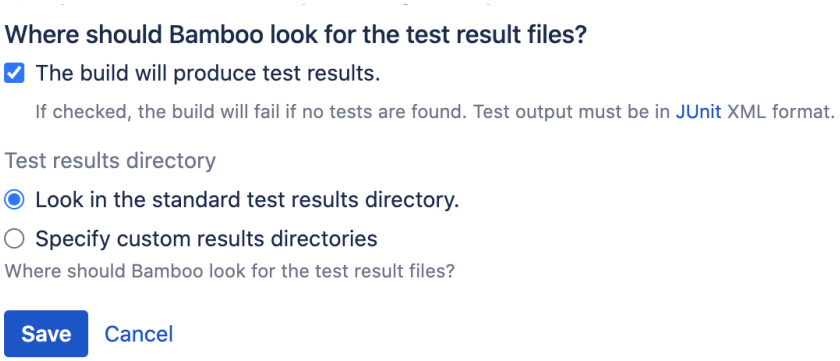
   Path
   Path to the PHPUnit executable e.g. /usr/bin/phpunit-x.x
3. Select **Add** to add the PHPUnit capability.

Now you can create a PHPUnit testing task:



Getting the test results

Your tests will be run when the builder task compiles the code. Each of the builder tasks above has a section to tell Bamboo to expect test results and where to look for them. You can specify a custom results location if your project directory doesn't use the conventional structure.



See Jobs and tasks for details.

Step 5: Go!

Enable the plan, and select **Create**.

You should see the plan run. The 'Plan summary' tab will report whether the build succeeded or not.

Tests in the appropriate directory in the source code repository will be run automatically as part of the build, and the test results will be displayed in Bamboo.

Now, whenever you commit a change to the repository, Bamboo will build your source code and report on your test results.

Get feedback

Bamboo displays a summary of the results of the build on the dashboard.

You can get further information about the build in the following ways:

- Build results for one or more plans can be displayed on a wallboard.
- You can get notifications about build results sent to you by email, IM and RSS feed.
- You can get build statistics about plans, and about developers contributing code to the build.
- You can drill down into the results to see the code changes that triggered the build, and the tests that were run for that build.

See Getting feedback for details.

Using the Bamboo dashboard

The dashboard is your Bamboo 'home' page. The dashboard has three tabs:

- **My Bamboo**— a convenient summary of information that is relevant to you (only appears if you have logged in to Bamboo):
  - plans that you have nominated as your favorites.
  - your latest build results (i.e. builds that were triggered by your latest code changes).
  - a summary of your build statistics (only appear if your Bamboo User Profile has been associated with your Author Name.
- **All build plans** — a list of plans and each plan's latest build result.
- **Build Activity** — Bamboo's agents and build queue, showing which plans Bamboo is currently building and which plans are waiting to be built.

You can return to the dashboard from anywhere in Bamboo by clicking **Dashboard** in the top navigation menu.

On this page:

Related pages:

- Configuring plans
- Working with builds
- Getting feedback

Viewing the dashboard

You can:

- select the **project name** (e.g. 'Bamboo Testing') to view the plans in the project.
- select the **plan name** (e.g. 'Acceptance Test JDK 1.6') to view the plan details.
- select the **build number** (e.g. '7823') to view the build result.
- select the **author's name** to view the author's details (the author is the person who triggered the build by checking-in code).

The icon next to a build number indicates the plan's current status:

This plan's latest build was successful.

This plan's latest build failed.

Bamboo is currently checking-out the source-code for this plan, in preparation for starting a build.

Bamboo is currently queuing a build for this plan in the Build Queue.

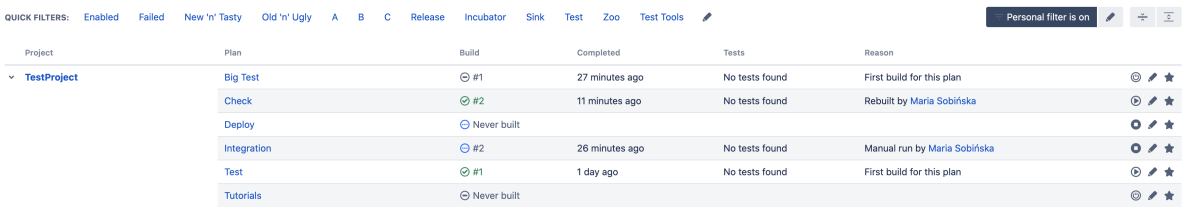
Bamboo is currently executing a build for this plan.

The plan is stopped at a manual stage.

The plan was not built, perhaps because the build was manually stopped.

This plan has been disabled.

Screenshot: Bamboo dashboard - 'Build > All build plans" tab



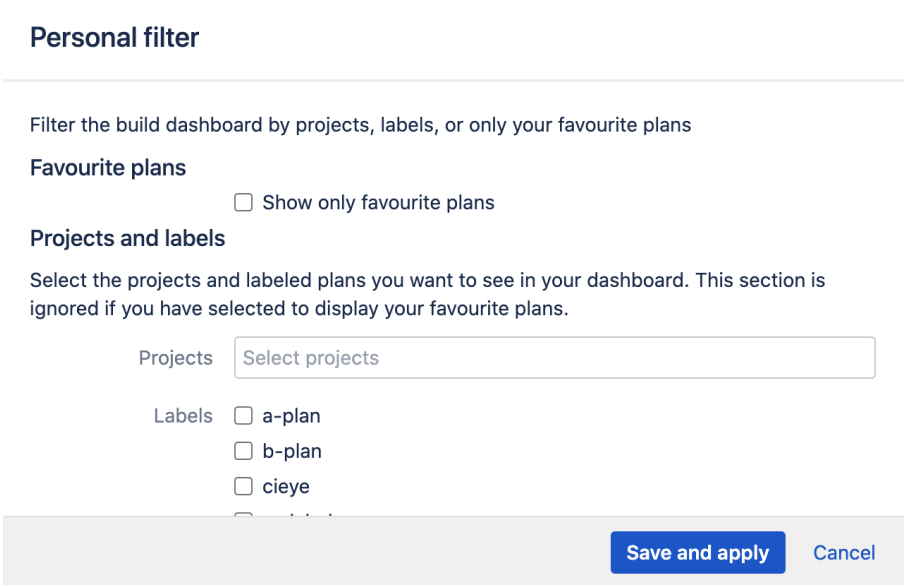
Filtering the plans

You can filter the plans on your dashboard by projects, labels or favourite plans (available in Bambo 6.10.5, 7.0.3, and later). For instructions on how to add a label to a plan, see Working with labels.

To filter the dashboard plans

1. In the build dashboard, click the  button.
2. In the Personal filter dialog, choose your filtering criteria:
   - filter by favorite plans (available in Bambo 6.10.5, 7.0.3, and later)
   - filter by projects and labels
3. Click **Save and apply**.
   The dashboard will refresh, showing only plans that match your filtering criteria.

Screenshot: Filtering plans on a dashboard



Working with favorites

The **My Bamboo** tab lists your *favorite* plans — that is, the plans you work with the most. You can easily add and remove plans from your favorites.

When you add a plan to your favorites, you become a 'watcher' of the plan. This means that you will receive notifications about the build results for your favorite plans, depending on how your administrator has configured each plan's notifications. You can receive notifications by email, Instant Messaging (IM) and RSS feed.

To add a plan to your favorites:

1. Select **Dashboard** in the top navigation bar to display the dashboard.
2. Select the **All Plans** tab. This will display a list of all plans in your Bamboo system.

3. Locate the plan and select the grey star icon at the right.

Viewing Bamboo's agents

A Bamboo *agent* is a service that can run job builds. There are the following types of Bamboo agents:

- *local agents* run as part of the Bamboo server.
- *remote agents* run on computers, other than the Bamboo server, that run the remote agent tool.
- elastic agents run in the Amazon Elastic Compute Cloud (EC2).

Local agents run in the Bamboo server's process, i.e. in the same JVM as the server. Each remote agent runs in its own process, i.e. has its own JVM.

Each agent has a defined set of capabilities and can only run builds for jobs whose requirements match the agent's capabilities.

To view agents that are currently active, see Using the Bamboo dashboard.

Related pages:

- Configuring agents
- Bamboo remote agent installation guide

View the agents in Bamboo

1. Choose **Build** > **Build activity** from the Bamboo header.
2. Click the name of the agent in the 'Building' section to see details for that particular agent.
3. Click **X of Y online agents building** in the 'Building' section of the page to see a list of all available agents.

View a specific Bamboo agent as a Bamboo administrator

1. Choose **Agents** from the 'cog' menu of the Bamboo header.
2. Click the name of the agent. You can configure this agent and its capabilities:
    - Click **Executable Plans** to view the plans that this agent can build.
    - Click **System Properties** to view the system properties of this agent.

Keyboard shortcuts

What are you doing?	Keyboard Shortcut	Action
Viewing any screen	Alt / Cmd + c	Opens Create menu at Create new plan
Viewing any screen	Alt / Cmd + u	See author report
Viewing the Dashboard	l	Filter projects
Viewing a plan or build	e	Edit the plan configuration
Viewing a build	Alt / Cmd + p	Previous build
Viewing a build	Alt / Cmd + n	Next build
Viewing a build	l	Label
Viewing a build	m	Write a comment
Viewing a build	Alt / Cmd + s	Save the comment
Editing a task	Alt / Cmd + s	Save the task

Getting started with Node.js and Bamboo

Node.js is described as:

> *"a platform built on Chrome's JavaScript engine for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." Node.js*

We agree, and bundle a number of tasks with Bamboo to make it easy for you to get continuous integration and deployment for your Node.js projects. You can find the official Node.js documentation here.

Distributions

Node.js distributions usually come bundled with npm, a package manager for the platform, which runs from the command line and manages dependencies for your applications.

All npm packages contain a file, usually in the project root, called `package.json` - this file holds metadata relevant to the project. You can find out more about the package.json file here.

Configure your Node.js project

Add the following dependencies (or devDependencies) to the `package.json` file in your Node.js project. These are required if you want to use the Grunt, Gulp, Bower, Nodeunit or Mocha Test Runner tasks:

Grunt

- grunt (v0.4 or newer)
- grunt-cli

Gulp

- gulp (v3.3.2 or newer recommended)

Bower

- bower

Mocha Test Runner

- mocha
- mocha-bamboo-reporter

Nodeunit

- nodeunit

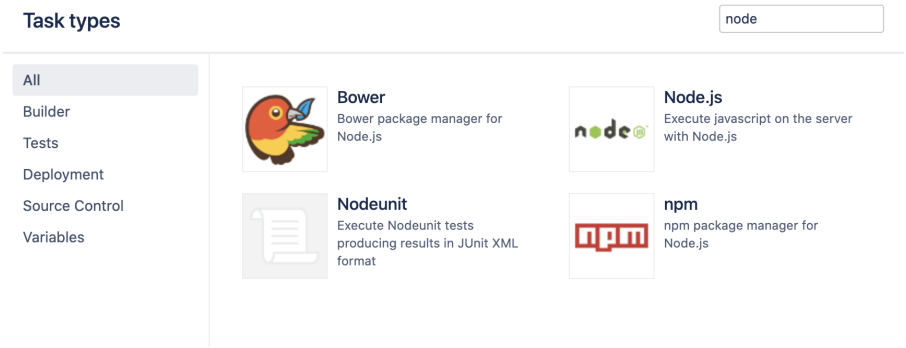
Your package.json file should look something like this:

```
...
"devDependencies": {
    ...
    "mocha": "~1.18",
    "mocha-bamboo-reporter": "*",
    ...
    "nodeunit" : "~0.8"
}
"dependencies": {
    ...
    "grunt": "~0.4",
    "grunt-cli": "~0.1",
    ...
    "gulp": "~3.3.2",
    ...
    "bower": "~1.3.12"
}
```

Install the necessary `node_modules` before executing any of the Node.js tasks, by adding an npm task and using the `install` command.

Node.js tasks

Bamboo ships with specific tasks for Node.js that make it easy to integrate the Node.js platform with Bamboo. You can use these tasks to set up builds for your Node.js project.

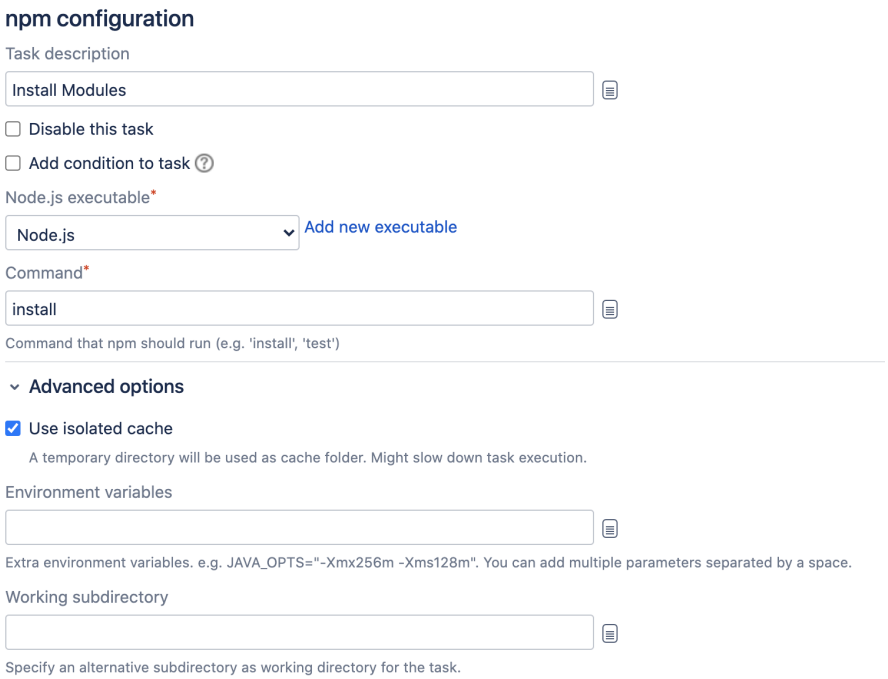


Note that it is possible to execute scripts installed by the npm task from the `node_modules`; however, we recommend that you use the dedicated tasks for executing such scripts, such as Grunt, Mocha or Nodeunit.

Install the necessary `node_modules` before executing any of the Node.js tasks, by adding an npm task and using the `install` command.

npm task

The npm task allows you to execute Node Package Manager commands in build plans and deployment projects. To run npm commands, simply enter the command to execute during task configuration:

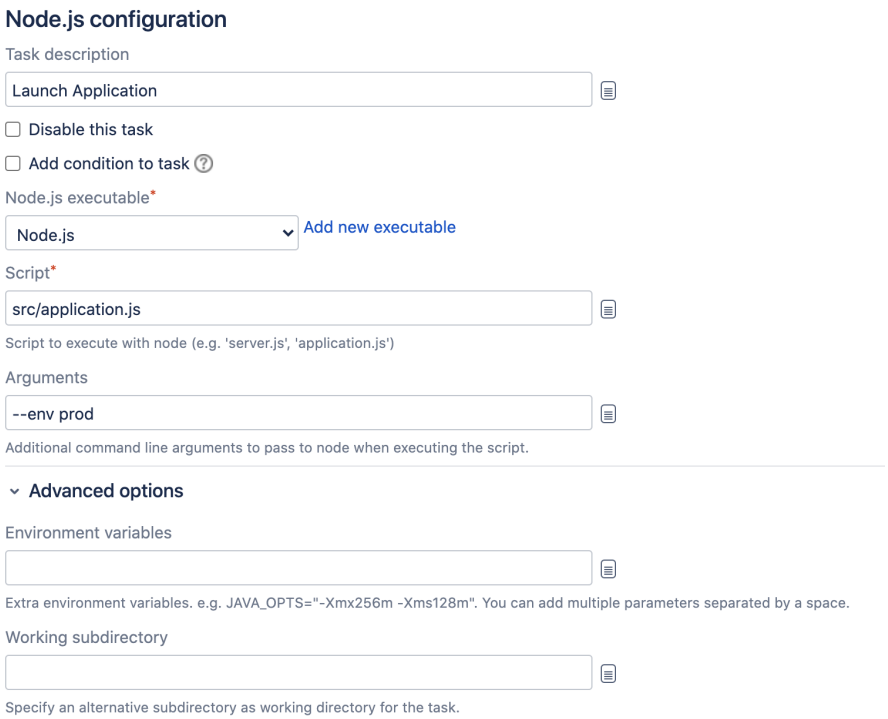


In order to execute npm commands, the Node.js capability must be present on your build agent (see below).

Note: since Node.js and npm are distributed together, Bamboo will use the Node.js capability for npm tasks as well. The path will be modified at run time to point to the npm executable.

Node.js task

The Node.js task is a general purpose task that can be used to execute Node scripts within Bamboo.



To run the Node.js task, the Node.js capability must be present on your local or remote agents (see below).

Node.js can be used to execute any custom Node.js scripts or applications. To do so, enter the path of the script to execute in the task configuration, and optionally define additional arguments to pass.

Note that it is possible to execute scripts installed by the npm task from the `node_modules,` however we recommend that you use the dedicated tasks for executing such scripts, such as Grunt, Mocha or Nodeunit.

Mocha

Mocha is a test framework that runs on the Node.js platform.

You can use the Mocha Test Runner task to run your Mocha tests – it will create an output file named `mocha.json`.

You can configure the task to parse test results after a successful execution. Alternatively, you can add a Mocha Test Parser task to run afterwards to parse the test results.

If you don't do a full checkout on each build, make sure you add a task to delete `mocha.json` *before* the Mocha Test Runner task. A simple script task that runs `rm -f mocha.json` should do the trick.

Grunt

Use the Grunt task to take advantage of the Grunt task runner.

Nodeunit

Nodeunit is a tool for defining and running unit tests for Node.js projects.

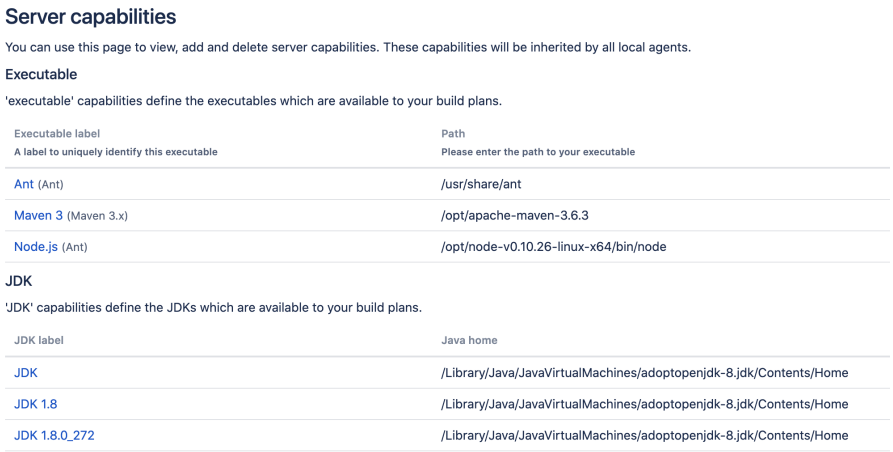
Running the Nodeunit task will create test results in JUnit XML format.

You can configure the task to parse test results after a successful execution. Alternatively, you can add a following JUnit Parser task to parse the test results.

Node.js capability

Bamboo comes with a definition for a new executable capability called Node.js. In order to use the Node.js task (as well as most of the other Node.js tasks in Bamboo), you need this capability to be present on your local or remote agents.

The capability can be auto detected on the server side.



The `NODE_HOME` environment variable will instruct Bamboo where to look for Node.js if it is not installed in a typical directory - it should point to the location of the Node.js installation. Bamboo will search the agent's default directories to find the Node.js installation; it will also search in location specified by `NODE_HOME`.

Getting started with Docker and Bamboo

Docker is an operating system container technology that allows running applications in isolated environments called containers. Docker containers are equipped with all dependencies required by an app to run. The container abstraction provides many benefits such as: the creation of reproducible environments, redistribution of full application stacks for running on different machines, or limiting the resource consumption of an application running in a container.

The Docker container technology and Bamboo can interact in the following ways:

Run Bamboo in Docker

Bamboo Docker images allows you to run the Bamboo Server or the Bamboo remote agent inside a Docker container. This makes it isolated from other applications that may be running on the same host system. In this type of interaction Bamboo is itself unaware of the existence of Docker container it is running in.

To get Bamboo up and running quickly, we have prepared Bamboo Server and Bamboo Agent Docker images. Both are minimalistic and highly customisable images that allow you to get Bamboo ready for action in no time. These images provide a fully controllable and reproducible environment which makes them perfect for testing purposes. Both images are designed to be easily extensible, allowing you to add capabilities needed to run your builds quickly. You can see an example how to extend an image to suit your needs on our Docker Hub space.

> To make your life easier and allow you not to worry which specific Bamboo version to run, we tag our images both with point and minor version numbers. In order to run latest version of Bamboo 6.7, you only need to use it as a version tag: atlassian/bamboo-server:6.7. This will start a container with the latest 6.7.x stable version available. Of course you can still start specific version of Bamboo, by using whole version tag: atlassian/bamboo-server:6.7.1

Also if you're considering implementing Bamboo in your environment, you can use our Docker images for Bamboo evaluation - just run the Docker and see if Bamboo is the tool for you.

Go to the Docker Hub for the images and instructions how to set them up:

- Bamboo Server
- Bamboo Agent base

Run Bamboo jobs in Docker

Docker Runner is a Bamboo feature that allows the user of Bamboo to run jobs of Bamboo plans inside Docker container environments. In this type of interaction, Bamboo is aware of the Docker container technology and communicates with it to create, manage, and terminate container environments in which jobs are run.

With Docker Runner, you can run builds and deployments in a Docker container to isolate the build process from the environment where the Bamboo build agent runs in. This increases the reliability of your environment by providing isolation, and more strict control over the resources the continuous integration (CI) process has access to. Also, it gives you possibility to recreate the same build environment at different moments in time. The isolation also helps with the reliability of your CI by making sure that environment it runs in can be reliably recreated each time you run your builds.

For information, see Docker Runner.

Use Docker in your Bamboo tasks

A Docker task is a specific task type in Bamboo which allows you to use your own, custom Docker image with Bamboo to run a task of a job inside the Docker container environment. In this type of interaction Bamboo is aware of the Docker container technology and, similarly to the Docker Runner feature, it interacts with it to create, manage, and terminate container environments in which one single task is run.

The main difference between the two is that one works at the level of the Bamboo task, which runs a single task inside the container, while the other works at the level of the Bamboo job, which runs all the tasks that make up the job inside the same Docker container environment.

To learn how to start using these tasks, see Configuring the Docker task in Bamboo.

Using Bamboo in the enterprise

Atlassian Bamboo is a continuous integration (CI) and deployment server. Bamboo assists software development teams by providing:

- Automated building and testing of software source code.
- Integration with your existing development environment.
- Status updates on successful and failed builds.
- Automated releases and deployments.
- Reporting tools for statistical analysis.

This page describes best practice for using Bamboo in enterprise environments, that is with 500+ user licenses. Of course, much of this information is also applicable to other Bamboo installations.

On this page:

Platform requirements for hosting Bamboo

Although Bamboo can be run on Windows, Linux and Mac systems, for enterprise use we only recommend, and support, Linux. This recommendation is based on our own testing and experience with using Bamboo.

Please see the Supported platforms page for details of the supported versions of Java, external databases and web browsers.

Performance considerations with Bamboo

In general, Bamboo is ...

High availability with Bamboo

If Bamboo is a critical part of your development workflow, maximizing Bamboo availability becomes an important consideration. Please see...

Setting up Bamboo in a production environment

When setting up Bamboo for a production or enterprise environment, we highly recommend that you configure the following aspects:

Run Bamboo as a dedicated user

- For production environments Bamboo should be run from a dedicated user account with restricted privileges. See...

Install Bamboo as a service

See:

- Running Bamboo as a Linux service
- Running Bamboo as a Windows service

Use an external database

- For production environments Bamboo should use an external database, rather than the embedded database. Set up your external DBMS (for example MySQL) before starting Bamboo for the first time. This allows you to connect Bamboo to that DBMS using the Setup Wizard that launches when you first run Bamboo. See Connect Bamboo to an external database.

Connect to your existing user directory

- Connect Bamboo to your existing user directory (for example Active Directory). See External user directories.

Secure the Bamboo home directory

- For production environments the Bamboo home directory should be secured against unauthorized access. See...

Secure Bamboo with HTTPS

- Access to Bamboo should be secured using HTTP over SSL, especially if your data is sensitive and Bamboo is exposed to the internet. See Securing Bamboo with HTTPS.

Change the context path for Bamboo

- If you are running Bamboo behind a proxy, or you have another Atlassian application (or any Java web application), available at the same hostname and context path as Bamboo, then you should set a unique context path for Bamboo. See Changing Bamboo's root context path.

Administering Bamboo in a production environment

Upgrading Bamboo

- For production environments we recommend that you test the Bamboo upgrade on a QA server before deploying to production. See the Bamboo upgrade guide.

Backups and recovery

- **We highly recommend** that you establish a data recovery plan that is aligned with your company's policies. See Data and backups.

Logging

- Bamboo logs record server, build and remote (including elastic) agent activity. See Logging in Bamboo.
- Bamboo audit logs record details of any changes made to the configuration of the Bamboo server. See Tracking changes to your Bamboo server.

Lorem ipsum

Unable to render {include} The included page could not be found.

Installing and upgrading

Installing

Installing Bamboo on Linux

Installing Bamboo on Mac OS X

Installing Bamboo on Windows

Connect Bamboo to an external database

Bamboo remote agent installation guide

Supported platforms

Upgrading

Bamboo upgrade guide

Supported platforms

This page describes the supported platforms for **Bamboo 9.4**. If you're looking for information about the platforms supported by another version of Bamboo, select the version from the menu located in the top-right corner of this page.

See also:

- End of support announcements for Bamboo
- Bamboo Best Practice - System Requirements
- Stock images

Definitions:

✅ **Supported** — you can use Bamboo with this platform.

ℹ️ **Limited** — you can evaluate Bamboo on this platform, but you can't use it to run a production Bamboo site.

⚠️ **Deprecated** — support for this platform will end in an upcoming release.

Java

Oracle JDK:

✅ Java 17

✅ Java 11

Open JDK:

✅ Java 17

✅ Java 11

Adoptium OpenJDK:

✅ Java 17

✅ Java 11

Good to know:

- Once the JDK is installed, you will need to set the **JAVA_HOME** environment variable, pointing to the root directory of the JDK. Some JDK installers set this automatically (check by typing 'echo %JAVA_HOME%' in a command prompt, or 'echo $JAVA_HOME' in a shell). You need to do this before installing Bamboo, as Bamboo will automatically configure JDK capabilities based on the system environment variables on your machine.
- For Bamboo server, it is not enough to have just the JRE. Please ensure that you have the full JDK.
- JVM implementations other than HotSpot are not supported, and JDKs from other vendors have not been tested.
- You only need to run the agent and server using a supported JDK. Agents can build software with any JDK version.
- AdoptOpenJDK is now known as Adoptium.

Operating systems

Operating systems:

Data Center nodes

✅ Microsoft Windows

✅ Linux

Agents

✅ Microsoft Windows

✅ Linux

✅ macOS (including Apple silicon)

Good to know:

- For Linux, you should create a dedicated user to run Bamboo. Bamboo runs as the user it is invoked under and can potentially be abused. See Installing Bamboo on Linux.

Databases

If you are using PostgreSQL and have upgraded the GNU libc to **2.28 or higher**, we recommend rebuilding the PostgreSQL indices before you start Bamboo.

MySQL:

- MySQL 8
- MySQL 5.7 DEPRECATED

PostgresSQL:

- PostgreSQL 16
- PostgreSQL 15
- PostgreSQL 14
- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11 DEPRECATED

Microsoft SQL Server:

- SQL Server 2019
- SQL Server 2017

Oracle:

- Oracle 19c
- Oracle 12c R2 DEPRECATED

H2:

- Shipped with Bamboo for evaluation purposes only

Good to know:

- MySQL is supported only with the InnoDB storage engine.
  - For MySQL 5.7 use MySQL Connector/J 5.1.49.
  - For MySQL 8 use MySQL Connector/J 8.0.26.

  In MySQL 8, the `com.mysql.jdbc.Driver` class implementing `java.sql.Driver` in MySQL Connector/J was deprecated. Instead, you should use the `com.mysql.cj.jdbc.Driver` class. See Migrate the Bamboo configuration from MySQL 5.7 to 8.0 for details.
- PostgreSQL is supported when used with the JDBC driver bundled with Bamboo.
- SQL Server is supported when used with Microsoft's JDBC driver.
- Bamboo ships with a built-in H2 database, which is fine for evaluation purposes but is somewhat susceptible to data loss during system crashes. For production environments we recommend that you configure Bamboo to use an external database.
- Starting from version 7.0, we've stopped bundling the JDBC driver for Oracle with Bamboo. We recommend that you use the Oracle JDBC driver version 21.x or later. Check out the Oracle documentation for more details: Oracle JDBC Frequently Asked Questions.

The H2 database is not for use in production instances.

Bamboo doesn't support database environment clusters or SCAN VIP Adresses.

Web browsers

Web browsers:

- Mozilla Firefox
- Chrome
- Safari
- Microsoft Edge

Good to know:

- If not specified otherwise, the latest stable version supported.

Source repositories

Git

- 2.8 and later
- Perforce DEPRECATED

Good to know:

- Bamboo 4.2, and later versions, support Subversion 1.7, but use the Subversion 1.6 Workspace Format by default to keep backwards compatibility with older Subversion working copies. You can set the b

Subversion:

- 1.5
- 1.6
- 1.7
- 1.8
- 1.9
- 1.10

`amboo.svn.wc.format` system property if your Bamboo plans need to use Subversion 1.7 commands as part of your build scripts. See Setting Bamboo to Support Subversion 1.7 Workspace Format for details.

Docker

Docker for Mac:

- 23.0 or later

Docker for Linux:

- 23.0 or later

Kubernetes

kubectl

- 1.24 or later

Infrastructure

Application servers:

- Bamboo runs on a bundled Apache Tomcat and it's the only supported configuration.

Internet protocols:

- You can run Bamboo in both IPv4 and IPv6 environments.
- Raw IPv6 addresses are not always recognized. See the IPv6 in Bamboo for limitations and known issues.

Agents and custom EC2 images:

> **Atlassian doesn't provide support for customized images.** Bamboo provides flexibility to use customized machine images, but it's impossible for us to support all individual configurations.
>
> Use Bamboo stock images as the base for all image customizations to ensure a minimal level of consistency of your Elastic Bamboo setup.

Install a Bamboo Data Center trial

Want to quickly get up and running with Bamboo Data Center? This page will guide you through a few simple steps to install and set up a trial Bamboo Data Center site.

A trial license gives you access to a full instance of Bamboo Data Center for 30 days. At the end of the trial period your Bamboo Data Center site will become read-only and you'll have the option to buy a full license to continue using it, so you won't lose any of your projects or data.

Before you begin

Bamboo installers come with all the bits and pieces you need to run the application, but there's a few things you'll need to get up and running:

> For the complete list of what Bamboo supports, see Supported platforms.

- **Hardware and operating system:** A 32- or 64-bit x86 PC running Microsoft Windows or Linux, or an Apple Silicon or Intel Mac.
- **Java SDK:** Make sure that JDK 8 or 11 is installed and that the JAVA_HOME environment variable is set correctly. Using Bamboo with a JRE is not supported.
- **Database:** You don't need to spin up a database if you're planning to evaluate Bamboo Data Center on a single node as it comes with a built-in H2 database. If you want to set up a production instance though, you'll have to migrate your data to an external database. To evaluate Bamboo cold standby clustering features, you'll also need an external database so your nodes can access it. We support Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL.

  The built-in H2 database is for evaluation purposes only and isn't meant for production environments.
- **Web browser:** You'll need it to access Bamboo.
- **Email address:** You'll need it to generate your 30-day trial license and create an account.

Ready? Let's start by downloading the installer.

Download the installer

Head over to https://www.atlassian.com/software/bamboo/download and download the latest installer for your operating system.

> If you're running Windows, we recommend that you download the `.exe` binary installer.

Got it? OK, now we'll install Bamboo locally.

Install Bamboo

See the following sections to learn how to install Bamboo on your operating system:

To install Bamboo on Windows:

1. Launch the executable installer from your downloads folder.
2. Follow the instructions on the screen to complete the installation.
3. To start Bamboo, from the Start menu, select **Bamboo** > **Start in Console**.
   Bamboo will now start up on port 8085. Keep the Console window open.

To install Bamboo on Linux:

1. Extract the downloaded `.tar.gz` archive and move the files outside your downloads folder. We'll refer to this location as the installation directory from now on.
2. Create a home directory for Bamboo away from the installation directory. This is the location where the app will store its data.
3. Tell Bamboo where the home directory is before you launch the app:

   a. Go to `/atlassian-bamboo/WEB-INF/classes/` in the installation directory.
   b. Edit `bamboo-init.properties` by uncommenting the `bamboo.home` line.
   c. Provide the absolute path to the home directory you created and save the file.
   For example: `/home/jim/bamboo-home`.
4. Launch the terminal and start Bamboo with the following commands:

```
cd <BAMBOO_INSTALLATION_DIRECTORY>
./bin/start-bamboo.sh
```

   Bamboo will now start up on port 8085. Keep this terminal window open.

To install Bamboo on macOS:

1. Extract the downloaded `.tar.gz` archive and move the files outside your downloads folder. We'll refer to this location as the installation directory from now on.
2. Create a home directory for Bamboo away from the installation directory. This is the location where the app will store its data.
3. Tell Bamboo where the home directory is before you launch the app:
   a. Go to `/atlassian-bamboo/WEB-INF/classes/` in the installation directory.
   b. Edit `bamboo-init.properties` by uncommenting the `bamboo.home` line.
   c. Provide the absolute path to the home directory you created and save the file.
   For example: `/Users/jim/bamboo-home`.

   > To get the absolute path to your Bamboo installation directory, in Finder, + **right-click** the folder and select **Copy as Pathname**.

4. Launch the terminal and start Bamboo with the following commands:

```
cd <BAMBOO_INSTALLATION_DIRECTORY>
./bin/start-bamboo.sh
```

   Bamboo will now start up on port 8085. Keep this terminal window open.

So far, so good. Now it's time to launch the Setup Wizard.

Set up Bamboo

Go through the setup wizard to configure some details and create an admin account.

1. In your web browser of choice, go to http://localhost:8085.
2. Get a 30-day trial license:
   a. On the Setup Wizard welcome screen, select **Generate a Bamboo Data Center license**.
   b. Log in to My Atlassian with your existing Atlassian ID account or create a new account if you don't have one already.
   c. On the **New trial license** page, provide your organization's name and select **Generate license**.
   d. Copy the license key, paste it into the setup wizard, and select **Continue**.
3. Complete the configuration by following the instructions in the setup wizard. You can change the configuration details as you please or keep the defaults.

It will take a few minutes to prepare everything for you. Once that's done, you're ready to go!

What's next?

Now that you've successfully set up your Bamboo trial, we have some hints on what to try out next:

- Getting started with Bamboo — check this out if you're new to Bamboo and want to find your way around.
- Bamboo Best Practice — learn how to get the most out of Bamboo.
- Bamboo Specs — test drive Bamboo's configuration as code capabilities.
- Set up a Bamboo Data Center cold standby — try creating a clustered Bamboo Data Center setup.

Bamboo installation guide

1. Check the system requirements

Supported platforms

Please read the Supported platforms page before you install Bamboo. The Supported Platforms page lists the applications servers, databases, operating systems, web browsers and JDKs that we have tested Bamboo with and recommend.

Note that Bamboo ships with a built-in H2 database, which is fine for evaluation purposes but is somewhat susceptible to data loss during system crashes. For the production environment, we recommend that you configure Bamboo to use an external database.

Hardware requirements

While some of our customers run Bamboo on SPARC-based hardware, Atlassian only officially supports Bamboo running on x86 hardware and 64-bit derivatives of x86 hardware.

Servlet container requirements

You will need a servlet container that supports the Servlet 2.4 specification. Most modern containers should comply with this.

Related pages:

- Running the Setup Wizard
- Upgrade guide
- Bamboo remote agent installation guide
- Bamboo Release Notes

2. Install and setup

Choose the relevant instructions for your operating system:

- Linux
- Mac
- Windows
- Getting started with Docker and Bamboo

3. Check for known issues and troubleshoot the Bamboo installation

If something is not working correctly after you have completed the steps above to install Bamboo, please check for known Bamboo issues and try troubleshooting your upgrade as described below:

- **Check for known issues**. Sometimes we find out about a problem with the latest version of Bamboo after we have released the software. In such cases we publish information about the known issues in the Bamboo Knowledge Base. Please check the known issues in the Bamboo Knowledge Base and follow the instructions to apply any necessary patches if necessary.
- **Did you encounter a problem during the Bamboo installation?** Please refer to the guide to troubleshooting upgrades in the Bamboo Knowledge Base.
- If you encounter a problem during the upgrade and cannot solve it, please create a support ticket and one of our support engineers will help you.

Installing Bamboo on Linux

In this guide we'll run you through installing Bamboo with an external database on Linux.

On this page:

Other ways to install Bamboo:

- Evaluation - get your free trial up and running in no time.
- Windows – install Bamboo on a Windows server.

Before you begin

Before you install Bamboo, there are a few questions you need to answer.

Are you using a supported operating system?

Check the Supported platforms page for the version of Bamboo you are installing. This will give you info on supported operating systems, databases and browsers.

Good to know:

- We only support Bamboo on x86 and 64 bit x86 derived hardware platforms.
- You will need permissions for both the Bamboo installation and home directories.

Is your JAVA_HOME variable set correctly?

Before you install Bamboo, check that you're running a supported Java version and that the `JAVA_HOME` environment variable is set correctly.

Bamboo can only run with the JDK (not JRE).

To check your Java version:

```
$ java -version
```

To check your JAVA_HOME variable is set correctly:

```
$ echo $JAVA_HOME
```

If you see a path to your Java installation directory, the `JAVA_Home` environment variable has been set correctly. If a path is not returned you'll need to set your `JAVA_HOME` environment variable manually before installing Bamboo.

Create a dedicated user to run Bamboo.

Bamboo runs as the user it is invoked under and can potentially be abused.

An example of how to create a dedicated user to run Bamboo in Linux:

```
$ sudo /usr/sbin/useradd --create-home --home-dir /usr/local/bamboo --shell /bin/bash bamboo
```

Install Bamboo

1. Download Bamboo

Download the file for your operating system - https://www.atlassian.com/software/bamboo/download

- `tar.gz` for MacOS or Linux distributions.
- `zip` for Windows.

2. Create the installation directory

a) Extract the downloaded file to an install location.

b) The path to the extracted directory is referred to as the `<Bamboo installation directory>` in these instructions.

3. Create the home directory

Specify your Bamboo home directory, where your Bamboo data is stored, before you run Bamboo for the first time.

a) Create your Bamboo home directory (without spaces in the name).

Note: You should not create your Bamboo home directory inside the `<Bamboo installation directory>` — they should be entirely separate locations. If you do put the home directory in the `<Bamboo installation directory>` it will be overwritten, and lost, when Bamboo is upgraded.

b) Open `<Bamboo installation directory>/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`

c) Uncomment the *bamboo.home* line.

d) Provide the absolute path to the bamboo-home directory.

Example:

```
bamboo.home=/var/bamboo/bamboo-home
```

4. Start Bamboo

a) In the command line, change the directory to `<Bamboo installation directory>` and run the commands to start Bamboo:

```
$ cd <Bamboo installation directory>
$ ./bin/start-bamboo.sh
```

b) After successfully starting Bamboo, you will find it online at `http://localhost:8085/`

5. Configure Bamboo

You are starting Bamboo for the first time, so you will need to follow the Setup Wizard to configure Bamboo. See Running the Setup Wizard.

Start using Bamboo

That's it! Your Bamboo site is accessible from a URL like this: `http://<computer_name_or_IP_address>:<port>`

If you want your Bamboo instance always running, check how to run Bamboo as a service.

If something is not working correctly after you have completed the steps above to install Bamboo, please check for known Bamboo issues and try troubleshooting your upgrade as described below:

- **Check for known issues**. Sometimes we find out about a problem with the latest version of Bamboo after we have released the software. In such cases we publish information about the known issues in the Bamboo Knowledge Base. Please check the known issues in the Bamboo Knowledge Base and follow the instructions to apply any necessary patches if necessary.
- **Did you encounter a problem during the Bamboo installation?** Please refer to the guide to troubleshooting upgrades in the Bamboo Knowledge Base.
- If you encounter a problem during the upgrade and cannot solve it, please create a support ticket and one of our support engineers will help you.

Installing Bamboo on Mac OS X

In this guide we'll run you through installing Bamboo with an external database on Mac OS X.

On this page

Before you begin

Before you install Bamboo, there are a few questions you need to answer.

Are you using a supported operating system?

Check the Supported platforms page for the version of Bamboo you are installing. This will give you info on supported operating systems, databases and browsers.

Good to know:

- We only support Bamboo on x86 and 64 bit x86 derived hardware platforms.
- You will need permissions for both the Bamboo installation and home directories.

Is your JAVA_HOME variable set correctly?

Before you install Bamboo, check that you're running a supported Java version and that the `JAVA_HOME` environment variable is set correctly.

Bamboo can only run with the JDK (not JRE).

To check your Java version:

```
java -version
```

To check your JAVA_HOME variable is set correctly:

```
echo $JAVA_HOME
```

If you see a path to your Java installation directory, the `JAVA_Home` environment variable has been set correctly. If a path is not returned you'll need to set your `JAVA_HOME` environment variable manually before installing Bamboo.

Install Bamboo

1. Download Bamboo

Download the file for your operating system - https://www.atlassian.com/software/bamboo/download

- `tar.gz` for MacOS or Linux distributions.
- `zip` for Windows.

2. Create the installation directory

a) Extract the downloaded file to an install location.

b) The path to the extracted directory is referred to as the `<Bamboo installation directory>` in these instructions.

3. Specify your Bamboo home location

Specify your Bamboo home directory, where your Bamboo data is stored, before you run Bamboo for the first time.

a) Create your Bamboo home directory (without spaces in the name).

Note: You should not create your Bamboo home directory inside the `<Bamboo installation directory>` — they should be entirely separate locations. If you do put the home directory in the `<Bamboo installation directory>` it will be overwritten, and lost, when Bamboo is upgraded.

b) Open `<Bamboo installation directory>/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`.
c) Uncomment the *bamboo.home* line.
d) Provide the absolute path to your home directory.

Example:

```
bamboo.home= /home/nathan/bamboo/bamboo-home
```

4. Start Bamboo

a) In the command line, change the directory to `<Bamboo installation directory>` and run the commands to start Bamboo:

```
$ cd <Bamboo installation directory>
$ ./bin/start-bamboo.sh
```

b) After successfully starting Bamboo, you will find it online at: `http://localhost:8085/`

5. Configure Bamboo

You are starting Bamboo for the first time, so you will need to follow the Setup Wizard to configure Bamboo. See Running the Setup Wizard.

Start using Bamboo

That's it! Your Bamboo site is accessible from a URL like this: http://<computer_name_or_IP_address>:<port>

If you want your Bamboo instance always running, check how to run Bamboo as a service.

Troubleshooting

If something is not working correctly after you have completed the steps above to install Bamboo, please check for known Bamboo issues and try troubleshooting your upgrade as described below:

- **Check for known issues**. Sometimes we find out about a problem with the latest version of Bamboo after we have released the software. In such cases we publish information about the known issues in the Bamboo Knowledge Base. Please check the known issues in the Bamboo Knowledge Base and follow the instructions to apply any necessary patches if necessary.
- **Did you encounter a problem during the Bamboo installation?** Please refer to the guide to t roubleshooting upgrades in the Bamboo Knowledge Base.
- If you encounter a problem during the upgrade and cannot solve it, please create a support ticket and one of our support engineers will help you.

Installing Bamboo on Windows

In this guide we'll run you through installing Bamboo with an external database on Windows.

On this page

Before you begin

Before you install Bamboo, there are a few questions you need to answer.

Are you using a supported operating system?

Check the Supported platforms page for the version of Bamboo you are installing. This will give you info on supported operating systems, databases and browsers.

Good to know:

- We only support Bamboo on x86 and 64 bit x86 derived hardware platforms.
- You will need permissions for both the Bamboo installation and home directories.

Is your JAVA_HOME variable set correctly?

Before you install Bamboo, check that you're running a supported Java version and that the `JAVA_HOME` environment variable is set correctly.

Bamboo can only run with the JDK (not JRE).

To check your Java version:

```
java -version
```

To check your JAVA_HOME variable is set correctly:

```
echo %JAVA_HOME%
```

If you see a path to your Java installation directory, the `JAVA_Home` environment variable has been set correctly. If a path is not returned you'll need to set your `JAVA_HOME` environment variable manually before installing Bamboo.

Install Bamboo

1. Download Bamboo

Download Bamboo from the Atlassian download site. You can choose either the Windows Installer versions (`.exe`) or the ZIP Archive (`.zip`).

> It is highly recommended to avoid placing the **Bamboo home directory** in any Windows security controlled directory, for example, `C:\Program Files, C:\Users, C:\Windows`

i. Launch the Bamboo Windows installer to begin the installation wizard.
ii. The installer requires you to specify two directories:

- **Destination directory**— This is the directory where Bamboo's application files will be installed. The default is:

```
C:\Program Files\Bamboo
```

- **Bamboo home directory**— This is the directory where Bamboo will store its configuration data. If the directory you specify doesn't exist, Bamboo will create the directory when it launches. The default is:

```
C:\Users\<current-user>\Bamboo-home
```

> ⚠ Ensure that the Bamboo home directory is not located inside the <Bamboo installation directory> .

i. Extract the files from the zip Archive to a `<Bamboo installation directory>` of your choice. By default, the root directory in your zip file is named "Bamboo".

> ⚠ **Warning: Some unzip programs cause errors**
>
> Some archive-extract programs cause errors when unzipping the Bamboo archive file. We highly recommend that you use the free 7Zip archive-extract program (if in doubt, download the '32-bit .exe' version).

ii. Set up your **Bamboo home directory** — this is the directory where Bamboo will store its root configuration data. To do this, edit the file named `bamboo-init.properties` in the `<Bamboo installation directory>/atlassian-bamboo/WEB-INF/classes/` directory. In this file, insert the property "bamboo.home", with an absolute path to your Bamboo home directory. Your file should look something like this:

```
bamboo.home=C:/test/bamboo-home
```

Alternatively, you can specify an environment variable 'BAMBOO_HOME' which specifies the absolute path to your Bamboo home directory. Bamboo will check if an environment variable is defined.

iii. If you are going to use Bamboo remote agents, set the following in the `bamboo-init.properties` file in the `<Bamboo installation directory>/atlassian-bamboo/WEB-INF/classes` directory:

```
bamboo.jms.broker.uri=tcp://localhost:54663
```

- Replace 'localhost' with the real host name or IP address of your Bamboo server.
- If port number 54663 is already in use, specify a different port number

2. Start Bamboo

a) In the command line, change the directory to `<Bamboo installation directory>` and run the command to start Bamboo:

```
$ cd <Bamboo installation directory>

$ bin\start-bamboo.bat
```

b) After successfully starting Bamboo, you will find it online at `http://localhost:8085/`

3. Configure Bamboo

You are starting Bamboo for the first time, so you will need to follow the Setup Wizard to configure Bamboo. See Running the Setup Wizard.

Start using Bamboo

That's it! Your Bamboo site is accessible from a URL like this: http://<computer_name_or_IP_address>:<port>

If you want your Bamboo instance always running, check how to run Bamboo as a service.

Troubleshooting

If something is not working correctly after you have completed the steps above to install Bamboo, please check for known Bamboo issues and try troubleshooting your upgrade as described below:

- **Check for known issues**. Sometimes we find out about a problem with the latest version of Bamboo after we have released the software. In such cases we publish information about the known issues in the Bamboo Knowledge Base. Please check the known issues in the Bamboo Knowledge Base and follow the instructions to apply any necessary patches if necessary.
- **Did you encounter a problem during the Bamboo installation?** Please refer to the guide to troubleshooting upgrades in the Bamboo Knowledge Base.
- If you encounter a problem during the upgrade and cannot solve it, please create a support ticket and one of our support engineers will help you.

Bamboo upgrade guide

You can upgrade Bamboo by installing a new version of Bamboo and setting it up with the configuration of the original Bamboo instance.

Overview

The recommended paths for upgrading Bamboo to a new version differ depending on whether you want to move to a new server or not:

Upgrading Bamboo locally	Upgrading Bamboo with a move to a new server
Perform the steps as described on this page. Make sure that your new Bamboo binaries aren't installed in the same directory as the original Bamboo binaries, so that you don't lose any changes made to scripts and configuration files inside the Bamboo installation directory.	1. Clone your Bamboo instance into the new location. 2. Perform the upgrade steps on the cloned Bamboo instance as described on this page. The cloned instance on the new server is referred to as original Bamboo instance.

In both scenarios, the new Bamboo instance uses the home directory and the database of the original Bamboo instance.

> We recommend that you test the Bamboo upgrade on a QA server before deploying to production.

If you are a Bamboo app developer, see our Bamboo API Changes by Version guide, which outlines changes in Bamboo that may affect Bamboo apps compiled for earlier versions of Bamboo.

Before you begin

Before you start the upgrade, make sure that you meet all of the following prerequisites:

- Determine the correct upgrade path
- Read the release and upgrade notes
- Optionally, prepare for the migration to Bamboo Data Center
- Check platform requirements and end-of-support announcements
- Check installed app compatibility

Determine the correct upgrade path

Upgrading older releases of Bamboo may require doing several intermediate upgrades to ensure backward compatibility. Choose the right upgrade path depending on which release of Bamboo you're currently on:

- If you're on a release of Bamboo **older than 2.0.6**, follow this path:
  - current 2.0.6 2.6.3 5.7.x 5.14.x 6.5.x 8.0.12 9.3.x

- If you're on **Bamboo 2.6.3 through 3.4.x**, follow this path:
  - current 5.7.x 5.14.x 6.5.x 8.0.12 9.4.x
- If you're on **Bamboo 4.0.x through 5.13.x**, follow this path:
  - current 5.14.x 6.5.x 8.0.12 9.4.x
- If you're on **Bamboo 5.14.x through 6.4.x**, follow this path:
  - current 6.5.x 8.0.12 9.4.x
- If you're on **Bamboo 6.5.x through 7.2.x**, follow this path:
  - current 8.0.12 9.4.x
- If you're on **Bamboo 8.x.x through 9.2.x**, upgrade to 9.4.x directly.

Read the release and upgrade notes

Head over to the Bamboo 9.4 release notes to learn more about all the new features and changes.

Optionally, prepare for the migration to Bamboo Data Center

If you're currently on Bamboo Server and are planning to migrate to Bamboo Data Center as part of the upgrade procedure:

- Check the infrastructure and hardware requirements for clustering with Bamboo Data Center
- Learn more about the changes that will be made to your home directory during the upgrade

Check platform requirements and end-of-support announcements

Go to:

- Supported platforms
- End-of-support announcements for Bamboo

Check installed app compatibility

Make sure that the apps you've installed are compatible with the release of Bamboo that you want to upgrade to.

To do this, run the Bamboo update check:

1. In the upper-right corner of the screen, select **Administration** > **Manage apps**.
2. At the bottom of the **Manage apps** page, select **Bamboo update check**.
3. Select the release that you want to upgrade to, and then select **Check**.

Learn more about apps and the Universal Plugin Manager (UPM)

1. Export and back up the existing Bamboo data

Create a support zip

Create a support zip containing logs from your instance, diagnostic, and configuration information. If you run into problems during the upgrade, you can send this file to our support team to help them understand how your application is configured and to troubleshoot your problem efficiently.

Learn how to create a support zip

Export the Bamboo database

There are two database backup scenarios, depending on whether you are using an embedded or external database.

Embedded HSQL database	External database

Create an export .zip file for the original Bamboo instance. For more information, see Exporting data for backup. The export may take a long time to complete and may require a large amount of disk space, depending on the number of builds and tests in your system. ⓘ HSQL is not recommended for production Bamboo instances.	Use native database tools to create a backup. For more information about external databases, see Connect Bamboo to an external database.

Stop Bamboo

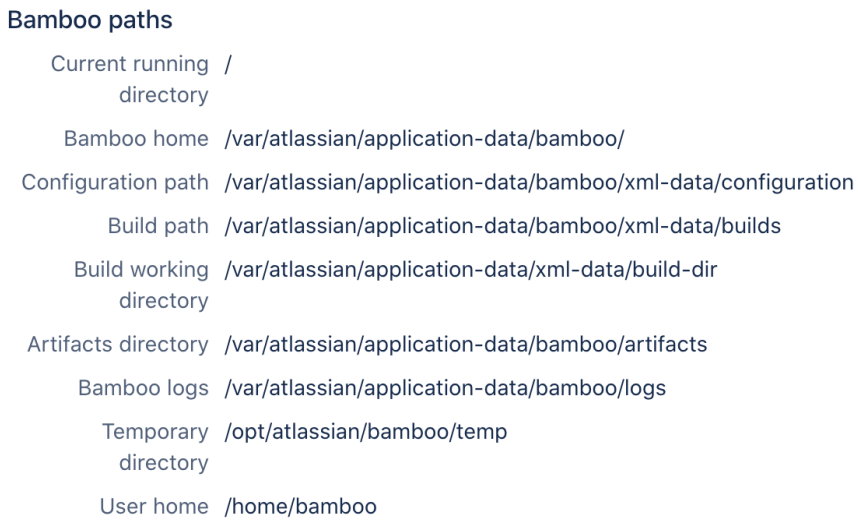
Stop the original Bamboo instance.

If you have Bamboo running as a Windows service, uninstall the service by using the `UninstallService.bat` executable that came with your Bamboo instance.

Back up the Bamboo configuration

When the original Bamboo instance is shut down, back up your <`bamboo-home` > directory, which contains the `builds` and `configuration` directories. You can compress it into a .zip file.

Select ⚙ > **System** > **System information** > **Bamboo paths**. Note the **Bamboo home** path, **Build path**, and **Configuration path**:



For more information about these directories, see Important Directories and Files.

2. Download and install a new Bamboo version

To upgrade Bamboo, you must install a new Bamboo instance in a <`bamboo-install`> directory that is different from the <`bamboo-install`> directory of the original Bamboo instance.

This upgrade scenario uses the home directory and the external database of the original Bamboo instance.

- Make sure that the original Bamboo instance is not running before you start the new installation.
- To prevent data loss during updates or reinstallation, the `<bamboo-home>` directory must be different from the `<Bamboo-install>` directory.

Follow these guidelines to download and unpack a new Bamboo version:

Download Bamboo

Download the file for your operating system - https://www.atlassian.com/software/bamboo/download

- `tar.gz` for MacOS or Linux distributions.
- `zip` for Windows.

Create the installation directory

a) Extract the downloaded file to an install location.

b) The path to the extracted directory is referred to as the `<Bamboo installation directory>` in these instructions.

3. Configure the new Bamboo instance

Set the home directory for the new Bamboo instance

Set the `<home-directory>` to use the `<home-directory>` of the original Bamboo instance:

1. Go to the new Bamboo instance <bamboo-install> directory. It is the directory where you installed Bamboo.
2. Open `atlassian-bamboo/WEB-INF/classes/ bamboo-init.properties`
3. Set the `bamboo.home` variable to use the `<bamboo-home>` path of the original Bamboo instance.
4. If needed set the `bamboo.shared.home` variable to use the `<bamboo-shared-home>` path of the original Bamboo instance.

Install a JDBC driver

If the JDBC driver for your database isn't bundled with Bamboo, you need to install it for the new Bamboo instance yourself. See Connect Bamboo to an external database for detailed instructions.

Migrate your existing Bamboo configurations over to your new Bamboo installation

If you have modified properties in configuration files of your existing Bamboo installation, make the same modifications in your new Bamboo installation. However, because the properties in the configuration files may have changed between versions, you cannot simply copy the configuration files from your existing installation and replace the equivalent files in the new installation.

For each file you have modified in your existing Bamboo installation, you need to **manually edit each equivalent file in your new Bamboo installation and re-apply your modifications**.

The table below lists the most commonly modified files and their locations within your Bamboo Installation Directory:

File	Location in Bamboo installation	Description

setenv.bat (Windows) or setenv.sh (Linux)	`bin`	Configuring your system properties
seraph-config.xml	`atlassian-bamboo /WEB-INF/classes`	Modified if you had integrated Bamboo with Crowd
server.xml	`conf`	Modified in the following situations: • If you had previously changed Bamboo's root context path. • If you had previously secured Bamboo with Tomcat using SSL or proxy server.

Check database access permission

Before you start the new Bamboo instance, make sure that it has the *write* access to the database, which is required to complete the upgrade tasks.

4. Start Bamboo

Install Windows service

If previous version of Bambo was executed as Windows service then configure Bamboo to run as a service on Windows, using the `service.bat` executable.

Start Bamboo

Once you have installed Bamboo and set the `bamboo.home` property, start the new Bamboo instance. The upgrade runs automatically.

You can check whether the upgrade was successful in the `atlassian-bamboo.log` file.

Update any installed apps

If you installed any apps in addition to the pre-installed system apps:

- Check if all apps are compatible with the new version of Bamboo.
- Update any apps that are out of date.
- Disable any apps that are incompatible with the new version of Bamboo.

> Upgrading Bamboo may require reindexing.
>
> Depending on the number of existing builds and tests, the reindexing process may take a significant amount of time, during which Bamboo will not be available.

Automatic update of remote agents

Since Bamboo 6.10, remote agents automatically detect when a new version is available and download new classes from the server. However, Bamboo 6.10 also updated the remote agent wrapper, which means that for upgrades from versions earlier than 6.10, you'll have to manually reinstall the wrapper on your remote agents.

Additionally, remote agents now support Java 17. If you want to migrate any of your existing remote agent instances to Java 17, you'll need to reinstall the affected remote agent wrappers.

For more information, see Bamboo remote agent installation guide.

Troubleshooting

If you followed the documentation and you still have problems with the upgrade process:

- Check other Knowledge Base articles.
- Ask questions at https://community.atlassian.com/.
- You can also create a support ticket. To help us address the issue, attach the `atlassian-bamboo.log` and `Support zip` file to the ticket.

IPv6 in Bamboo

Starting from Bamboo 6.7, we support communication over IPv6. We've taken the dual-stack approach (IPv4 + IPv6), so IPv4 addresses will still work. If your systems are IPv6-only, make sure you read about the limitations below.

IPv6 stands for "Internet Protocol Version 6", and is the next-generation Internet protocol designed to replace the current IPv4 protocol. The Internet Engineering Task Force (IETF) created IPv6 standard described in RFC 8200 to accommodate the growing number of users and devices accessing the Internet.

IPv6 addresses are 128 bits, which allows for approx. 3.4×10^{38} unique IP addresses. Here's an example of an IPv6 address:

```
2401:1d80:ffff:1:202:02ff:fe07:0305
```

Best practices

We recommend that you use hostnames or domain names instead of IP addresses everywhere in the Bamboo configuration. It's a more reliable way of configuring and accessing both Bamboo and other Atlassian products.

Limitations

These limitations apply if your systems use only IPv6 instead of dual-stack:

- You won't be able to connect to Atlassian Marketplace, as it requires IPv4.
- End of Life check won't work, because it needs to connect to Atlassian Marketplace.
- You won't be able to integrate with Atlassian Cloud products, as they require IPv4.
- Log Analyzer won't work, as it requires IPv4.
- When configuring LDAP, use hostnames, not IPv6 addresses. Otherwise, you'll get a validation error.
- Bamboo requires running using with **java.net.preferIPv6Addresses** sys property set to true, to operate normally in IPv6 environment.

There are no IPv6 endpoints for EC2, the only service that we use that has IPv6 endpoints is S3.

- Remote agents: tested in both IPv4 and IPv6 set ups.
- Remote agents authentication:
- In case of IPv4 it is possible to use wildcards, e.g. 192.168.50.*
- In case of IPv6 Use CIDR instead of wildcards: 2a05:d014:f7d:f801:b1a4:dec3::/32
- Elastic agents: required IPv4 as there are no IPv6 endpoints for EC2.

RSS requires dual stack IPv4/IPv6 set up to work properly in both Docker and without Docker modes. RSS is not supported in IPv6 pure set up.

- Requires dual stack (IPv4/IPv6) set up on host.
- Tested with Docker version: 18.03.1-ce.

- Requires dual stack (IPv4/IPv6) set up on host.
- Tested with npm version: 5.6.0, node version: 8.11.3

- Tested with Openfire 4.2.3

- Oracle: tested with Oracle 12c
- MS SQL: tested with MS SQL 2016 Standard edition.
- PostgreSQL: tested with Postgres 9.6.9 and 10.4.
- MySQL: tested with JDBC Driver mysql-connector-java-8.0.11.jar and mysql-connector-java-5.1.46.jar, MySQL 5.7.2.

- Github: doesn't support IPv6.
- SVN: tested with: SVNKit 1.8.15, SVN server - SVN 1.9.7, SVN server required the svnserve daemon to be run with -6 option (--prefer-ipv6).

- Mercurial: tested SSH and HTTPS with Hg 4.2.3.
- Git: tested SSH/HTTPS/GIT protocols with git version 2.14.4.
- Bitbucket Server: RSS won't work in pure IPv6 setup.
- Bitbucket Cloud: Required setting system property **-Djava.net.preferIPv6Addresses=true,** webhooks don't work over pure IPv6 setup.

- Bitbucket Data Center and Server: supported since version 5.8 (however, Bamboo supports version 7.5 or later)
- FeCru: tested with version 4.5.4
- Jira: tested with Jira Software 7.11.0
- Confluence: tested with Confluence 6.10.1
- Crowd: tested with Crowd 3.2.3

Bamboo doesn't expose port for IPv6 out of the box. To enable this, in the **server.xml** file, change:

```
<Connector/>'s  attribute protocol="HTTP/1.1"
```
to
```
protocol="org.apache.coyote.http11.Http11NioProtocol"
```

Migrating custom logging configurations to Log4j 2

In Bamboo 9.0, we've upgraded the Log4j runtime logging library to version 2. This means that if you're using a custom Log4j 1.x configuration with your Bamboo instance, you'll need to manually migrate it to the Log4j 2 configuration file format. This guide describes the necessary steps to complete the migration and provides examples to help you along the way.

> If you didn't customize the logging configuration of your Bamboo instance or remote agents before upgrading to Bamboo 9.0, you can skip this page.

To learn more about what's new in Log4j 2, see Apache Log4j 2.

On this page:

Step 1: Migrate logging level configurations

In the following example, we'll migrate the `MarketplaceClient` class logging level configuration.

In Log4j 1.x, the logging level configuration for the `MarketplaceClient` class would look like this:

```
log4j.category.com.atlassian.marketplace.client.MarketplaceClient=WARN
```

The same configuration in Log4j 2 requires that we migrate the `MarketplaceClient` category configuration to a logger under the name `marketplaceClient`:

```
logger.marketplaceClient.level = WARN
logger.marketplaceClient.name = com.atlassian.marketplace.client.MarketplaceClient
```

As part of this migration, we gave the new logger the following properties:

- `level` defines the logging level (in this case, the logging level is `WARN`)
- `name` indicates the class we are configuring the logging level for (in this case, `com.atlassian.marketplace.client.MarketplaceClient`)

Step 2: Migrate Log4j appender configurations

Appender property definitions have also changed in Log4j 2. In the following example, we'll migrate the `BambooRollingFile` appender with the same configuration as in Log4j 1.x.

This is what the `BambooRollingFile` appender configuration would look like in Log4j 1.x:

```
#using 'bamboo home aware' appender. If the File is relative a relative Path the file goes into {bamboo.
home}/logs
log4j.appender.filelog=com.atlassian.bamboo.log.BambooRollingFileAppender
log4j.appender.filelog.File=atlassian-bamboo.log
log4j.appender.filelog.MaxFileSize=100MB
log4j.appender.filelog.MaxBackupIndex=5
log4j.appender.filelog.layout=org.apache.log4j.PatternLayout
log4j.appender.filelog.layout.ConversionPattern=%d %p [%t] [%c{1}] %m%n
```

And this is what the configuration should look like in a Log4j 2-compatible format:

```
#using 'bamboo home aware' appender. If the File is relative a relative Path the file goes into {bamboo.
home}/logs
appender.filelog.type = BambooRollingFile
appender.filelog.name = filelog
appender.filelog.fileName = atlassian-bamboo.log
appender.filelog.filePattern = atlassian-bamboo.log.%i
appender.filelog.layout.type = PatternLayout
appender.filelog.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.filelog.policies.type = Policies
appender.filelog.policies.size.type = SizeBasedTriggeringPolicy
appender.filelog.policies.size.size = 100MB
appender.filelog.strategy.type = DefaultRolloverStrategy
appender.filelog.strategy.max = 5
appender.filelog.strategy.fileIndex = min
```

Step 3: Migrate custom logging configurations on remote agents

If you're using custom logging configurations on your remote agents, migrate them to the Log4j 2-compatible format by applying the previous steps, and then change the remote agents' configuration parameters in `wrapper.conf`.

Under Log4j 1.x, the `wrapper.conf` file would point to the `log4j.properties` file by means of the `log4j.configuration` parameter:

```
wrapper.java.additional.4=-Dlog4j.configuration=/path/to/log4j.properties
```

Now, the parameter is called `log4j2.configurationFile`, so to make this work with Log4j 2, adjust the parameter name and value:

```
wrapper.java.additional.4=-Dlog4j2.configurationFile=/path/to/log4j2.properties
```

If your custom configuration has few changes over the original file, you might find it easier to start with a clean `log4j2.properties` file and customize it accordingly. Simply use the default Log4j 2 configuration file as a template, and migrate your custom configurations from the old properties file to the new one.

Here's an example of the default contents of the `log4j2.properties` file:

```
#
# Change the following line to configure the bamboo logging levels (one of INFO, DEBUG, ERROR,
FATAL)
#
packages = com.atlassian.bamboo.log
status = warn
rootLogger = INFO, console, filelog

appender.console.type = Console
appender.console.name = console
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.console.filter.threshold.type = ThresholdFilter
appender.console.filter.threshold.level = OFF

#using 'bamboo home aware' appender. If the File is relative a relative Path the file goes into
{bamboo.home}/logs
appender.filelog.type = BambooRollingFile
appender.filelog.name = filelog
appender.filelog.fileName = atlassian-bamboo.log
appender.filelog.filePattern = atlassian-bamboo.log.%i
appender.filelog.layout.type = PatternLayout
appender.filelog.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.filelog.policies.type = Policies
appender.filelog.policies.size.type = SizeBasedTriggeringPolicy
appender.filelog.policies.size.size = 100MB
appender.filelog.strategy.type = DefaultRolloverStrategy
```

```
appender.filelog.strategy.max = 5
appender.filelog.strategy.fileIndex = min

################################################################################################################
#######################
# Access Log
################################################################################################################
#######################
#using 'bamboo home aware' appender for access logs
appender.accesslog.type = BambooRollingFile
appender.accesslog.name = accesslog
appender.accesslog.fileName = atlassian-bamboo-access.log
appender.accesslog.filePattern = atlassian-bamboo-access.log.%i
appender.accesslog.layout.type = PatternLayout
appender.accesslog.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.accesslog.policies.type = Policies
appender.accesslog.policies.size.type = SizeBasedTriggeringPolicy
appender.accesslog.policies.size.size = 20MB
appender.accesslog.strategy.type = DefaultRolloverStrategy
appender.accesslog.strategy.max = 5
appender.accesslog.strategy.fileIndex = min

logger.accesslogfilter = INFO, accesslog, console
logger.accesslogfilter.name = com.atlassian.bamboo.filter.AccessLogFilter
logger.accesslogfilter.additivity = false

################################################################################################################
#######################
# Emergency, high volume logging
################################################################################################################
#######################
appender.emergency.type = BambooRollingFile
appender.emergency.name = emergency
appender.emergency.fileName = emergency-atlassian-bamboo.log
appender.emergency.filePattern = emergency-atlassian-bamboo.log.%i
appender.emergency.layout.type = PatternLayout
appender.emergency.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.emergency.policies.type = Policies
appender.emergency.policies.size.type = SizeBasedTriggeringPolicy
appender.emergency.policies.size.size = 20480KB
appender.emergency.strategy.type = DefaultRolloverStrategy
appender.emergency.strategy.max = 5
appender.emergency.strategy.fileIndex = min

logger.emergency=INFO, emergency
logger.emergency.name = emergency
logger.emergency.additivity = false

################################################################################################################
#######################
# Javascript Debugging
################################################################################################################
#######################
appender.javascript.type = BambooRollingFile
appender.javascript.name = javascript
appender.javascript.fileName = js-atlassian-bamboo.log
appender.javascript.filePattern = js-atlassian-bamboo.log.%i
appender.javascript.layout.type = PatternLayout
appender.javascript.layout.pattern = %d{DEFAULT} %p [%t] [%C{1}] %m%n
appender.javascript.policies.type = Policies
appender.javascript.policies.size.type = SizeBasedTriggeringPolicy
appender.javascript.policies.size.size = 20480KB
appender.javascript.strategy.type = DefaultRolloverStrategy
appender.javascript.strategy.max = 5
appender.javascript.strategy.fileIndex = min

logger.javascript=DEBUG, javascript
logger.javascript.name = JavaScript
logger.javascript.additivity = false

################################################################################################################
#######################
# Database Logging
################################################################################################################
#######################
logger.SessionImpl.level=ERROR
```

```
logger.SessionImpl.name=org.hibernate.impl.SessionImpl
#Remove when criteria queries are mostly gone
logger.hibernate-deprecation.level=ERROR
logger.hibernate-deprecation.name=org.hibernate.orm.deprecation
#Remove when most entities are on annotations instead of hbms
logger.MetadataContext.level=ERROR
logger.MetadataContext.name=org.hibernate.metamodel.internal.MetadataContext

## log hibernate prepared statements/SQL queries (equivalent to setting 'hibernate.show_sql' to
'true')
#logger.sql.name = org.hibernate.SQL
#logger.sql.level= DEBUG

## log hibernate prepared statement parameter values
#logger.hibernate-type.name=org.hibernate.type
#logger.hibernate-type.level=TRACE
#logger.hibernate-batcher.name=org.hibernate.impl.BatcherImpl
#logger.hibernate-batcher.level=DEBUG

## log active objects sql statements
#logger.ao-sql.name=net.java.ao.sql
#logger.ao-sql.level=DEBUG

## Bamboo plan cache detailed logging
#logger.plan-cache.name=com.atlassian.bamboo.plan.cache
#logger.plan-cache.level= DEBUG

########################################################################################################
########################
# OSGi and Atl plugins related logs
########################################################################################################
########################
logger.FelixOsgiContainerManager.level=WARN
logger.FelixOsgiContainerManager.name = com.atlassian.plugin.osgi.container.felix.
FelixOsgiContainerManager
logger.ExportPackageListBuilder.level=ERROR
logger.ExportPackageListBuilder.name = org.twdata.pkgscanner.ExportPackageListBuilder
logger.atlassian-plugin.level=WARN
logger.atlassian-plugin.name = com.atlassian.plugin
logger.streams.level=WARN
logger.streams.name=com.atlassian.streams

########################################################################################################
########################
# SSH proxy related settings
########################################################################################################
########################
logger.sshd.level=WARN
logger.sshd.name=org.apache.sshd
logger.plugin-ssh.level=WARN
logger.plugin-ssh.name = com.atlassian.bamboo.plugins.ssh
logger.bamboo-ssh.level=WARN
logger.bamboo-ssh.name=com.atlassian.bamboo.ssh

########################################################################################################
########################
# General ignore logging for various packages
########################################################################################################
########################
logger.marketplaceClient.level = WARN
logger.marketplaceClient.name = com.atlassian.marketplace.client.MarketplaceClient
logger.webwork.level=WARN
logger.webwork.name = webwork
logger.velocity.level=WARN
logger.velocity.name = velocity
logger.xwork-util.level = ERROR
logger.xwork-util.name = com.opensymphony.xwork2.util
logger.springframework.level = WARN
logger.springframework.name = org.springframework
logger.hibernate.level = WARN
logger.hibernate.name = org.hibernate
logger.acegi.level = WARN
logger.acegi.name = org.acegisecurity
logger.activemq-failover.level = WARN
logger.activemq-failover.name = org.apache.activemq.transport.failover.FailoverTransport
logger.servlet-filter.level = WARN
```

```
logger.servlet-filter.name = com.atlassian.plugin.servlet.filter.
ServletFilterModuleContainerFilter
logger.plugin-resource-download.level = WARN
logger.plugin-resource-download.name = com.atlassian.plugin.servlet.PluginResourceDownload
logger.ignore-missing-field-xstream.level = INFO
logger.ignore-missing-field-xstream.name = com.atlassian.bamboo.persister.xstream.
IgnoreMissingFieldXStream
logger.dependency-resolver.level = ERROR
logger.dependency-resolver.name = com.atlassian.plugin.webresource.
DefaultResourceDependencyResolver
logger.apache.level = INFO
logger.apache.name = org.apache
logger.botocss.level = WARN
logger.botocss.name = com.atlassian.botocss
logger.amazonaws.level = WARN
logger.amazonaws.name = com.amazonaws
logger.amazonaws-ec2metadatautils.level = ERROR
logger.amazonaws-ec2metadatautils.name = com.amazonaws.util.EC2MetadataUtils
logger.vutbr-web.level = WARN
logger.vutbr-web.name = cz.vutbr.web

# Embedded Crowd
logger.crowd.level = WARN
logger.crowd.name = com.atlassian.crowd

# log S3 retries
logger.s3.name=com.amazonaws.services.s3
logger.s3.level=INFO

logger.struts2-beanselection.name=org.apache.struts2.config.BeanSelectionProvide
logger.struts2-beanselection.level=WARN

logger.tunnel.name=com.atlassian.tunnel
logger.tunnel.level=WARN

# Docker client
logger.docker.name=com.spotify.docker.client
logger.docker.level = WARN

# profiling (needs system property -Datlassian.profile.activate=true to be activated)
logger.profiling.name=com.atlassian.util.profiling
logger.profiling.level=DEBUG

#upm cache flushes
logger.upm-stacktrace.name=com.atlassian.cache.stacktrace
logger.upm-stacktrace.level=WARN
```

Once you've completed all the steps above, your custom configuration should be perfectly compatible with Log4j 2.

Upgrade from Bamboo Server to Bamboo Data Center

If you're a current Bamboo Server customer looking to upgrade to Bamboo Data Center, this page will help you create a trial license and set up Data Center. There are several ways to get started with Bamboo Data Center, depending on your current setup.

If you're installing Bamboo Data Center for the first time with no existing Bamboo Server data to migrate, check out how to install a Bamboo Data Center trial.

Set up Data Center

Things you should know about when setting up your Data Center:

License

It's your Bamboo license that determines the type of Bamboo you have: Server or Data Center. Bamboo will auto-detect the license type when you enter your license key, and automatically unlock any license-specific features.

To upgrade from Bamboo Server to Bamboo Data Center, you will need a Data Center license. You can either purchase a full Data Center license or get a free trial license for 30 days. When your 30-day trial finishes, you'll have the option to purchase a Data Center license and carry on using Bamboo Data Center without losing any data you've created during the trial. If you decide Bamboo Data Center is not for you, you can easily revert to your existing Server license.

> Note that as of February 15, 2024 PT, your Server products will reach the end of support.

See our Supported platforms page for information on the database, Java, and operating systems you'll be able to use. These requirements are the same for Server and Data Center deployments.
Apps extend what your team can do with Atlassian products, so it's important to make sure that your team can still use their apps after migrating to Data Center. When you upgrade to Data Center, you'll be required to switch to the Data Center compatible version of your apps, if one is available.

See Evaluate apps for Data Center migration for more information.
To use Bamboo Data Center, you must:

- Have a Data Center license (you can purchase a Data Center license or create a trial license at my.atlassian.com)
- Use a supported external database, operating system, and Java version
- Use OAuth authentication if you have application links to other Atlassian products (such as Confluence)

To run Bamboo in a cluster, you must also:

- Have a shared directory accessible to all cluster nodes in the same path (this will be your shared home directory). This must be a separate directory, and not located within the local home or install directory.

Upgrade to Data Center

Review and upgrade your apps

If you have any apps installed on your site, you'll need to upgrade to the Data Center app version, if one is available. To avoid any impact to your apps, we recommend you do this before you enter your Bamboo Data Center license key. Learn more about upgrading Server apps when you migrate to Data Center

Upgrade your Bamboo license

To upgrade from Bamboo Server to Bamboo Data Center:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **System**, select **License details**.
3. Enter your Bamboo Data Center license key.

4. Select **Save new license**.

Data Center features such as cold standby high availability, build resiliency, and single sign-on will now be available.

> For rate limiting to become available, restart your Bamboo server. For more information about rate limiting, go to Improving instance stability with rate limiting.

Set up your cluster

If your organization requires continuous uptime, scalability, and performance under heavy load, you'll want to run Bamboo Data Center in a cluster.

To find out more about clustering, including infrastructure requirements, see Clustering with Bamboo Data Center.

If you're ready to set up your cluster now, head to Installing Bamboo Data Center.

Looking to upgrade all your Atlassian applications to Data Center? We've got you covered:

- Upgrade from Jira Server to Jira Data Center
- Upgrade from Confluence Server to Confluence Data Center
- Upgrade from Crowd Server to Crowd Data Center

Considering moving to cloud? Plan your cloud migration.

End of support announcements for Bamboo

This page announces the end of support for various platforms and browsers used with Atlassian Bamboo.

Why is Atlassian ending support for these platforms?

Atlassian is committed to delivering improvements and bug fixes as fast as possible. We are also committed to providing world-class support for all the platforms our customers run our software on. However, as new versions of databases, web browsers, etc. are released, the cost of supporting multiple platforms grows exponentially, making it harder to provide the level of support our customers have come to expect from us. Therefore, we no longer support platform versions marked as end-of-life by the vendor or very old versions that are no longer widely used.

Platform	Announcement date	Details
PostgreSQL 11	October 2023	The deprecation of support for PostgreSQL 11. Support for this version of PostgreSQL will be removed entirely in a future version of Bamboo.
Oracle 12c R2	October 2023	The deprecation of support for Oracle 12c R2. Support for this version of Oracle will be removed entirely in a future version of Bamboo.
MySQL 5.7	October 2023	The deprecation of support for MySQL 5.7. Support for this version of MySQL will be removed entirely in a future version of Bamboo.
Perforce	October 2023	The deprecation of support for Perforce version control. Support for Perforce will be removed entirely in a future version of Bamboo.
32-bit binaries	June 2023	Starting with Bamboo 9.3, the 32-bit Windows installer and ZIP are no longer available. The 32-bit ZIP distribution is replaced by a 64-bit version.
PostgreSQL 10	June 2023	The end of support for PostgreSQL 10 with the release of Bamboo 9.3.
Java 8	June 2023	The end of support for Java 8 (including Oracle JDK 8, OpenJDK 8 Adoptium OpenJDK 8) with the release of Bamboo 9.3.
Java 8	December 2022	Deprecation of Java 8 with the release of Bamboo 9.1.
PostgreSQL 10	April 2022	Deprecation of PostgreSQL 10 with the release of Bamboo 9.0.
End of Support for MySQL 5.6.3	December 2021	The end of support for MySQL 5.6.3 with the release of Bamboo 8.1.
End of support for PostgreSQL 9.6	December 2021	The end of support for PostgreSQL 9.6 with the release of Bamboo 8.1.
End of support for Microsoft SQL Server 2016	December 2021	The end of support for Microsoft SQL Server 2016 with the release of Bamboo 8.1.
Solaris	December 2021	The deprecation of the Solaris operating system on Bamboo agents with Bamboo 8.1 and later

Apache Lucene	November 2020	The deprecation of the Apache Lucene library with Bamboo 7.2 and later.
Deprecation of PostgreSQL 9.5	24 July 2020	The deprecation of the listed Postgres database versions with Bamboo 7.1 and later.
CVS repository type	24 July 2020	The deprecation of the CSV repository type with Bamboo 7.1 and later.
Deprecation of PostgreSQL 9.2, 9.3, 9.4	March 2020	The deprecation of the listed Postgres database versions with Bamboo 7.0 and later.
Oracle Database server 12c R1 (12.1.x)	March 2020	The deprecation of Oracle Database server 12c R1 with Bamboo 7.0 and later.
Bamboo elastic agent AMIs	March 2020	Starting from Bamboo version 7.0, official support for Bamboo elastic agent AMIs will be deprecated. Eventually the support will be dropped and Bamboo won't be preparing and releasing Bamboo elastic agent AMIs. In order to use elastic agents Bamboo users will need to build their own AMIs.
Internet Explorer 11	September 2019	In 2015 Microsoft released Edge as the browser to supersede Internet Explorer, and in recent times Microsoft has discouraged the use of Internet Explorer as a default browser. To allow us to continue to take advantage of modern web standards to deliver improved functionality and the best possible user experience across all of our products, we have decided to end support for Internet Explorer 11. End of support means we will not fix bugs specific to Internet Explorer 11, and will begin to introduce features that aren't compatible with this browser. **When is this happening?** • Bamboo 6.10 will be the last version to support Internet Explorer 11. • Subsequent versions will not support Internet Explorer 11. **What this means for you** We recommend switching to one of our supported browsers.
Bitbucket Cloud REST API 1.0	April 2019	The deprecation of Bitbucket Cloud REST API with Bamboo 6.8 release.
Deprecation of jGit	January 2018	The deprecation of the JGit with Bamboo 6.8 release.
End of support for Oracle 11g	18 May 2016	The end of support for Oracle 11g databases with the upcoming release of Bamboo Server 5.12.
End of support for MySQL 5.5	18 May 2016	The end of support for MySQL 5.5 with the upcoming release of Bamboo Server 5.12.

Deprecation of PostgreSQL 9.0 and 9.1	16 May 2016	The deprecation of postgreSQL 9.0 and 9.1 with the upcoming release of Bamboo Server 5.12.
End of support for PostgreSQL 8	13 April 2016	The end of support for PostgreSQL 8 with the upcoming release of Bamboo Server 5.11.
End of support for Microsoft SQL Server 2005 and 2008	13 April 2016	The end of support for Microsoft SQL Server 2005 and 2008 with the upcoming release of Bamboo Server 5.11.
Deprecation of Oracle 11g	13 April 2016	The deprecation of the Oracle 11g database with the upcoming release of Bamboo Server 5.11.
End of support for Windows 2008 and Amazon Linux 32bit stock images		The end of support for Windows 2008 and Amazon Linux 32bit images in Bamboo Server with the upcoming release of Bamboo 5.10. We will provide the last refreshed version of these images with Bamboo 5.10.0. After 5.10.0, the images will continue to be available on upgraded instances, but will not be shown anymore on new installations. Note that the 32bit variant of Amazon Linux has been abandoned by Amazon in 2014 and no longer receives security fixes.
End of support for MySQL 5.1		The end of support for MySQL 5.1 in Bamboo Server with the upcoming release of Bamboo 5.10.
End of support for Internet Explorer 9 and 10		The end of support for Internet Explorer 9 and 10 in Bamboo Server with the upcoming release of Bamboo 5.10.

End of support for JDK 6 and 7	26 October 2015	We're announcing the end of support for JDK 6 and 7 in Bamboo Server with the upcoming release of Bamboo 5.10. It means that agents, custom images, and Bamboo itself won't run against a JDK less than 8. We highly recommend upgrading to the latest version of JDK 8 as soon as you upgrade to Bamboo 5.10 to avoid any issues with existing agents or custom images. The stock images are upgraded automatically. **Note:** Atlassian doesn't provide support for customized images. Bamboo provides flexibility to use customized machine images, but it's impossible for us to support all individual configurations. *Tips:* • Try to match the layout and scripts of our stock images as closely as possible. • Choose Oracle if you have the choice between Oracle and OpenJDK flavor of JDK. **Related links:** • Bamboo supported platforms • Managing your elastic images • Managing your elastic agents • Latest Oracle JDK 8 download
Deprecation of Windows 2008 images	16 October 2015	The deprecation of support for Windows 2008 images. After 5.1 s0.0, the images will continue to be available on upgraded instances, but will not be shown anymore on new installations.
Deprecation of Amazon Linux 32bit images	16 October 2015	The deprecation of support for 32bit Amazon Linux images. We will provide the last refreshed version of these images with Bamboo 5.10.0. After 5.10.0, the images will continue to be available on upgraded instances, but will not be shown anymore on new installations. Note that the 32bit variant of Amazon Linux has been abandoned by Amazon in 2014 and no longer receives security fixes.
Deprecation of PostgreSQL 8	17 March 2015	The deprecation of support for PostgreSQL 8 in Bamboo. PostgreSQL 8 will no longer be supported in a future release of Bamboo.
Deprecation of Microsoft SQL Server 2005 and 2008	17 March 2015	The deprecation of support for Microsoft SQL Server 2005 and 2008 in Bamboo. Microsoft SQL Server 2005 and 2008 will no longer be supported in a future release of Bamboo.
Deprecation of MySQL 5.1	17 March 2015	The deprecation of support for MySQL 5.1 in Bamboo. MySQL 5.1 will no longer be supported in a future release of Bamboo.
Deprecation of Java 7	17 March 2015	The deprecation of support for Java 7 in Bamboo. Java 7 will no longer be supported in a future release of Bamboo.
Deprecation of Java 6	11 November 2014	The deprecation of support for Java 6 in Bamboo. Java 6 will no longer be supported in a future release of Bamboo.

<table>
<tr><td>Deprecation of Apache Tomcat 5.5 and 6.0</td><td>11 February 2014</td><td>In version 5.5, Bamboo will no longer support Apache Tomcat 5.5 and 6.0, and will only support Apache Tomcat 7.0 and above. Bamboo 5.5 is expected to be released later in 2014.</td></tr>
<tr><td>Deprecation of Internet Explorer 8</td><td>15 October 2013</td><td>In version 5.3, Bamboo will no longer support Internet Explorer 8, and will only support Internet Explorer 9 and above. Bamboo 5.3 is expected to be released later in 2013.</td></tr>
<tr><td>Deprecation of Maven Artifact Sharing plugin 8</td><td>15 October 2013</td><td>In version 5.3, Bamboo will no longer support the Maven artifact sharing plugin. Bamboo 5.3 is expected to be released later in 2013.</td></tr>
<tr><td>Deprecated Databases for Bamboo</td><td>4 October 2011</td><td>This section announces the end of Atlassian support for certain database versions for Bamboo. End of support means that Atlassian will not fix bugs related to certain database versions past the support end date.

We will stop supporting the following database versions in Bamboo 3.4, from December 2011:

• MySQL 5.0
• Oracle 10g

The details are below. Please refer to the list of supported platforms for details of platform support for Bamboo. If you have questions or concerns regarding this announcement, please email <code>eol-announcement at atlassian dot com</code>.

End of Life Announcement for Database Support
<table><tr><th>Database</th><th>Support End Date</th></tr><tr><td>MySQL 5.0</td><td>When Bamboo 3.4 releases, after December 2011</td></tr><tr><td>Oracle 10g</td><td>When Bamboo 3.4 releases, after December 2011</td></tr></table>• Notes for MySQL 5.0 and Oracle 10g:
  ◦ Atlassian intends to end support for MySQL 5.0 and Oracle 10g in Bamboo 3.4. Bamboo 3.3 is the last version that will support MySQL 5.0 and Oracle 10g.
  ◦ 'Support End Date' means that Bamboo 3.3 and previously released versions will continue to work with MySQL 5.0 and Oracle 10g. However, Atlassian will not fix bugs affecting MySQL 5.0 and Oracle 10g past the support end date.
  ◦ Bamboo 3.4 will not be tested with MySQL 5.0 and Oracle 10g.</td></tr>
</table>

<table>
<tr>
<td>Deprecated Java Platforms for Bamboo</td>
<td>16 February 2011</td>
<td>
This section announces the end of Atlassian support for certain Java Platforms for Bamboo.

We will stop supporting the following Java Platforms:

• From Bamboo 3.1, due in the first half of 2011, support for Java Platform 5 (JDK/JRE 1.5) will end.

We are ending support for Java Platform 5, in line with Sun's Java SE Support Road Map (i.e. "End of Service Life" for Java Platform 5 dated October 30, 2009). We are committed to helping our customers understand this decision and assist them in updating to Java Platform 6, our supported Java Platform.

The details are below. Please refer to the Supported platforms page for more details regarding platform support for Bamboo. If you have questions or concerns regarding this announcement, please email <code>eol-announcement at atlassian dot com</code>.

End of Life Announcement for Java Platform Support
<table>
<tr><th>Java Platform</th><th>Support End Date</th></tr>
<tr><td>Java Platform 5 (JDK /JRE 1.5)</td><td>When Bamboo 3.1 releases, due in the first half of 2011</td></tr>
</table>
• Java Platform 5 End of Support Notes:
  ◦ 'Support End Date' means that Bamboo 3.0.x and previous released versions will continue to work with Java Platform 5 (JDK /JRE 1.5), however we will not fix bugs related to Java Platform 5 past the support end date.
  ◦ Bamboo 3.1 will only be tested with and support Java Platform 6 (JDK/JRE 1.6).
  ◦ If you have concerns with this end of support announcement, please email <code>eol-announcement at atlassian dot com</code>.
</td>
</tr>
</table>

<table>
<tr>
<td>Deprecated Web Browsers for Bamboo</td>
<td>16 February 2011</td>
<td>
This section announces the end of Atlassian support for certain web browser versions for Bamboo. End of support means that Atlassian will not fix bugs related to certain web browser versions past the support end date.

We will stop supporting the following web browser versions from Bamboo 3.0, due February 2011:

• Microsoft Internet Explorer 7 (IE7)

The details are below. Please refer to the list of supported platforms for details of platform support for Bamboo. If you have questions or concerns regarding this announcement, please email <code>eol-announcement at atlassian dot com</code>.

End of Life Announcement for Web Browser Support
<table>
<tr><th>Web Browser</th><th>Support End Date</th></tr>
<tr><td>Microsoft Internet Explorer (version 7 only)</td><td>When Bamboo 3.0 releases, due February 2011</td></tr>
</table>
• Internet Explorer Notes:
  ◦ Atlassian intends to end support for IE7 in Bamboo 3.0. Bamboo 2.7 is the last version that will support IE7.
  ◦ IE8 will still be supported.
  ◦ 'Support End Date' means that Bamboo 2.7 and previously released versions will continue to work with IE7. However, we will not fix bugs affecting IE7 past the support end date.
  ◦ Bamboo 3.0 will not be tested with IE7.
</td>
</tr>
</table>

Running the Setup Wizard

When you launch Bamboo for the first time, the Bamboo Setup Wizard will display. The Wizard will lead you through the Bamboo settings that you need to configure before you can start using it.

Before you begin

If you are currently using Atlassian's Crowd with Bamboo and wish to import existing data into Bamboo (see Step 5. Starting Data below), you will need to *disable Crowd* before starting the Setup Wizard. To do this, select **Administration > Security > User directories**.

You can then re-enable Crowd and restart Bamboo at the completion of the Setup Wizard.

Step 1. License Details and Setup Method

You must have a valid Bamboo license (evaluation or commercial) to use Bamboo. You can generate your own Bamboo evaluation license from your MyAtlassian self-service account here.

Once you have entered a valid license key, you can choose which setup method you prefer for your Bamboo installation:

- **Express installation** — use this method if you are evaluating or demonstrating Bamboo.
  - The Express installation method requires only a minimum of configuration information. It sets up Bamboo with default settings and an embedded database (H2).
  - If you choose the Express installation method you can skip to Step 6. Set Up Administrator User below.

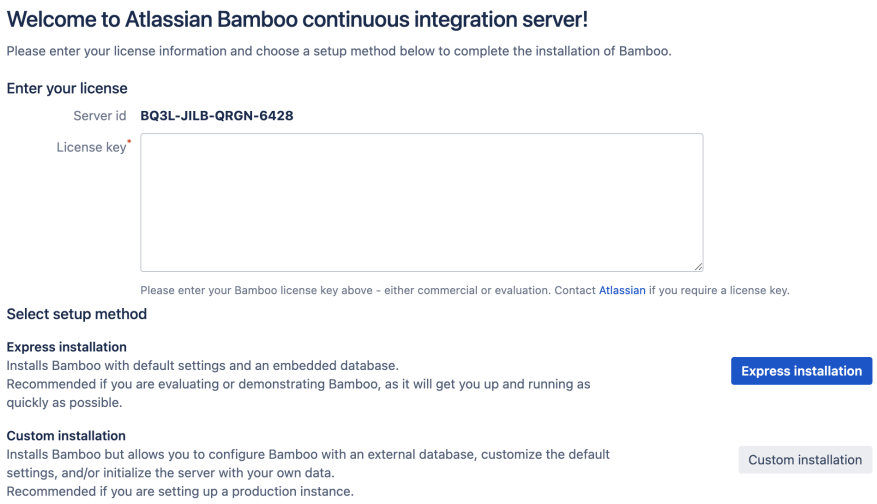
On this page:

Related pages:

- Installing Bamboo on Linux
- Installing Bamboo on Mac OS X
- Installing Bamboo on Windows

Custom installation — use this method if you are setting up a production instance of Bamboo.

- The Custom installation method takes longer, but allows you to configure Bamboo with an external database, customize the default settings, and/or initialize the server with your own data.
- If you choose the Custom installation method, proceed to Step 2. General Configuration below.



Step 2. General Configuration

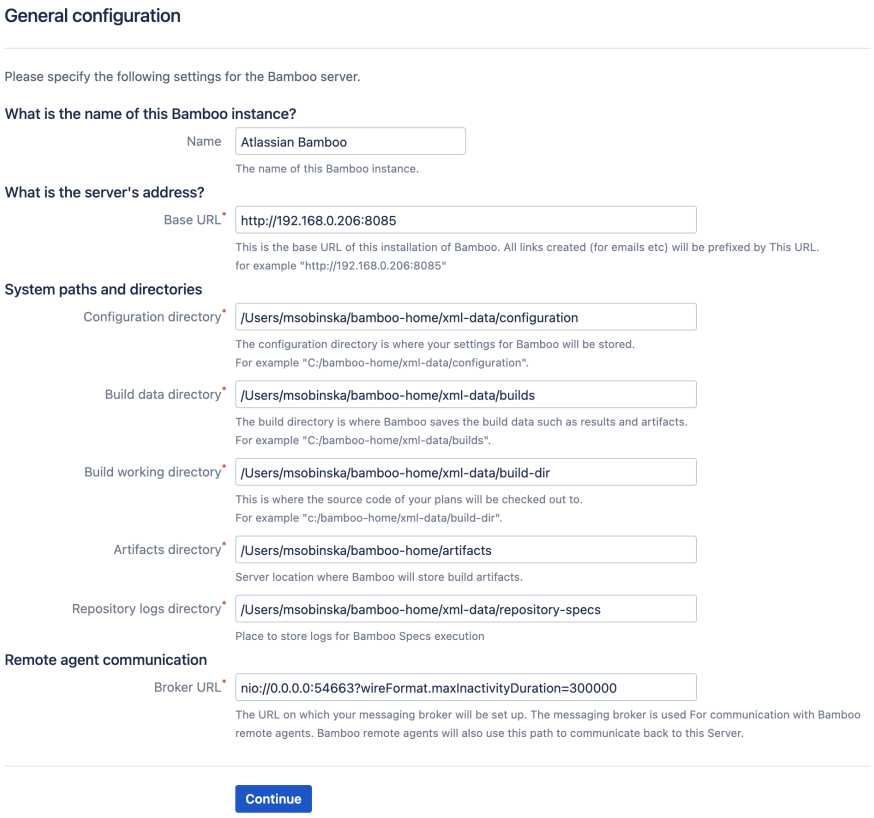
This step applies to the Custom installation method only.

On this page you specify a number of Bamboo server settings, such as the address of the server, where data is stored and the message broker used to communicate with remote agents.

⚠ You may find it simplest to keep the default settings for the three directory settings in the table. For more information please see Locating important directories and files.

Setting	Details
Name	See Specifying Bamboo's Title
Base URL	See Specifying Bamboo's URL
Configuration Directory	The location for Bamboo configuration files.
Build Data Directory	The location for Bamboo project data files.
Build Working Directory	The location of project files checked out from source control.
Broker URL	Only visible if you are permitted remote agents under your Bamboo license. The URL of the embedded messaging broker that Bamboo sets up to communicate with its remote build agents. This URL will be written to `bamboo.cfg.xml` as a property. You can update this file if you want to change your Broker URL. • Replace `localhost` with the real host name or IP address of your Bamboo server. You should not use `localhost` as the host name in the Broker URL, as remote agents are provided with the Broker URL on startup and use it to communicate to the server. • If port number `54663` is already in use, specify a different port number.

Broker client URL	The URL used by agents to connect to the broker. This URL will be written to `bamboo.cfg.xml` as a property. You can update this file if you want to change your Broker URL. The default includes an actual IP of the Bamboo server. Specify if the default IP is not reachable from the agents.



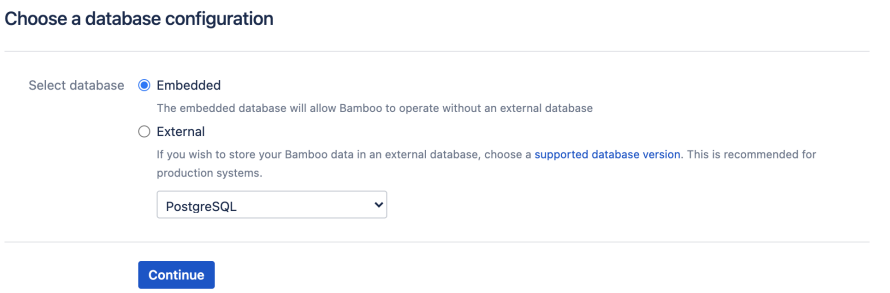
Step 3. Choose a Database Configuration

This step applies to the Custom installation method only.

Picking a database configuration is an important choice. If you pick the Embedded database configuration, you do not have to set up a database. However, the embedded H2 database is **only suitable for evaluation purposes**. You will need to move to an external database if you decide to deploy Bamboo in production at a later stage (as described in Move data to a different database).

Choose one of the following:

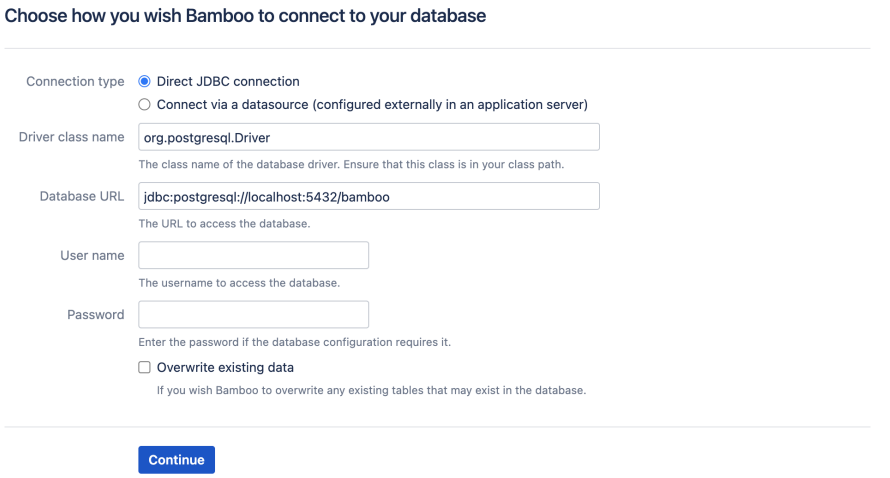
- **Embedded** — Choose this for quick and easy first-time installation of Bamboo. This option is suitable for evaluation purposes only. Skip to Step 5. Starting Data.
- **External** — Choose this if you wish to use an external database. Proceed to Step 4. Database Configuration below.



Step 4. Database Configuration

This step applies to the Custom installation method only.

If you selected External database in Step 3, you will need to provide the configuration details for your database. Please see Connect Bamboo to an external database for further instructions.



Step 5. Starting Data

This step applies to the Custom installation method only.

On this page you specify how Bamboo will populate the home directory that you set up when you installed Bamboo.

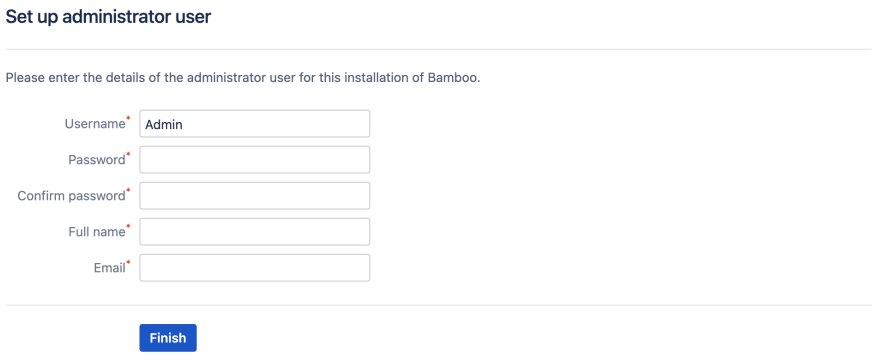
Choose one of the following:

- **Create a new Bamboo home** — choose this if you are performing a normal installation or upgrade.
- **Import existing data** — only choose this under exceptional circumstances, e.g. if you are connecting Bamboo to a different database, or moving your pre-existing Bamboo installation to a different server. Avoid importing backups from different versions of Bamboo.

Step 6. Set Up Administrator User

The final step of the Setup Wizard is to enter the details of the first registered user for the Bamboo system. This user will have global administrative privileges over the entire installation of Bamboo and should not be removed.

Once you have entered the details for your administrator user, select **Finish**. The Bamboo dashboard will be displayed.



Congratulations, you have successfully set up Bamboo!

Bamboo remote agent installation guide

⚠ Starting from Bamboo 9.4, remote agents support Java 17. If you want to migrate any of your existing remote agent instances to Java 17, you'll need to reinstall the affected remote agent wrappers.

Before you begin

Before you begin installing remote agents, there are a few questions you need to answer.

Do you need to install a remote agent?

See Agents and capabilities to understand how remote agents interact with your Bamboo server.

Do you have sufficient agent licenses?

See Bamboo licensing for details.

Does your system meet the minimum requirements?

See Supported platforms.

Do you have a supported version of Java installed on the agent machine?

See Supported platforms.

Are you upgrading your version of Bamboo?

- If you're upgrading Bamboo from a release **earlier than 6.10** to a release that's also **earlier than 6.10**, your remote agents will be upgraded automatically along with Bamboo.
- If you're upgrading Bamboo from a release **earlier than 6.10** to **6.10 or later**, manually upgrade your remote agents to make sure that the Tanuki wrapper is also updated and the `wrapper.conf` file is successfully recreated.

  ⓘ Bamboo 6.10 comes with an updated Tanuki wrapper that fixes some known issues but is not updated automatically and must be reinstalled.

  1. Stop the original agent.
  2. Backup the `BAMBOO_AGENT_HOME/conf/wrapper.conf` file.
  3. Remove the `BAMBOO_AGENT_HOME/conf` directory.
  4. Download a new Remote Agent JAR from your Bamboo Server: http://<*BAMBOO_URL*>/admin/agent/addRemoteAgent.action.
  5. Launch the remote agent. T
     This will create new `wrapper.conf` file. If you want to re-enact your custom configuration in that file, you can do it now.
- If you're upgrading Bamboo from a release **earlier than 8.0** to **8.0 or later** and you're also upgrading the **Java version** on your remote agents **from 8 to 11**, either update the wrapper configuration manually or download a new remote agent JAR from your upgraded Bamboo and reinstall the wrapper.

On this page:

Related pages:

- Configuring remote agent capabilities using bamboo-capabilities.properties
- Implementing a remote agent service wrapper
- Legacy remote agent installation guide
- Enabling creation of remote agents
- Specifying a Broker URL

1. Stop the original agent.
2. Backup the `BAMBOO_AGENT_HOME/conf/wrapper.conf` file.
3. Remove the `BAMBOO_AGENT_HOME/conf` directory.
4. Download a new Remote Agent JAR from your Bamboo Server: http://*<BAMBOO_URL>*/admin/agent/addRemoteAgent.action.
5. Launch the remote agent. This will create new `wrapper.conf` file. If you want to re-enact your custom configuration in that file, you can do it now.

1. Stop the original agent.
2. Open wrapper configuration file - `BAMBOO_AGENT_HOME/conf/wrapper.conf`
3. Set the value of the `wrapper.java.version.max` property to 11.
4. Start the agent.

- If you're upgrading Bamboo **from an earlier version to versions 8.2–9.3**, download a new Remote Agent JAR from your upgraded Bamboo and reinstall the wrapper.

    1. Stop the original agent.
    2. Back up the `BAMBOO_AGENT_HOME/conf/wrapper.conf` file.
    3. Remove the `BAMBOO_AGENT_HOME/conf` directory.
    4. Download a new Remote Agent JAR from your Bamboo Server at http://*<BAMBOO_URL>*/admin/agent/addRemoteAgent.action.
    5. Launch the remote agent. This will create new `wrapper.conf` file. If you want to re-enact your custom configuration in that file, you can do it now.

- If you're upgrading Bamboo from **versions 8.0–9.3 to version 9.4 or later**, either download a new remote agent JAR from your upgraded Bamboo and reinstall the wrapper or update the wrapper configuration manually.

    1. Stop the original agent.
    2. Back up the `BAMBOO_AGENT_HOME/conf/wrapper.conf` file.
    3. Remove the `BAMBOO_AGENT_HOME/conf` directory.
    4. Download a new Remote Agent JAR from your Bamboo Server at http://*<BAMBOO_URL>*/admin/agent/addRemoteAgent.action.
    5. Launch the remote agent. This will create new `wrapper.conf` file. If you want to re-enact your custom configuration in that file, you can do it now.

    1. Stop the original agent.
    2. Open the wrapper configuration file `<BAMBOO_AGENT_HOME>/conf/wrapper.conf`.
    3. Set the following property values:

```
wrapper.java.version.min=11
wrapper.java.version.max=17
wrapper.java.version.fallback=11
wrapper.java.additional.4=--add-opens=java.
base/
java.util=ALL-UNNAMED
wrapper.java.additional.5=--add-opens=java.
base/
java.lang=ALL-UNNAMED
wrapper.java.additional.6=--add-opens=java.
base/
sun.net.www.protocol.http=ALL-UNNAMED
wrapper.java.additional.7=--add-opens=java.
base/
sun.net.www.protocol.https=ALL-UNNAMED
```

4. Start the agent.

> Mixing operating systems between Bamboo Server and Remote Agents is supported and possible.
>
> For example, you can connect a Linux based Remote Agent to a Windows Bamboo Server or a Windows Remote Agent to a Linux based Bamboo Server.
>
> The only requirement is that it's running on a supported Operating System and Java version:
>
> - Supported platforms

> Note that you can run multiple Bamboo agents on the same machine – you just need to provide a separate home directory for each agent installation.

1. Enable remote agent support

1. From the Bamboo header select > **Build resources** > **Agents**.
2. Select either **Enable Remote Agent Support** or **Disable Remote Agent Support**.

For more information on enabling and disabling remote agent support, see Disabling and enabling remote agents support.

2. Download and install the remote agent

1. Create a directory on the agent machine (e.g. `bamboo-agent-home`) to serve as the Bamboo agent home for the remote agent.
2. From the Bamboo header select > **Build resources** > **Agents**.
3. The Agents screen displays showing the lists of all local agents and all remote agents that currently exist on your Bamboo system.
4. If not already enabled, select the **Enable remote agent support** link.
5. Select **Install remote agent**. The Installing a remote agent screen will display.
6. Select **DOWNLOAD Remote Agent JAR** and save the JAR file to the directory on the agent machine that you created above.
7. Copy the command under Running a Remote Agent to the clipboard for use in Step 3 that follows.

3. Launch the remote agent

Once installed, run the remote agent by executing the command line obtained above. This command will look something like this:

```
java -jar atlassian-bamboo-agent-installer-X.X-SNAPSHOT.jar http://bamboo-host-server:8085/bamboo
/agentServer/
```

Where X.X represents your Bamboo version number.

> If you are having issues launching the agent, then take a look at our troubleshooting guide.

> The name of the `.jar` file (for example, `atlassian-bamboo-agent-installer-5.4-SNAPSHOT.jar)` will vary depending on the version of Bamboo you are running.

You can run the remote agent with a number of additional command line parameters. Configuration options include remote agent data storage, capability detection and logging, suppression of self-signed certificate and running without the Remote Agent Supervisor or with different start-up commands.

See Additional remote agent options for more information.

4. Configure the remote agent's capabilities

All remote agents feature a capability that can be defined. Examples include an executable, such as Maven, a JDK, a DVCS client or a custom capability. They typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of them.

Capabilities can be defined specifically for an agent, or shared between all local or all remote agents.

See Configuring capabilities for more on defining capabilities.

Configuring remote agent capabilities using bamboo-capabilities.properties

You can define the capabilities for a specific remote agent by using a configuration file on the agent machine. When the Bamboo agent starts up, it will look in the current working directory of the remote agent Java runtime process (i.e. `<bamboo-agent-home>/bin` when the remote agent is started with the `<bamboo-agent-home>/bin/bamboo-agent.sh` script.) for a file named `bamboo-capabilities.properties`. The capabilities defined in that file will then be published for the Bamboo agent after registering.

If the remote agent is started without the wrapper supervisor (as explained here: https://confluence.atlassian.com/bamboo/additional-remote-agent-options-436044733.html), then the current working directory for the Java runtime of the remote agent will be the directory where the Java command is run. In this case the `bamboo-capabilities.properties` file needs to be located in this directory.

To configure remote agent capabilities:

1. Shut down the remote agent, if it is running.
2. Create a file named `bamboo-capabilities.properties` in the current working directory of the Java runtime process of the remote agent (i.e. `<bamboo-agent-home>/bin`) on the agent machine.

3. Edit the `bamboo-capabilities.properties` file to add capabilities. You need to use the formats shown below:
   Notes:
   - Use '\' to escape spaces, periods and backslashes ('\').
   - All capabilities other than custom capabilities should start with '`system`'.

   JDK capabilities

   ```
   system.jdk.JDK\ <jdk number>=<jdk location>
   ```

   Examples:

   ```
   system.jdk.JDK\ 1.6=/System/Library/Frameworks/JavaVM.framework/Versions/1.6
   system.jdk.JDK\ 1.6=C:\\Program Files\\Java\\jdk6.0.17
   ```

   Executable capabilities

   ```
   system.builder.<executable type>.<executable label>=<executable path>
   ```

   Examples:

   ```
   system.builder.ant.Ant=/opt/apache-ant-1.7.1
   system.builder.maven.Maven\ 1=/opt/maven-1.0.2
   system.builder.mvn2.Maven\ 2=/opt/maven-2.0
   system.builder.node.Node.js\ 0.12=/opt/node-0.12/bin/node
   system.builder.devenv.Visual\ Studio\ 2022=/usr/bin/code
   ```

   Version control capabilities

   ```
   system.<DVCS>.executable=<DVCS command location>
   ```

   Examples:

   ```
   system.git.executable=/usr/bin/git
   system.hg.executable=/usr/bin/hg
   ```

   Perforce capabilities

   ```
   system.p4Executable=<perforce executable location>
   ```
   Example:

   ```
   system.p4Executable=/usr/bin/p4
   ```

   Custom capabilities

   ```
   <custom capability name>=<custom capability value>
   ```

   Example:

   ```
   system.os=osx
   ```

4. Save your changes to the `bamboo-capabilities.properties` file.
5. Start up your remote agent. The capabilities defined in the `bamboo-capabilities.properties` file will be configured for your agent.

Create a `bamboo-capabilities.properties` file, based on an existing Agent

If you are looking to create a `bamboo-capabilities.properties` file, based on an existing Remote Agent, please check the following KB article:

- How to export a remote agent capability list to bamboo-capabilities.properties format

Legacy remote agent installation guide

If you have implemented your own remote agent service wrapper or have problems with the service wrapper used by the remote agent supervisor in Bamboo, you can install the legacy remote agent (pre-Bamboo 2.2) which does not have a service wrapper.

Before you begin:

- **Not sure whether to install a Remote Agent?** See About Agents to understand how Remote Agents interact with your Bamboo server.
- **Ensure that you have specified the Broker URL**, as described in the Bamboo Setup Wizard.
- **Do you have sufficient Agent licenses?** See Bamboo licensing for details.
- **Have you enabled the creation of Remote Agents**, as described in Disabling and enabling remote agents support.

- **Ensure that you have Java Runtime Environment 5.0 or later** installed on the agent machine.

Step 1. Download and install the Legacy Remote Agent

1. Create a directory on the agent machine (e.g. `bamboo-agent-home`), to serve as the Bamboo agent home for the remote agent.
2. In the upper-right corner of the screen, select **Administration** > **Overview**.
3. Under **Build resources**, select **Agents**.
4. On the **Agents summary** page, Select **Install remote agent**.
5. Select **bamboo-agent-.jar** under the Running the agents without the service wrapper section and save the JAR file to the directory you created in step 1.1.

Note that if you configure the capabilities of the remote agent using a `bamboo-capabilities.properties` file, that file should be located in the same directory as the JAR file (that is, `bamboo-agent-home` in the above instructions).

Step 2. Launch the Remote Agent

Once installed, you can run the remote agent by executing the command line obtained in the previous step. This command will look something like the following:

```
java -jar bamboo-agent-2.0-SNAPSHOT.jar http://bamboo-host-server:8085/agentServer/
```

You may wish to configure the remote agent machine to start the Bamboo remote agent automatically when the machine boots. Please consult your operating system documentation for instructions on how to do this.

You can also choose to run the remote agent with different command line parameters, to change where the remote agent stores its data or suppress the self-signed certificate of the server.

Changing where the remote agent stores its data

By default, the remote agent will store its data in a directory called `bamboo-agent-home`. If you wish to specify a different directory, add the following command line parameter:

```
-Dbamboo.home=RemoteAgentHome
```

where *RemoteAgentHome* is the path to the Bamboo agent home directory you created in step 1.1.

Your command line will look something like this:

```
java -Dbamboo.home=RemoteAgentHome -jar bamboo-agent-2.0-SNAPSHOT.jar http://bamboo-host-server:8085
/agentServer/
```

Suppressing the self-signed certificate of the server

If your Bamboo server uses SSL (https) with a self-signed certificate, you will need to carry out one of the following two options:

- **Add the parameter -Dbamboo.agent.ignoreServerCertName=true to the remote agent's command line**, for example:
  java -Dbamboo.agent.ignoreServerCertName=true -jar bamboo-agent-2.0-SNAPSHOT.jar http://bamboo-host-server:8085/agentServer/
  Please be aware that this **reduces the security of your configuration**, as the identity of your Bamboo server will not be authenticated by the remote agent.
- **Use the keytool utility to add the self-signed certificate to the trusted certificates in your keystore**. This is a more secure option, but is complex to set up. For detailed instructions of how to do this, please refer to the relevant Oracle documentation.

Step 3. Configure the Remote Agent's Capabilities

Please see Configuring capabilities.

Synchronizing remote agent capabilities with Bamboo Server

When an existing remote agent reconnects to the server, its capabilities are synchronized with capabilities defined on the server. A remote agent's capabilities can be defined in the following ways:

- manually in Bamboo server (in the UI or by using REST)
- in the `bamboo-capabilities.properties` file
- automatically detected by the agent

The capabilities' sources given above are listed in order of priority. If capability is defined in several places then the capability is taken from the source with highest priority. For example, if an agent's capability has been defined both in Bamboo server and in the `bamboo-capabilities.properties`, the value of capability defined in Bamboo server is going to be used. A current source of a given capability can be checked in a remote agent capabilities page in Bamboo.

Note that modifying the `bamboo-capabilities.properties` file requires restarting the agent in order to synchronize capabilities with the server.

Running Bamboo as a service

You can configure Bamboo to start automatically on system startup, allowing it to recover automatically after a reboot.

Running Bamboo as a Windows service

Running Bamboo as a Windows service as the local user

Running Bamboo as a Linux service

Running Bamboo as a Windows service

Once you have installed Bamboo, you can choose to run Bamboo as a service so that it starts up every time Windows restarts.

Upgrading Bamboo server

- If you have just upgraded your Bamboo server, you must re-install the Bamboo service. You can do this by removing the service and installing it again.
- If you run Bamboo as a Windows service, make sure that the user has Full control access to the following folders:
  - In version 5.1 or later:
    - `<Bamboo_install_directory>\temp`
  - In version 5.15 up to 7.x:
    - `<Bamboo_install_directory>\temp`
    - `<Bamboo_home_directory>\xml-data\configuration\cipher`
  - In version 8.x and later:
    - `<Bamboo_install_directory>\temp`
    - `<Bamboo_home_directory>\shared\configuration\cipher`

Procedure:

1. Click on the **Start menu** in Windows
2. Select **Bamboo** from the programs list
3. Click on the **Install service** option to install Bamboo as a service in Windows
4. Configure the Service to run as a local user

   Due to changes in recent versions of Tomcat and the Commons Daemon, the default user for the Service in Bamboo 7.0.2 and above is Local Service instead of Local System. To work around the restrictions of the Local Service user, our recommendation is to run as a local user.

5. Click **Start service** to start the service
6. You can alternatively run the following batch files to achieve the same steps:
   a. `<Bamboo_install_directory>\InstallAsService.bat`
   b. `<Bamboo_install_directory>\UninstallService.bat`

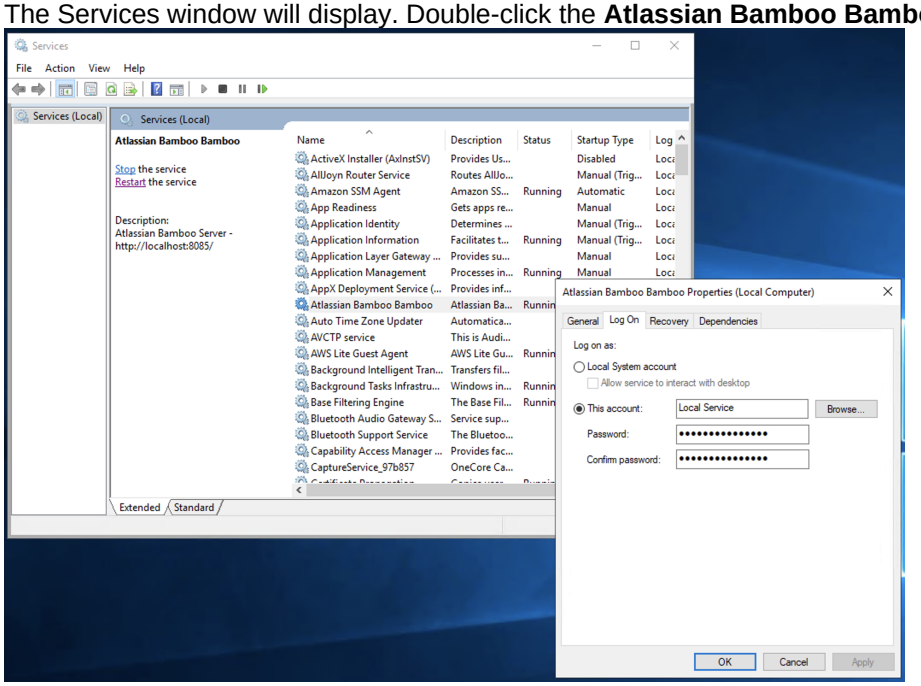
Running Bamboo as a Windows service as the local user

1. Install Bamboo Application Server

1. Download Bamboo and run the Setup Wizard.
2. Install Bamboo as a Windows service, as described in Running Bamboo as a Windows service.

2. Edit the Bamboo service to run as the local user

1. Select **Start > Run** and enter '**services.msc**'.
2. The Services window will display. Double-click the **Atlassian Bamboo Bamboo** row.

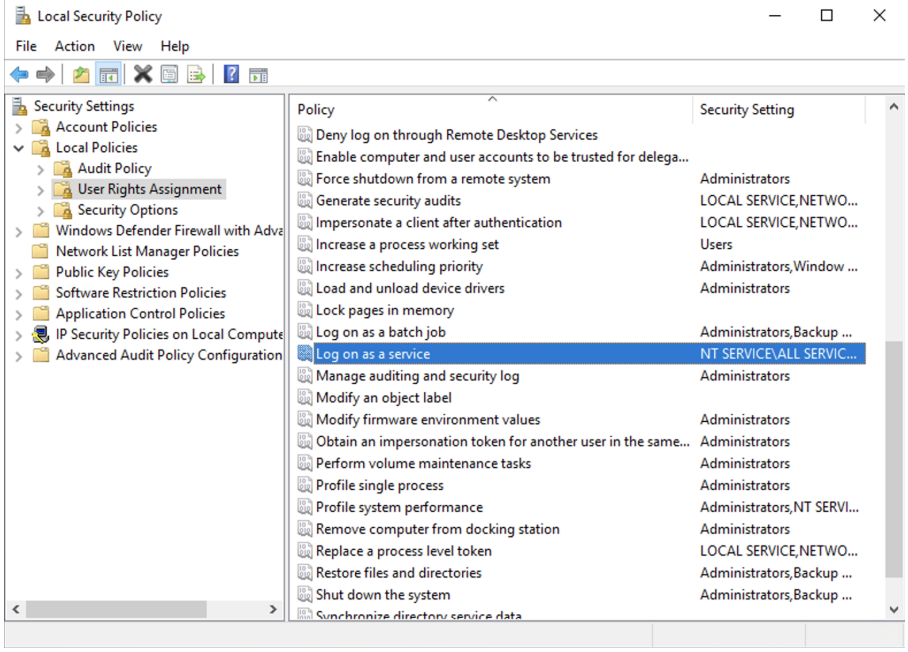


3. The Bamboo build server Properties window will display. Select **This account** and provide a local admin account credentials, then select **OK** to apply your changes.

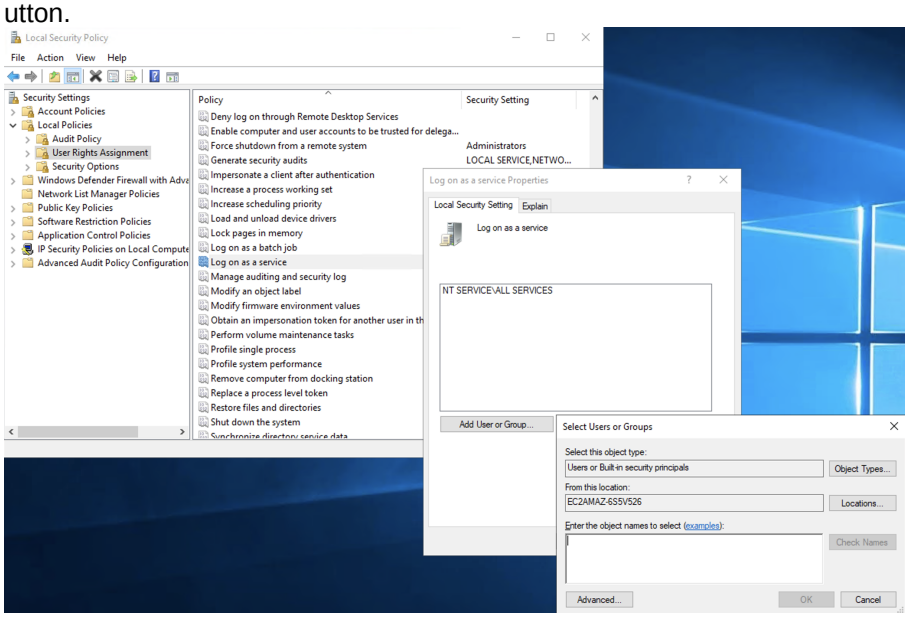
3. Give the local user access to "log on as a service"

1. Select **Start > Run** and enter '**secpol.msc**'.
2. The Local Security Settings window will display. Expand the **Local Policies** tree and select **User Rights Assignment**.

3. Scroll down and find the **Log on as a service** policy. Double-click the **Log on as a service** policy.



4. The properties window for the **Log on as a service** policy will display. Select the **Add User or Group...** button.



5. The **Select Users or Groups** window will display. Enter your local user and select **OK** to allow your user to log on as a service.
6. Select **OK** and close all open windows.

Bamboo will now start as a service, under the local user.

Running Bamboo as a Linux service

⚠ Linux system administration is outside the scope of Atlassian support. This page is provided for your information only.

On Linux/Solaris, the best practice is to install, configure and run each service (including Bamboo) as a dedicated user with only the permissions they require.

To install, configure and get Bamboo to start automatically on Linux/Solaris:

1. Create a bamboo user account that will be used to run Bamboo. For example, enter the following at a Linux console:

   ```
   sudo useradd --create-home -c "Bamboo role account" bamboo
   ```

2. Create a directory into which Bamboo will be installed. For example:

   ```
   sudo mkdir -p /opt/atlassian/bamboo
   sudo chown bamboo: /opt/atlassian/bamboo
   ```

3. Log in as the bamboo user to install Bamboo:

   ```
   sudo su - bamboo
   ```

4. You need to extract Bamboo:

   ```
   cd /opt/atlassian/bamboo
   tar zxvf /tmp/atlassian-bamboo-X.Y.tar.gz
   ln -s atlassian-bamboo-X.Y/ current
   ```

5. Edit `current/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties` and set `bamboo.home=/var/atlassian/application-data/bamboo (or any other directory of your choice, but not the same as Bamboo's installation directory)`
6. Proceed with the service configuration. There are two options included below for creating the service configuration, which one you will use will depend on your Linux distribution:

ⓘ Suitable for modern distributions such as:

- Ubuntu 15
- CentOS 7
- RHEL 7

For anything older see the SysV Init Script section below.

ⓘ Systemd will ignore environment variable definitions placed in /etc/environment as well as other traditional environment variable definitions from Sys-V init. If one needs to define environment variables when running Bamboo as a systemd unit then the variable definitions need to be placed in the unit file.

The examples below are designed for Bamboo Server. If you want to automate the Remote Agent service, simply replace ExecStart/ExecStop instructions with:

```
ExecStart=<bamboo-agent-home>/bin/bamboo-agent.sh
ExecStop=<bamboo-agent-home>/bin/bamboo-agent.sh
```

`Environment=CATALINA_PID` and `PIDFile` properties can be removed as the Remote Agent startup is controlled by a service wrapper.

1. Create a `bamboo.service` file in your `/etc/systemd/system` directory

```
[Unit]
Description=Atlassian Bamboo
After=syslog.target network.target

[Service]
Type=forking
User=<bamboo-user>
Environment=CATALINA_PID=<bamboo-install>/bin/Catalina.pid
PIDFile=<bamboo-install>/bin/Catalina.pid
ExecStart=<bamboo-install>/bin/start-bamboo.sh
ExecStop=<bamboo-install>/bin/stop-bamboo.sh
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

   The values for `<bamboo-user>` and `<bamboo-install>` should be replaced with your Bamboo user and the path to your Bamboo Install directory, respectively. `CATALINA_PID` and `PIDFile` are also declared, this way systemd knows which java process to monitor.
2. Enable the service to start at boot time by running the following in a terminal:

```
systemctl enable bamboo.service
```

3. Stop Bamboo using the provided Bamboo stop script (`<bamboo-install>/bin/stop-bamboo.sh`) and restart your system to check that Bamboo starts as expected
4. Use the following commands to manage the service:

Disable the service:

```
systemctl disable bamboo.service
```

Check that the service is set to start at boot time:

```
if [ -f /etc/systemd/system/*.wants/bamboo.service ]; then echo "On"; else echo "Off"; fi
```

Manually start and stop the service:

```
systemctl start bamboo
systemctl stop bamboo
```

Check the status of Bamboo:

```
systemctl status bamboo
```

1. As root, create the file `/etc/init.d/bamboo` (code shown below), which will be responsible for starting up bamboo after a reboot (or when manually invoked).

```
#!/bin/sh
set -e
### BEGIN INIT INFO
# Provides: bamboo
# Required-Start: $local_fs $remote_fs $network $time
# Required-Stop: $local_fs $remote_fs $network $time
# Should-Start: $syslog
# Should-Stop: $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Atlassian Bamboo Server
### END INIT INFO
# INIT Script
#######################################

# Define some variables
# Name of app ( bamboo, Confluence, etc )
APP=bamboo
# Name of the user to run as
USER=bamboo
# Location of application's bin directory
BASE=/opt/atlassian/bamboo/current

case "$1" in
  # Start command
  start)
    echo "Starting $APP"
    /bin/su - $USER -c "export BAMBOO_HOME=${BAMBOO_HOME}; $BASE/bin/startup.sh &> /dev/null"
    ;;
  # Stop command
  stop)
    echo "Stopping $APP"
    /bin/su - $USER -c "$BASE/bin/shutdown.sh &> /dev/null"
    echo "$APP stopped successfully"
    ;;
   # Restart command
   restart)
        $0 stop
        sleep 5
        $0 start
        ;;
  *)
    echo "Usage: /etc/init.d/$APP {start|restart|stop}"
    exit 1
    ;;
esac

exit 0
```

2. Make the init script executable:

```
chmod a+x /etc/init.d/bamboo
```

3. Place symlinks in the run-level directories to start and stop this script automatically.

    a. For Debian-based systems:

    ```
    update-rc.d bamboo defaults
    ```

    The following commands will be executed to place symlinks in the run-level directories:

```
Adding system startup for /etc/init.d/bamboo ...
   /etc/rc0.d/K20bamboo -> ../init.d/bamboo
   /etc/rc1.d/K20bamboo -> ../init.d/bamboo
   /etc/rc6.d/K20bamboo -> ../init.d/bamboo
   /etc/rc2.d/S20bamboo -> ../init.d/bamboo
   /etc/rc3.d/S20bamboo -> ../init.d/bamboo
   /etc/rc4.d/S20bamboo -> ../init.d/bamboo
   /etc/rc5.d/S20bamboo -> ../init.d/bamboo
```

b. For RedHat-based systems:

the init.d script contains chkconfig settings

```
sudo /sbin/chkconfig --add bamboo
```

4. Ensure the script is executed in the correct order, in particular after the database startup script.

Note: If starting your new `bamboo` service fails immediately with an error, it may be that your `/etc/init.d/bamboo` script has had carriage return characters introduced into it. You can confirm this by running:

```
cat -v /etc/init.d/bamboo
```

If there are carriage return characters in your `/etc/init.d/bamboo` script, they will appear as `^M` in the output:

```
#!/bin/sh^M
set -e^M
### BEGIN INIT INFO^M
# Provides: bamboo^M
# Required-Start: $local_fs $remote_fs $network $time^M
# Required-Stop: $local_fs $remote_fs $network $time^M
# Should-Start: $syslog^M
# Should-Stop: $syslog^M
```

You can remove carriage return characters from `/etc/init.d/bamboo` with the following command:

```
sed -i -e 's/\r//g' /etc/init.d/bamboo
```

Retry starting the service after making this change.

Get a Bamboo Data Center trial license

A trial license gives you access to a full instance of Bamboo Data Center for 30 days. At the end of the trial period your Bamboo Data Center site will become read-only and you'll have the option to buy a full license to continue using it, so you won't lose any of your projects or data.

We support single-node Bamboo Data Center both for trial and full license instances, so you don't have to modify your current number of application nodes if you don't want to scale up to a cluster yet.

To create a Bamboo Data Center trial license:

1. Head to my.atlassian.com and log in with your Atlassian ID.
2. From the list of Atlassian products, select **Bamboo**, then select the **Data Center** option and fill out the form with your organization's information.
3. Select **Generate license**.

If you're ready to scale up your instance, check out how to upgrade from Bamboo Server to Bamboo Data Center.

If you're a new customer, the next step is to download and set up your new Bamboo Data Center trial instance.

Using Bamboo

Atlassian Bamboo is a continuous integration (CI) and continuous delivery (CD) server. Bamboo assists software development teams by providing:

- automated building and testing of software source-code status.
- updates on successful/failed builds.
- reporting tools for statistical analysis.
- visibility into, and control over, release artifacts and environments.

This section has information about using Bamboo. Please see Administering Bamboo for information about managing the Bamboo server itself.

Continuous integration

Understanding the Bamboo CI Server

Configuring plans

Linking to source code repositories

Jobs and tasks

Working with builds

Sharing artifacts

Continuous delivery

Understanding deployment releases

Deployment projects

A sample deployment project

Creating a deployment project

Creating a deployment environment

See also

Getting started

Managing your user profile

Administering Bamboo

Supported platforms

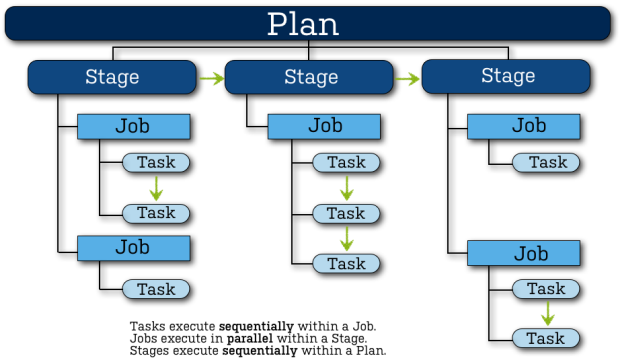
Bamboo Release Notes

Installing and upgrading

Integrating Bamboo with other applications

Configuring plans

A *plan* defines everything about your continuous integration build process in Bamboo.



A plan:

- Has a single stage, by default, but can be used to group jobs into multiple stages.
- Processes a series of one or more stages that are run **sequentially** using the **same** repository.
- Specifies the default repository.
- Specifies how the build is triggered, and the triggering dependencies between the plan and other plans in the project.
- Specifies notifications of build results.
- Specifies who has permission to view and configure the plan and its jobs.
- Provides for the definition of plan variables.

Every plan belongs to a project.

Projects and plans can only be configured by Bamboo administrators (see Creating a plan).

On this page:

Navigate to a plan's configuration

Go to **Build** > **All build plans** from the Bamboo header, then select the edit icon (✎) for the plan you want to edit.

The plan's configuration is found on several tabs.

Configure a plan

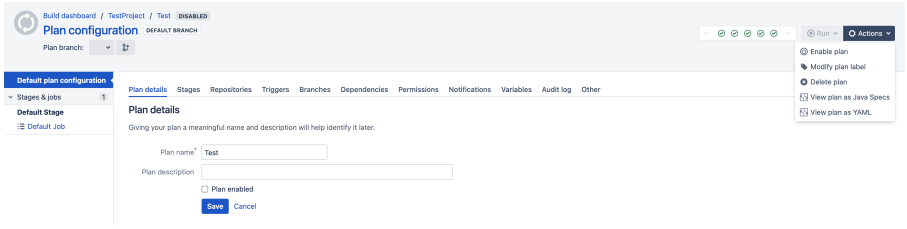
1. Navigate to the plan's configuration pages as described above.
2. Select a tab to configure that aspect of your plan:

Plan details	A plan's **Project Key** and **Plan Key** are not editable once the plan is created, however see Moving plans to a different project.
Stages	See Using stages in a plan.
Repositories	See Linking to source code repositories.
Triggers	See Triggering builds.
Branches	See Using plan branches.
Dependencies	See Setting up build dependencies.

Permissions	See Configuring a plan's permissions.
Notifications	See Configuring notifications.
Variables	See Defining plan variables.
Miscellaneous	See Configuring expiry of a plan's build results.
Audit log	A record of changes to the plan's configuration. This feature is disabled by default. To enable it, select **Bamboo administration** > **System** > **Audit log**.

Exporting plan configuration to Bamboo Specs

Bamboo instance administrators can export the plan configuration to Bamboo Specs in **Plan configuration** > **Actions**:



Viewing a plan's build information

A *plan* defines everything about your continuous integration build process in Bamboo.

To view information about a plan:

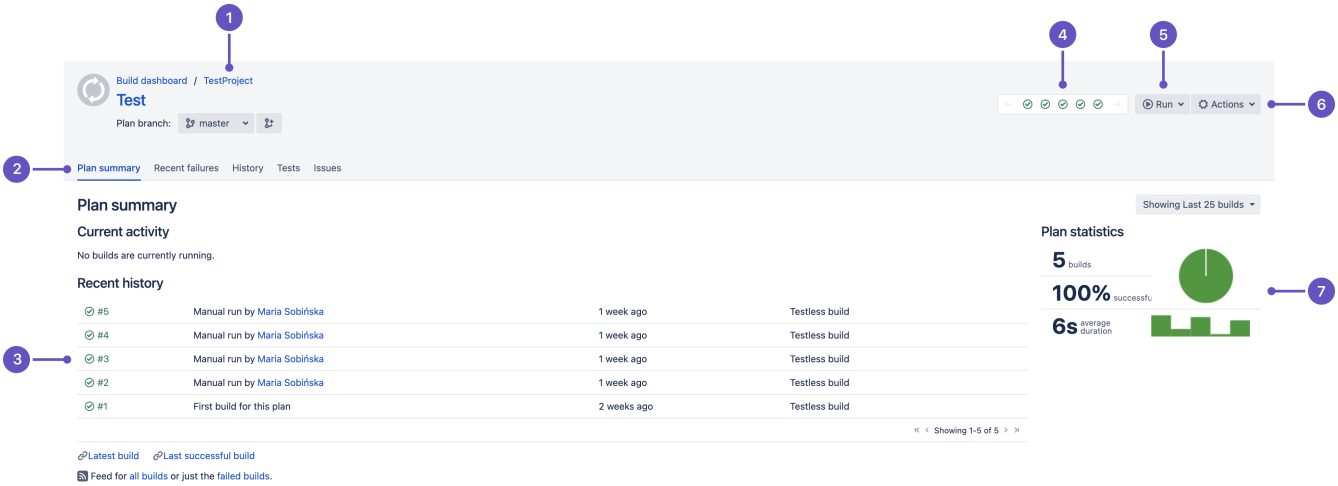
1. Navigate to the desired plan, as follows:
    - If you are viewing the Dashboard, locate and select the plan's name in the list, or
    - If you are viewing a job or build result, select the plan's name in the breadcrumb links at the top of the screen.
2. Select a tab to view information about the plan:

Related pages:

- Using the Bamboo dashboard
- Viewing a build result
- Configuring plans
- Configuring a plan's permissions

Tab	Notes
Plan summary	Information about the plan, as shown in the diagram below.
Recent failures	Information about recent failures of the plan, including the builds that failed, links to the build results, time taken to fix, etc.
History	The full history of builds of the plan.
Tests	A summary of the 10 most frequently broken tests.
Issues	View the Jira issues linked to builds of your plan. *(You will only see this if your administrator has integrated Bamboo with Jira.)*

Use the **Actions** menu to access functions for the plan, such as **Disable plan** and **Configure plan**. *(This menu is only displayed if you are an administrator for the plan.)*



1. **Navigation:** Select the project to view a summary of its plans.
2. **Tabs:** View further details for the plan.
3. **Current activity and recent history:** See whether the plan is building. Shows results for the ten most recent plan builds. Select a build number to view that build result.
4. **Plan status:** Icons show if the plan is building and the status of recent builds. Select an icon to see the build result.

5. **Run menu:** Select options for running the plan, e.g., run a parameterized build.
6. **Actions menu:** E.g., Configure plan or Disable plan.
7. **Statistics and charts:** For the build history of the plan.

Creating a plan

A plan defines everything about your build process, including what gets built, how the build is triggered, and what jobs are executed.

This page describes how to:

- Create a new plan
- Clone an existing plan

Note, you need the **Create** or **Admin** global permission to create or clone a plan.

Create a new plan

1. In the top menu bar, select **Create** > **Create plan**.
2. Complete the build plan details on the Configure plan page.
3. Link repository to new build plan:
    - link previously used repository OR
    - a. Select your code repository host.
      b. Give your repository a display name.
      c. Provide repository URL.
      d. Specify access to the repository. You may select from:

      Allow all users to reuse the configuration of this repository
      All user access. This is the default access setting.

      Only you are allowed to reuse the configuration of this repository
      Limit access to just yourself.

      For more information about source code repositories in Bamboo, see Linking to source code repositories.

4. Select **Configure plan** and you are done.
   You can now configure the tasks and jobs required by your build plan.

Clone an existing plan

When you clone an existing plan, you make a copy of that plan and its entire configuration, with the exception of any branches:

1. Select **Create** > **Clone plan** in the top menu bar.
2. Use **Plan to clone** to select a plan. Only plans for which you have the Clone and/or Admin plan permission are shown.
3. Select an existing project for the plan, or create a new project.
4. Enter details for the new plan.
5. Select whether to enable this plan. Enabling the plan instructs Bamboo to start running builds of the plan, based on the plan's trigger configuration.
6. When you select **Create**, the Plan summary page for the new plan will be displayed. Bamboo will automatically run an initial build for your new plan.

Using plan branches

Plan branches are used to represent a branch in your version control repository, with the plan branch using the same build configuration as your plan.

Tools such as Git and Mercurial encourage a practice called feature branching, where a developer can use a new branch to work in isolation from his or her team members before merging their changes back into main line development.

With plan branches in Bamboo:

- Any new branch created in the repository can be automatically built and tested using the same build configuration as that of the parent plan.
- Any branches deleted from the repository can be deleted automatically from Bamboo according to the settings.
- You have the flexibility to individually configure branch plans, by overriding the parent plan, if required.
- Optionally, changes from the feature branch can be automatically merged back to the master (e.g. trunk, default, or mainline branch) when the build succeeds.

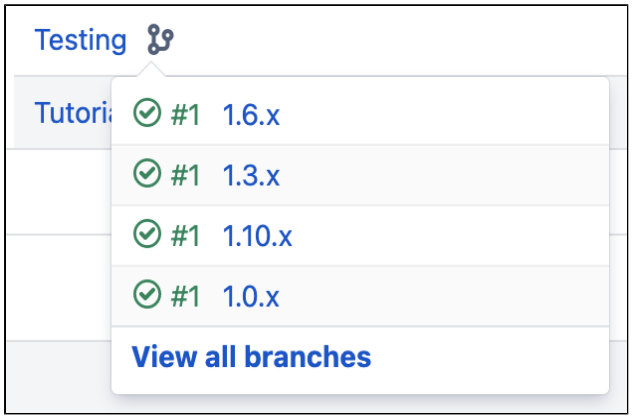
Further reading:

- Atlassian Git Tutorial
- Feature branches explained

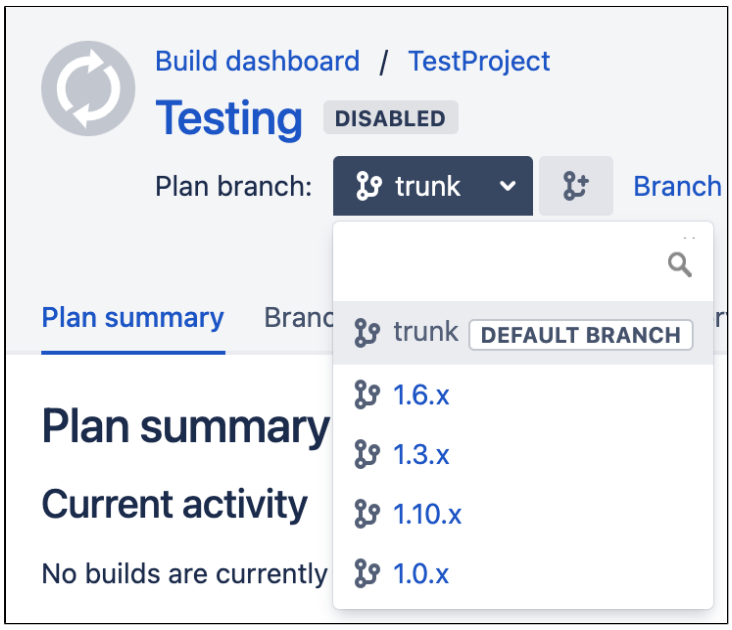
1. Viewing plan branches

Use the Branch status page for quick access to plan branch information.

You can access the list of all branches in a plan from different places. For example, you can select the **Branch** icon next to the plan name in the **Build dashboard** view:



You can also access the branch list from the **Plan summary** view:



2. General branches configuration

You can create plan branches manually or automatically. The branch configuration can be provided on the plan level and customized on the branch level. The settings provided in the branch configuration override the settings provided for the plan.

Automatic branch management

Plan branches can be created and deleted automatically based on the updates in the primary source repository. Automatic branch management is available for Git, Mercurial, and Subversion. For other repository types, you can use manual branching. You can override the default settings for a branch, such as values of the variables.

You can override the branch deletion settings in the branch details configuration view.

You can specify how often Bamboo checks the primary source repository for new or deleted branches in the general branch settings.

To hand over the branch management to Bamboo:

1. In the Plan summary view, select **Actions** > **Configure plan**.
2. Select the **Branches** tab.
    a. Configure the following:

Primary source repository branches	
Manually	Bamboo doesn't create new plan branches automatically. You can create branches manually.

When a pull request is created	Bamboo creates a plan branch automatically when a new pull request is created. If a pull request is merged or declined, Bamboo will disable this plan branch. You can select the **Forked repositories are allowed** option to enable Bamboo to detect pull requests originating from forked repositories. If forked repositories are not allowed, the source repository must match the target repository. This option is available for environments that support pull requests. Currently, these are Bitbucket Server, Bitbucket Cloud, and GitHub. If you're using Bamboo with Bitbucket Server, detecting pull requests from forks requires Bitbucket Server 7.20 or later and that you're connected to Bitbucket Server's primary mirror (secondary mirrors are not supported). Bitbucket Cloud public repositories are supported by default. To use this feature with a private repository, ensure that the owner of the target repository has access to its forks. To increase build safety, Bamboo will not create new plan branches from divergent branches in a forked repository. See Working with branch divergence. The build statuses of PRs from forked repositories won't appear in Bitbucket Data Center and Server, Bitbucket Cloud, or Github. To see the build results, go directly to Bamboo.
When a new branch in the repository is created	Bamboo creates a plan branch for each new branch detected in the primary source repository.
When a new branch in the repository is created and matches the expression	Bamboo creates a plan branch for each new branch detected in the primary source repository that matches the regular expression that you provided.

b. Configure plan branch clean up:

Primary source repository branches	Description
After a branch was deleted from the repository	When a branch is deleted from the repository, Bamboo will wait for provided number of days before deleting the plan branch.
After branch inactivity in repository	When a branch is inactive for provided number of days, Bamboo will delete the plan branch. If a branch in the primary source repository is inactive, Bamboo does not automatically delete the corresponding plan branch.

If you selected **Clean up plan branch automatically** in the configuration on the branch level, the branch is disabled and deleted according to the daily cleanup rules, regardless of the automatic branch management settings. **Clean up plan branch automatically** is selected by default for manually created plan branches.

3. Select **Save** to apply the changes.

Global settings - branch detection interval

Once automatic plan branch management is enabled, Bamboo checks for new or deleted branches in the primary source code repository.

You can specify how often Bamboo checks for new branches in the primary source repository in the system settings. The default value is 300 seconds.

To configure the branch detection interval:

1. Select > **System** > **General configuration**.
2. In **Global system configurations**, set the branch detection interval. Provide the value in seconds, the default value is 300.

Manual branch management

Use manual branching for all supported repository types. You may want to consider using automatic branch management for Git, Mercurial, and Subversion repositories.

To manually create a branch of a plan:

1. In the Plan summary view, select **Actions** > **Configure plan**.
2. Select the **Branches** tab, then **Create plan branch**.
3. In the **Create plan branch** view, you can create branches in one of the following ways:

Action	Description
Select from available VCS branches	Select one or more branches from the list of available VCS branches. At the bottom of the list of the branches you can select the **Enable branches** check box, which makes all selected branches available for building and change detection. Select **Create plan branch** manually to go to manual branch creation screen.
Create plan branch manually	Provide: • a display name (required) - overrides the VCS branch name • a branch description - a meaningful description of the branch • VCS branch name - the name of the branch in the VCS repository You can select the **Enable branches** check box, which makes the new branch available for building and change detection. Select **Auto-detect VCS** branches to go back to the list of available VCS branches.

4. Select **Create**.

Automatic branch merging

Bamboo provides two merging models if you choose to automate your branch merging:

- *Branch Updater* — a branch repo is kept up-to-date with changes to master. Note that changes on your master branch do not trigger branch builds.
- *Gatekeeper* — the default repo is only updated with changes in the branch that have built successfully.

Any updates are performed only if the merged branches have built successfully.

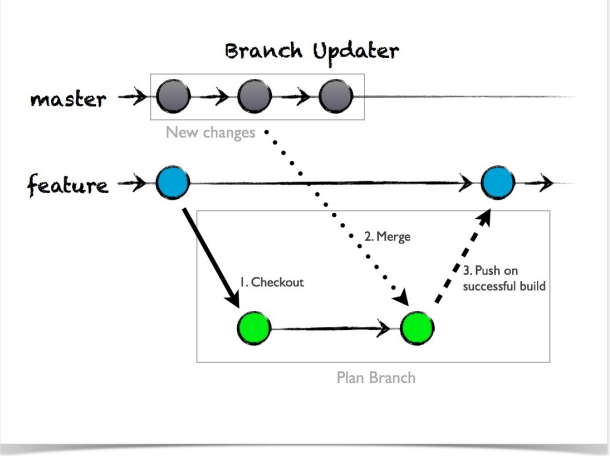
The automatic branch merge strategy for the master plan can be overridden in an individual plan branch, if required. Automatic branch merging is not available for Subversion.

Branch updater

When to use

The Branch Updater should be used when you want to:

- Automatically merge changes from the team's master branch into your feature branch, after a successful build of the master and branch merge.
- Get notified when the changes on your feature branch are no longer compatible with the team's master branch.

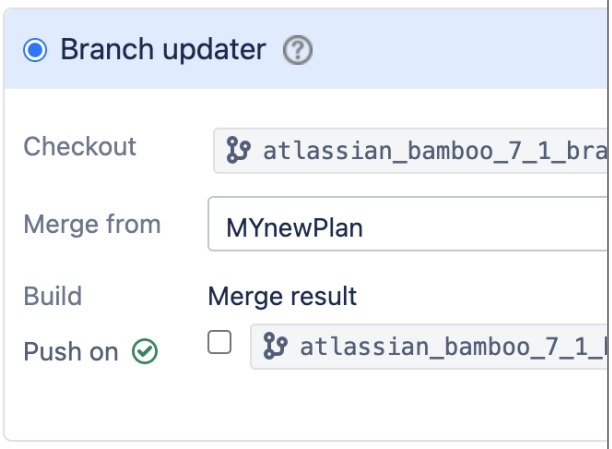


Change detection is available only for the branch you're currently working on.

Configuring

To have recent changes in another repo merged into your branch repo:

1. Go to the **Branch details** tab of the branch plan's configuration pages.
   (Select the branch icon beside a plan name on the **All build plans** tab, then select the icon.)
2. Under Merging select **Branch merging enabled**, then **Branch updater**.
3. Use the **Merge from** list to select the repo from which changes should be merged with your feature branch.
4. Select **Push on** only if you want those changes merged back into your branch once the build completes successfully.
5. Select **Save**.

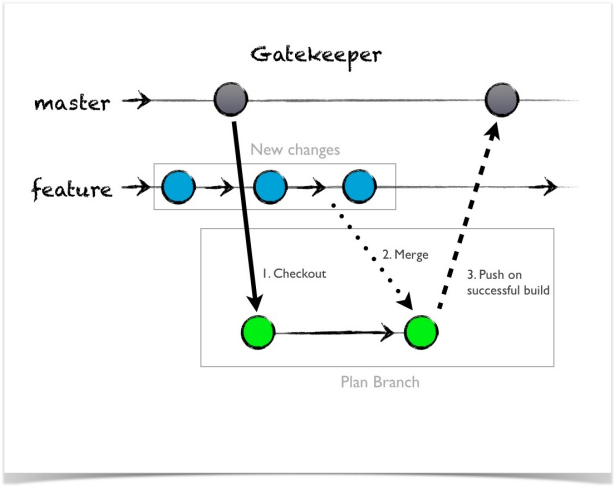


Gatekeeper

When to use

The Gatekeeper should be used when you want to:

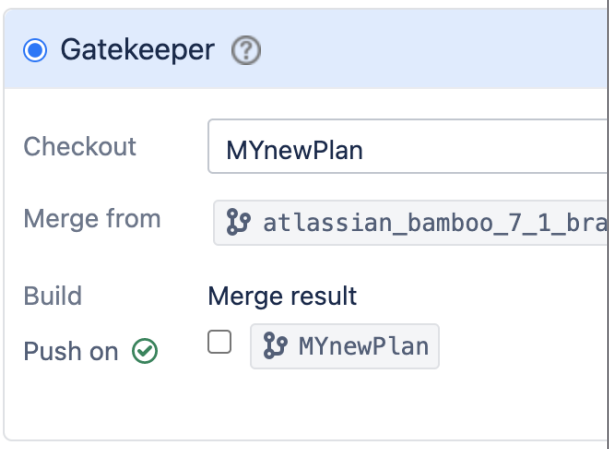
- Automatically merge your feature branch back into the team's master branch, after a successful build of the merged changes from both branches.
- Get notified when a build of combined changes from both branches fails, preventing the feature branch from being merged back into the team's master branch.



Configuring

To have your successfully built changes pushed to another repo:

1. Go to the **Branch details** tab of the branch plan's configuration pages.
   (Select the branch icon beside a plan name on the **All build plans** tab, then select the icon.)
2. Under Merging select **Branch merging enabled**, then **Gatekeeper**.
3. Use the **Checkout** list to select the repo with which to merge your changes (and to which changes should be pushed).
4. Select **Push on** only if you want your changes pushed to the other repo once the build completes successfully,
5. Select **Save**.



Integrating branches with Jira applications

Check **Create Remote Links from Jira Issues** to have the plan branch automatically linked, using an issue key in the branch name.

When a developer begins working on a feature described in a Jira application issue, they use Git or Mercurial to branch the repository. If they use the issue key as part of the VCS branch name, Bamboo will detect the issue key and automatically link the new branch to the issue:

- The Jira application issue key needs to be in the name of the branch – 'jb-BDEV-790' and ' BDEV-769 1 ' are valid forms.
- The link shows up right under the breadcrumb on the Build Result Summary for the plan branch, and on the issue too.

To use Jira applications Feature Branching, Bamboo needs an application link to the Jira application server.

Branch notifications

You can get build notifications from branch plans just as you do for master plans.

To specify how notifications are sent by all branches created from a plan, go to the **Branches** tab for the plan's configuration and choose one of the following options:

- Notify committers and people who have favorited this branch.
- Use the plan's notification settings.
- Notifications should not be sent for this branch.

You can override how notifications are sent from a particular branch plan, if necessary, by going to the **Notifications** tab on the Plan branch configuration.

See Configuring notifications for a plan and its jobs for information about plan notifications.

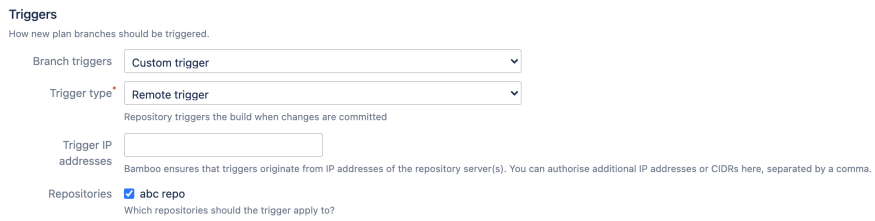
Branch triggers

You can configure how new plan branches should be triggered.

To specify how branches created from a plan are triggered, go to the **Branches** tab for the plan's configuration and choose one of the following options:

- Same as defined in parent plan
- None. Run new plan branches manually
- Custom trigger

After choosing a *Custom trigger* you can pick any trigger type, that is normally available for the plan.



You can overrider branch trigger for a particular branch by going to **Branch details** tab on the Plan branch configuration.

Note that you can only configure one trigger for a plan branch, and that this overrides all triggers that may be configured for the master plan.

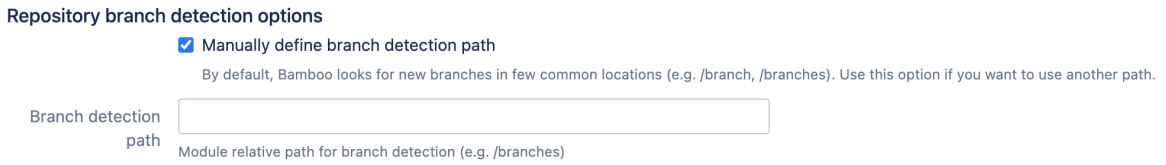
See Triggering builds for more information about plan triggers.

Subversion branches location

This section is displayed only for plans that use a Subversion source repository. Bamboo assumes that your Subversion repository structure follows the convention for branches, and automatically calculates the branch root URL.

For example, for the `fastBuild` repo with this URL: `https://svn.mycompany.com/svn/fastBuild/trunk`, Bamboo will expect that branches will be created at this location: `https://svn.mycompany.com/svn/fastBuild/branches`.

If your Subversion repository structure follows a different convention, you can specify where repository branches will be created by selecting **Manually define branch detection path**.



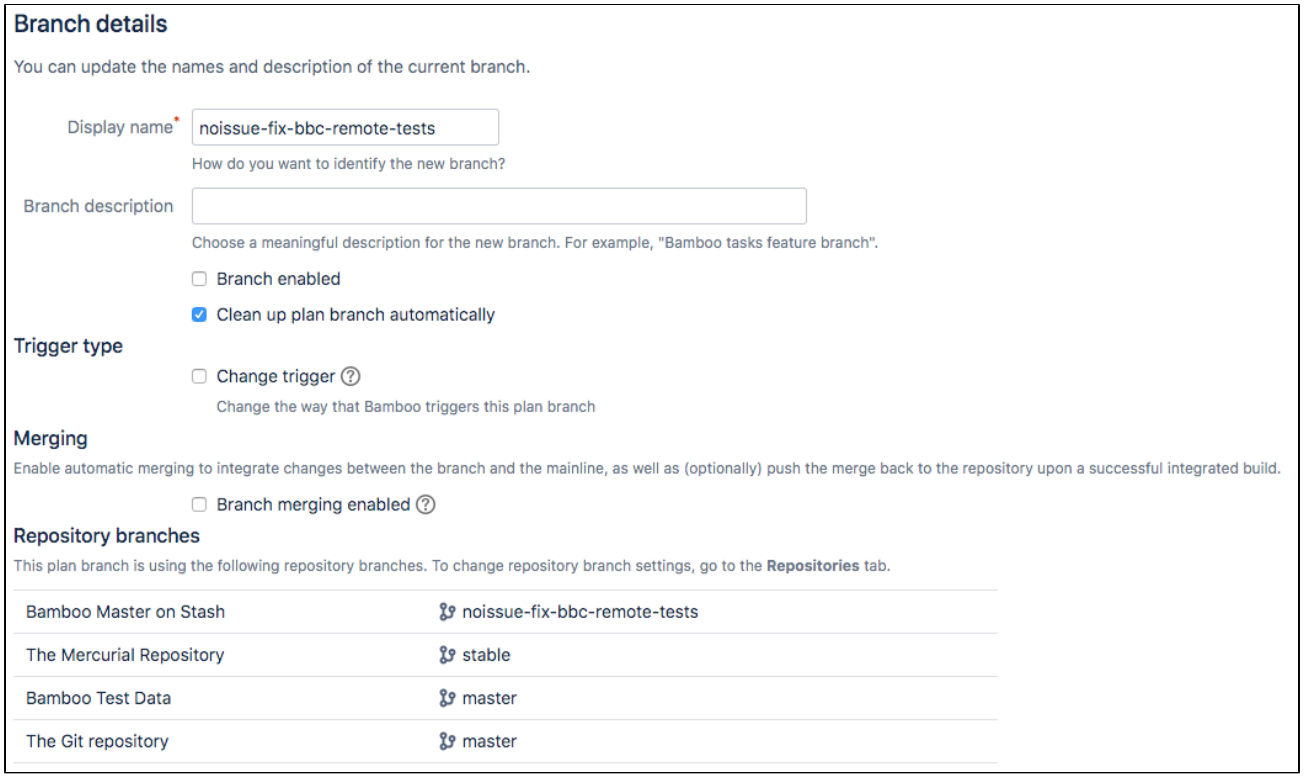
Branch dependencies

You can use build dependencies for plan branches in a similar way to that for plans: a branch plan is triggered only when another branch plan has been successfully built. This can be used to ensure that breaking source code changes associated with one branch plan are detected before they can break the build of a dependent branch plan. Dependencies between master plans are maintained if their branch plans have the same name. See Setting up plan build dependencies for further information about dependencies.

Select **Trigger dependencies for branches**, in the **Advanced options** section on the **Dependencies** tab for the plan configuration, if you want plan branches to honor the build dependencies of their respective master plans.

3. Configuring a plan branch in Bamboo

Branch details configuration



Branch clean-up

On the **Branch details** tab of the branch's configuration, you can specify that a plan branch is *not* cleaned up automatically.

By default, plan branches are deleted automatically after:

- 7 days after the branch was deleted in the primary source repository **OR**
- 10 days of branch inactivity in the primary source repository

The values can be specified on the plan level.

Trigger type

You can override the way that Bamboo triggers specific plan branch. You can choose any trigger type, that is normally available for the plan and any of the available trigger conditions:

- Only run build if other plans are currently passing
- Only build branches when there are changes
- Only run plan when no child plans are queued or in progress

Note that you can only configure one trigger for a plan branch, and that this overrides all triggers that may be configured for the master plan.

Merging

You can override automatic merging strategy for chosen plan branch. Available merging strategies are described above.

Branch repositories

Once the plan branch is created, automatically or manually, all repositories defined in the plan inherit settings from the master plan by default. The only exception is the default repository, which uses one of the following branches:

- branch selected by you during the manual creation of the plan,
- branch detected in VCS in case of automatically created plan.

You can change settings of all repositories defined in the plan branch in the **Repositories** tab. In this tab, simply put a desired branch name or enable the *Change repository settings for this branch plan* toggle if you need to modify more settings.

In the **Branch details** tab the *Repository branches* section gives you a quick overview of repositories and branches configured in the plan branch. The *Default repository settings changed* flag indicates that this repository overrides more settings from the parent plan. The *No branch support* flag means that a given repository type does not support branches.

Configuring plan repositories and branches

Changing the repositories configured in a plan affects all the existing branches and automatic branch detection. Keep in mind that:

- removing a repository from a plan removes it from all plan branches
- repositories added to the plan configuration are also added to all plan branches (the VCS branch configuration is inherited from the plan configuration unless you change it manually)
- the automatic branch detection and expiry feature creates and removes branches according to the configuration of a plan's default repository

If you remove the default repository from the plan, Bamboo will try to restore the VCS branch for all existing plan branches. If the VCS branch does not exist in the new repository, the plan branch is marked as invalid and won't be built. In this case, you should manually correct the configured branch in the **Repositories** tab.

If you need to replace the the default plan repository, we strongly recommend that you set the replacement repository as the new default first, and only then remove the old default repository.

Notifications

You can override sent notification for builds for specific branch on the **Notifications** tab of the branch's configuration. The options are:

- Notify committers and people who have favorited this branch
- Use the plan's notification settings
- Notifications should not be sent for this branch

See Configuring notifications for a plan and its jobs for information about plan notifications.

Variables

You can override values of plan and global variables at the **Variables** tab of the branch's configuration. See Defining plan variables.

Other

Limitations with plan branches

The following limitations apply to using automated plan branching and merging:

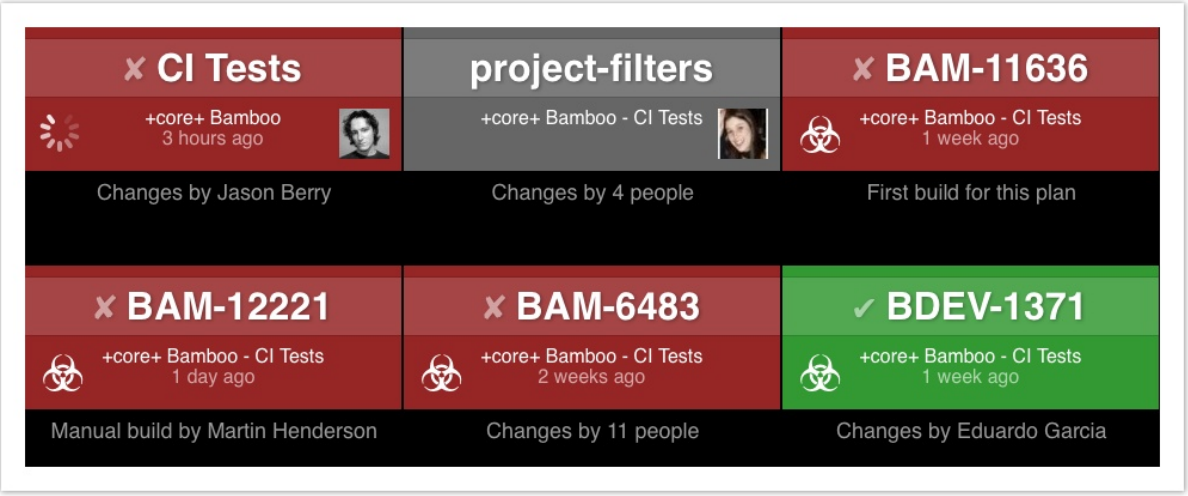
Action	Limitations
Auto plan branching	Can only be used with Git, Mercurial and Subversion repositories. For other repository types, use manual branching. Cannot be used with the Git implementation embedded in Bamboo. (You need to have set up native Git.)
Manual plan branching	Can be used for all repository types supported by Bamboo.
Auto branch merging	Can only be used with Git and Mercurial repositories. Can only be used with branches that were configured in Bamboo. Cannot be used with the Git implementation embedded in Bamboo. (You need to have set up native Git.)

Branches wallboard

The branches wallboard displays the status of all the branches and the plan that the branches belong to. The plan's own status always appears first. Plans shown as grey are disabled.

To display the branches wallboard:

1. Go to the Plan summary for the plan that has branches you want to display.
2. Select **Actions** > **Branch wallboard**.



Enhanced plan branch configuration

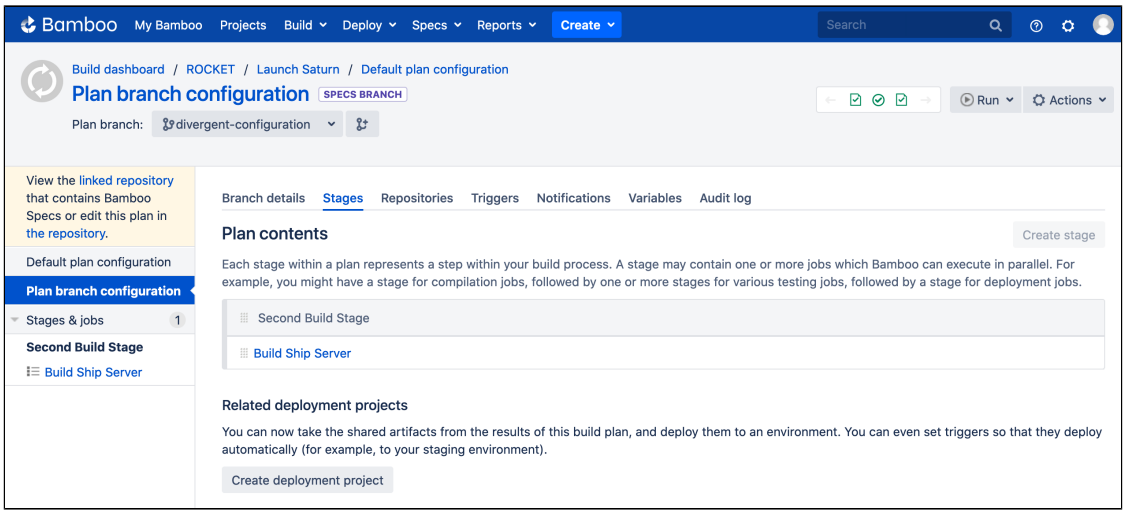
You can configure plan branches using Bamboo Specs. This way you create a custom plan brach configuration that differs from that on the default branch. We're referring to plan branches with custom configuration via Bamboo Specs as **Specs branches**.

- To use the enhanced plan branch configuration, your code must be stored in Bitbucket Server or Bitbucket Cloud, and you must be using Bamboo Specs for plan configuration. To use this feature with Bitbucket Cloud, set up the webhook and enable the **Enable webhooks** flag in the repository configuration. See Triggering a Bamboo build from Bitbucket Cloud using Webhooks.
- By default, all plan branches are configurable.
- Bamboo Specs are not applied to plan branches that were created before upgrading to Bamboo 7.0. If you want Bamboo to process such branches, you must enable it manually from the Branch details configuration.

Branches are commonly used in version control systems to develop features or bug fixes without affecting your default branch typically referred to as master. Before Bamboo 7.0 plan branches came with certain limitations — their configuration was inherited from the master build configuration and you could only modify it to a very limited extent. This would mean that you wouldn't be able to test your custom changes to your plan branch until you copied your changes to the master, or have a custom configuration for individual branches.

In Bamboo 7.0, we're enhancing plan branch configurations with Specs branches, allowing for your build configuration to live inside plan branches. Specs branches allow you to create your own feature configuration separate from master, where you can make your commits, Bamboo Specs changes, and tests without having to worry about modifying the master branch. This way you can create custom build plans and feature branch configurations that will be different from those in master.

Was does it mean for me in practice? It means that from now on, your plan branch configuration will have some options that previously (before Bamboo 7.0) were available for master branch configuration only for example Triggers, Stages, or Jobs.



Prerequisites for using enhanced plan branch configuration

Before you can use enhanced plan branch configuration capabilities, ensure the following:

- You are using Bamboo Specs to configure plan branches.
- Your code and Bamboo Specs are stored in the same Bitbucket Server or Bitbucket Cloud and at the same level as the `bamboo-specs` folder. See Integrating Bamboo with Bitbucket Data Center and Integrating Bamboo with Bitbucket Cloud.
- Your repository is configured as a linked repository of the Bitbucket type
- Plan branch detection is enabled (it's on by default). See Using plan branches.

Known limitations and changes to previous behavior of Bamboo

- Plan branch configuration is not available for deployment projects. Bamboo ignores any Bamboo Specs on deployment projects in plan branches.
- When using Specs branches you can't link any repositories additional to those on master branch. You can change the configuration of that repository but you can't add or remove it.
- To create a new plan on your Specs branch, you must first create it on master.
- Default settings from *Automatic branch detection* configuration, like triggers and notification settings, are ignored by Bamboo Specs branches.
- The default repository of a Specs branch is inherited from the master branch and it's not possible to select a different repository on your Specs branch.
- In regular feature branches you can only define one trigger and inherit other triggers or inherit triggers from master. When using Specs branches, you can define multiple triggers.

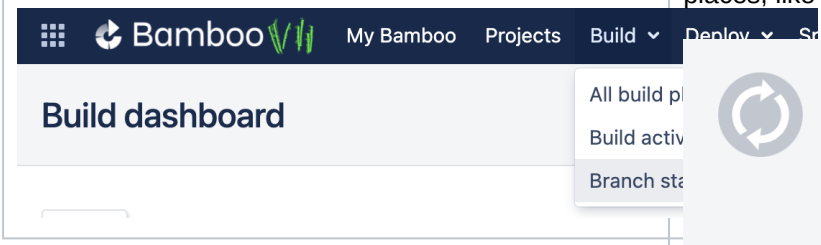
Using the branch status page

With Bamboo you can view and manage your plan branch assignments on a single page.

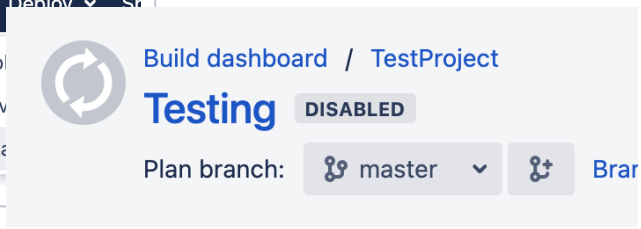
On this page

Getting to the branch status dashboard

You can find the page in the top menu, in **Build** > **Branch status**.



You can also find a link to the **Branch status** page in places, like screens with the details of builds or plans:

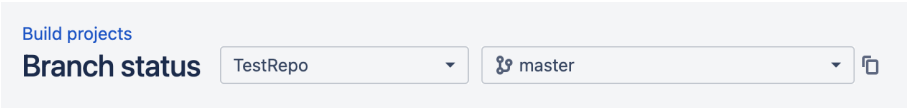


Using the Branch status dashboard

Here are the most important things that you need to know about the Branch status dashboard.

Picking a repository and a VCS branch

To start working with branch statuses, you must specify a repository and a VCS branch. You can select it from a list or start typing to search by name.



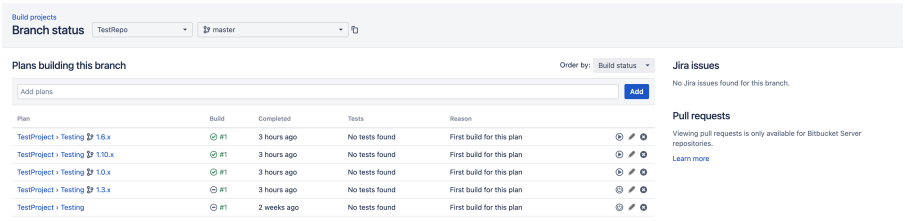
The dashboard works for all linked repositories that support branches:

- Git
- Mercurial
- SVN
- Bitbucket Cloud
- Bitbucket Server / Stash
- Github

We've added a small **Copy** button next to the selected branch name - it copies the name of the branch in case you need to share it.

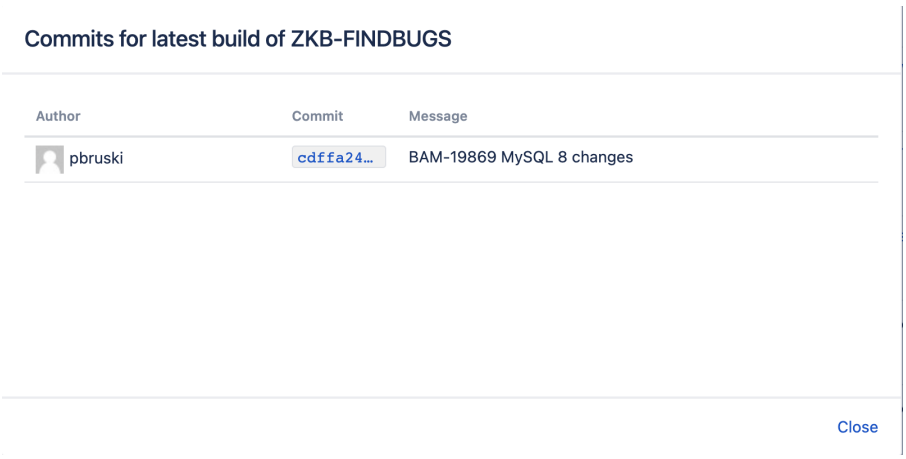
Viewing build plans assigned to a VCS branch

Once you select a repository and a VCS branch, you'll see a list of plans that build the branch:



If the information about the commits is available to Bamboo, you can see the number of commits next to the build status:

Select the number to display the details of the commits:

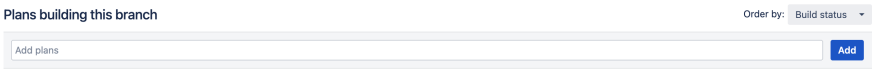


It might happen that there aren't any build plans assigned to the VCS branch that you selected. You can associate the branch with a plan directly from the **Branch status** page (see *Managing plan branches* below).

Managing plan branches

You can manage assignments between plan branches and VCS branches directly from the **Branch status** p age.

On top of the list of existing plans (or on top of the empty list if there aren't any plans yet), you can find the search field and the **Add** button. To assign a build plan to the selected VCS branch, select within the search field and either pick a build plan from a list or start typing to search:



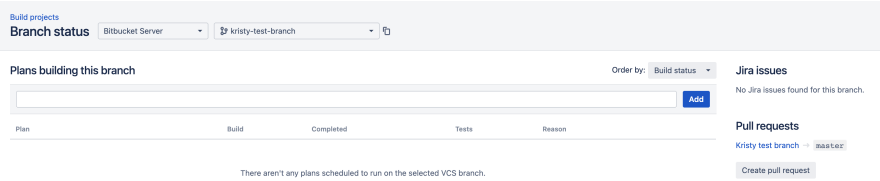
Select **Add** to save the assignment.

Managing existing build plans

Once you've specified some plans that will build your branch, you can see them in the list on the **Branch status** page. Next to each plan, there's a small **Actions** menu that allows you to run, edit, or remove build plans assigned to a VCS branch:

Viewing pull requests

Starting from Bamboo 6.0, the branch status page also displays open pull requests outgoing from the selected VCS branch. Pull request information includes pull request name, target branch, and whether it's in conflicted state.



Currently, only the Bitbucket Server repository type is supported.

Managing plans

A *plan* defines everything about your continuous integration build process in Bamboo. See Configuring plans for information about how to set up build plans.

You can also perform actions on one or more plans together, or make global settings that affect all plans on the Bamboo server.

See the following pages for information about managing your Bamboo plans:

- Configuring a plan's permissions
- Disabling or deleting a plan
- Modifying multiple plans in bulk
- Moving plans to a different project
- Configuring concurrent builds
- Configuring the hanging build event
- Configuring the build queue timeout event
- Build monitoring

Configuring a plan's permissions

This page describes how to change the permissions for a particular plan. For ongoing ease of management, we recommend that you grant permissions to groups rather than to individual users.

You need to have Admin permission on the plan to edit its permissions.

Note that a Bamboo Admin can also set global permissions for access to Bamboo.

To change plan permissions:

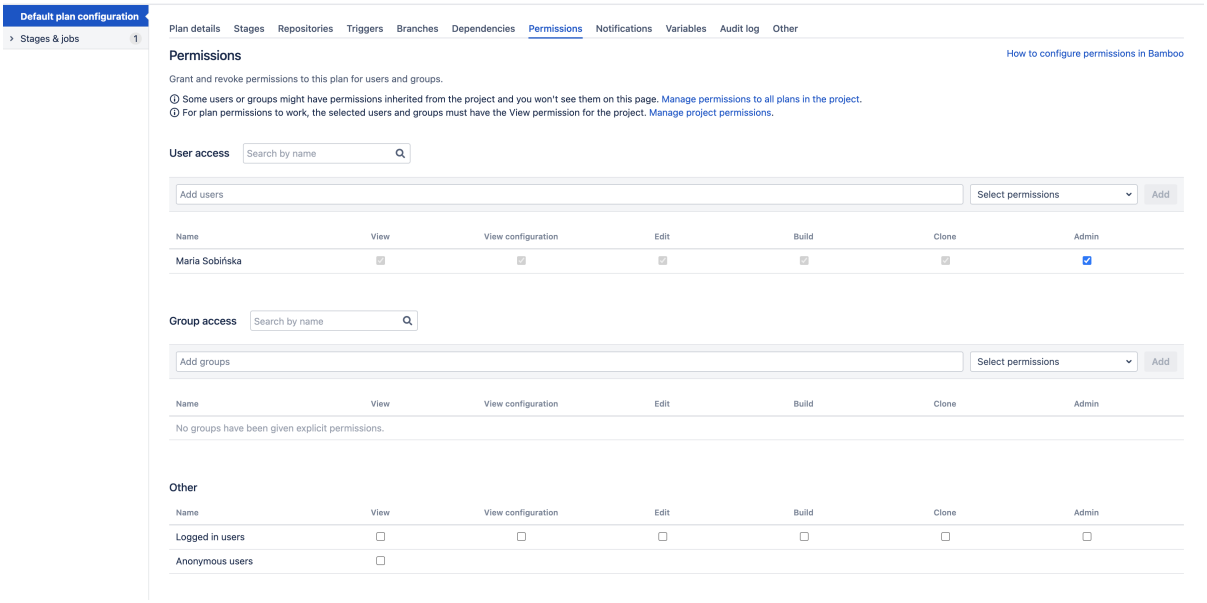
1. From the Bamboo header go to **Build** > **All build plans**, then select the name of the plan you want to edit.
2. Select **Actions** > **Configure plan**.
3. Select the **Permissions** tab.
4. Add users or groups for which you wish to set permissions. Select (or clear) the check box for each permission that you wish to change for a user or group. See the table below for details.
5. Select **Save**.

Related pages:

- Configuring plans
- Granting plan permissions in bulk
- Managing permissions

- Managing users
- Managing groups

Plan permission	Actions
View	• View the plan and its builds • Add a comment or label to a build result
View configuration	• View the configuration for a plan and its jobs. Available for Bamboo Data Center version only.
Edit	• Edit the configuration for a plan and its jobs (except for plan permissions and stages) • Delete a comment or label from a build result • Add and delete plan labels
Build	• Trigger a manual plan build • Pause and resume a plan build
Clone	• Clone the plan
Admin	• Edit the configuration for a plan and its jobs (including plan permissions and stages)



Disabling or deleting a plan

Bamboo allows you to disable or delete plans that you don't want to be built:

- **Disabling a plan** prevents it from being built. You can re-enable the plan, if you want to build it again.
  For example, if a plan's latest build is broken and cannot be fixed quickly, you may want to disable it temporarily to stop the plan from being built.
- **Deleting a plan** removes it completely from your Bamboo system. You will need to recreate a new plan from scratch, if you want to build it again.
  For example, if a plan is no longer relevant, you may want to delete it.

On this page:

Related pages:

- Configuring plans
- Disabling or deleting a job
- Stopping an active build
- Exporting data for backup

Disable a plan

1. Go to **Builds** > **All build plans** and select the plan's name.
2. Select **Actions** > **Disable plan**.

You can also disable the plan using the **Plan enabled** check box on the **Plan details** tab of a plan's configuration pages.

Note that disabling a plan doesn't disable its branch plans or builds which are already running.

Delete a plan

Deleting a plan deletes everything related to that plan, including the plan's configuration, all of the plan's job configurations and the plan's branch plans, job build results, artifacts, labels, and comments:

- ⚠️ Deleting a plan also deletes its branch plans. Be careful!
- The Admin global permission is required to delete a plan.
- A plan that is currently being built cannot be deleted. If you need to delete such a plan, stop the plan's build first. Refer to Stopping an active build for more information.
- Bamboo cleans up everything related to deleted plans every two minutes. You'll have to wait at least that long if you want to reuse the key from a deleted plan.
- If you need to keep a permanent record of the job build results for your plan, see Exporting data for backup.

There are two ways to delete a plan:

- From the dashboard:
  1. Go to **Builds** > **All build plans** and select the plan's name.
  2. Select **Actions** > **Configure plan**.
  3. Select **Actions** > **Delete plan**.
- In the Administration console:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Plans**, select **Remove plans**.
3. Choose the plan you wish to delete.
4. Select **Delete** at the bottom of the list. You will be prompted to confirm the deletion.

Modifying multiple plans in bulk

Bulk actions allow you to make changes to multiple plans at once. You need to be a Bamboo administrator to modify plans in bulk.

To use bulk actions:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Plans**, select **Bulk actions**.
3. Select the required bulk action and follow the on-screen instructions to complete the 5 steps.

The following bulk actions are available:

Add new notification
See Configuring notifications for a plan and its jobs for further details.

Disable Plan
See Disabling or deleting a plan for further details.

Enable Plan

Remove all notifications
See Configuring notifications for a plan and its jobs for further details.

Replace triggers
You can select a new trigger that will replace all existing triggers for the selected plans. This change affects all branches of the selected plans.

Run manual build
You have the option to disable dependencies when running the manual builds for the selected plans.

Update SVN credentials
See the Subversion documentation for further details.

Update SVN repository URL
See the Subversion documentation for further details.

Update web repository
See the Subversion or Perforce documentation for further details.

Moving plans to a different project

Moving a plan to a different project involves changing the plan's project key (as well as possibly the plan name and plan key), which will also change the build key for all of the plan's build results.

Moving a plan does not affect the plan's configuration, nor any comments or labels that have been applied to job build results within the plan.

You need to be a Bamboo administrator to move a plan.

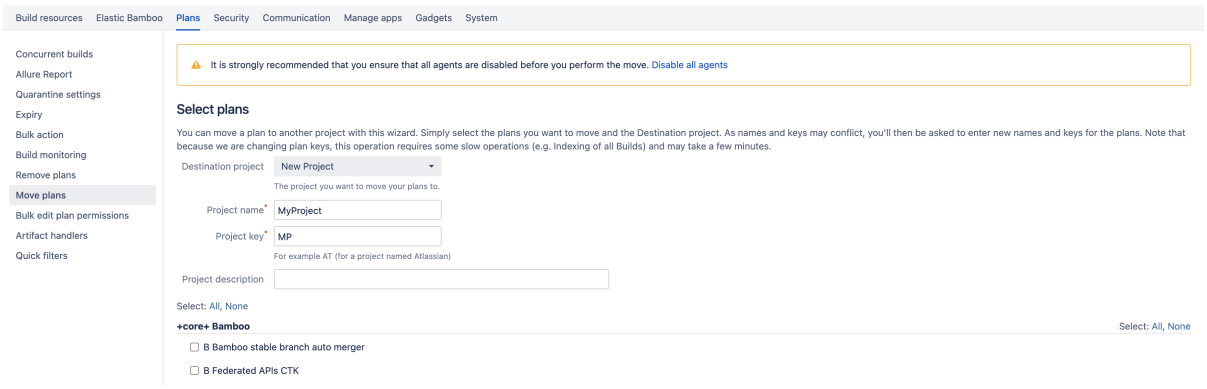
⚠️ Note that moving a plan will require Bamboo to re-index all its data, so your Bamboo system may run slowly for a few minutes.

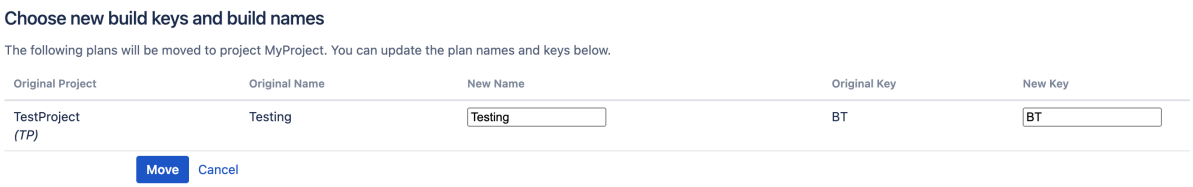
Before you begin:

- We recommended that you back up your Bamboo build results before you move a plan. See Exporting data for backup for instructions.

To move a plan to a different project:

1. From the Bamboo header select > **Plans**.
2. Select the **Move plans** tab.
3. Select either an existing project or **New Project** from the **Destination project** list. For a new project, enter a new **Project name** and a unique **Project key**.
4. Select one or more plans to move.
5. Select **Move** to display the Choose new build keys and build names page.
6. Edit the new name and new key for each plan, if necessary. You may need to do this if the destination project already has a plan with the same plan name or key, or if you wish to change these.
7. Select **Move**.





Configuring concurrent builds

Bamboo's concurrent builds feature allows you to build a plan concurrently on several agents. You might find this useful if a plan is likely to be triggered again before the current build completes.

You can configure a default value for the maximum number of builds of a plan that your Bamboo server can run concurrently, using the Bamboo administration console. This value is a default – it can be overridden on the **Oth er** tab of a plan's configuration.

You need to be a Bamboo administrator to configure concurrent builds.

To configure the number of concurrent builds of a plan allowed by Bamboo:

1. From the Bamboo header select > **Overview** > **Plans**.
2. Select the **Concurrent builds**, then select **Enable**.
3. Select **Edit**.
4. Edit the value for **Default number of concurrent builds allowed**.
5. Select **Save**.

Configuring the hanging build event

The hanging build event is thrown when Bamboo determines that a build has become unresponsive according to two criteria:

- Expected Build Time — defined as Build Time Multiplier x Average Build Time.
  - Build Time Multiplier is a user-defined setting.
  - Average Build Time is calculated by Bamboo using an average of previous build times (in minutes).
- Log Quiet Time — the length of time (in minutes) between log entries for a build.

The Expected Build Time and Log Quiet Time must *both* be exceeded for Bamboo to throw a hanging build event.

This event is currently used by Bamboo to send notifications.

You can also disable build monitoring altogether so that the hanging build event never occurs.

On this page:

Related pages:

- Configuring notifications for a plan and its jobs
- Configuring the build queue timeout event
- Build monitoring
- Configuring tasks

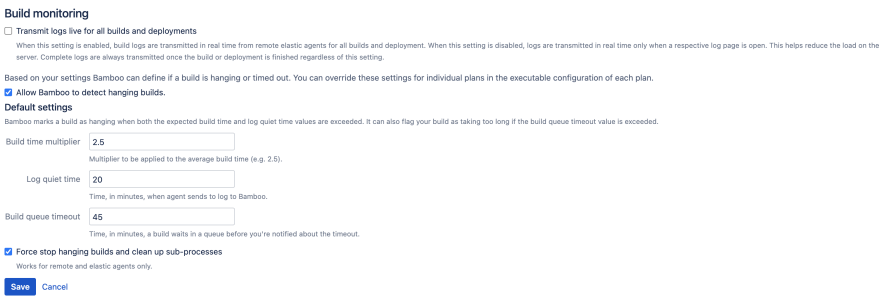
Configure the hanging build event

You can change the criteria governing when a hanging build event is thrown.

Note, the hanging build criteria can be also be set for a specific job, when specifying a job's builder. Job-level criteria will override the global criteria described on this page (including disabling this event).

To edit the hanging build event settings:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Plans**, select **Build monitoring**.
3. Select **Edit** and update the values for **Build time multiplier** and **Log quiet time** as required.
4. Select **Save**.



The check interval for hung builds

By default, Bamboo will check whether a hanging build event has been thrown every 60 seconds.

You can change this check interval by configuring the system property, `bamboo.buildHangingMonitor.checkInterval`. (This property is specified in seconds.)

Please read Starting Bamboo for instructions on how to configure the `bamboo.buildHangingMonitor.checkInterval` system property.

Configuring the build queue timeout event

The build queue timeout event is thrown when a build has been waiting in the build queue for longer than a specified period of time.

This event is currently used by Bamboo to send notifications.

Configuring the build queue timeout event

You can change the criteria governing when the build queue timeout event is thrown. You can also disable build monitoring altogether so that the build queue timeout event never occurs.

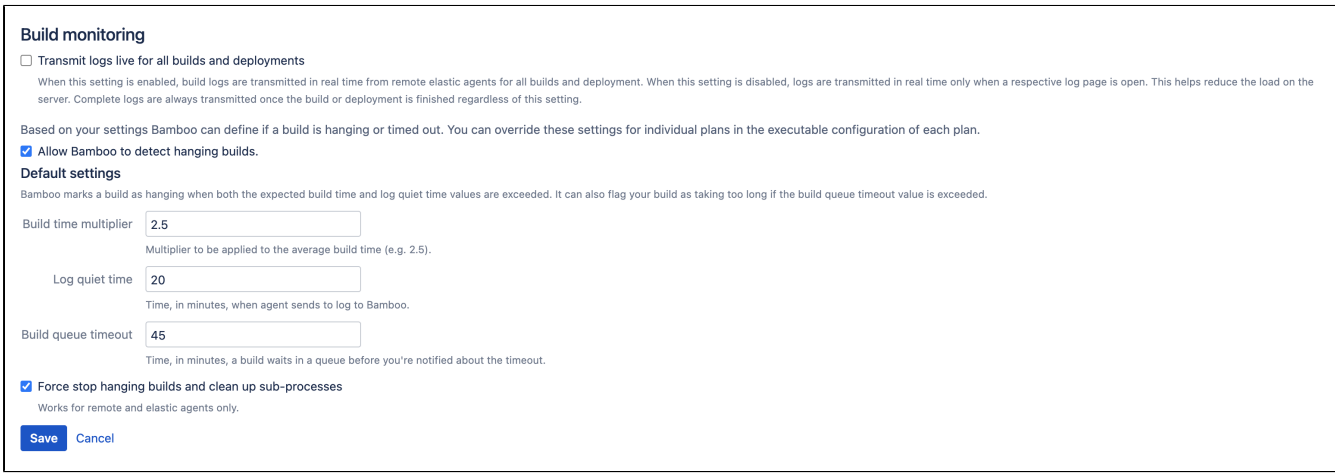
On this page:

Related pages:

- Configuring notifications for a plan and its jobs
- Build monitoring

To edit the build queue timeout event settings:

1. From the Bamboo header select > **Plans**.
2. Select the **Build monitoring** tab.
3. Select **Edit** and update the value for **Build queue timeout** as required.
4. Select **Save**.



Disabling the build queue timeout event

You can disable the build queue timeout event by disabling build monitoring for your Bamboo installation. See Build monitoring.

Please note, you cannot disable the build queue timeout event without disabling all build monitoring features for your Bamboo installation.

The check interval for build queue timeouts

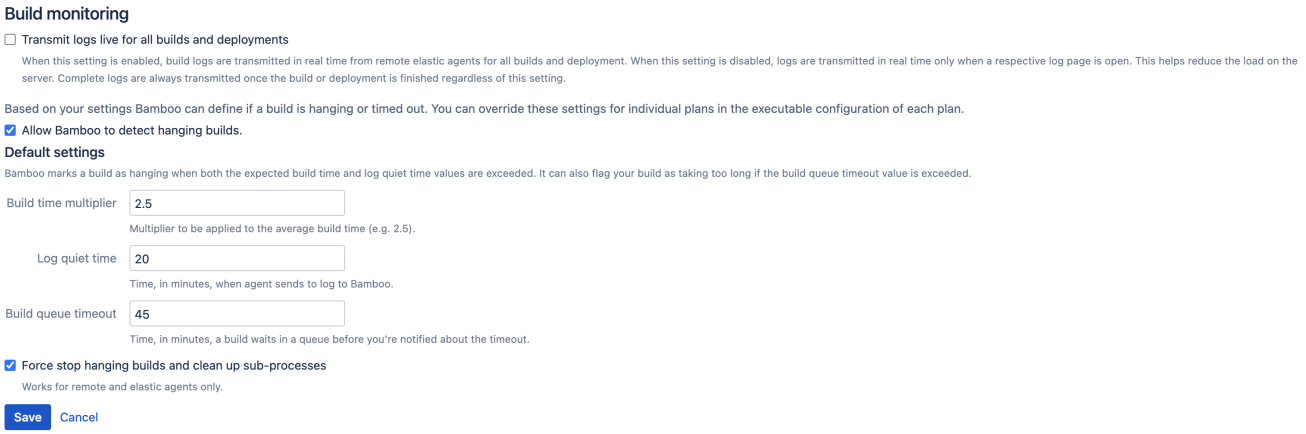
By default, Bamboo will check whether a build queue timeout event has been thrown every 60 seconds.

You can change this by configuring the system property, `bamboo.buildQueueMonitor.checkInterval`. (This property is specified in seconds.)

Please read Starting Bamboo for instructions on how to configure the `bamboo.buildQueueMonitor.checkInterval` system property.

Build monitoring

Based on your settings, Bamboo can determine if a build is hanging or timed out. You can override these settings for individual plans in the executable configuration of each plan. Build monitoring is enabled by default.



Change build monitoring settings

To configure build monitoring:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. On the **Bamboo administration** page, under **Plans**, select **Build monitoring**

> **Force-stop hanging builds**
>
> This option allows you to automatically stop any builds which, based on the criteria you set, are considered hanging. This option works only on remote agents.
>
> Force-stop builds configuration applies globally. To overwrite its functionalities on lower levels, you must go to the plan-level configuration

Disable build monitoring

To disable build monitoring:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. On the **Bamboo administration** page, under **Plans**, select **Build monitoring**.
3. Uncheck **Allow Bamboo to detect hanging builds**.

Artifact handlers

Artifact handlers allow Bamboo administrators to control where the artifacts produced by plans are stored. This can help to optimize the utilization of network bandwidth and file system space. You can activate each handler for shared and non-shared artifacts separately. Additionally, you can store artifacts in multiple locations. The default artifact handler selection is configured by a Bamboo admin but can be overridden in a plan's configuration by users that have administration permission on the plan.

To configure artifact handlers, select > **Plans** > **Artifact handlers**.

Types of artifact handlers used by Bamboo

Agent-local artifact handler

This handler stores the artifact on Bamboo's remote agent's file system. This handler does not publish artifacts to the server (in other words, the artifacts will not be downloadable from result pages if remote or S3 handlers are not enabled). It can be used to save bandwidth when exchanging artifacts between builds & deployments running on the same agent or running on different agents that share a common file system. In the configuration, you need to define the root directory in which all the artifacts will be saved.

Bamboo server artifact handler

This handler makes artifacts accessible on Bamboo remote and elastic agents. It also allows remote agents to publish artifacts they produce when running builds. The artifacts are stored on the Bamboo server, in the shared home directory.

You can define the maximum number of files per artifact. If the threshold is exceeded, the artifact is automatically compressed into a single `.zip` file, reducing the number of requests to the file system, which is particularly beneficial if the home directory is located on a remote file system (for example, NFS).

Amazon S3

Artifacts are stored in Amazon Simple Storage Service (S3) and are downloadable from there. Amazon S3 offers unlimited flexible storage capacity. For more information about S3, see the Amazon S3 product page.

You can use the AWS credentials provided in the Elastic Bamboo configuration or you can configure a separate account. In either case, you need to provide a bucket name. If you use the same S3 bucket for other purposes, you can provide a root path for all artifacts.

> Because the Amazon S3 artifact handler doesn't support IAM role credentials, the **Use the same AWS credentials as Elastic Bamboo (EC2)** option is not available in the Amazon S3 artifact handler if Elastic Bamboo has been configured with an instance profile.

You can also define the maximum number of files per artifact. If the threshold is exceeded, the artifact is automatically compressed into a single `.zip` file, reducing the number of requests to S3 when uploading and downloading the artifacts and improving the efficiency of the whole process.

SFTP artifact handler

Artifacts are stored on a dedicated SSH server and transferred to and from it through SFTP. This allows to move the load and disk usage away from the Bamboo server and improves its performance. To determine if your SSH server supports SFTP and how to enable it, see the documentation of your particular SSH server implementation. If you're using OpenSSH, SFTP is enabled by default, and no action is required.

You can define the maximum number of files per artifact. If the threshold is exceeded, the artifact is automatically compressed into a single `.zip` file, reducing the number of requests to the SSH server.

Impact of configuration changes on artifact visibility

If you change the configuration of one of the available global default artifact handlers or one configured for a particular plan, Bamboo won't automatically synchronize existing artifacts between the different locations. Your artifacts will remain intact in their original location, but they may become inaccessible through Bamboo's interface depending on the type of configuration change you make.

Changing the artifact handler type

Changing the type of an artifact handler (for example, from the Bamboo server artifact handler to the SFTP artifact handler) doesn't impact the visibility of old artifacts as long as the configuration details of the handler remain the same.

Changing storage location settings

Changing a configured artifact handler's location settings (for example, by replacing the SFTP server URL or specifying a different Amazon S3 bucket) will make the old artifacts inaccessible through the Bamboo interface.

Related content

Bamboo Artifact Handlers - Use Case Scenarios

Configuring the SFTP artifact handler

The SFTP artifact handler can store artifacts on a remote POSIX-compliant (Linux, Unix) SSH server, which helps reduce the load and disk usage on the Bamboo server and improve its performance. This works by allowing Bamboo agents to communicate with the SSH server directly, completely bypassing the Bamboo server during artifact publishing and retrieval.

Artifacts saved to a remote SSH server by the SFTP artifact handler are accessible directly on the SSH server or through the Bamboo interface. In the latter case, the Bamboo server acts as a proxy between the client browser and the SSH server.

> Make sure that your SSH server supports the SFTP protocol extension. If you're using OpenSSH, SFTP is enabled by default, and no action is required.
>
> Changing artifact handler configuration details requires Administrator permissions.

To enable and configure the SFTP artifact handler:

1. Select **Administration** > **Plans** > **Artifact handlers**
2. Enable the SFTP artifact handler for shared and non-shared artifacts by selecting the appropriate checkbox.
3. Scroll down to the **Handler-specific configuration** section and configure the SFTP artifact handler.

   The following table describes the available SFTP artifact handler settings:

Setting	Description
Authentication type	The preferred method of authenticating with the SSH server: • **Username and password** — use shared credentials or provide a username and password. • **SSH private key** — use shared credentials or upload an SSH key and provide a passphrase.
Username	The username on the remote host to use
Host	Host name or IP address of the SSH server.
Remote path	The path on the remote file system where Bamboo will store artifacts.
Verify remote host fingerprint on connect	Enables strict host fingerprint verification when establishing an SSH connection.
Host fingerprint	The host fingerprint to use if strict host verification is on.
Port	The port number used by the SSH server. The default value is 22.
Package threshold	The maximum number of files per artifact. If the number of files in the artifact exceeds the package threshold, the artifact is automatically compressed into a ZIP archive to reduce resource overhead during upload and download.

4. Optionally, de-select the handlers you want to stop using.
5. Select **Save**.
6. If you want to test the connection to the SSH server, select **Test connection**.
   Bamboo will now attempt to create and immediately remove a file on the SSH server to verify that it's accessible and that file system permissions are set correctly.

Docker Runner

Docker Runner is a Bamboo feature that allows you to run builds and deployments in a Docker container. This isolates the build process from the rest of the environment it runs in. This increases the security of your environments by providing more strict control over resources the continuous integration (CI) process has access to. The isolation also helps with the reliability of your CI by making sure that environment it runs in can be reliably recreated each time you run your builds. This increases the reliability of your builds by minimizing the influence of external applications on the build environment and allowing a finer control of resources dedicated to the build. Moreover it allows to easily reproduce the environment at a later time and on a different environment.

Docker Runner operates at the level of Bamboo jobs. A job is the unit of distribution of work per Bamboo build agents and it is itself made of a set of tasks that run sequentially. When a job is distributed with Docker Runner to a remote / local agent a Docker container is created on the build agent that picks up the job. Then the job runs all the tasks that it is comprised of in a sequential manner inside the container and finally Bamboo copies the build results of the job and cleans up. Bamboo transparently creates, manages and cleans up the Docker container when the build has finished but there are some configuration settings that can influence this process which are explained below.

If you want to user Docker images in your Bamboo builds at a different level than jobs, you can do that with Docker tasks. See Configuring the Docker task in Bamboo.

Before you begin

- Make sure you have Docker installed. We advise to use the most recent version. Bamboo provides support for Docker for Mac, and Docker for Ubuntu in version 17 and later.
- Define a Docker capability in Bamboo. See Defining a new Docker capability

Enabling Docker Runner

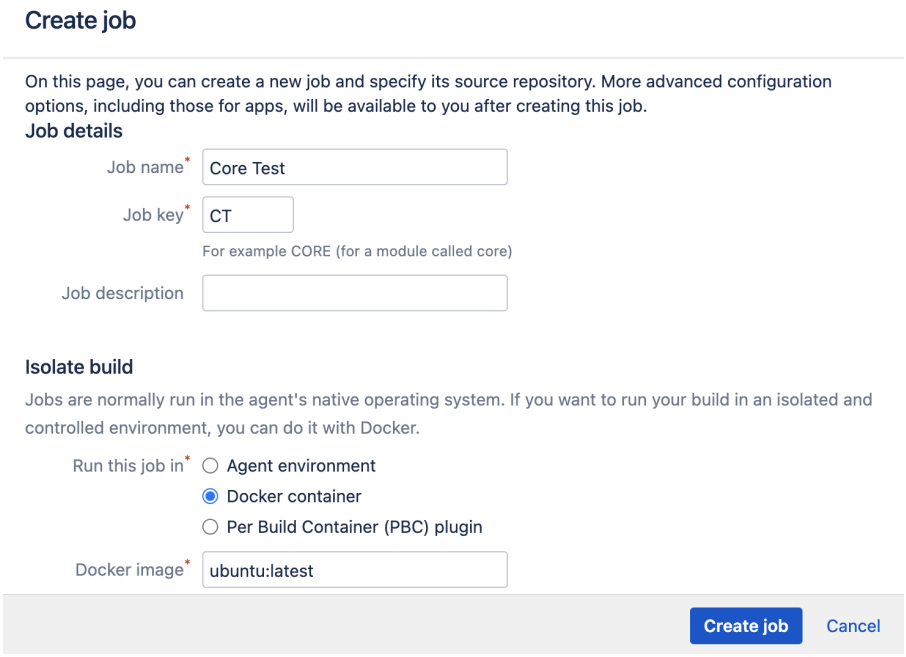
Runner can be enabled for jobs and deployment environments when:

- creating or editing jobs
- creating or editing deployment environments

Enabling Docker for a job or a deployment environment disables their standard requirements which come from tasks and replaces them with a single requirement for Docker.

Custom requirements, manually configured by users, will not be erased, and will still be used for agent matching.

To enable Docker Runner, in the **Isolate build** section, just choose to run the job in a Docker container and provide the Docker image.



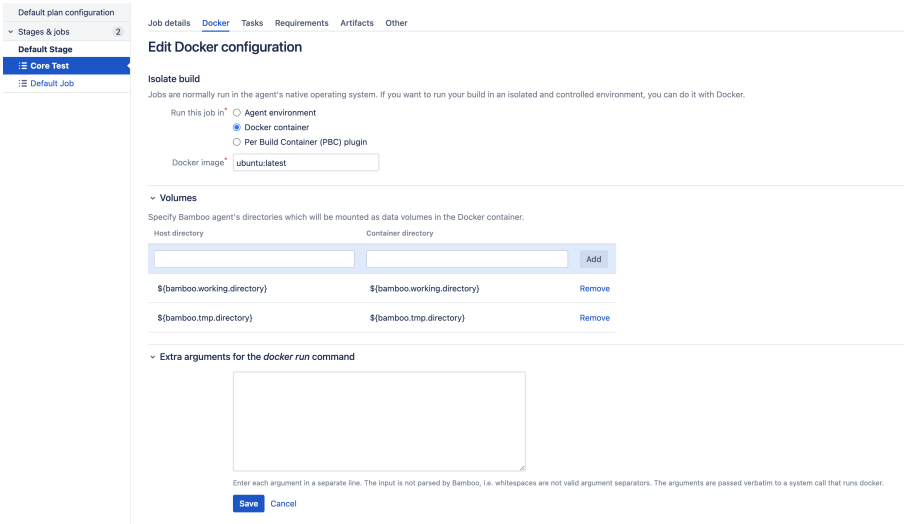
When editing a job or a deployment environment, you can also specify which directories from Bamboo agent should be mounted as additional volumes to the Docker container.

You can also define extra arguments that should be applied to the *docker run* command when starting the container. Note, that each argument must be put in a separate line. The input is not parsed by Bamboo, i.e. whitespaces are not valid argument separators. The arguments are passed verbatim to a system call that runs docker. For instance, in order to define environment variable VAR you need to enter either:

```
-eVAR=VALUE
```

or

```
-e
VAR=VALUE
```



Docker Runner is also fully supported with Bamboo Specs. See our Bamboo Specs reference guide.

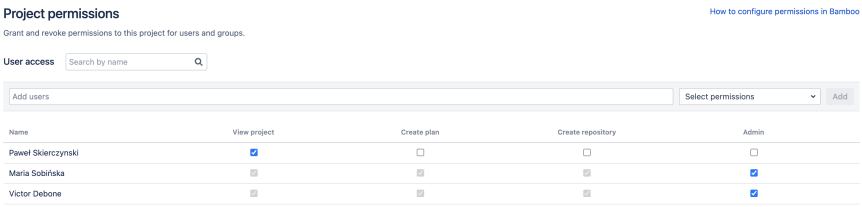
For more information about Bamboo and Docker integration, see Getting started with Docker and Bamboo

Configuring project permissions

You must be an **Administrator** or **Project administrator** to be able to edit permissions on the project-level.

To change project permissions:

1. From the Bamboo header select **Projects**.
2. Select the project you want to set permissions for.
3. In the upper-right corner, select **Project settings** > **Project permissions**.
4. Search for a user or a group you want to grant permissions.
5. From the **Select permissions** drop-down, choose the permission type and select **Add**.



Project permission	Actions
View	• view the project ⚠ The project View permission is a prerequisite for accessing all plans in the project. Without the project View permission, you won't be able to see, run, or administer any plans.
Create plans	• create plans for the project
Create repository	• create repository for the project. Available in Bamboo Data Center version only.
Admin	• manage permissions for the project • manage permissions for all plans in a project • change project settings

Working with branch divergence

Branches are commonly used in version control systems to develop features or bug fixes without affecting your main branch (also known as the *master branch* or *master* for short). This is handy since using a single master branch with multiple people committing their changes at once can create clutter. Branching your code out from master creates a safe space that can contain separate feature configurations, where you can safely commit changes and test Bamboo Specs locally without having to worry about the master branch.

However, if you ran your build plan in an older Bamboo release, the configuration in master would be carried over to your branch, and any additional or custom configuration would be ignored. Currently, Bamboo supports *divergent branches*; that is, branches containing custom Bamboo Specs build plans and feature branch configurations that are different from those on the master branch.

Bamboo supports divergent branches in repositories stored in Bitbucket Server and Bitbucket Cloud.

Prerequisites for allowing branch divergence in Bamboo

Before you can use divergent branches, ensure the following:

- Your code is stored in Bitbucket Server or Bitbucket Cloud
- Bamboo is integrated with Bitbucket Server or Bitbucket Cloud
- Your project is configured as a linked repository of the Bitbucket type
- Bamboo Specs are stored in the bamboo-specs folder in the same repository
- Plan branch detection is enabled (it's on by default)

How to use divergent branches?

That's great! Make sure that you are using Bitbucket Server repository type as other types are not supported yet for divergent branches.

To start using divergent branches just change your specs in the bamboo-specs directory on your branch (it can be either Java or YAML specs) and push your changes!
Consider storing your build plan configuration as code for easier automation, change tracking, validation, and much more. This is the way to best way to create your configuration of divergent branches. We called this feature Bamboo Specs and you can read more about it here.

Know limitations and changes to previous behavior of Bamboo

- Branch divergence is not available for deployment projects. Bamboo ignores any Bamboo Specs on deployment projects in divergent branches.
- The **Other** tab is not available in the plan configuration screen for divergent branches.
- When using divergent branches, you can't link any repositories additional to those on master branch. You can change the configuration of that repository but you can't add or remove it.
- To create a new plan on your divergent branch, you must first create it on master.
- Default settings from *Automatic branch detection* configuration, like triggers and notification settings, are ignored by divergent branches.
- The default repository of a divergent branch is inherited from the master branch and it's not possible to select a different repository on your divergent branch.

Linking to source code repositories

A crucial part of setting up your continuous integration build process is specifying the code repositories with which Bamboo will work. You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.

The recommended approach is to set up linked source repositories at the global level, described below.

Important changes to Linked repositories that affect usage and permissions

Linked repositories are now the preferred way to define and share repository configuration between plans in Bamboo. As a result, we've made two changes that you should be aware of:

- When users create plans, they are only given the option to **select** from Linked repositories. This requires that users have the **Create plan** global permission.
- Users will need the **Create repository** global permission in order to create Linked repositories. Note that this permission alone does not permit a user to create or edit a plan.

In the long term, **Atlassian plans to deprecate the repository configuration defined against the plan**. These configurations can be converted to Linked repositories by clicking **Convert to linked repository** in each plan's repository configuration page.

Link a source code repository for all Bamboo plans

Linked repositories are available globally to all plans and jobs configured on the Bamboo server. Doing this can save you from having to reconfigure the source repositories in multiple places if these ever change - any changes to a linked repository are applied to every plan or job that uses that repository.

You need the Create plan or Admin global permission to configure linked repositories.

1. From the Bamboo header select > **Build resource**s > **Linked repositories**.
2. Select **Add repository.**
3. Select your repository type from the available menu options. For configuration details for a particular repository type, please refer to one of the following pages:
    - Bitbucket (for Bitbucket Cloud)
    - Stash (for Bitbucket Data Center)
    - Git
    - GitHub (for GitHub and GitHub Enterprise)
    - Subversion
    - Perforce

If you need to use an unsupported type of repository, a number of third-party Source Repository plugin modules are available (e.g. the ClearCase plugin). You can also write a Source Repository Module plugin to enable Bamboo to connect to your repository.

Note that shared source repositories are no longer the preferred way to share repository configuration between plans. Use Linked repositories instead!

Configure a repository for a plan

When you create a new plan, the source repository you specify becomes the default. It is used by the plan's default job and can be used by other jobs added to this plan. Note that default repository is an essential concept when working with plan branches, refer to Using plan branches for more details.

1. Navigate to the plan. See Configuring plans for instructions.
2. Select **Actions** > **Configure plan**.
    - Select the **Repositories** tab to see all the repositories that have been added to the plan.
    - Select a repository name in the list to edit its configuration details.

- Select **Add repository** to configure a repository to be used by the plan.
  For configuration details for a particular repository type, please refer to one of the following pages:
  - Bitbucket (for Bitbucket Cloud)
  - Stash (for Bitbucket Data Center)
  - Git
  - GitHub (for GitHub and GitHub Enterprise)
  - Subversion
  - Perforce

In the long term, we plan to deprecate the repository configuration defined against a plan. These configurations can be converted to Linked repositories by selecting **Convert to linked repository** on each plan repository configuration page.

Bitbucket Data Center

Configure Bamboo to use a Bitbucket Data Center repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.
- job – repositories are available to all tasks in the Bamboo job.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

When you link a repository hosted in Atlassian's Bitbucket Data Center with a build plan in Bamboo, then without any further configuration:

- Bamboo will automatically run a build when changes are pushed to the Bitbucket Data Center repository, without needing to configure polling.
- Bamboo will automatically update plan branches when a developer pushes a new branch to the repository (or deletes a branch).
- You can click through to Bitbucket Data Center to see the commit diff for all files that are part of the changeset.
- Bitbucket Data Center commits that are part of a build are displayed in Bamboo.
- Build results are notified to Bitbucket Data Center (and displayed for the associated commits and pull requests).

Bitbucket Data Center and Bamboo only need to have been connected by creating an application link. Repositories in Bitbucket Data Center are then made available in Bamboo, so it is easy for you to link your Bamboo plan to a Bitbucket Data Center repository.

When you create a plan that uses a Bitbucket Data Center source repository, Bamboo will automatically use the Bitbucket Data Center repository to trigger the build when changes are committed trigger option instead of polling the repository for changes option. This reduces the load on the Bamboo and Bitbucket Data Centers because Bamboo doesn't need to send poll requests (for each branch of each plan) to the Bitbucket Data Center every 3 minutes (the default polling period). Instead, Bitbucket Data Center will trigger Bamboo whenever there is a push to the repository.

Configuration requirements

Navigate to the source repository settings for a plan or job, as described on Linking to source code repositories, then:

1. Either select **Add repository** to add a new repository, or edit an existing repository configuration.
2. Select **Stash** from the **Repository host** list.
3. Complete the required information:

Name	A name that identifies this repository within Bamboo.
Server	This menu will show all Bitbucket Data Centers that have been linked to Bamboo via an application link.
Repository	The repository that will be built. This menu will show all repositories on the Bitbucket Data Center that you have permission to access.
Branch	Select a branch if you want to check out code from a branch other than the default branch.
Use shallow clones	Allows Bamboo to perform shallow clones (i.e. history truncated to a specified number of revisions). This should increase the speed of the initial code checkouts. However, if your build depends on the full repository history, we recommend that you don't use this option. Shallow clones are enabled by default.

Enable repository caching on remote agents	Allow caching of repositories on remote agents to save bandwidth. Note that caches are always full clones of the source repository.
Use submodules	Select to enable submodules support if these are defined for the repository. If native Git capability is not defined for agent submodules support will be disabled.
SSH key applies to submodules	Bamboo will use the primary repository's SSH key for submodule authentication.
Command timeout	This helps to stop the hung Bitbucket processes. On slower networks, you may consider increasing the default timeout to allow Bamboo time to make an initial clone of the Git repository.
Verbose logs	Turns on more verbose logs from Git commands. Use this option if you encounter problems with Git in Bamboo.
Fetch whole repository	Fetches whole repository instead of only one selected branch.
Enable LFS support	Enables support for Git Large File Storage (LFS), which replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server. To use this option you must have the following: • Git version 1.8.2 or later installed locally in your environment • Git LFS 1.2 or later installed. To learn more about Git LFS, see Git LFS tutorials.
Mirror	Allows you to use mirror locations for storing your repository data instead of using remote locations. Read more.
Enable quiet Period	Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.
Include /exclude files	Allows you to specify the files that Bamboo should, or should not, use to detect changes. When you configure the Include option, it means that you want Bamboo to use **only** the mentioned files for change detection (by default Bamboo checks all the files). In the same way, if you configure the Exclude option, Bamboo will not consider the excluded files when detecting changes. In **File pattern, enter** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See this page for examples.
Exclude changesets	Enter a regular expression to match the commit messages for changesets that should not start a build.

If your repository can be viewed in a web browser, select the repository type.

This allows links to relevant files to be displayed in the Code changes section of a build result.

Fisheye – specify the following details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (e.g. ' `https://atlaseye.atlassian.com/` ')
- **Repository Name** — the name of your Fisheye repository (e.g. '`Bamboo`'). This is effectively the alias for your repository path
- **Repository Path** — the path for your Fisheye repository (e.g. '`/atlassian/bamboo/`')

See Integrating Bamboo with Fisheye for more information.

Bitbucket Data Center — see Integrating Bamboo with Bitbucket Data Center for more information.

Regenerate SSH keys for Bitbucket Data Center

It may happen that one or more SSH keys you used to authenticate Bamboo with Bitbucket Data Center source code repositories are missing (for example, when an SSH key expires due to the global expiry policy configured by an admin). In such a case, you can regenerate them directly in the Bamboo user interface or by using dedicated REST API endpoints.

On this page:

Regenerating the key for a single repository

When viewing the settings for linked, project, or plan repositories stored in Bitbucket Data Center, you may see an error message stating that Bamboo couldn't access the repository. One of the potential reasons for this issue could be missing SSH keys.

To regenerate the SSH key for an affected repository, navigate to the repository configuration page and select **Save repository** without making any changes to the settings.

Bamboo will attempt to repair the connection to the repository. If the problem persists, it may mean that the repository may have been moved or deleted or that the respective instance of Bitbucket is inaccessible.

Learn more about configuring source code repositories

Bulk regenerating SSH keys

If SSH keys are missing from many repositories and potentially across multiple linked instances of Bitbucket Data Center, you can repair them in bulk using the following Bamboo REST API endpoints.

These operations performed by these endpoints will only add the missing keys and will not replace existing, working ones. Performing key repair operations requires Administrator permissions.

> Depending on the number of Bitbucket Data Center instances and repositories to process, bulk key regeneration may take a long time to complete.

Regenerating keys for all linked instances of Bitbucket Data Center

To regenerate the keys for all affected repositories in all linked instances of Bitbucket Data Center, call the following Bamboo REST API endpoint:

```
POST /rest/stash/latest/bulk/keys/repair
```

Regenerating keys for a specific instance of Bitbucket Data Center

To regenerate the keys for all affected repositories in a single, specific linked instance of Bitbucket Data Center, call the following Bamboo REST API endpoint:

```
POST /rest/stash/latest/bulk/keys/repair/<APPLICATION_LINK_ID>
```

To obtain a particular application link ID, use the following command to get the list of all linked Bitbucket Data Center instances together with their application link IDs:

```
curl -H "Content-Type: application/json" -H "Accept: application/json" http://<BAMBOO_URL>/rest/stash
/latest/servers
```

Response

The responses to the previously described REST calls come in the form of an execution log with information about the processing status of each repository.

Optional filtering parameters

Both endpoints include optional filtering parameters in the request payload to help with narrowing the number of repositories to process:

Parameter	Type	Description	Default value
`search`	string	Case-insensitive phrase to search for in repository names.	Empty
`skipLinked`	boolean	Skip linked repositories.	`false`
`skipPlan`	boolean	Skip plan-based repositories.	`false`
`skipProject`	boolean	Skip project repositories.	`false`
`selectedProjects`	array[string]	List of case-insensitive project keys whose project repositories should be included. By default, all projects are included.	Empty
`selectedPlans`	array[string]	Case-insensitive list of plan keys whose plan repositories should be included. By default, all plans are included.	Empty

For example, the request payload may look like this:

```
{
    "search": "bamboo",
    "skipLinked": false,
    "skipProject": false,
    "skipPlan": true,
    "selectedProjects": ["PROJECTKEY1", "PROJECTKEY2"]
}
```

Bitbucket Cloud

This page describes how to configure Bamboo to use a Bitbucket Cloud repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available for all plans in Bamboo.
- plan – repositories are available for all jobs in the Bamboo plan.
- job – repositories are available for all tasks in the Bamboo job.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

On this page:

Note that you will not be able to create plans or jobs that use a Bitbucket repository without first specifying the shared local Git capability. Learn more about configuring a Version Control capability.

Related pages:

- Git

Configure a Bitbucket source repository

1. Navigate to the repository configuration for a linked repository, plan, or job. See Linking to source code repositories.
2. Either select **Add repository** to add a new repository, or edit an existing repository configuration.
3. Select **Bitbucket Cloud** from the **Source repository** list.
4. Enter a **Name** to help identify the repository in Bamboo.
5. Specify the repository access level and corresponding authentication details for loading the list of repositories:

Step	Public	Private
5.1	Provide the name of the Bitbucket Cloud user who owns the repository in the **Owner** field.	To load the list of repositories available in Bitbucket Cloud you can provide a username and an app password or use shared credentials.
5.2	You can configure the following settings of a public or private source repository for your plan: • **Repository** - retrieves all repositories you have explicit permissions to access from Bitbucket Cloud when you select **Load repositories** • **Branch** - pick a branch if you want to check out code from a branch other than the default branch	

5.3		For private repositories, you can specify the authentication method that Bamboo will use to connect to the repository you selected. You can choose from: • **App password** - create an app password • **SSH private key** - provide an SSH key to authenticate; use shared credentials or upload an SSH key and type a passphrase • **Bitbucket password** - if you created your Atlassian account before September 13, 2021, reuse the credentials provided in step 5.1; see Deprecating Atlassian account password for Bitbucket API and Git activity for more information
5.4	The **Enable webhooks** flag determines whether the source repository will use the webhook-based integration of Bamboo with Bitbucket Cloud. Enabling the flag results in the following behavior: • Plans using this repository default to using the Bitbucket Cloud repository trigger instead of the polling trigger • Branch and pull request detection relies on webhook events rather than polling • Enhanced plan branch configuration with Bamboo Specs becomes available • Webhook registration is enabled on the repository configuration screen and allows to set up the integration in Bitbucket Cloud	

Advanced Options

Use submodules
Select to enable submodules support if these are defined for the repository. If native Git capability is not defined for agent submodules support will be disabled.

SSH key applies to submodules
Bamboo will use the primary repository's SSH key for submodule authentication.

Command timeout
This is useful to stop hung Bitbucket processes. On slower networks, you may consider increasing the default timeout to allow Bamboo time to make an initial clone of the Mercurial repository.

Verbose logs
For Mercurial: Turns on `--verbose` and `--debug` options in Hg or Git commands and passes the output to build logs. Use this option if you encounter problems with Git or Mercurial in Bamboo.

Enable quiet period
Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.

Include/Exclude files
Allows you to specify the files that Bamboo should, or should not, use to detect changes.

Enter into **File pattern** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See subpage for examples.

Exclude changesets
Enter a regular expression to match the commit messages for changesets that should not start a build.

Git LFS

Enables support for Git Large File Storage (LFS), which replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server.
To use this option you must have the following:

- Git version 1.8.2 or later installed locally in your environment.
- Git LFS 1.2 or later installed.

Bamboo 5.15 is shipped with a number of images which also include the Git LFS client.

To learn more about Git LFS, see Git LFS tutorials.

Web Repository

If your repository can be viewed in a web browser, select the repository type.

This allows links to relevant files to be displayed in the Code changes section of a build result.

Note: This option is not available for Git repositories. See

Unable to locate Jira server for this macro. It may be due to Application Link configuration. for more information.

Fisheye – specify the following details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (for example, ' `https://atlaseye.atlassian.com/` ').
- **Repository Name** — the name of your Fisheye repository (for example, 'Bamboo'). This is effectively the alias for your repository path.
- **Repository Path** — the path for your Fisheye repository (for example, '`/atlassian/bamboo/`').

See Integrating Bamboo with Fisheye for more information.

How do I determine my Repository Path?

If you have previously run builds with changes from your repository, the easiest way of determining your repository path is to view the code changes and copy the path from the start of the path of one of the changed files, up to (but not including) the appropriate root directory. The root directories for repositories are the ones shown by Fisheye when browsing a repository (e.g. `trunk`). For example, if a code change listed `/atlassian/bamboo/trunk/bamboo-acceptance-test/pom.xml`, the path would be `/atlassian/bamboo/`.
If you have not previously run builds with changes from your repository, you will need to ask your Fisheye administrator for the repository path indexed by Fisheye.

Bitbucket Cloud and Bamboo webhook integration

Learn how to configure triggering a build from Bitbucket Cloud using webhooks that enables change, branch, and pull request detection. With the integration, you can use branch divergence.

Bamboo statuses in Bitbucket Cloud

You can view the status of Bamboo builds in Bitbucket Cloud.

The Bamboo URL needs to be a fully qualified domain name (FQDN).

The feature is setup automatically if you setup a plan in the following way:

- the repository type is **Bitbucket Cloud** or **Bitbucket Server**

- you provided Bitbucket credentials (username and app password)

The automatic setup of Bamboo build statuses in Bitbucket works with private and public repositories as long as you provide valid Bitbucket credentials.

The build statuses in Bitbucket are displayed in the commit, branch, and pull request views.

For more information about Bamboo statuses in Bitbucket Cloud, see:

- Pull requests and code review
- Integrate your build system with Bitbucket Cloud
- statuses/build Resource

Git

This page describes how to configure Bamboo to use a Git repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.
- job – repositories are available to all tasks in the Bamboo job.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

You need to have previously defined a Git capability before you can configure a Git source repository – see Defining a new version control capability.

Note that Bamboo comes with its own built-in Git implementation. However, you need to use native Git to be able to use symbolic links, submodules, automatic branch detection and automatic merging - these are not supported by the built-in Git.

You can download Git from the following locations:

- For Windows
- For Linux and Mac

Related pages:

- Bitbucket Cloud
- GitHub
- Defining a new version control capability

Configure a Git source repository

1. Navigate to the repository configuration for a linked repository, plan or job. See Linking to source code repositories.
2. Either select **Add repository** to add a new repository, or edit an existing repository configuration.
3. Select **Git** from the **Source repository** list.
4. Enter a **Name** to help identify the repository in Bamboo.
5. You can configure the following settings for a Git source repository for your plan:

Repository URL

The full path to your Git repository (eg: https://bitbucket.org/atlassian/bamboo-git-plugin.git)

Valid URLs are of the form:

- git://host.xz[:port]/path/to/repo.git
- ssh://[user@]host.xz[:port]/path/to/repo.git
- [user@]host.xz[:port]/path/to/repo.git
- http[s]://host.xz[:port]/path/to/repo.git
- /path/to/repo.git
- file:///path/to/repo.git

Branch

Type the name of the relevant branch (or tag) you want to work on. Leave empty to work on the `master` branch.

Authentication type

- **None** – choose if you want to access the repository anonymously.
- **Personal access token** - use personal access token to authenticate with Git API; to learn how to create such token in Git, see the GitHub documentation.
- **SSH private key** – use shared credentials or upload an SSH key and provide the SSH passphrase.

Use shallow clones

Allows Bamboo to perform shallow clones (i.e. history truncated to a specified number of revisions). This should increase the speed of the initial code checkouts, however if your build depends on the full repository history, we recommend that you do not use this option. Shallow clones are enabled by default.

Location of POM file

The path to your project's `pom.xml` file, relative to the root of your Git **Repository URL** (defined above).

(Only available when importing a Maven 2 project)

Git LFS

Enables support for Git Large File Storage (LFS), which replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server.
To use this option you must have the following:

- Git version 1.8.2 or later installed locally in your environment.
- Git LFS 1.2 or later installed.

> Bamboo 5.15 is shipped with different images which also include the Git LFS client.

Advanced Options

Use submodules
Select to enable submodules support if these are defined for the repository. If native Git capability is not defined for agent submodules support will be disabled.

SSH key applies to submodules
Bamboo will use the primary repository's SSH key for submodule authentication.

Command timeout
This is useful to stop hung Bitbucket processes. On slower networks, you may consider increasing the default timeout to allow Bamboo time to make an initial clone of the Git repository.

Verbose logs
Turns on more verbose logs from Git commands. Use this option if you encounter problems with Git in Bamboo.

Enable quiet period
Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.

Include/Exclude files
Allows you to specify the files that Bamboo should, or should not, use to detect changes. When you configure the Include option, it means that you want Bamboo to use **only** the mentioned files for change detection because by default Bamboo checks all the files. The same way, if you configure the Exclude option, Bamboo will not consider the excluded files for detecting changes.

Enter into **File pattern** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See sub page for examples.

Exclude changesets
Enter a regular expression to match the commit messages for changesets that should not start a build.

Git LFS

Enables support for Git Large File Storage (LFS), which replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server.
To use this option you must have the following:

- Git version 1.8.2 or later installed locally in your environment.
- Git LFS 1.2 or later installed.

> Bamboo 5.15 is shipped with Git LFS client image.

To learn more about Git LFS, see Git LFS tutorials.

Web repository

If your repository can be viewed in a web browser, select the repository type.

This allows links to relevant files to be displayed in the Code changes section of a build result.

Fisheye – specify the URL and other details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (e.g. ' `https://atlaseye.atlassian.com/` ').
- **Repository name** — the name of your Fisheye repository (e.g. '`Bamboo`'). This is effectively the alias for your repository path.
- **Repository path** — the path for your Fisheye repository (e.g. '`/atlassian/bamboo/`').

See Integrating Bamboo with Fisheye for more information.

> **How do I determine my Repository Path?**
>
> If you have previously run builds with changes from your repository, the easiest way of determining your repository path is to view the code changes and copy the path from the start of the path of one of the changed files, up to (but not including) the appropriate root directory. The root directories for repositories are the ones shown by Fisheye when browsing a repository (e.g. `trunk`)). For example, if a code change listed `/atlassian/bamboo/trunk/bamboo-acceptance-test/pom.xml`, the path would be `/atlassian/bamboo/`.
> If you have not previously run builds with changes from your repository, you will need to ask your Fisheye administrator for the repository path indexed by Fisheye.

Hiding password for https Git tasks

Starting from Bamboo 6.4, you Git username and password will be stored in a URL kept in turn in a temporary file which will be deleted This way only Bamboo will have access to your Git credentials preventing other users on the same machine from viewing your data.

> If you're using Git Credential Manager for Windows, Gut might ask you for your credentials not at the bash but in the Credential Manager. Because it's operating on your server or agent, you won't be able to see the see the Credential Manager display.

For more information about storing credentials in Git, see Git Tools Credential Storage.

Configuring Git SSH on Windows

SSH overview

You can use SSH keys to establish a secure connection between the Bamboo server and the SCM that hosts Git repositories.

- If no Git capability is configured, Bamboo will use its built-in Git implementation: the built-in Git implementation does not support symbolic links, submodules, automatic branch detection and automatic merging.
- Your SCM administrator must have already enabled SSH access to Git repositories.
- Supported key types are DSA and RSA2. Note that RSA1 is not supported. We've tested key sizes of 768, 1024, 2048, 4096 and 8192 bytes.

On this page:

Enabling SSH access to Git repositories

To enable SSH access:

You need to set up SSH access when you configure your linked repositories:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Build resources**, select **Linked repositories**.
3. Select **Add repository**, and then select Git from the Source repository menu.
4. Complete the following fields:

Field	Description
Display name	The name that identifies the repository when you are using multiple repositories in a plan
Repository URL	The URL of the Git repository
Branch	The name of the branch or tag containing the source code

5. Select **SSH private key** from the **Authentication Key** menu.
6. Select the file containing your SSH key using the Choose file button.
7. Enter the passphrase to allow access to your SSH key.
8. Select **Save repository**.

Once you have enabled SSH access, you will also need to add an SSH server capability:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Build resources**, select **Server capabilities**.
3. Select **Add** under the Add capability heading.
4. Complete the following fields:

Field	Value
Capability type	Git

Executable	SSH
Path	The path to the SSH executable, for example: /usr/bin/ssh

5. Select **Add** to add the SSH capability.

GitHub

This page describes how to configure Bamboo to use a GitHub repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

Related pages:

- Git

Configure a GitHub source code repository

1. Go to the repository configuration for a linked repository or plan. See Linking to source code repositories.
2. Select **Add repository,** or edit an existing repository configuration.
3. From the **Source repository** list, select **GitHub repository**.
4. Enter a display name for this repository so that you can identify it quickly later on.
5. In the **Username** field, enter one of the following values:
   - If you're adding a repository that belongs to your organization, enter the name of the organization as it appears in GitHub.
   - If you're adding your own repository, enter your GitHub username.
6. Enter your GitHub **access token.**

   ⚠ To learn how to create personal access tokens (PAT) in GitHub, see GitHub documentation. Your PAT must have access to the following scopes to work with Bamboo: **public_repo**, **read:org**, **read:user**, **repo:status**. For private repositories, use the following scopes: **repo**, **read:org**, **read:user**.

7. If you are a **GitHub Enterprise** User:
   a. Check the **Using GitHub Enterprise?** box.
   b. Enter your GitHub Enterprise base URL in the **Base URL** text field – if empty it defaults to *https://github.com*.
8. Select **Load repositories**.
9. (optional) Configure the following advanced options for a GitHub source repository for your plan:

Advanced Options

Use submodules
Select to enable submodules support if these are defined for the repository. If native GitHub capability is not defined for agent submodules support will be disabled.

SSH key applies to submodules
Bamboo will use the primary repository's SSH key for submodule authentication.

Command timeout
This is useful to stop hung Bitbucket processes. On slower networks, you may consider increasing the default timeout to allow Bamboo time to make an initial clone of the GitHub repository.

Verbose logs
Turns on `--verbose` and `--debug` options in hg commands and passes the output to build logs. Use this option if you encounter problems with GitHub in Bamboo.

Enable quiet period
Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.

Include/Exclude files
Allows you to specify the files that Bamboo should, or should not, use to detect changes.

Enter into **File pattern** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See sub page for examples.

Exclude changesets
Enter a regular expression to match the commit messages for changesets that should not start a build.

Web Repository

Fisheye – specify the URL and other details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (e.g. '`https://atlaseye.atlassian.com/`').
- **Repository Name** — the name of your Fisheye repository (e.g. '`Bamboo`'). This is effectively the alias for your repository path.
- **Repository Path** — the path for your Fisheye repository (e.g. '`/atlassian/bamboo/`').

See Integrating Bamboo with Fisheye for more information.

How do I determine my Repository Path?

If you have previously run builds with changes from your repository, the easiest way of determining your repository path is to view the code changes and copy the path from the start of the path of one of the changed files, up to (but not including) the appropriate root directory. The root directories for repositories are the ones shown by Fisheye when browsing a repository (e.g. `trunk`)). For example, if a code change listed `/atlassian/bamboo/trunk/bamboo-acceptance-test/pom.xml`, the path would be `/atlassian/bamboo/`.
If you have not previously run builds with changes from your repository, you will need to ask your Fisheye administrator for the repository path indexed by Fisheye.

Perforce

This page describes how to configure Bamboo to use a Perforce repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.
- job – repositories are available to all tasks in the Bamboo job.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

Configure a Perforce source repository

1. Navigate to the repository configuration for a linked repository, plan or job. See Linking to source code repositories.
2. Either select **Add repository** to add a new repository, or edit an existing repository configuration.
3. Select **Perforce** from the **Source repository** list.
4. Enter a **Display Name** to help identify the repository in Bamboo.
5. You can configure the following settings for a Perforce source repository for your plan:

On this page:

Port
Type either the port to which the Perforce client will connect, or the Perforce server itself. This is the Perforce P4PORT environment variable that tells Bamboo which p4d (Perforce server) to use.

Client (Workspace) (3)
The name of the Perforce Client Workspace which Bamboo will use. The Client Workspace determines which portions of the depot are visible in your Workspace Tree.

Do not create two plans or jobs that use the same client (e.g. one client set to manage, the other client set to not manage). This setup will create major issues in your builds.

Depot view
The client view of the depot that contains the source code files for this Plan/Job. This is typically in the form `//<clientname>/<workspace_mapping>/...` For details please see the *Perforce User's Guide.*

Bamboo sets the client root to its working directory, which means that code will be checked out to the working directory/`<workspace_mapping>` location. Please take note of this, when specifying the Artifact copy pattern for your Build artifacts.)

Username
The Perforce username that Bamboo will use when it accesses the server (Port). Leave this field blank if you want Bamboo to use the default Perforce user (i.e. the OS username).

Password
Type the password required by the Perforce username (if applicable).

Let Bamboo manage your workspace (4)
This field indicates whether or not you want Bamboo to manage your workspace.

Use Client Mapping For Change Detection
Select this option if you use overlay mappings for your workspace. Your workspace must be available on the Bamboo Server.

Advanced Options

Enable quiet period
Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.

Include/Exclude files
Allows you to specify the files that Bamboo should, or should not, use to detect changes.

Enter into **File pattern** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See sub page for examples.

Exclude changesets
Enter a regular expression to match the commit messages for changesets that should not start a build.

Web repository
If your repository can be viewed in a web browser, select the repository type.

This allows links to relevant files to be displayed in the Code changes section of a build result.

Generic Web Repository

- **Web repository URL** – the URL of the repository.
- **Web repository module** — the particular repository required for this plan or job, if the **Web repository URL** above points to multiple repositories.

Stash – specify the following details for the repository:

- **Stash URL** – the URL of your Stash (now Bitbucket Server) instance (e.g. https://bitbucket.mycompany.com).
- **Stash project key** – the key of the project in Stash (e.g. CONF).
- **Repository name** – the name of the repository in Stash (e.g. conf-dev).


Use this option to connect to a Bitbucket Server repository.

See Integrating Bamboo with Bitbucket Data Center for more information.

Fisheye – specify the URL and other details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (e.g. `https://atlaseye.atlassian.com/`).
- **Repository name** — the name of your Fisheye repository (e.g. `Bamboo`). This is effectively the alias for your repository path.
- **Repository path** — the path for your Fisheye repository (e.g. `/atlassian/bamboo/`).

See Integrating Bamboo with Fisheye for more information.


How do I determine my Repository Path?

If you have previously run builds with changes from your repository, the easiest way of determining your repository path is to view the code changes and copy the path from the start of the path of one of the changed files, up to (but not including) the appropriate root directory. The root directories for repositories are the ones shown by Fisheye when browsing a repository (e.g. `trunk`)). For example, if a code change listed `/atlassian/bamboo/trunk/bamboo-acceptance-test/pom.xml`, the path would be `/atlassian/bamboo/`.
If you have not previously run builds with changes from your repository, you will need to ask your Fisheye administrator for the repository path indexed by Fisheye.

Notes

1. **You will not be able to create plans or jobs that use a Perforce repository without specifying the shared local Perforce capability first.** Read more about configuring a VCS capability.
2. **Keep your Perforce configuration up to date** — If you are using Perforce as your repository, you must ensure your Perforce configuration in Bamboo is in sync with any changes to your Perforce repository (such as client, depot or user credential changes). If not, your Perforce repository changes may cause unexpected behavior in Bamboo when Bamboo tries to access the repository. See the notes in the configuration instructions below for further details.
3. **Issue when running Bamboo with Perforce prior to Bamboo 2.0.7** — A known issue exists when running Bamboo with Perforce prior to Bamboo 2.0.7 (See BAM-2866 and BAM-2849). If you change the name of your Perforce client (i.e. via an update) without updating your Perforce configuration in Bamboo, Bamboo will not be able to find the Perforce client to run against. Perforce will then create a default client in your running directory. This can lead to situations where Bamboo will attempt to clear out data from your running directory (e.g. force build). To avoid this problem, ensure that you update the **Client** in your Perforce configuration whenever you change your Perforce client.
4. Please be aware of the following implications when either letting Bamboo manage or preventing Bamboo from managing your workspace:
    - **If you let Bamboo manage your workspace,**
        - We recommend this configuration if your jobs will be running on many different machines or different operating systems, as Bamboo sets the client root for you.
        - Bamboo will make configuration changes to the Client Workspace to manage builds (e.g. Bamboo will modify the `host` and `root`). You need to ensure that you enter a Client Workspace in the **Client** field that will be used **solely for Bamboo**.
        - Under this configuration, you should configure one client per job to avoid conflicts when updating the client root.
    - **If you do not let Bamboo manage your workspace,**
        - We recommend this configuration if you wish to reuse your client for several jobs, as Bamboo will retrieve the client root directory from Perforce and use it to run builds.
          ⚠️ **Setting the client root in Perforce:** We strongly recommend that you choose a directory that is dedicated for Bamboo's use only, when you are specifying the client root in your Perforce repository. This directory may get cleaned (i.e. files and sub-directories deleted) if you choose to force clean builds.
        - Under this configuration, you need to ensure that the client root directory exists on all machines that the job will be built on.
        - Please note that alternate roots does not currently work in Bamboo. See issue BAM-2377 for further details.

Using Perforce with Bamboo - limitations and workarounds

There are some limitations to using Bamboo with Perforce. Please read the following information carefully before setting up a build plan to use Perforce.

On this page:

1. Running builds on multiple remote agents or machines

Limitation

You will not be able to run builds on **multiple remote agents and/or multiple remote machines** using a **Perforce repository**, without using one of the workarounds described below. If you try to do so, you will run into problems with change detection that could **cause your agents to build incorrect code**. This problem **does not** affect the running of builds on multiple local agents.

Background

Perforce is a client/server SCM (software configuration management) system that manages your changes/files by storing the change information on its server. However, storing change information on the Perforce server can cause problems when you have clients on multiple agents/machines. If you have downloaded a particular change with a Perforce client, the change will be marked as downloaded by the Perforce server. If you use the same Perforce client on another machine, the Perforce server will incorrectly assume that you have already downloaded that particular change and will not download it. Hence, your agents may not pick up changes correctly and could build incorrect code.

Workarounds

There a few workarounds available for this issue, if you are using Perforce with Bamboo:

- **Restrict your plan to use a single machine** — you can use one or more remote agents to build a plan, if they are *running on the same machine* and you *set the client root yourself* (i.e. do not let Bamboo manage your workspace) so that your agents will build to the same directory.
- **Make Bamboo force a clean build every time it builds** — this will ensure that your agents are always building the correct code. However, it can be an inefficient setup for big projects.
- **Use alternate roots for different machines** — specifying alternate roots for different machines will allow you to work around the change detection issue, as long as the roots on each machine are unique. Please note however, you will be *restricted to three machines* (with three different roots) due to Perforce limitations.

Please see the following Jira issues for further information, BAM-2843 and BAM-2774.

2. Using Perforce Overlay and Exclusionary Mappings in Bamboo

Limitation

You will not be able to control how Bamboo *detects changes* using exclusionary mappings or overlay mappings.

Please note, this issue does not affect you if you only trigger your builds on a schedule or manually, as Bamboo agents still build the correct code when triggered.

You may want to try the Use client mapping for change detection option available in the Bamboo Perforce repository type.

Background

Bamboo currently uses the depot view, not the client view, when detecting changes. Hence, any exclusionary and overlay mappings will not be available during change detection.

For example, if a p4 client uses an overlay mapping like this one:

```
//depot/Prj/... //clientName/depot/Prj/...
+//depot/Dep/... //clientName/depot/Prj/Dep/...
```

and the **Depot** specified in a plan's repository configuration is:

```
//clientName/depot/Prj/...
```

then Bamboo will lookup the corresponding depot view and detect changes by running the following command:

```
p4 changes //depot/Prj/...
```

Consequently, **no changes** to files in `//clientName/depot/Prj/Dep/...` will be picked up by change detection, despite the overlay mapping.

Hence, if you set up your build to trigger when code is updated it will not trigger correctly.

Workarounds

A partial workaround is available in Bamboo, if you wish to use exclusionary mappings for your client workspace. Specify your build plan to exclude files that match a specified pattern by selecting Exclude all changes that match the following pattern from the **Include / Exclude files** dropdown (under the Common repository configuration section). See Perforce configuration for further details. Please note, this will only **exclude one pattern** whereas multiple exclusions can be specified in an exclusionary mapping.

Unfortunately, there is no workaround for overlay mappings in Bamboo.

Please note, we are aware of these problems and are working to address them — see the following Jira issue for further information, BAM-3323.

Subversion

This page describes how to configure Bamboo to use a Subversion repository.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.
- job – repositories are available to all tasks in the Bamboo job.

The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

On this page:

Related pages:

- Setting the SVN workspace format

Configure a Subversion source repository

1. Navigate to the repository configuration for a linked repository, plan or job. See Linking to source code repositories.
2. Either select **Add repository** to add a new repository, or edit an existing repository configuration.
3. Select **Subversion** from the **Source repository** list.
4. Enter a **Display name** to help identify the repository in Bamboo.
5. You can configure the following settings for a Subversion source repository for your plan:

Repository URL
The location of the root of your Subversion repository. For example:

http://svn.collab.net/repos/svn

Note that you can use global variables in this field (see Using Global or Build-specific Variables).

If you are importing a Maven 2 Project, this location should contain your project's `pom.xml` file.

Branch name
The display name of a branch or a module that you want to check out. For example *My project*. The name will be used in the Bamboo UI.

Branch path
The path to a branch or a module that you want to checkout. For example, `trunk, branches/my_branch`. The path is relative to the root URL of the repository.

Note that you can use global variables in this field (see Using Global or Build-specific Variables).

If you are importing a Maven 2 Project, this location should contain your project's `pom.xml` file.

Username (Optional)
The Subversion username (if any) required to access the repository.

Authentication Type

- **Password** – choose this option if you want to authenticate with a username and password.

- **SSH** – if you choose to authenticate using SSH, you need to provide the following details:
  - **Private key** — the absolute path of your SSH private key.
  - **Passphrase** — the passphrase for your SSH private key.

  If you are planning to use remote agents the **ssh private key** file has to be copied to the agent box into the same location as specified.

- **SSL Client Certificate** – if you choose to authenticate using an SSL Client Certificate, you need to provide the following details:
  - **Private key** — the absolute path of your SSL client certificate.
  - **Passphrase** — the passphrase for your SSL client certificate.

  Please note, the client certificate has to be in PKCS12 format and the client certificate file must be passphrase protected, otherwise a runtime exception is thrown by the JDK security engine while opening the user key.

Advanced Options

Detect changes in externals
Select this if your Subversion repository uses svn:externals to link to other repositories (your externals must be in the root of the checkout directory, not in a subdirectory). Please note that you only need to select this check box if you require Bamboo to detect changes in the externals. If your externals reference a particular (static) revision, you do not need to check this box.

Use SVN export
This option will speed up the first-time checkout, but updates are not supported. Implies Force Clean Build.

Enable commit isolation
Ensures that a build will only have one change, allowing you to isolate your build failures.

Automatically detect root URL for branches
Specifies whether the VCS Branching Task automatically determines the location of created branches.

Automatically detect root URL for tags
Specifies whether the VCS Tagging Task automatically determines the location of created branches.

Enable quiet period
Specifies a delay after a single commit is detected before the build is started. This allows multiple commits to be aggregated into a single build.

Include/Exclude files
Allows you to specify the files that Bamboo should, or should not, use to detect changes.

Enter into **File pattern** a regular expression to match the files that Bamboo includes or excludes. The regex pattern must match the file path in the repository. See include/exclude files examples.

Exclude Changesets
Enter a regular expression to match the commit messages for changesets that should not start a build.

Web Repository
If your repository can be viewed in a web browser, select the repository type.

This allows links to relevant files to be displayed in the Code changes section of a build result.

Generic web repository

- **Web repository URL** – the URL of the repository.
- **Web repository module** — the particular repository required for this plan or job, if the **Web repository URL** above points to multiple repositories.

Stash – specify the following details for the repository:

- **Stash URL** – the URL of your Stash (now Bitbucket Server) instance (e.g. https://bitbucket.mycompany.com).

- **Stash project key** – the key of the project in Stash (e.g. CONF).
- **Repository name** – the name of the repository in Stash (e.g. conf-dev).

Use this option to connect to a Bitbucket Server repository.

See Integrating Bamboo with Bitbucket Data Center for more information.

Fisheye – specify the URL and other details for the repository:

- **Fisheye URL** — the URL of your Fisheye repository (e.g. `https://atlaseye.atlassian.com/`).
- **Repository name** — the name of your Fisheye repository (e.g. `Bamboo`). This is effectively the alias for your repository path.
- **Repository path** — the path for your Fisheye repository (e.g. `/atlassian/bamboo/`).

See Integrating Bamboo with Fisheye for more information.

How do I determine my Repository Path?

If you have previously run builds with changes from your repository, the easiest way of determining your repository path is to view the code changes and copy the path from the start of the path of one of the changed files, up to (but not including) the appropriate root directory. The root directories for repositories are the ones shown by Fisheye when browsing a repository (e.g. `trunk`)). For example, if a code change listed `/atlassian/bamboo/trunk/bamboo-acceptance-test/pom.xml`, the path would be `/atlassian/bamboo/`.
If you have not previously run builds with changes from your repository, you will need to ask your Fisheye administrator for the repository path indexed by Fisheye.

Notes

- If you are having problems **connecting to Subversion**, consult our documentation on troubleshooting Subversion connections.
- If you use **pre-1.5 Subversion client** to access code checked out by Bamboo, you may encounter problems with your builds. This is due to the SVNKit upgrade in Bamboo 2.1.4. Please read this knowledge base article for further details.
- You can add the `-Dsvnkit.http.methods=Basic,NTLM` system property to SVNKit to have NTLM authentication work with Bamboo.

Configuring source code management triggers for Subversion

This page provides instructions on how to configure Subversion to send message events that trigger the execution of Bamboo plans. You only need to configure Subversion to send these message events if **The repository triggers the build when changes are committed** build strategy has been specified for one or more of your Bamboo plans.

Configuring Subversion to trigger a build

This section explains how to configure Subversion to trigger a build when the repository is changed. A Subversion hook script is used to perform the trigger action whenever a Subversion repository is changed.

The following commands and script files assume that your Subversion server runs on a UNIX- or Linux-based operating system. If your Subversion server runs on any other operating system, then you will need to modify the script files and if necessary, the commands below to suit that operating system.

On this page:

Related pages:

- Subversion

Step 1. Enable the Subversion post-commit hook

To do this, run the following commands:

```
cd svn-repository-containing-the-build-source-code
cd into the hooks/ directory
```

The Subversion post-commit file is not installed by default. If it does not exist, make a copy of the `post-commit.tmpl` file in the `hooks/` directory, name it `post-commit` and make it executable:

```
cp post-commit.tmpl post-commit
chmod a+rx post-commit
```

Step 2. Install the post-commit trigger

Add a line like the following to the `post-commit` file, for running Bamboo's build trigger script file.

```
/path-to-your-bamboo-installation/scripts/svn-triggers/postCommitBuildTrigger.sh base-url BUILD-KEY
```

where:

- `base-url` is the base URL of the Bamboo server. For example: http://<name-of-machine>:8085
- `BUILD-KEY` is the key of the Bamboo plan to be executed.

Make Bamboo's build trigger script file executable (using `chmod`) so that the Subversion user can execute it.

Step 3. Do a test commit

Conduct a 'test' commit. Bamboo should start building the relevant plan after a few seconds.

The Bamboo log file should contain an entry like this:

```
[INFO] com.atlassian.bamboo.build.UpdateAndBuild - Bamboo build was triggered by remote http call from
127.0.0.1
```

Notes

Build Trigger Security — Bamboo will only accept remote build triggers if the triggers originated from the Subversion server(s) identified in the Subversion **Repository URL** of any Bamboo plans. Requests originating from other Subversion servers will be rejected by Bamboo.

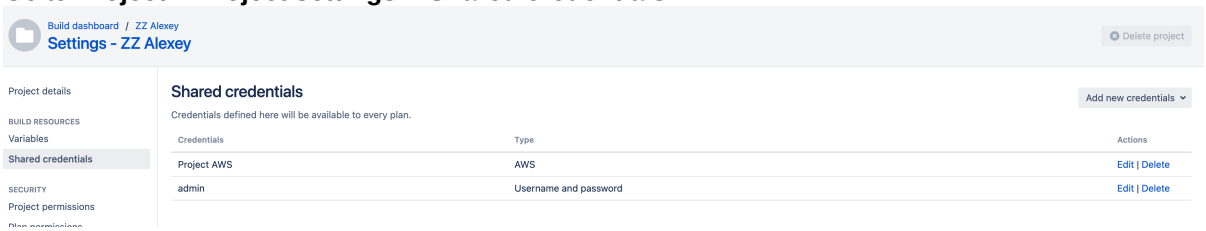
Project-level build resources

⚠ Adding and managing repositories and shared credentials on the project level without global admin help is available for the Data Center licenses only.

Credentials shared at the project level allow more granular distribution of work and responsibilities over projects. Starting from version 8.0, Bamboo uses project-level shared credentials which are visible only for plans from the same project. These credentials are not available for any plans outside of the project. Any user with project admin permissions can manage shared credentials.

To manage projects' shared credentials:

Go to **Project** > **Project settings** > **Shared credentials**.



For detailed instructions on how to add, edit, and delete shared credentials, see Global shared credentials.

To move or clone plans with shared credentials:

Bamboo doesn't allow moving or cloning plans that use shared credentials for new projects. To move/clone such plan to a new project, you must first clone it within the same project, and disable the use of shared credentials.

Project repositories

Repositories defined at the project level are available only for plans in the same project. Project repositories can use project shared credentials from the same project.

Permissions for project repositories are the same as for global repositories and allow flexible configuration of access for different users and groups.

Bamboo doesn't allow cloning or moving plans that use project repositories.

Bamboo Specs

Project repositories support Bamboo Specs execution with the following differences from linked repositories:

- Project repositories can access all plans at the project level.
- Project repositories can't access plans that are outside the project.
- Linked repository admin should grant access of project repo to specified linked repo
- It’s possible to turn on the access of the project repository to all repositories of a given project,

Shared credentials

You can store credentials within Bamboo for easier access to repositories and Amazon Web Services. The access details that you provide are available to all plans on your Bamboo server.

To manage shared credentials:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Build resources**, select **Shared credentials**.
3. You can add, edit, or delete existing credentials.

Related pages:

- Linking to source code repositories
- Checking out code
- Configuring plans

Adding shared credentials

1. Select **Add shared credentials** and select the type of credentials that you want to add.
2. Provide the details:

Type	Field	Description
	Credential name	The name of the credential set. Make the name meaningful, as Bamboo refers to the credential set by its name without quoting the details.
AWS	Access key ID	Credentials assigned to each IAM user in the AWS management console. For more information, see AWS account for Bamboo.
	Secret access key	
SSH	SSH key	The **private** key from the SSH key pair that you created to authenticate with a repository. You must specify the **public** key on the repository host side.
	SSH passphrase	The passphrase for accessing the SSH private key
Username and password	Username	The username with which you want to authenticate
	Password	The password with which you want to authenticate

3. Select **Save credentials** to add the credentials to the shared credentials list.

Editing shared credentials

You can modify the details of the existing shared credential by selecting **Edit** next to the credential name, then selecting the **Change** check box.

Deleting shared credentials

You can delete existing shared credentials by selecting **Delete** next to the credential name.

After you select **Delete**, Bamboo displays a message with a list of repositories or plan tasks that depend on the credential and might break if you delete it.

Smart Mirroring

Starting with version 6.2, Bamboo provides support for Smart Mirroring in Bitbucket Data Center and Server. Smart Mirroring allows you to use local repositories for storing your repository data to avoid having to wait when cloning a repository from a remote location.

For more on Smart Mirroring, see the Bitbucket Data Center and Server documentation.

Prerequisites

- Bitbucket Data Center and Server 7.5 or later
- Bitbucket Data Center and Server license with mirrors support
- routing from Bamboo to Bitbucket Data Center and Server instance
- routing between agent and mirror enabled

To enable Smart Mirroring in Bamboo:

1. From the Bamboo header select > **Build resources** > **Linked repositories**.
2. Select your Bitbucket Data Center and Server repository.
3. In the Edit repository section, select **Advanced options**.
4. From the **Mirror** drop-down list, select your mirror location.

Bulk update of Smart Mirroring settings for repositories

Starting from Bamboo 6.10.3, you can update the mirrors for all repositories of a single Bitbucket Data Center and Server.

You can update the **Mirror** field value by using the following REST endpoint:

```
POST /rest/stash/latest/bulk/APPLICATION_LINK_ID/mirror
```

- `APPLICATION_LINK_ID`: the ID of a linked Bitbucket Data Center and Server instance. Read below for more info on how to find it.

Payload format

1. Perform a dry run (without an actual update), and reset all repositories to the master node. This will allow you to disable the usage of mirrors for all Bitbucket Data Center and Server repositories in Bamboo.

```
{
        resetToPrimary:true,
        dryRun:true
}
```

Here's a sample output:


```
{
  "size": 21,
  "limit": 21,
  "lastPage": true,
  "start": 0,
  "values": [
    {
      "id": 1770061851,
      "name": "Bamboo Version Stable",
      "global": false,
      "selfUrl": "http://BAMBOO_URL/chain/admin/config/editChainRepository.action?buildKey=ZJM-VCT",
      "oldMirrorId": "",
      "oldMirrorName": "",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    },
    {
      "id": 2357198853,
      "name": "bamboo-stash-plugin",
      "global": true,
      "selfUrl": "http://BAMBOO_URL/admin/editLinkedRepository.action?repositoryId=2357198853",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    },
    {
      "id": 1642397765,
      "name": "bamboo-doc-code",
      "global": true,
      "selfUrl": "http://BAMBOO_URL/admin/editLinkedRepository.action?repositoryId=1642397765",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    },
    {
      "id": 1775861823,
      "name": "bamboo-some-code",
      "global": true,
      "selfUrl": "http://BAMBOO_URL/admin/editLinkedRepository.action?repositoryId=1775861823",
      "oldMirrorId": "",
      "oldMirrorName": "",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    },
    {
      "id": 2439381015,
      "name": "static assets",
      "global": false,
      "selfUrl": "http://BAMBOO_URL/chain/admin/config/editChainRepository.action?buildKey=TEST-
CIWEBDRIVERCDN",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    },
    {
      "id": 1738702849,
      "name": "atlassian-plugin",
      "global": true,
      "selfUrl": "http://BAMBOO_URL/admin/editLinkedRepository.action?repositoryId=1738702849",
      "oldMirrorId": "BFHH-O74T-KDGH-CT47",
      "oldMirrorName": "US East Mirror",
      "newMirrorId": "primary",
      "newMirrorName": "Primary"
    }
]
```


2. Update all Bitbucket Data Center and Server repositories, so they start using the provided mirror.

```
{
        mirrorId:"MIRROR_ID",
        ignoreErrors: true
}
```

This will tell all repositories linked to a given Bitbucket Data Center and Server Application Link to use the provided mirror. The `ignoreErrors` flag allows to skip updating repositories that aren't supported by the given mirror. The response will contain all affected repositories.

Obtaining the Bitbucket Server Application Link value

Use the following snippet to print the list of all Bitbucket Data Center and Server instances known to Bamboo, together with their Application Link IDs:

```
curl -H "Content-Type: application/json" -H "Accept: application/json" http://BAMBOO_URL/rest/stash/latest
/servers
```

Obtaining the Mirror ID value:

Use the following snippet to print the mirror servers that are configured in Bitbucket Data Center and Server, together with their IDs:

```
curl -H "Accept: application/json" https://BITBUCKET_SERVER_URL/rest/mirroring/latest/mirrorServers
```

⚠ Downgrading from Bitbucket Data Center and Server license to a regular license may cause problems. In the unlikely event of such a downgrade, we recommend to switch all repositories to primary before switching versions.

Enabling webhooks

This page relates to Repository Stored Specs-enabled webhooks only

RSS webhooks will trigger a scan for changes on the `**/bamboo-specs` folder within a repository and engage with build triggers if Bamboo finds any changes to the Specs Plan. To know how to configure build triggers for non-Repository Stored Specs commits, check the following pages:

- Repository triggers the build when changes are committed
- Git repository remote trigger

Webhooks allow your repositories other than Bitbucket Server to communicate with Bamboo. This page describes enabling webhooks only to Repository Stored Specs repositories. Once you setup a webhook for this repository, a repository sends the HTTP request to Bamboo with every new commit. This HTTP request, in turn, triggers a scan for changes on the ****/bamboo-specs** folder within a Repository and triggers the build in case Bamboo finds any changes to the Specs Plan. If Bamboo detects any changes in a repository, it automatically updates necessary plans and deployments.

Bamboo supports the use of Java and YAML Specs with the following repositories:

- Bitbucket Cloud
- Git
- GitHub
- Subversion

To enable webhooks in your repository:

1. In Bamboo, generate a URL used as a destination for your webhook:
    a. In the top navigation bar, select **Specs** > **Setup Specs Repository**.
    b. Select your project type.
    c. Select the repository for Bamboo Specs.
    d. Copy the URL that is generated for you as the destination for the webhook.
2. In the source repository you want to use for storing Bamboo Specs, use the Bamboo URL to enable the webhook:
    a. In your repository provider application (Bitbucket, Subversion, Github, etc.), go to your repository settings.
    b. Find webhook-specific configuration.
    c. Paste in the URL Bamboo provided you with.

Bamboo webhook is now enabled in your repository.

Triggering builds

Triggering in Bamboo allows plan builds to be started automatically. Bamboo has the following trigger methods:

Trigger a build if code has changed:

- poll the repository for changes
- a push to the repository triggers the build
- based on a tag

Trigger a build based on a schedule:

- cron-based scheduling
- single daily build

Trigger a build depending on the outcome of other plans:

- Plan builds are triggered by preceding successful builds of other plans.
- Plan only builds if other specified plans are building successfully.

On this page:

Related pages:

- Running a plan build manually
- Setting up plan build dependencies

Note that a plan that has no configured triggers can only be started manually, or if it is dependent on the successful build of another plan.

From Bamboo 4.3, you can configure multiple triggers for each plan. This allows a plan to be triggered by different trigger types, and to have triggering scenarios such as "every 5 minutes between 9:00 am and 10:00 am, and every 20 minutes between 1:00 pm and 10:00 pm".

Triggers can only be configured by a Bamboo administrator.

Choosing a triggering strategy

This table lists the ways in which plan builds can be triggered in Bamboo.

Triggering option	Description
Polling the repository for changes	Bamboo will poll the selected source code repositories for code changes, using either a specified interval (that is, periodically) or a schedule. If Bamboo detects code changes, a build of the plan is triggered. • Your VCS must service a check out or update command whenever it is polled, even if no code has changed in a repository. See Repository polling.

Repository triggers the build when changes are committed	Bamboo waits to receive a message about changed code from any of the selected source code repositories. When Bamboo receives such a message, a build of the plan is triggered. • This option minimizes server load, because message events are sent only when code changes to a repository are committed. • You must configure your source code management system to send message events to Bamboo about code changes in the repositories. • This is the default option when you use a linked Bitbucket Server repository. See Repository triggers the build when changes are committed.
Cron-based scheduling	Bamboo will trigger scheduled builds of this plan based on a cron expression. • This option allows you to schedule builds when server load is likely to be minimal, for example, outside office hours. • Scheduled builds are triggered irrespective of any code changes in the source code repositories. See Cron-based scheduling.
Single daily build	Bamboo will trigger a build of the plan once per day at a specified time. • Can be set up to run at a time of you choice. • This option is suitable if a build of this plan takes a long time to complete. • Scheduled builds are triggered irrespective of any code changes in the source code repositories. See Single daily build.
Tag triggering	You can schedule Bamboo to run a build automatically whenever a selected tag appears in your repository. Tag triggering is enabled by default in Bamboo and becomes available when you link a repository to your Bamboo instance. You can use tag triggering in Bamboo with the following repository types: • Bitbucket Cloud • Bitbucket Server/ Stash • GitHub repository • Git See Tag triggering.

Repository polling

You can configure Bamboo to poll the repository for source code changes, either:

- periodically (e.g. every 180 seconds), or
- based on a schedule (e.g. the second Sunday of every month at 5:00 am).

If Bamboo detects a change in the source code, a build of your plan is triggered.

Related pages:

- Triggering builds
- Repository triggers the build when changes are committed
- Cron-based scheduling
- Single daily build

To configure Bamboo to poll the repository for source code changes:

1. From the dashboard select **Build** > **All build plans**.
2. Locate the plan in the list and select the edit icon ( ) to display the plan's configuration pages.
3. Select the **Triggers** tab, then select either an existing trigger or **Add trigger**.
4. Select the **Repository polling** trigger type.
5. Bamboo displays the available repositories for the plan, as previously configured on the **Source repositories** tab. Optionally, enter a trigger description. Select the repositories that this trigger should apply to.
6. Select a polling strategy:
    a. **Periodically**
    Enter a **Polling frequency** value (in seconds) for the time between when Bamboo checks for repository changes.
    b. **Scheduled**
    Select the edit icon ( ) to use the Schedule Editor to set the polling schedule.
    Note, this is a schedule for polling your repository: a plan build will only be triggered if there are source code changes. See Triggering builds.
    Note that for the **Cron expression** option, a cron expression consists of 6 mandatory and one optional field. The fields in sequential order are: seconds, minutes, hours, day-of-month, month, day-of-week and (optional) year. For example, `0 0 1 ? * 1#2`. For information on Cron expressions, see this FAQ: Constructing a cron expression in Bamboo.
7. Select **Save trigger**.

Repository triggers the build when changes are committed

Using the source repository to trigger the build of a plan is one of the available methods for triggering builds in Bamboo.

Repository triggers the build when changes are committed method has the advantage of placing minimal load on your Bamboo server. However, it requires that your source repository is configured to fire an event to the Bamboo server (which the configured plan will listen for).

Configuring the repository to trigger the build when changes are committed requires two changes:

1. Configuring your source repository
2. Configuring Bamboo to respond to post-commit messages

The overall process is: a commit to the repository causes a post-commit message to be sent to Bamboo. Bamboo responds by checking the repository for unbuilt changes. If changes are found, Bamboo triggers a build.

If you're using Bitbucket Cloud, we recommend that you configure build triggering with webhooks. See Triggering a Bamboo build from Bitbucket Cloud using Webhooks.

Related pages:

- Triggering builds
- Repository polling
- Cron-based scheduling
- Single daily build

1. Configuring your source repository

Configure your source code management system's repository to send post-commit event messages to Bamboo. These messages tell Bamboo to begin building the plans that use this repository.

Add the Bamboo webhook to your repository in Bitbucket Cloud. No further action is necessary on your local repository. Each push of new commits in to Bitbucket will trigger the build based on your configuration.

When you create a plan that uses a linked Bitbucket Server repository, Bamboo uses the Bitbucket repository triggers the build when changes are committed trigger option by default.

Learn how to setup Remote repository triggering in Bamboo when using Git repositories: Git Repository - Remote Trigger. Edit the Git repository's `.git/hooks/post-receive` trigger file, for example with the following:

```
/pathto/postCommitBuildTrigger.sh http://bambooserver JIRA-MAIN JIRA-BRANCH
```

where:

- Jira-MAIN and Jira-BRANCH are the Bamboo plans that you would like to trigger
- Jira is the project key
- BRANCH or MAIN are the plan key

For Git, use the SVN `postCommitBuildTrigger.sh` script. See below for more information about the scripts.

Edit the Hg repository's `.hg/hgrc` settings, for example with the following:

```
[hooks]
commit = /pathto/postCommitBuildTrigger.sh http://bambooserver JIRA-MAIN JIRA-BRANCH
```

where:

- Jira-MAIN and Jira-BRANCH are the Bamboo plans that you would like to trigger
- Jira is the project key
- BRANCH or MAIN are the plan key

See below for more information about the scripts.

If you are using a remote SVN server, copy file "atlassian-bamboo/repositoryScripts/svn-triggers /postCommitBuildTrigger.sh" (.py for Windows installations) located in the Bamboo install directory to the SVN repository .../hook/post-commit folder so that the post CommitBuildTrigger file is accessible from the `post-commit` trigger file.

Edit the Subversion repository's `hooks/post-commit` trigger file, for example with the following:

```
/path/to/postCommitBuildTrigger.sh http://bambooserver JIRA-MAIN JIRA-BRANCH
```

where:

- Jira-MAIN and Jira-BRANCH are the Bamboo plans that you would like to trigger
- Jira is the project key
- BRANCH or MAIN are the plan key

See below for more information about the scripts. Also, refer to Configuring source code management triggers for Subversion.

Add the script as a change-commit trigger.

```
triggerName change-commit //myDepot/... "/usr/local/bin/postCommitBuildTrigger.sh
http://bambooServer/ MYPLAN-DEFAULT"
```

See below for more information about the scripts.

Edit two files in the CVSROOT module: `commitinfo` and `loginfo`.

- For `commitinfo`, add a line like this:

  ```
  ^jira(/|$)  /pathto/preCommit.sh
  ```

  where *jira* is your module.
- For `loginfo`, you can add a line. For example:

  ```
  ^jira(/|$) /pathto/postCommitBuildTrigger.sh %{} http://bambooserver JIRA-MAIN JIRA-BRANCH
  ```

  where:
  - Jira-MAIN and Jira-BRANCH are the Bamboo plans that you would like to trigger
  - Jira is the project key
  - BRANCH or MAIN are the plan key

See below for more information about the scripts. Please refer to Configuring source code management triggers for Subversion.

- You can download the scripts using this link. Use the same SVN script for Git. Copy the scripts to your repository. The scripts can also be found in the `/scripts` folder of your Bamboo Installation Directory.
- Depending on which operating system your repository is running on, you may need to edit the scripts. The scripts assume that `wget` is in `/usr/bin/`; if this isn't the case for your repository (e.g. Solaris 10 has it in `/usr/sfw/bin/`), edit the scripts and change `/usr/bin/` to the appropriate location.
- Ensure that the user which Bamboo is running has appropriate file permissions to execute the scripts, i.e. the scripts should be executable by non-root user(s).

2. Configuring Bamboo to respond to post-commit messages

Before you begin:

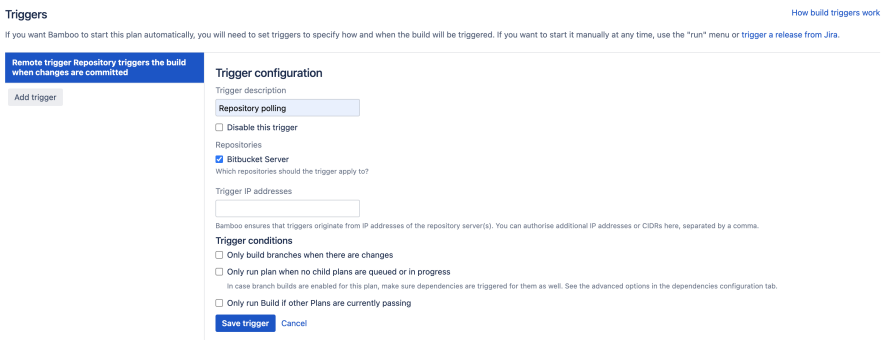
- Triggering a build when there is no repository update — Bamboo will ignore build triggers if the local working copy and the repository copy have the same revision numbers. When testing your build triggers, ensure that the local working copy is not the latest version - if this is the case, Bamboo will take no further action.
- If you're using the Bitbucket Cloud Bamboo post-push hook, ensure that the user you are using to authenticate triggering the build has the build permission on the plan you are attempting to trigger.

To configure Bamboo to trigger a build on code check-in:

1. From the Bamboo dashboard select **Builds** > **All build plans**.
2. Locate the plan in the list and select the edit icon (✎) to display the plan's configuration pages.
3. Select the **Triggers** tab, then **Add trigger**.
4. Select **Remote trigger**.
5. Bamboo displays the available repositories for the plan, as previously configured on the **Source repositories** tab. Optionally, enter a trigger description. Select the repositories that this trigger should apply to.
6. Only enter an IP address in **Trigger IP addresses** if you want Bamboo to trigger on post-commit messages from other than the primary IP address for the repository.

   ⚠ If you use a Mercurial or Git repository then you *must* type the IP address of your repository host in **Trigger IP addresses**. For **Bitbucket Cloud** the current *outbound* IP addresses can be found at Access Bitbucket Cloud from Behind a Firewall.

7. Select **Save trigger**.



Cron-based scheduling

Using a cron-based schedule to trigger the build of a plan is one of the available methods for triggering builds in Bamboo. This schedule is configured using the Schedule editor.

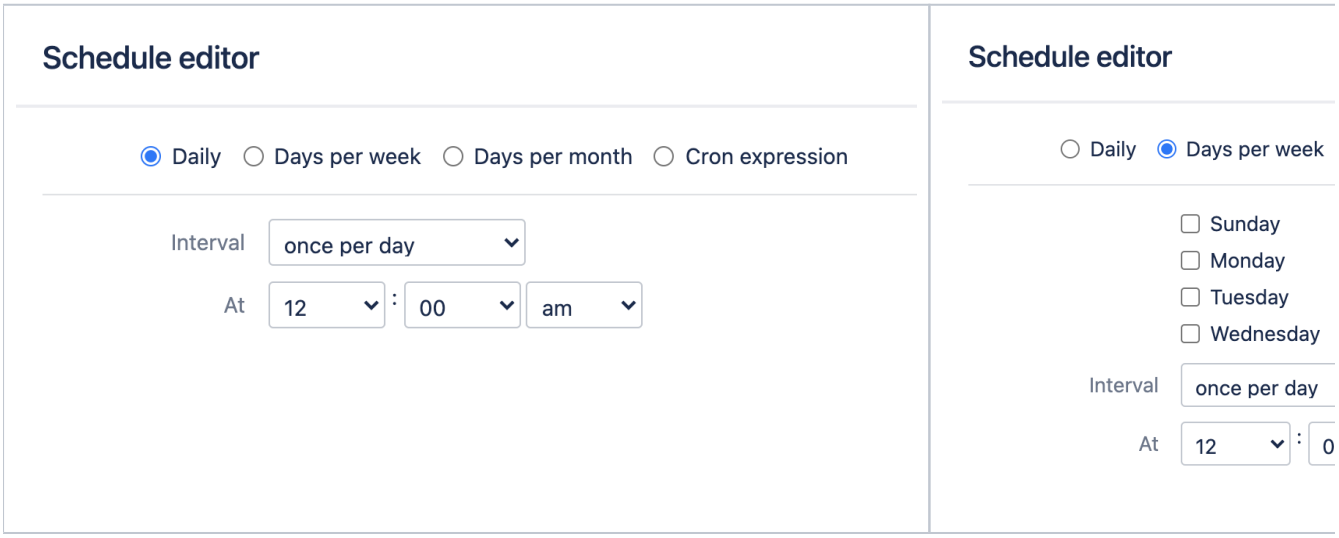
The schedule can be daily (times per day), weekly (days per week), monthly (days per month) or based on a cron expression.

Related pages:

- Triggering builds
- Repository polling
- Repository triggers the build when changes are committed
- Single daily build

To schedule a plan build using a cron expression:

1. From the Bamboo header select **Build** > **All build plans**.
2. Locate the plan in the list and select the edit icon ( ) to display the plan's configuration pages.
3. Select the **Triggers** tab, then select either an existing trigger or **Add trigger**.
4. Select the **Scheduled** trigger type.
5. Optionally, enter a trigger description.
6. Select the edit icon ( ) next to the current schedule to display the Schedule editor.
7. Use the Schedule editor (see screenshots below), to specify the build schedule for your plan. For information about cron expressions, see this FAQ: Constructing a cron expression in Bamboo.
8. Select **Save trigger**.



Constructing a cron expression in Bamboo

Cron is a time-based job scheduler used in Unix/Linux computer operating systems with a unique and powerful terminology. A number of scheduling features in Bamboo, such as build expiry and elastic instance scheduling, require you to specify your requirements as a cron-based expression. For example, a cron expression such as "0 0/30 9-19 ? * MON-FRI" signifies that a scheduled event will be triggered every half an hour from 9am to 7pm, Monday to Friday.

A cron expression comprises of 6 mandatory and one optional field to specify a schedule. The fields in sequential order are: seconds, minutes, hours, day-of-month, month, day-of-week and (optional) year, i.e.

`<seconds> <minutes> <hours> <day-of-month> <month> <day-of-week> <year (optional)>`

Each field can be expressed as an integer (e.g. **`1,2,3,`** etc) and special characters can be used in most fields as well (i.e. '**`, - * / ? L W #`**').

Bamboo uses **OpenSymphony's Quartz** to schedule cron tasks. The syntax it accepts may vary from other cron implementations. Please refer to the Quartz CronTrigger Tutorial documentation for further information on each of these parameters and more detailed examples.

Single daily build

Triggering the build of a plan to run at a particular time each day is one of the available methods for triggering builds in Bamboo.
A Single daily build runs at a time of your choice. This is particularly suitable for builds that take a long time to complete.

Related pages:

- Triggering builds
- Repository polling
- Repository triggers the build when changes are committed
- Cron-based scheduling

To schedule a plan build at a specified time each day:

1. From the Bamboo header select **Build** > **All build plans**.
2. Locate the plan in the list and select the edit icon ( ) to display the plan's configuration pages.
3. elect the **Triggers** tab, then select either an existing trigger or **Add Trigger**.
4. Select the **Single daily build** trigger.
5. Optionally, enter a trigger description.
6. Specify the time of day at which the build should run in **Build time**. Use hh:mm format, with a 24-hour clock.
7. Select **Save trigger**.

Running a plan build manually

Typically in Bamboo, your build plans are configured to be automatically triggered when code changes are committed to the working repository, or according to a schedule.

However, there can be scenarios where you do not want the plan to be automatically triggered:

- The plan should only ever be run manually.
- You want to choose the revision of the default repository that should be used for the build.
- You want to run a customized build, so as to override global variables or plan variables.
- You want to select particular manual stages to run.
- You want the plan to be triggered by other plans that build successfully first.

This page describes how to run a plan build manually, and the options available when running a customized plan build.

Running a plan build manually

To start a plan build manually:

1. Locate the relevant plan on the Dashboard.
2. Select the run icon () for the plan.

Alternatively, if you are viewing the plan, simply select the **Run** menu.

On this page:

Related pages:

- Triggering builds
- Setting up plan build dependencies
- Stopping an active build
- Defining plan variables

Running a customized manual build

If you trigger a plan build manually, you can customize the following aspects of how the plan is run (when these are available):

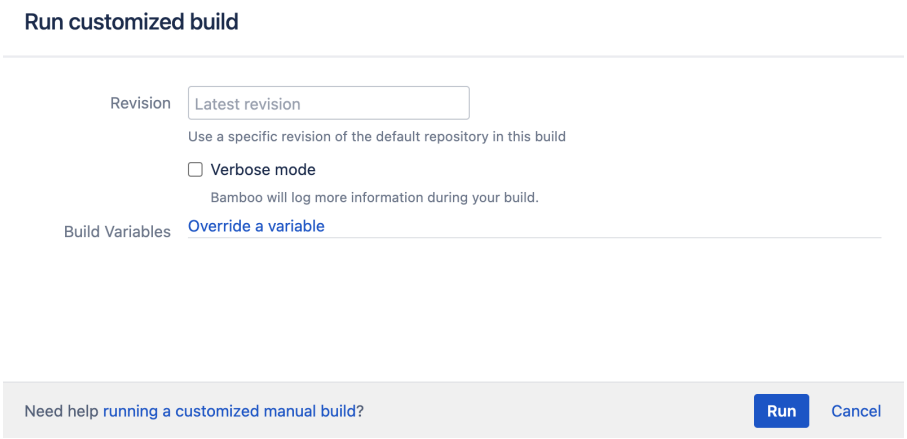
- Select the revision of the default repository that should be used.
- Override any global variables or plan variables with your own parameters when triggering a build manually. This is referred to as running a parameterized plan build.
- Select which manual stages to run, if manual stages have been configured for the plan.

To run a customized plan build:

1. Locate the relevant plan on the Dashboard.
2. Select the plan name to go to the Plan summary.
3. Select **Run** > **Run customized...**.
4. Customize the following aspects of the plan:

Revision	Select a repository revision to use for the build. Note that: • You can only select revisions from the default repository. • The build is not included in plan statistics or telemetry. • SVN repositories use the revision number • Perforce projects use the changelist number • Git repositories use the changeset number • Mercurial repositories use the tag **Note for Subversion repositories that make use of externals** When running a build with a custom revision on a Subversion repository with externals, Bamboo will choose the latest revision in the external repository. This is because Subversion externals always use the latest version and cannot be fixed at a specific revision.
Verbose mode	Allow Bamboo to log additional information, like logs from various VCS and environment variables, during your build.
Build Variables	Select **Override a variable** for another variable to override.
Stages	Select the stages that should be run.

5. Select **Run**.



Rerunning a failed stage

If a stage has failed in your build, you can choose to rerun the stage (with exactly the same data) instead of the entire plan.

To rerun a stage:

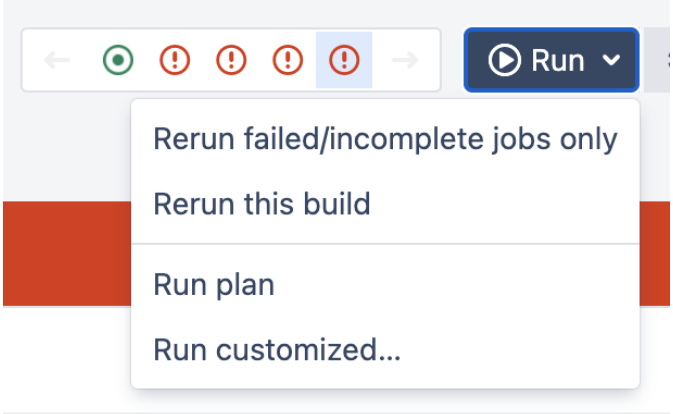
1. Navigate to the failed build result, as described on Viewing a build result.
2. Select **Run** > **Rerun failed/incomplete jobs only** to run the stage again.

Note that:

- Only failing jobs will be rerun.
- Subsequent stages will be executed automatically, unless they are manual stages.
- You might want to add a comment to the build result to record the reason for failure. The existing build result will be overwritten (Bamboo will not create a new build) and the previous failure reason will not be retained.
- For plans based on a Subversion repository, you can only rerun the failed job or the whole plan.
- Starting from Bamboo 7.2, logs for rerun jobs are kept in separate files to allow your for quicker analysis of potential problems.

Related pages:

- Running a plan build manually
- Configuring plans
- Configuring jobs
- Using stages in a plan



Triggering a Bamboo build from Bitbucket Cloud using Webhooks

Webhooks provide a way to allow Bitbucket Cloud to make requests to your server whenever certain events occur in Bitbucket Cloud. The purpose of this guide is to help you setup a webhook that will trigger a Bamboo build from Bitbucket Cloud after certain events like creating a branch or pull request, or pushing a commit.

1. When creating or editing a repository, turn on the **Enable webhooks** flag in the main configuration section.
2. Register and configure a webhook either from Bamboo or Bitbucket Cloud in one of the following ways:
    - From Bamboo, use the **Register webhook** button just below the **Enable webhooks** flag on the repository configuration page. In the dialog configuration screen, use the appropriate option for credentials that have access to the Bitbucket Cloud repository webhooks. Also, provide the URL from which Bamboo is accessible by Bitbucket Cloud. Bamboo takes care of registering all required trigger types.
    - From Bitbucket Cloud, add a webhook from the repository by selecting **Bitbucket Cloud** > **Repository** > **Settings** > **Webhooks** > **Add webhook**, and then configure the webhook:
        a. Specify any title for your Webhook that aptly describes it.
        b. Enter the URL of the Bamboo API to trigger a plan as follows:
        `<BAMBOO_URL>/rest/bitbucket-cloud/latest/webhooks`
        c. To achieve all the integration capabilities, select all the required trigger types:
            - From the list of repository triggers, select **Push**.
            - From the list of pull request triggers, select:
                - **Created**
                - **Updated**
                - **Merged**
                - **Declined**
        d. Save your changes.
3. Configure your Bamboo plan to trigger a build using the Bitbucket Cloud repository trigger by selecting **Actions** > **Configure plan** > **Triggers** > **Add trigger** > **Bitbucket Cloud repository trigger**.

    If the plan was created after creating the repository, this is the default trigger for the plan.

That's it! If your webhook doesn't appear to be working, the best place to start troubleshooting is by viewing the request and response at **Bitbucket Cloud** > **Repository** > **Settings** > **Webhooks** > **View requests**.

Triggering a build from Bitbucket Cloud using the Remote trigger (legacy)

Learn how to set up the Remote trigger to kick off a Bamboo build from Bitbucket Cloud after certain events like creating a branch or pull request, or pushing a commit.

⚠ Using this method of triggering builds from Bitbucket Cloud is no longer recommended and should only be used to troubleshoot legacy integrations with Bitbucket Cloud. For new integrations, we recommend that you use webhooks. See Triggering a Bamboo build from Bitbucket Cloud using Webhooks.

1. Configure the Bamboo plan to use the Remote trigger:
    a. From the plan configuration page, select **Actions** > **Configure plan** > **Triggers** > **Add trigger** > **Remote trigger**.
    b. Select **Scheduled** or **Single daily build** depending on the times you want your build to run.
    c. Bamboo displays the available repositories for the plan, as previously configured in the **Source repositories** tab. Optionally, enter a trigger description. Select the repositories that this trigger should apply to.
    d. In the **Trigger IP addresses** field, add the IP addresses of the request sources as a comma-separated list.

    ⓘ For the current list of Bitbucket Cloud IP addresses for outbound POST requests by webhooks, see Bitbucket Cloud IP Addresses.

    When configuring trigger IP addresses, consider the following:

    - CIDR Notation is only supported in Bamboo 5.14 or newer.
    - Bamboo's IP allowlist also inspects the X-Forwarded-For HTTP header to determine the origin of the request. If you have a reverse proxy forwarding request to your Bamboo instance, depending on its configuration, you may also need to add the IP address of the proxy server to the allowlist.

    e. Select **Save trigger**.
2. Add the IP addresses of the request sources to the trigger
3. In Bitbucket Cloud, add a webhook from the repository:
    a. Select **Bitbucket Cloud** > **Repository** > **Settings** > **Webhooks** > **Add webhook**.
    b. Specify any title for your Webhook that aptly describes it.
    c. Enter the URL of the Bamboo API to trigger a plan as follows:
    `<BAMBOO_URL>/rest/triggers/1.0/remote/changeDetection?planKey=<PLAN-KEY>&skipBranches=false`
    where `<BAMBOO_URL>` is the URL of your Bamboo instance and `<PLAN_KEY>` is the key identifying the plan you want to trigger.

    ✓ You can get the plan key from your browser's address bar when viewing the plan in Bamboo.

    The `skipBranches` parameter determines whether change detection is triggered by this event for every branch on the plan (`false`) or just for the specified plan key (`true`).
    d. Select your preferred trigger types.
    e. Save your changes.

Triggering a Bamboo build from Jira Automation

The purpose of this guide is to help you set up a webhook that will trigger a Bamboo build from Jira after a certain event occurs. For example, you may want to trigger a Bamboo build when a Jira issue changes its status.

To do so, you need to configure a Remote trigger for your plan in Bamboo, and create a new rule trigger in Jira Automation that will connect to the Remote trigger with a webhook.

Before you begin

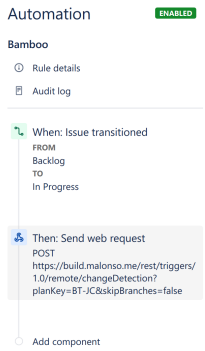
Make sure that your Bamboo instance is publicly available and has a valid certificate.

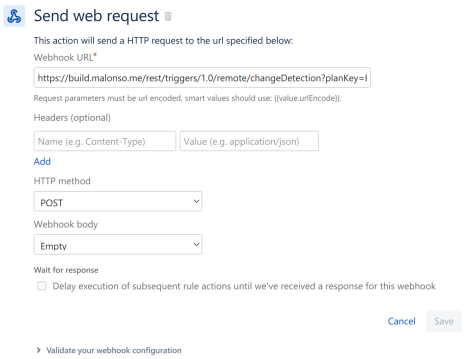
To configure a Remote trigger for your Bamboo plan:

1. From the Bamboo header select **Build** > **All build plans**.
2. Select the plan you want to set up a trigger for.
3. Select **Actions** > **Configure plan**, then select the **Triggers** tab.
4. Select **Add trigger** > **Remote trigger** and complete the form.
   Note that Bamboo requires you to add to the allowlist the IP addresses of the source of the request to trigger under the **Trigger IP addresses** field (comma separated for each IP). Select the IP addresses you want to add to the allowlist from the list of IP addresses for Atlassian cloud products.
5. Select **Save trigger**.

To create a new Jira Automation rule trigger:

1. In your Jira project, go to **Project settings** > **Automation** > **Create rule**.
2. Select your trigger type, then select **Save**.
3. Select the **New action** component type, and select **Send web request**.
4. Enter the URL of your Bamboo API: **`BAMBOO_URL`**`/rest/triggers/1.0/remote/changeDetection?planKey=`**`PLAN-KEY`**`&skipBranches=false`, where:
   - **`BAMBOO_URL`** is the URL of your Bamboo instance.
   - **`PLAN-KEY`** is the key of the plan you wish to trigger. The `PLAN-KEY` is visible in the URL when browsing the plan in Bamboo. E.g. https://bamboo.atlassian.com/browse/PLAN-KEY.
   - **skipBranches true / false** determines whether change detection will be triggered by this event for every branch on the plan (**false**), or just the plan key specified (**true**).
5. Set the **HTTP method** to POST.
6. Set the **Webhook body** option to Empty.
7. *(Optional)* Select the **Validate your webhook configuration** option to test your connection.
8. Select **Save**. Your trigger is ready!





Tag triggering

You can schedule Bamboo to run a build automatically whenever a selected tag appears in your repository.

Before you begin

Tag triggering is enabled by default in Bamboo. However, tag trigger type is available only if there's a repository added to your Bamboo instance.

You can use tag triggering in Bamboo with the following repository types:

- Bitbucket Cloud
- Bitbucket Server/ Stash
- GitHub repository
- Git

To schedule builds triggered by tags:

1. From the Bamboo header select **Build** > **All build plans**.
2. Locate the plan in the list and select the edit icon ( ) to display the plan's configuration pages.
3. Select the **Triggers** tab.
4. Select **Add Trigger**.
5. Select the **Tag trigger** type.
6. Define your tag name.
7. Enter the tag which will trigger a build:

   You can use regular expressions to filter tag names better.

8. Decide if you want to **Run build if the branch contains the matched tag** (default).
   When this option is enabled, your build will be automatically triggered only if a selected tag appears in a branch for which this trigger is set. If you clear this option, a build is going to be triggered by a selected tag regardless whether that tag appears on master or a branch.
9. Select **Save trigger**.

Your trigger is now created and a build will run automatically if a set tag appears in your repository.

To disable tag triggering in Bamboo:

If for any reasons tag triggering causes any problems on your Bamboo instance, you can disable it by setting the following system property:

```
atlassian.bamboo.tag.detection.disable=true
```

Using stages in a plan

Stages group (or map) jobs to individual steps within a plan's build process. For example, you may have an overall build process plan that comprises a compilation step, followed by several test steps, followed by a deployment step. You can create separate Bamboo stages to represent each of these steps.

A stage:

- By default has a single job but can be used to group multiple jobs.
- Processes its jobs in **parallel**, on **multiple** agents (where available).
- Must successfully complete all its jobs before the next stage in the plan can be processed.
- May produce artifacts that can be made available for use by a subsequent stage.

Each new plan created in Bamboo contains at least one stage (for the default job) and is known as the Default stage. Stages can only be configured by Bamboo administrators.

On this page:

Types of stages in Bamboo

Normal stage

The normal stage must successfully complete all its jobs before the next stage in the plan can be run. If a normal stage is not run successfully, the following normal stage can't be run.

Manual stage

A user has to trigger this type of stage manually to run it.

Any stage in a plan can be configured to be a manual stage. If you run a plan with manual stages, Bamboo will pause the execution of the plan every time it reaches a manual stage. The plan build will only continue once a user has manually triggered the stage.

- A manual stage can only be triggered if the previous stage has been completed successfully.
- Manual stages must be executed in the order that they are configured in the plan. You can't skip a manual stage.
- Manual stages will be displayed in the Plan navigator with either this icon  (not due to be triggered) or this icon  (pending execution).
- You need Build permission on the plan to run a manual stage.
- Not even the final stages will be run after an untriggered manual stage.
- A manual stage can also be a final stage. In this case, the build will stop at the manual stage and wait for a user to execute it manually. You can start this particular final-manual stage whether a build is successful or not.

Final stage

The final stage is run regardless of whether previously run stages were successful or not.

The final stages can be useful if you want to run cleanups or aggregate results regardless of whether a build succeeds or not. **Any stage** in a plan **can be** configured to be a **final stage**.

A final stage can also be a manual one. In this case, the build will stop at the manual stage and wait for a user to execute it manually. You can start this particular final-manual stage whether a build is successful or not.

If a final stage follows a manual stage, the final stage will not run until the manual stage is run. If the build fails and the manual stage cannot be run, you can't run the final stage.

Find, create, edit, and delete a stage

1. From the Bamboo header, select **Build** > **All build plans**.
2. Select the name of the plan you want to edit.
3. Select **Actions** > **Configure plan**.
4. Select the **Stages** tab.

1. Go to the stages for your plan.
2. Select **Create stage**.
3. Complete the form and select **Create**.
4. *(optional)* You may want to do one or more of the following with your new stage:
    - Order your new stage in the list of stages, by dragging and dropping it.
    - Add a new job to your stage.
    - Move a job from another stage to your new stage by dragging and dropping the job.

⚠ You may break artifact dependencies by moving stages, or by moving jobs between stages. Bamboo will warn you if a dependency will be broken by moving a stage or a job.

1. Navigate to the stages for the plan, as described above.
2. Edit the stage as required:
    - To edit the name and description of the stage or configure whether it is a manual stage, select the cogwheel icon and select **Configure stage**.
    - To move the stage, drag and drop the stage to the desired place in the plan.

⚠ You may break artifact dependencies by moving stages. Bamboo will warn you if a dependency will be broken by moving a stage.

⚠
- Deleting a stage will delete all job configurations, artifacts, logs, and results related to the stage. These cannot be recovered after the stage is deleted.
- You may break artifact dependencies by deleting a stage.

1. Go to the stages for the plan, as described above.
2. Select the cogwheel icon for the relevant stage and select **Delete stage**.
3. Select **Confirm** to delete the stage. Note that a deleted stage can't be recovered.

Jobs and tasks

The following pages contain information about configuring jobs and tasks for your Bamboo plans. If you are looking for information about Bamboo builds, please see Working with builds.

- Creating a job
- Configuring jobs
- Disabling or deleting a job
- Configuring tasks

Jobs

A Bamboo *job* is a single build unit within a plan. One or more jobs can be organized into one or more stages. The jobs in a stage can all be run at the same time, if enough Bamboo agents are available. A job is made up of one or more tasks.

A job:

- Processes a series of one or more tasks that are run **sequentially** on the **same** agent.
- Controls the order in which tasks are performed.
- Collects the requirements of individual tasks in the job, so that these requirements can be matched with agent capabilities.
- Defines the artifacts that the build will produce.
- Can only use artifacts produced in a previous stage.
- Specifies any labels with which the build result or build artifacts will be tagged.

Each new plan created in Bamboo contains at least one job known as the Default job.

Projects and plans can only be configured by Bamboo administrators (see Creating a plan).

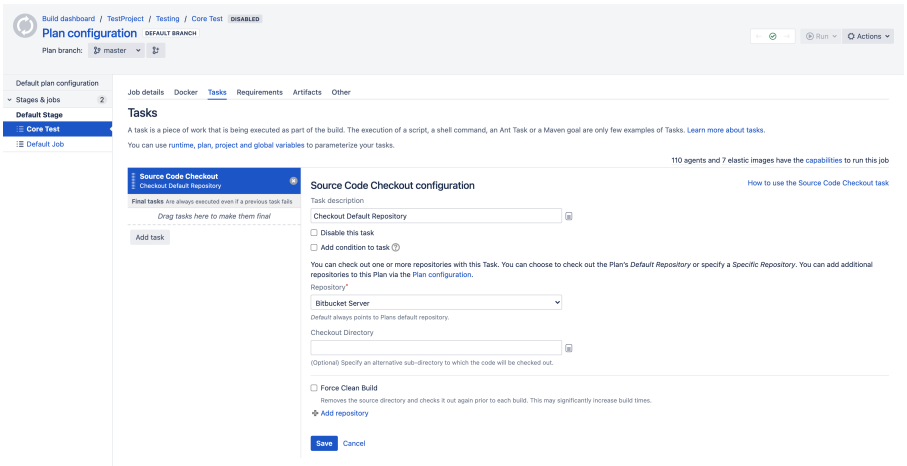
Tasks

A task:

- Is a small discrete unit of work, such as source code checkout, executing a Maven goal, running a script, or parsing test results.
- Is run sequentially within a job on a Bamboo working directory.

Tasks may make use of an executable if required. Tasks are configured within the scope of a job. A job can be configured to execute a number of tasks, on the same working directory. For example, before executing a Maven goal, the user could substitute specific files within the working directory, substitute version numbers, check out source repositories, or execute a script.

Final tasks for a job are always executed, even if previous tasks in the job failed.



Creating a job

This page describes how to create a Bamboo job in a stage of a plan.

- You can either create a new job, or clone an existing job.
- You must have the Admin or Create plan global permission to create jobs.
- A job allows you to collect together a number of tasks that you want to be run sequentially on the same agent.

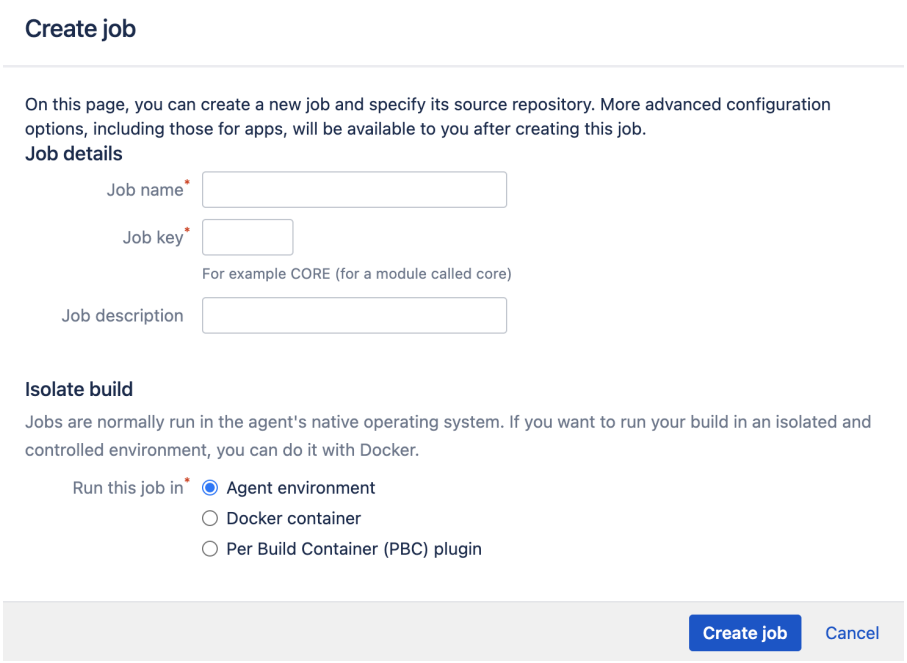
Related pages:

- Configuring plans
- Using stages in a plan
- Configuring jobs
- Disabling or deleting a job

To create a new job for a plan:

1. From the Bamboo header select **Build** > **All build plans**.
2. Select the plan you want to create a new job for.
3. Select **Actions** > **Configure plan**.
4. Select the **Stages** tab.
5. Select **Add job** in the stage where you want the new job.
6. Select either **Create a new job** or **Clone an existing job**.
7. If cloning a job, complete the from:
    - **Plan to clone from** — Select the plan containing the job you wish to clone. Plans are grouped by project in the list.
      Only plans for which you have the Clone and/or Admin plan permission are shown.
    - **Job to clone** — Select the job you wish to clone from your selected plan. Jobs are grouped by stage in the list.
8. Provide your job details.
9. Select **Create job**.

If you wish to configure tasks for the job, such as configuring a Repository checkout, see Configuring jobs.



Configuring jobs

A Bamboo *job* is a single build unit within a plan. One or more jobs can be organized into one or more stages. The jobs in a stage can all be run at the same time if enough Bamboo agents are available. A job allows you to collect together a number of tasks that you want to be run sequentially on the same agent.

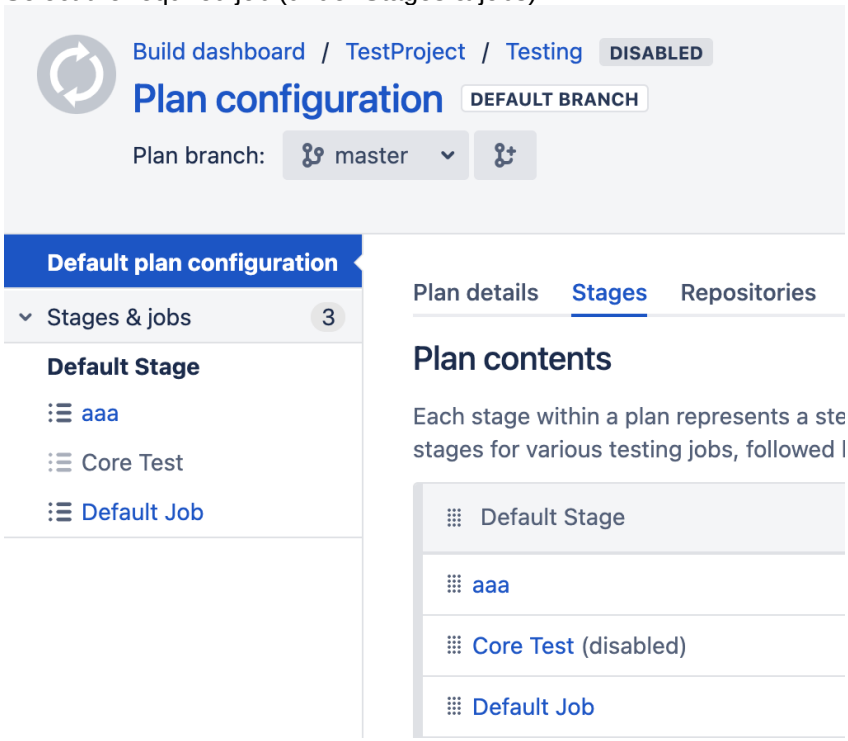
You must have the Admin or Create plan global permission to configure jobs.

Related pages:

- Creating a job
- Disabling or deleting a job
- Viewing a job's Maven dependencies

To configure an existing job in a Bamboo plan:

1. From the Bamboo header select **Build** > **All build plans**.
2. Select the edit icon ( ) for the plan you want to edit.
3. Select the required job (under Stages & jobs):



4. Select a selected tab to begin editing that aspect of your job:
    - **Job details** — Note that Job key is not editable.
    - **Docker** - see Docker runner.
    - **Tasks** — see Configuring tasks, including Repository checkout tasks and builder tasks.
    - **Requirements** — see Configuring a job's requirements.
    - **Artifacts** — see Configuring a job's build artifacts.
    - **Other** — see Configuring miscellaneous settings for a job and Configuring automatic labeling of build results.

Configuring a job's requirements

This page describes how to configure the requirements of a job.

A *requirement* is specified in a job or a task. A requirement specifies a capability that an agent must have for it to build that job or task. A job inherits all of the requirements specified in its tasks.

Together, capabilities and requirements control which agents can execute builds for particular jobs. Each job can only be built by agents whose capabilities match the job's requirements.

There are four types of capabilities in Bamboo that can be specified by job and task requirements:

- **Executable capabilities** — Define external programs that can be called by Bamboo, for example Ant, Maven, MSBuild or PHPUnit. See Defining a new executable capability.
- **JDK capabilities** — Define the JDK versions to be used by the job or task. See Defining a new JDK capability.
- **Version control capabilities** — Specify the VCS client application that Bamboo should use to check out source code. See Defining a new version control capability.
- **Custom capabilities** — Can be used to control which jobs will be built by a particular agent. For example, if the builds for a particular job should only run in a Windows environment, you could create a custom capability of '`operating.system=WindowsXP`' for the appropriate agent(s), and specify it as a requirement for this job. See Defining a new custom capability.

Before you can specify a requirement in your job, you must first define that capability in your Bamboo system.

On this page:

Related pages:

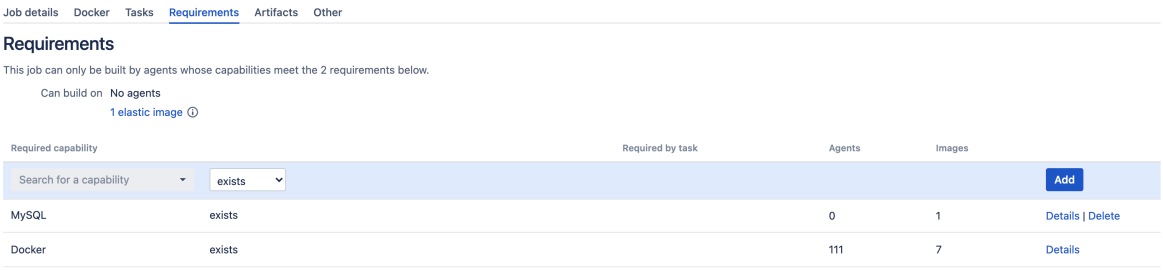
- Configuring jobs
- Configuring tasks
- Viewing a capability's agents and jobs

Specifying extra requirements for a job

A job will inherit the requirements of its tasks by default. However, you can specify extra requirements for a job, in addition to its task requirements.

To specify extra requirements for a job:

1. Navigate to the desired job's configuration pages, as described on Configuring jobs.
2. Select the **Requirements** tab (see screenshot below). This page shows a list of all the job's current requirements and the number of Agents and Images (i.e. agents/elastic images which meet the job's requirements and can run a build for this job). See Viewing current capable agents below for more information.
3. **Search for a capability** and select a requirement from the list.
4. Select the value for the requirement from the list:
    - **exists** — this job can be built by any agent that has a capability with the same key.
    - **equals** — this job can be built by any agent that has the capability with the same key and value.
    - **matches** — this job can be built by any agent that has a capability with the same key, and the value matches the regular expression. For more information about regular expressions, see Oracle's tutorial on regular expressions.
5. Select **Add**. The numbers of Agents and Images will be updated, as the plan can now only be built by agents with capabilities that meet the new custom requirement you have specified.



Viewing current capable agents

To view details about agents or elastic images that are currently able to build your job:

1. On the job's **Requirements** tab (described above), select the name of the requirement in the table (e.g. MySQL).
2. The summary page for the capability will be displayed, showing the agents and images that have the capability. See Viewing a capability's agents and jobs for more information.

Configuring a job's build artifacts

Artifacts are files created by a *job build* (e.g. JAR files). Artifact definitions are used to specify which artifacts to keep from a build and are configured for individual jobs.

See Sharing artifacts.

This page describes how to define the artifacts that should be kept from a job's build. For example, you may wish to keep reports, websites or files (e.g. JAR files) generated by a job build.

You can also configure artifact sharing between jobs in a plan. For example, you may want to run acceptance tests on a build, and then share the WAR from one job to another, without rebuilding the WAR each time. See Sharing artifacts.

Atlassian blog posts:

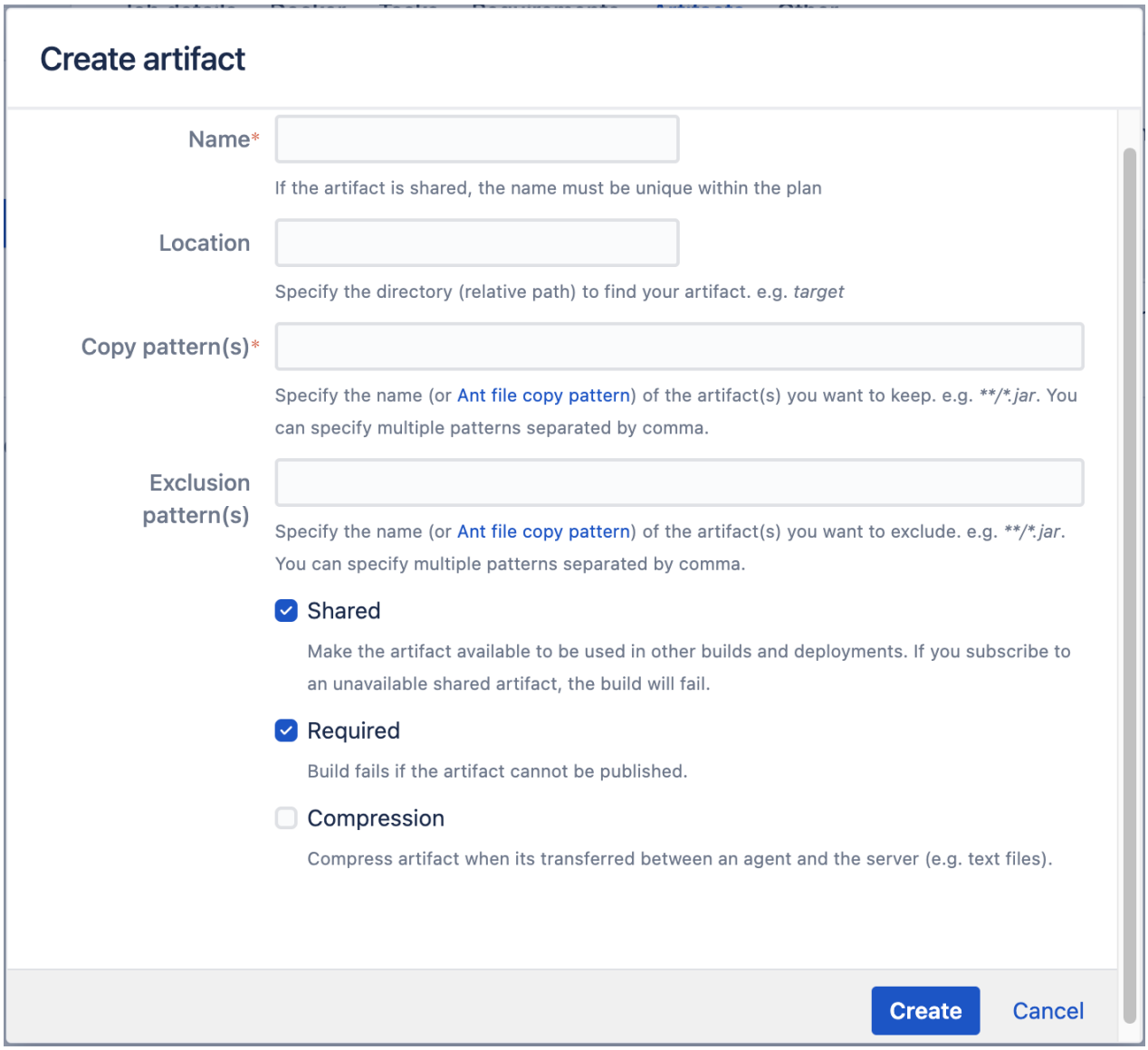
- Artifact passing for agile teams

Define the artifacts to keep for a job

You can specify which artifacts to keep by setting up an artifact definition for the job. The artifacts will be available after each build of a job.

To set up a new artifact definition for a job:

1. Navigate to the desired job, as described on Configuring jobs.
2. Select the **Artifacts** tab, and then **Create artifact**.
3. Complete the fields on the screen (see screenshot below) and select **Create**. For example, if you want to keep the latest version of a JAR you have built, you could specify **Copy Pattern** to be '****/***.jar' and the **Location** to be target.
   Note:
   - The location is relative to the build directory. Do not use the absolute path to refer to the location.
   - The copy pattern is relative to the location specified.
   - Asterisks are not supported for **Location**. For this field, provide the folder name where the file would be located.
   - If you want to share artifacts with other jobs in the plan, you will need to mark the artifacts as shared. See Sharing artifacts.
   - You can disable file compression that happens when the artifact is being transferred between the agent and the server.



Notes

Artifacts are copied to a subdirectory (`/JOB_KEY/download-data/`) under your Build directory folder (see Locating important directories and files). Artifacts which you define in the plan are listed in each build result as *artifacts* (see Viewing a build's artifacts).

Configuring miscellaneous settings for a job

For each job of a plan, you can optionally specify a number of miscellaneous settings including:

- Build hanging detection
- NCover output
- Clover code coverage

To configure the miscellaneous settings for a job:

1. Navigate to the desired job, as described in Configuring jobs.
2. Edit the desired settings as follows: **Override default hanging build detection**

    Override the default build hanging detection settings. These settings determine when a build hung event is thrown (e.g. you can configure your notifications to trigger from this event).

    Build Time Multiplier — Calculate the 'Expected Build Time' for the build (i.e. 'Expected Build Time' = 'Build Time Multiplier' multiplied by 'Average Build Time'). 'Average Build Time' is calculated by using an average of previous build times.

    Log Quiet Time — The amount of time since Bamboo last recorded an entry in the build log for a build. The 'Expected Build Time' and 'Log Quiet Time' must *both* be exceeded for Bamboo to throw the build hung event.

    Build Queue Timeout — The amount of time that a build will wait in a build queue before an timeout event is thrown. Setting this value will override the global build queue timeout setting (see Configuring the build queue timeout event).

    NCover output will be produced
    Do not select this option. NCover is a code coverage tool that supports .NET projects.

    Use Clover to collect Code Coverage for this build
    Select this check box if:

    - This job will be building a Java or Groovy-based project using a builder such as Ant, Maven or Grails.
    - You are running Atlassian Clover and want to collect code coverage data to view from within Bamboo (see Viewing the Clover code-coverage for a build).

    Automatically integrate Clover into this build

    - **Generate a Clover Historical Report** — shows the current coverage results compared with previous Clover code coverage reports.
    - **Generate a JSON report** — gives the Clover results in a format ready for embedding into applications or external report views.

    You will also need to insert a Clover license (evaluation licenses are available) into the field provided. See Enabling Clover for Bamboo.

    Clover is already integrated into this build and a clover.xml file will be produced
    Use this option when you already have Clover-for-Ant or Clover-for-Maven configured to generate a report.

    - **Clover XML Location** — specify where the Clover XML report is generated. Include the name of the directory, including path, *relative* to your job build's root directory, for example: `target/site/clover/clover.xml`
3. Select **Save**.

Configuring automatic labeling of job build results

For each job of a plan, you can (as an option) specify a label that can be applied to the job's build results automatically after each build of that job.

Automatic labeling of job builds is built into Bamboo itself. There are a number of third-party plugin modules available that can provide additional post actions (e.g. the Pre-Post Build Command plugin). You can also write your own plugins to provide additional post actions for a job. See the Bamboo Plugin Guide for further details.

Labels can also be applied to build results manually by Bamboo users.

On this page:

Related pages:

- Configuring jobs

Specifying labels for a job's build results

To specify labels for a job's build results:

1. Navigate to a job's configuration pages, as described on Configuring jobs.
2. Select the **Other** tab.
3. Using **Regex Pattern**, you can either:
    - Specify a regular expression to match content in the log files of this job's builds. Labels will be applied to a build of this job if this regular expression matches content in the build's log files (see the examples below).
      For more information about regular expressions, please refer to the Java documentation on regular expression constructs.
    - Leave this field blank to label *every* build of this job.
4. In the **Labels** field, type the word (or multiple words, separated by commas and/or spaces) with which the plan's build results are to be labeled.
5. Select **Save**.

Regex examples

See http://www.regular-expressions.info/reference.html for examples.

A simple regex example:

```
'There are \d+ results'
```

In the above regex, '\d+' represents *any number with one or more digits*. ('\d' means 'any digit', and '+' means 'one or more times'. When combined, they mean 'any sequence of one or more digits'.) Therefore, positive matches would include:

- 'There are 0 results'
- 'There are 123 results'

A regex example with multiple labels:
You can use "capturing groups" with Bamboo 1.2.1 or later to create different labels for different purposes.

For example, the following settings will label your builds with PERFORMANCE_IMPROVED if "PERFORMANCE_IMPROVED" appears in the build log, and PERFORMANCE_DETERIORATED if "PERFORMANCE_DETERIORATED" appears in the build log. If both strings appear in a log, then both labels are applied to the build.

- Enter the following into the **Regex Pattern** field:

```
(PERFORMANCE_IMPROVED|PERFORMANCE_DETERIORATED)
```

- Enter the following into the **Labels** field:

```
\1
```

Viewing a job's Maven dependencies

If you have configured a job to use a Maven builder (Maven 2 or later), you can choose to have dependencies generated from your Maven pom.xml (see documentation for setting up Maven as a builder for instructions). After the initial build, Maven will parse the pom.xml file, determine the artifacts produced by the build and generate the dependencies. You can view these dependencies in two places:

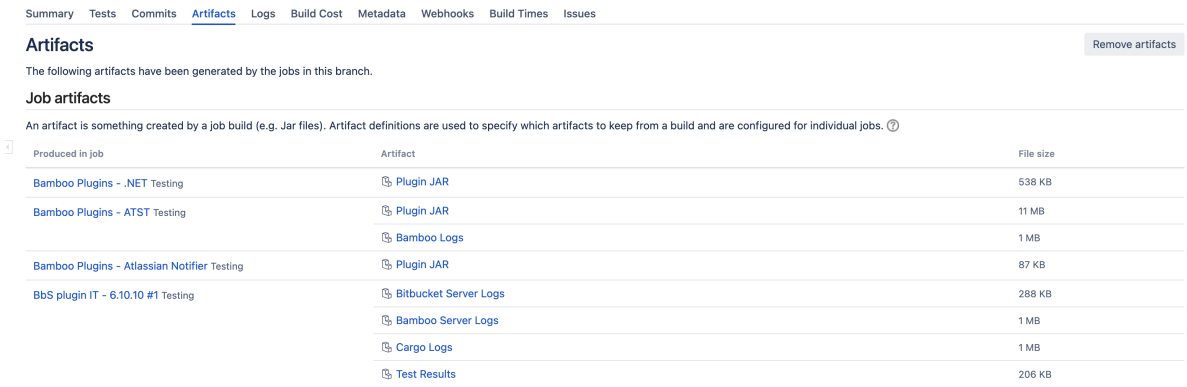
- On the **Dependencies** tab when configuring your plan, as described in Setting up plan build dependencies.
- On the **Artifacts** tab when viewing a job's build result, as described below.

Before you begin:

- The Maven dependencies for a build will only become known to Bamboo after a build. If you cannot see the Maven dependencies for a build, try running it first without triggering any other dependencies. See Modifying multiple plans in bulk if you want to run multiple builds.

To view the Maven dependencies for a job's build result:

1. Navigate to the desired job, as described on Configuring jobs.
2. Select the desired build result number in the Recent history of the Job summary.
3. Select the **Artifacts** tab for the build results. The produced Maven artifacts and Maven artifact dependencies will be listed.



Disabling or deleting a job

Bamboo allows you to disable or delete jobs that you don't want to be built.

Disabling a job

Disabling a job prevents Bamboo from building that particular job within a plan, allowing the rest of the plan's jobs to be built. You can re-enable the job, if you want to build it again.
For example, if a job's latest build is broken and cannot be fixed quickly, you may want to disable it temporarily to stop the job from being built.

Deleting a job

Deleting a job deletes everything related to that job, including the job's configuration, build results, artifacts, labels and comments. However, everything else related to the job's plan, and this plan's other jobs, is retained by Bamboo.

You will need to recreate a new job from scratch, if you want to build it again.
For example, if a job is no longer relevant, you may want to delete it.

Note that:

- The Admin global permission is required to delete a job.
- A job that is currently being built cannot be deleted. If you need to delete such a job, stop the plan's build first. Refer to Stopping an active build for more information.
- If you need to keep a permanent record of your job's build results, see Exporting data for backup.

Related pages:

- Creating a job
- Configuring jobs
- Disabling or deleting a plan

Disable or delete a job

1. Navigate to the job configuration, as described on Configuring jobs.
2. Select either **Actions** > **Disable job** or **Actions** > **Delete job**.

Deleting a job's current working files

If you only run a single Bamboo server (i.e. with no remote or elastic agents) and:

- you need to ensure that a plan's job cleanly checks out its source code when Bamboo next executes a build of that plan
- and you *do not* use the **Force clean build** option when linking to the source repository for a job

then you can simply delete the current working files for that job to ensure its source code is cleanly checked out.

You need the Admin global permission or the Admin plan permission to delete current working files.

To delete a job's current working files:

1. Navigate to the job configuration, as described on Configuring jobs.
2. Select the **Other** tab.
3. Select **Clean working directory after each build**.

Configuring tasks

A task:

- Is a small discrete unit of work, such as source code checkout, executing a Maven goal, running a script, or parsing test results.
- Is run sequentially within a job on a Bamboo working directory.

Tasks may make use of an executable if required. Tasks are configured within the scope of a job. A job can be configured to execute a number of tasks, on the same working directory. For example, before executing a Maven goal, the user could substitute specific files within the working directory, substitute version numbers, check out source repositories, or execute a script.

Final tasks for a job are always executed, even if previous tasks in the job failed.

Create a task for a job

When creating a new job or configuring an existing one, you need to specify the tasks that will execute the job's builds. You must specify an executable for each task. If you specify an Ant, Grails or Maven executable, you will also need to choose a JDK.

When creating a new plan, you can configure the tasks for the plan's default job.

On this page:

Related pages:

- Checking out code
- Configuring a builder task
- Configuring a test task
- Configuring jobs
- Creating a plan
- Pattern matching reference

To create a task for a job:

1. Navigate to the tasks configuration for a job. Do this by:
    - selecting the **Tasks** tab when configuring an existing job, or
    - creating a new plan (you will be configuring tasks for the default job).
2. Select **Add task**.
3. Select the task type.
4. Complete the following fields that are common to all task types:

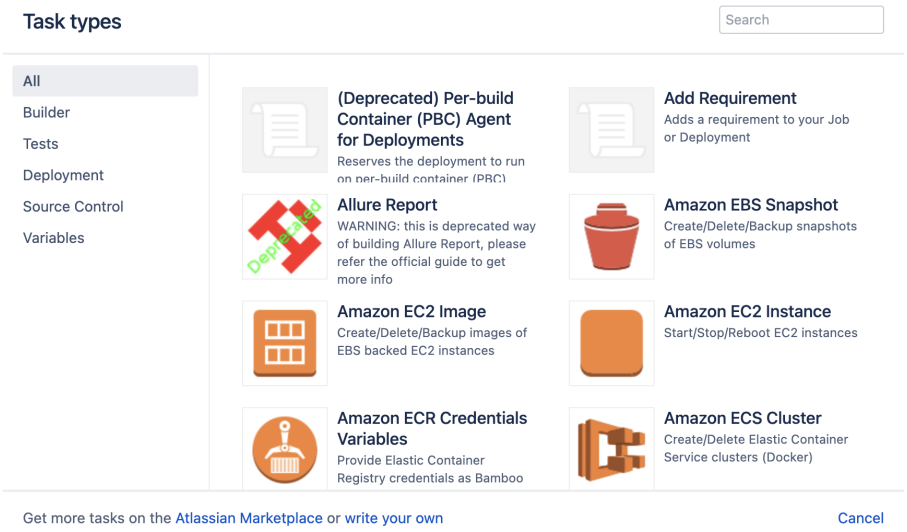
    Task description
    Enter a description of the task, for display in Bamboo.

    Disable this task
    Check, or clear, to selectively run this task.

    Add condition to task
    Make task run only when a certain condition is met.
5. Complete the remaining fields, which are specific for individual task types. See the following pages for further details:
    - Checking out code
    - Configuring a builder task
    - Configuring a test task

- Configuring a variables task
- Configuring a deployment task
- Pattern matching reference
- Configuring the Docker task in Bamboo
- Configuring a Source Control task
- Configuring Build warnings parser task

6. Select **Save**.



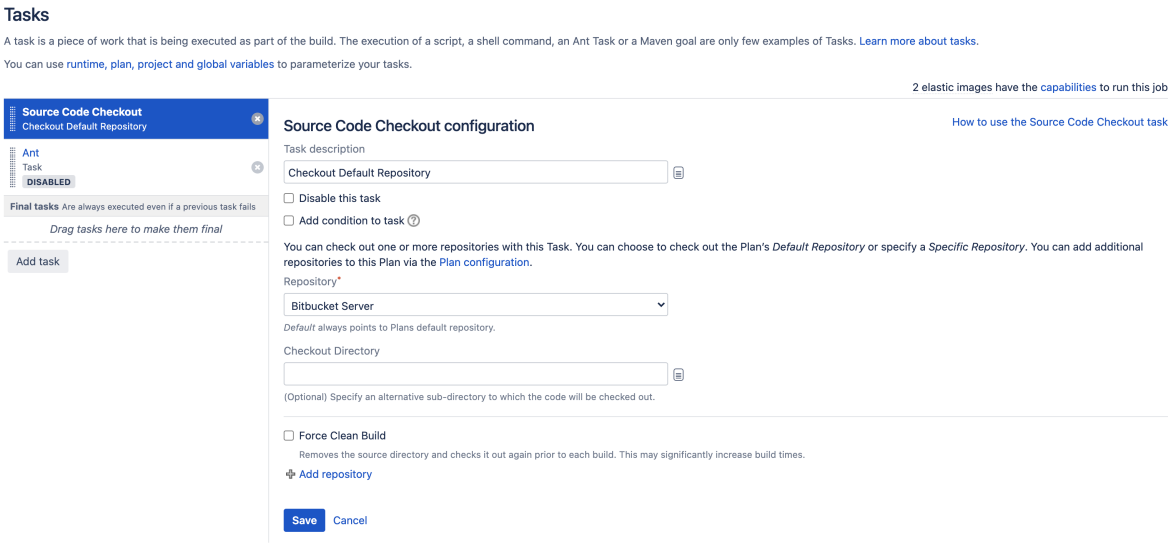
Order the tasks in a job

Tasks can be designated as *build tasks* or *final tasks* in a job:

- Build tasks will run sequentially in the order specified in the job. If a Build task fails, all subsequent tests will not be executed.
- Final tasks will run sequentially, once the build tasks have completed. Final tasks will always be executed, regardless of whether any Build tasks or other Final tasks fail. Final tasks will be executed even if you stopped the build manually.

To order the tasks for a job:

1. Navigate to the tasks for the desired job.
2. Drag and drop the tasks into the desired order in the table on the left. If you want to change a Build task to a Final task or vice versa, drag and drop it under the desired header in the table. Your changes will be saved immediately.



Notes

- **Adding new executables** — At least one executable is configured automatically after installing Bamboo. You can add more executables of different types as described in Configuring a new executable.
- **Adding new JDKs** — At least one JDK is configured automatically after installing Bamboo. You can add more JDKs as described in Defining a new JDK capability.
- **About the Compatibility task** — The Compatibility task is created by Bamboo when upgrading from Bamboo 3.0 or earlier and Bamboo cannot match a builder to a task. This may occur if you are using a builder enabled by a custom plugin.

Checking out code

You use the Source Code Checkout task to check out a repository for use by just one job. By default, repositories are checked out to the Bamboo working directory.

Using Source Code Checkout task you can also:

- Check out repositories to a custom directory path in the working directory.
- Specify multiple checkouts that occur at different stages of the build. (Simply add another Source Code Checkout task to a job at any point in the plan.)

For information about specifying a repository for use by all the plan's jobs, or by all plans, see Linking to code repositories.

To configure a new Source Code Checkout task:

1. Navigate to the job that should perform the task.
2. Select the **Tasks** tab, and select an existing Source Code Checkout task in the tasks list, or add a new one using the **Add task** button.
3. Configure the task:

   Task description
   Enter a description of the task, for display in Bamboo.

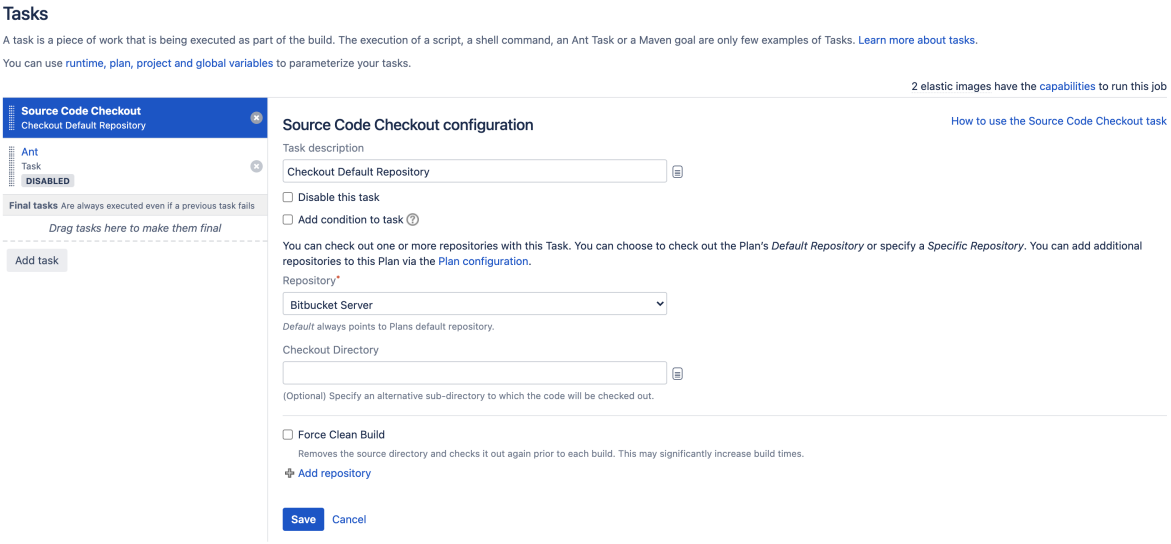
   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Repository
   Select the desired repository. If you wish to add different types of repositories, they must have been previously defined on the plan's Source repositories tab. See Linking to source code repositories for a list of supported SCMs.

   Checkout Directory
   The location to which the contents of the selected repository will be checked out to when the task executes.

   Force Clean Build
   Deletes the previously checked out directory and checks it out again prior to the next build. This may significantly increase build times.
4. Select **Add repository** at the bottom of the screen to check out another repository using this task.
5. Select **Save**.



Notes

- A number of source repositories are supported out of the box, as described on the Linking to code repositories page.
- If you need to use a type of repository that is not supported, a number of third-party Source Repository plugin modules are available (e.g. ClearCase plugin). You can also write a Source Repository Module plugin to enable Bamboo to connect to your repository.
- If Source Code Checkout task is used in Deployment Environment, it will only checkout the latest revision instead of the revision used in the related build result.

  BAM-13279 - Deployment environment checkout task should use same revision as build result
  GATHERING INTEREST

Configuring a builder task

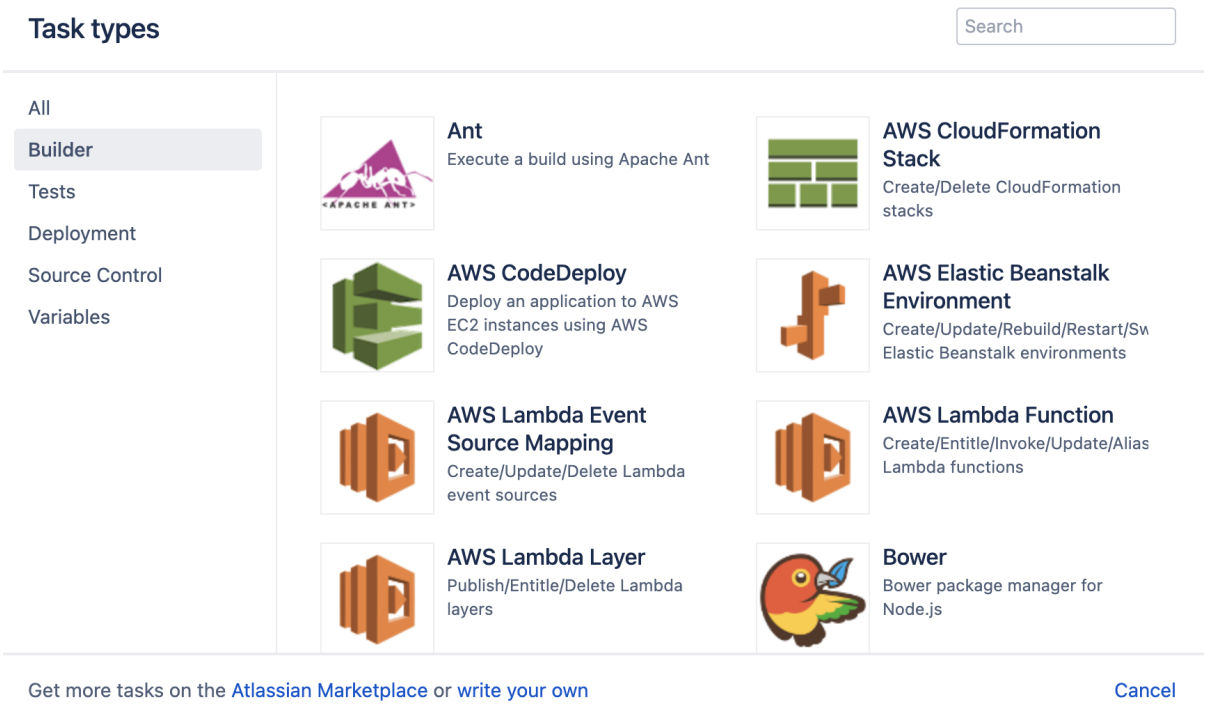
A builder task allows you to connect your Bamboo plan (or job) to a build tool such as Ant, Maven, or MSBuild. The build tool uses its existing configuration when the plan (or job) is built.

You can connect Bamboo to the following build tools:

- Ant
- Custom command executable
- Fastlane
- Grails
- Maven
- MSBuild
- NAnt
- Script
- Visual Studio
- Xcode

Related pages:

- Configuring tasks
- Configuring a test task
- Checking out code



Ant

This page describes how to configure a Bamboo task to use Ant.

See Configuring a builder task for an overview of Bamboo builder tasks.

Related pages:

- Configuring tasks
- Configuring jobs
- Pattern matching reference

To configure an Ant task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing Ant task, or select **Add task** > **Ant** to create a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable

   The Ant executable that is available to perform the task.
   The executable that you select will become one of the task's (and so, the job's) requirements.

You can add other executables, if required.

Build file
The name of your existing build file (e.g. `build.xml`).
You can include variables (see Using Global or Build-specific Variables).

Target
The Ant target that you want this Bamboo task to execute (e.g. `test`).
You can use '`-D`' to define one or more JVM parameters (e.g.: `-Djava.awt.headless="true`"). You must use double quotes around the parameter value; single quotes are considered as part of the actual value.
Multiple Ant targets can be specified with a space-delimited list.
You can also include variables (see Using Global or Build-specific Variables).

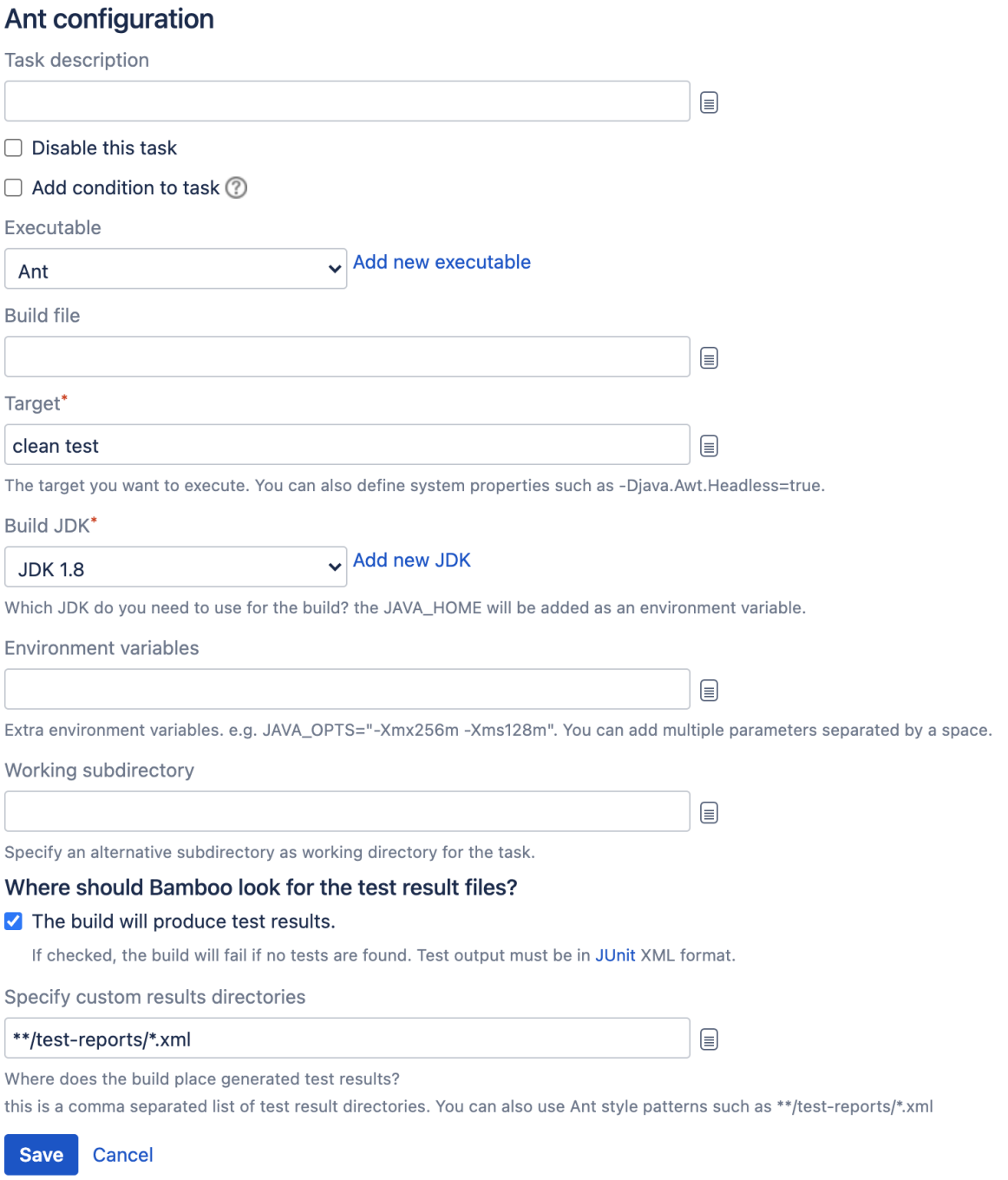
Build JDK
The JDKs that are available to perform the task. The JDK that you select will become one of the task's (and so, the job's) requirements.
You can add other JDKs, if required.

Environment variables (Optional)
Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Using Global or Build-specific Variables).
Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

Working subdirectory (Optional)
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

The build will produce test results
Select to specify the directory, relative to the root directory, where test results will be created. You can use Ant-style patterns such as `**/test-reports/*.xml`. Bamboo requires test results to be in JUnit XML format.
⚠️ For jobs that use CVS, the root directory is `<bamboo-home>/xml-data/build-dir/JOB_KEY/<cvs-module>.`

4. Select **Save**.



Custom command executable

This page describes how to configure a Bamboo task that uses a command (e.g. Bash) executable.

See Configuring a builder task for an overview of Bamboo builder tasks.

Related pages:

- Configuring tasks
- Configuring jobs

To configure a command task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing Command task, or select **Add task** > **Command** to create a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable
   The command executable that is available to perform the task (e.g. Bash). The executable that you select will become one of the task's (and so, the job's) requirements.

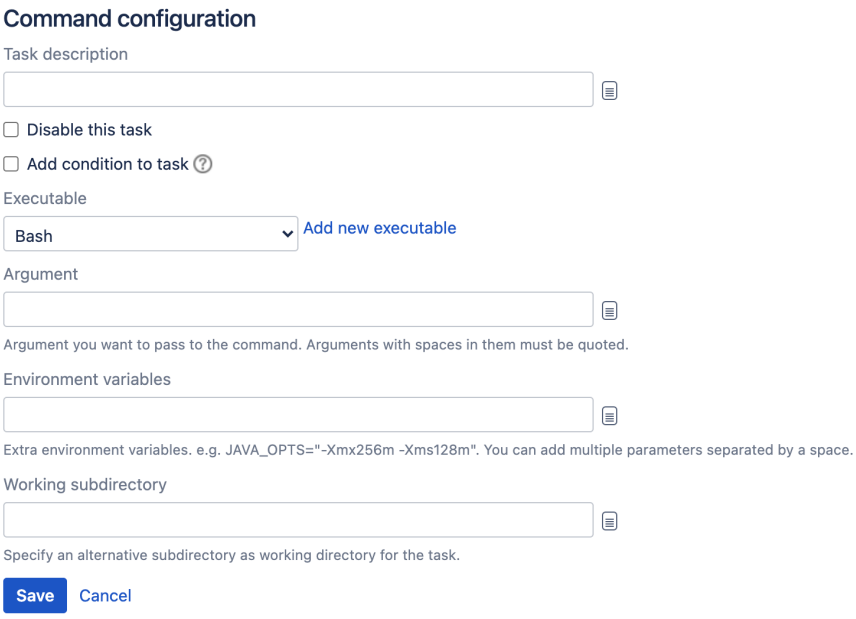
   You can add other executables, if required.

   Argument (Optional)
   The relevant argument to pass to the command. Note that arguments which contain spaces must be quoted. You can include variables (see Bamboo variables).

   Environment variables (Optional)
   Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Using global, plan or build-specific variables).
   Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

   Working subdirectory (Optional)
   An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

4. Select **Save**.



Fastlane

This page describes how to configure a Bamboo task to use Fastlane.

To configure a Fastlane task:

1. Go to the **Tasks** configuration tab for the job.
2. Select the **Add task** button.
   From the list of task types, select **Fastlane**.
3. Provide the Fastlane settings:

Field	Description
Tasks description	A description of the task, which is displayed in Bamboo.
Disable this task	Check, or clear, to selectively run this task.
Add condition to task	Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
Executable	The executable that is available to perform the task. The executable that you select will become one of the task's (and so, the job's) requirements.
Lane	The lane you want to execute. This field also allows you to define Fastlane properties such as param:paramValue.
Environment variables	Extra parameter variables. You can define multiple variables.
Working subdirectory	A sub-directory which can be used as an alternative for the task.

4. Select **Save**.

Test result parsing:

Bamboo supports test report in the JUnit XML format. To allow Bamboo to recognise tests from the Fastlane process you must:

1. Configure the Fastlane Scan to produce test output in the JUnit format:
   a. Create ScanFile in your Fastlane directory with the following content:

```
output_types "junit"
```

2. Add the JUnit Parser task to parse the results and point it to the Fastlane test output directory.

Grails

This page describes how to configure a Bamboo Grails task.

Bamboo supports Grails versions 1.2.x, 1.3.x, and 2.x.

Related pages:

- Configuring tasks
- Configuring jobs
- Defining a new JDK capability

To configure a Grails task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing Grails task, or select **Add task** > **Grails** to create a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable
   The Grails executable that is available to perform the task. The executable that you select will become one of the task's (and so, the job's) requirements.

   You can add other executables, if required.

   Grails commands
   The Grails commands that you want Bamboo to execute. See the Grails Command Line Reference documentation for more details on Grails commands.

   - You can use `'-D'` to define one or more JVM parameters, e.g.: `-Djava.awt.headless=true` will pass the parameter 'java.awt.headless' with a value of 'true'.
   - You can include variables (see Bamboo variables).

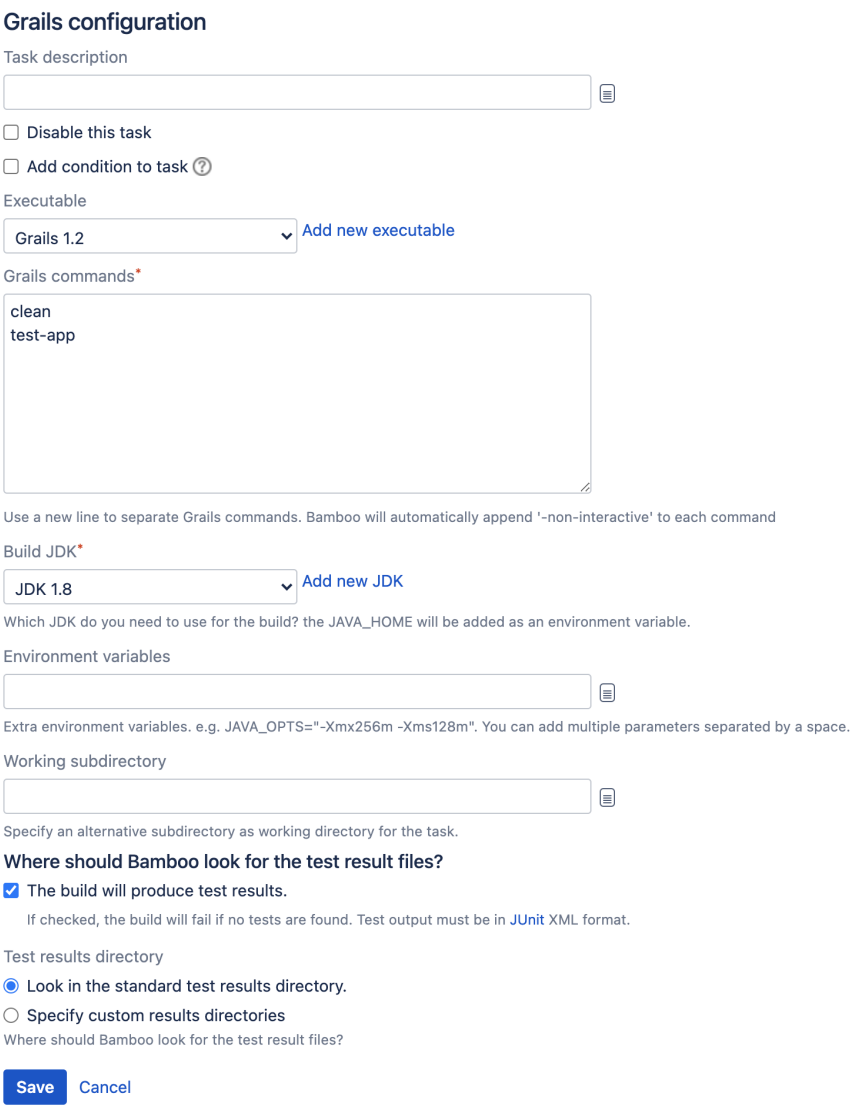
Build JDK
The JDKs that are available to perform the task. The JDK that you select will become one of the task's (and so, the job's) requirements.
You can add other JDKs, if required.

Environment variables (Optional)
Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Using global, plan or build-specific variables).
Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

Working subdirectory
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

The build will produce test results
Choose one of the following: **Look in the standard test results directory** – Bamboo looks in the standard directory for the test results. Use this unless you've customized your test runner to output the results to a different location. **Specify custom results directories** -- Specify the custom directory, relative to the root directory, where test results will be created. You can use Ant-style patterns such as `**/test-reports/*.xml`. Bamboo requires test results to be in JUnit XML format. ⚠️ For jobs that use CVS, the root directory is `<bamboo-home>/xml-data/build-dir/JOB_KEY/<cvs-module>`.

4. Select **Save**.



Maven

This page describes how to configure a Bamboo task to use a Maven executable. Apache Maven is a tool used for building and managing Java-based projects.

Related pages:

- Configuring tasks
- Configuring jobs
- Viewing a job's Maven dependencies
- Defining a new JDK capability

Atlassian blogs:

- Forgetful Maven Users, Rejoice! A new Bamboo task “releases” you from worry.

To configure a Maven task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing Maven task, or select **Add task**, and then a Maven option (e.g. **Maven 3.x**) to create a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable
   The Maven executable that is available to perform the task.
   The executable that you select will become one of the task's (and so, the job's) requirements.

   You can add other executables, if required.

   Goal
   The Maven goal that Bamboo will execute.

   - You can use '`-D`' to define one or more JVM parameters. For example, `-Djava.awt.headless=true` will pass the parameter 'java.awt.headless' with a value of 'true'.
   - Multiple maven goals can be specified, separated spaces.
   - You can include variables (see Using Global or Build-specific Variables).

Build JDK
The JDKs that are available to perform the task. The JDK that you select will become one of the task's (and so, the job's) requirements.
You can add other JDKs, if required.

Environment variables (Optional)
Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Using Global or Build-specific Variables).
Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g MAVEN_OPTS="-Xms200m -Xmx700m").

Working subdirectory (Optional)
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

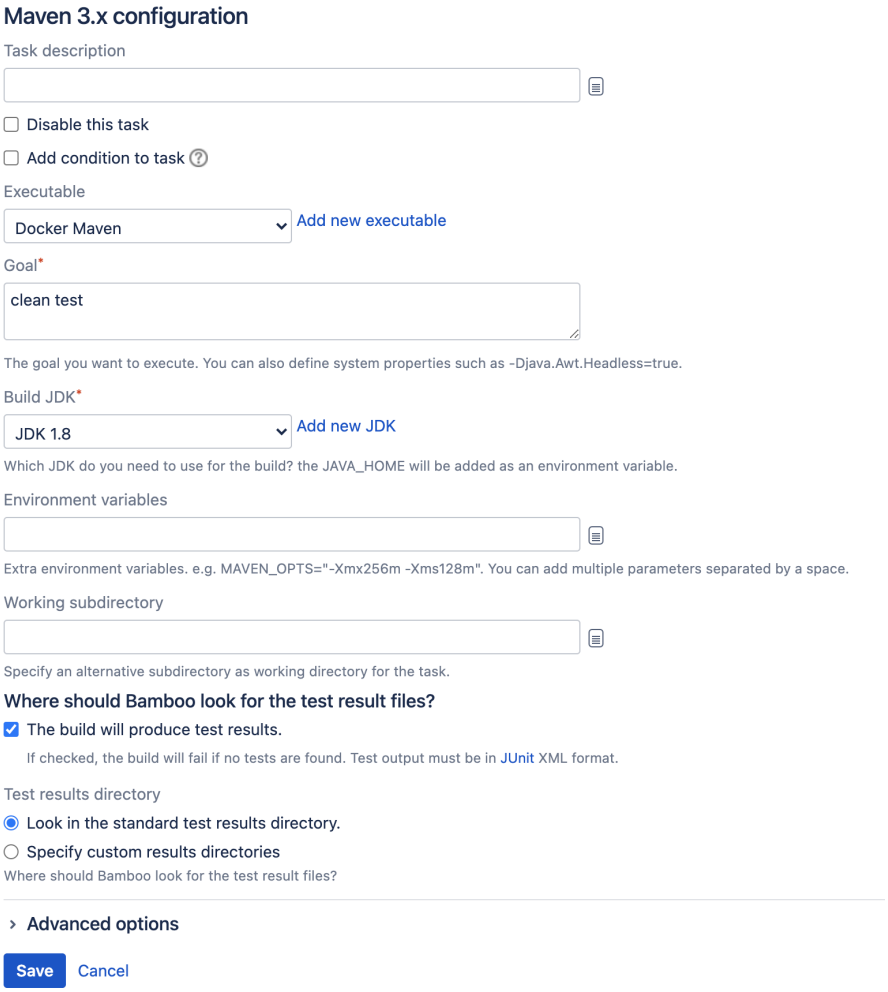
The build will produce test results
Select one of the following: **Look in the standard test results directory** or **Specify custom results directories** – Specify the alternative directory, relative to the root directory, where test results will be created. You can use Ant-style patterns such as `**/test-reports/*.xml`. Bamboo requires test results to be in JUnit XML format. ⚠️
For jobs that use CVS, the root directory is `<bamboo-home>/xml-data/build-dir/JOB_KEY/<cvs-module>`.

Use Maven return code
Select to have Bamboo skip log parsing.

Override project file (Optional: Maven 2.x and later only)

The path to your Maven project file, relative to the working sub directory specified. If this is not specified, Maven will use the `pom.xml` in the root of the working sub directory.

4. Select **Save**.



MSBuild

This page describes how to configure a Bamboo task to use an MSBuild executable.

Related pages:

- Configuring tasks
- Configuring jobs

Note that you cannot use Clover to collect code coverage for MSBuild builds, as Clover only supports builders of Java/Groovy-based projects, such as Ant, Maven, or Grails.

To configure an MSBuild task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of the desired MSBuild task, or select **Add task** > **MSBuild** if creating a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable
   The MSBuild executable that is available to perform the task. The executable you select will become one of the task's (and so, the job's) requirements.

   You can add other executables, if required.

   Project file
   The name of the solution, project file, or MSBuild project to execute, for example `ExampleSolution.sln`. You can include variables (see Bamboo variables).

   Options
   The MSBuild command line options that you want to include.

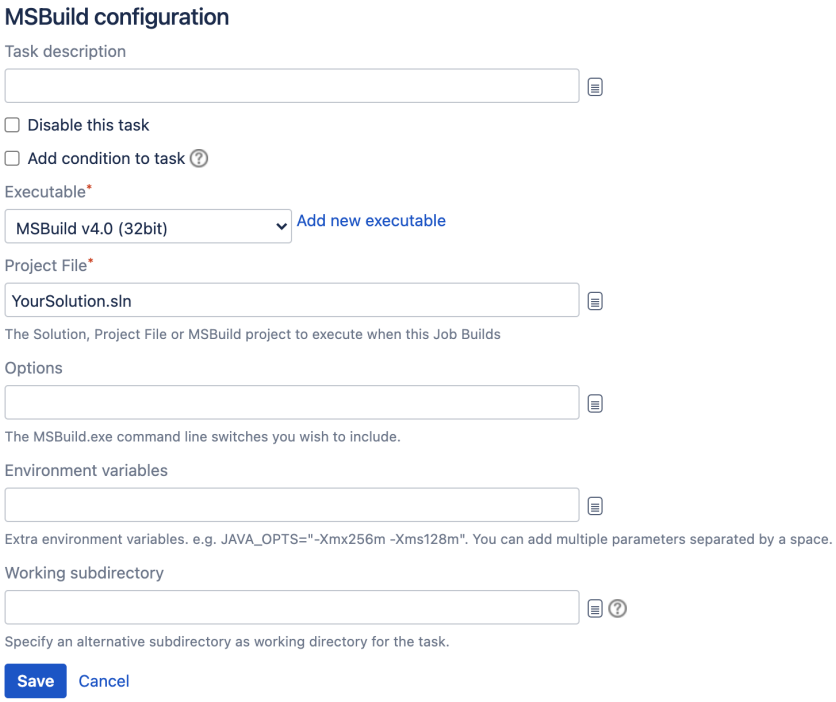
   By default, Bamboo 5.7 (and later versions) writes the contents of the **Projects File** and **Options** field s to an MSBuild response file. See below for more information.

   You can include variables (see Bamboo variables).
4. If required, specify environment variables and working directory settings: **Environment variables** (Opt ional)
   Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables).
   Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

   Working subdirectory (Optional)
   An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

   Run as Powershell script (Optional, Windows only)
   Check the Run as Powershell script checkbox to run the script with Powershell instead of cmd.exe which interprets .bat files. The inline editor supports Powershell syntax.

5. Select **Save**.



Passing options to MSBuild

By default, Bamboo 5.7 (and later versions) writes the contents of the **Projects file** and **Options** fields to an MSBuild response file:

```
# MSBuild response file generated by Atlassian Bamboo
%CONTENTS_OF_OPTIONS_FIELD%
%CONTENTS_OF_PROJECTS_FILE_FIELD%
```

and then runs the following command:

```
msbuild.exe @<full-path-to-response-file>response-file.rsp
```

This allows you to use the same settings with the Bamboo MSBuild task as you would use when calling MSBuild on the command line.

You can disable creation of the response file. In that case, Bamboo will create a .bat file instead:

```
"<full-path-to-msbuild>msbuild.exe" %CONTENTS_OF_OPTIONS_FIELD% %CONTENTS_OF_PROJECTS_FILE_FIELD%
```

and run that.

To *disable* use of the MSBuild response file, set the `bamboo.plugin.dotnet.msbuild.useResponseFile` system property to false.

There are a couple of ways to do that:

- If you start the Bamboo server or remote agents manually you can set the property on the command line, as an argument to the JVM, like this:

  ```
  -Dbamboo.plugin.dotnet.msbuild.useResponseFile=false
  ```

Do this on all Bamboo agents, and on the Bamboo server if you use local agents.

- If your agents are run as a service, set the system property in the <Bamboo agent home directory>/conf/wrapper.conf configuration file, like this:

```
# The Bamboo Agent home configuration file
wrapper.java.additional.1=-Dbamboo.home=/home/bamboo/bamboo-agent-home
wrapper.java.additional.2=-Dbamboo.agent.ignoreServerCertName=false
wrapper.java.additional.3=-Dbamboo.plugin.dotnet.msbuild.useResponseFile=false
```

- If your Bamboo server runs as a service, add the system property to the <Bamboo home directory>/conf/wrapper.conf configuration file.

NAnt

This page describes how to configure a Bamboo task to use a NAnt executable.

Related pages:

- Configuring tasks
- Configuring jobs

To configure a NAnt task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of the desired NAnt task, or select **Add task** > **NAnt** if creating a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable
   The NAnt executable that is available to perform the task.
   The executable that you select will become one of the task's (and so, the job's) requirements.

   You can add other executables, if required.

   Build file
   The relevant file name (e.g. `default.build`). You can include variables (see Bamboo variables).

   Targets
   The NAnt target that you want Bamboo to execute. for example: `run`. You can also include variables (see Bamboo variables).

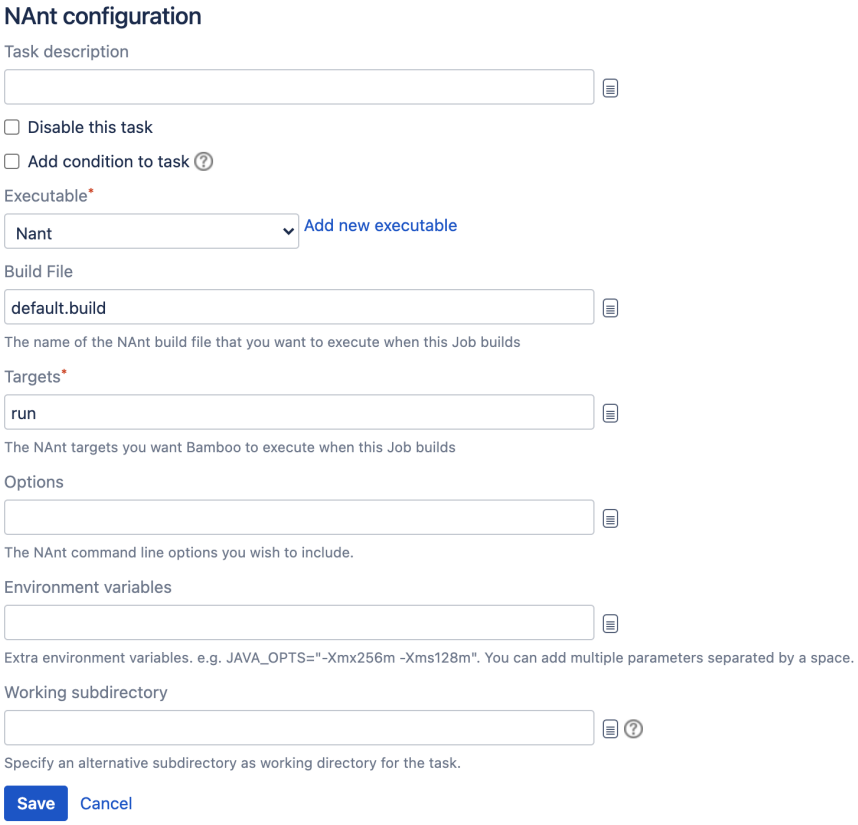
   Options
   The NAnt command line options that you want to include. You can also include variables (see Bamboo variables).
4. If required, specify environment variables and working directory settings: **Environment variables** (Optional)
   Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables).
   Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

Working subdirectory (Optional)
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Run as Powershell script (Optional, Windows only)
Check the Run as Powershell script checkbox to run the script with Powershell instead of cmd.exe which interprets . bat files. The inline editor supports Powershell syntax.

5. Select **Save**.

Note that you cannot use Clover to collect code coverage for NAnt builds, as Clover only supports builders of Java/Groovy-based projects, such as Ant, Maven or Grails.



Script

This page describes how to configure a Bamboo task to run a script. The Script task is flexible enough to allow the possibility to use: the default shells on Linux (/bin/sh) or Windows (cmd.exe), a more modern shell on Windows (PowerShell), and an arbitrary shell in Linux by using the shebang on the first line of the script file. This can be controlled by the Interpreter field explained below.

Related pages:

- Configuring tasks
- Configuring jobs

To configure a script task:

1. Go to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing script task, or select **Add task** > **Script** if creating a new task.
3. In the **Task description** field, enter a description for the task that will appear in Bamboo.
4. Check or clear **Disable this task** to selectively run the task.
5. Optionally, select **Add condition to task** to make the task run only when a certain condition is met.

   You can find conditions on Atlassian Marketplace or implement your own.

6. From the **Interpreter** menu, select one of the available interpreters:
   - **Shell** (chooses the interpreter based on the first line of the script, typically using the shebang)
   - **Windows PowerShell**
   - **/bin/sh or cmd.exe** (uses the default shell of the operating system: sh for Linux, cmd.exe for Windows)
7. From the **Script location** menu, select the location of the script file:
   - **File** — enter the location of the file in the **Script file** field. This can be either relative to the repository root of the plan, or absolute. You can include variables (see Bamboo variables).
   - **Inline** — enter the script in the **Script body** field.
8. In the **Script body** field, enter or paste the code of your script using the syntax appropriate for the selected interpreted.
9. In the **Argument** field, specify an argument to pass to the script.

   Arguments containing spaces must be quoted. You can include variables (see Bamboo variables).

10. Optionally, in the **Environment variables** field, specify additional system environment variables that you want to pass to your build.

    - Separate multiple variables with spaces. Parameters values containing spaces must be quoted, for example:
      ```
      ANT_OPTS="-Xms200m -Xmx700m"
      ```
    - Existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables.
    - If the specified Bamboo variable is present during the task execution, the created environment variable will be set to its value. Declared variable values are passed to the task verbatim, which means that they will not get parsed by any other method and will be sent to the script as a literal string. Hence, no further variable expansion is performed. If you want to assign a script variable using the contents from another variable, do that internally within the script task. Learn how to load variables from within a script task

11. Optionally, in the **Working subdirectory** field, specify an alternative subdirectory relative to the root directory of the job where Bamboo will run the executable.

> ⚠ The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

12. Select **Save**.

Visual Studio

This page describes how to configure a Bamboo task to use a Visual Studio (devenv.exe) executable.

Related pages:

- Configuring tasks
- Configuring jobs

To configure a Visual Studio task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of the desired MSBuild task, or select **Add task** > **Visual Studio** if creating a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Executable

   The Visual Studio executable that is available to perform the task. The executable that you select will become one of the task's (and so, the job's) requirements.

   You can add other executables, if required.

   Solution
   The name of the Visual Studio solution file that you want Bamboo to execute. For example: `RegexDemo /RegexDemo.sln`. You can also include variables (see Bamboo variables).

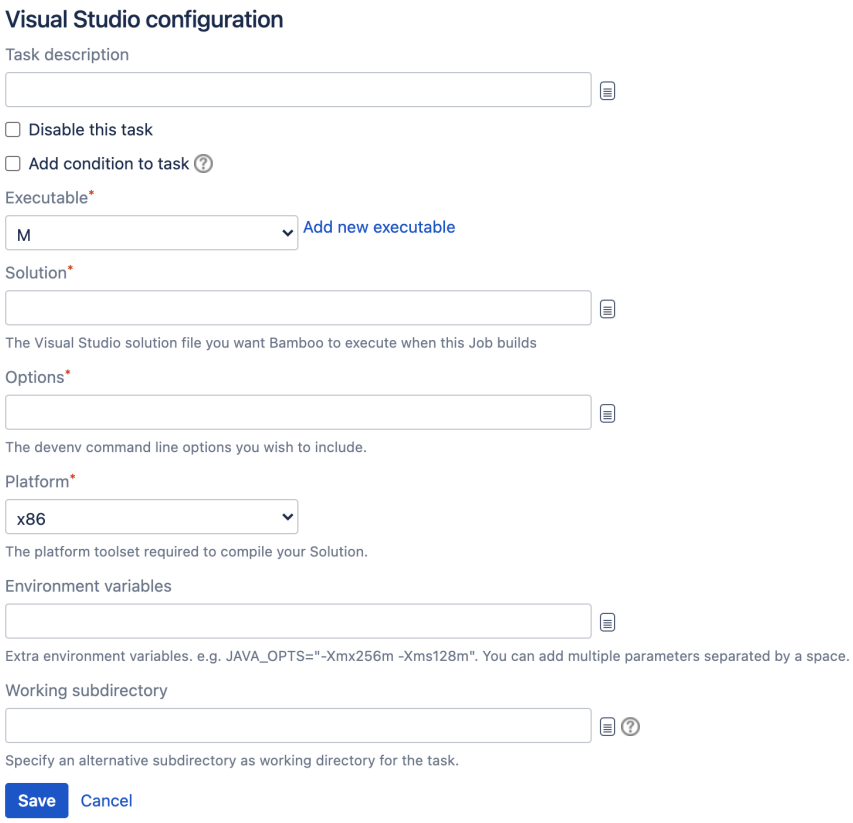
   Options
   Specify any Visual Studio command-line options that you want to include (e.g. `/build Debug`). You can also include variables (see Bamboo variables).

   Platform
   Select the platform toolset required to compile your solution. This is provided as an argument to `Vcvarsall.bat` (see this MSDN article for more details).
4. If required, specify environment variables and working directory settings: **Environment variables** (Optional)
   Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables).
   Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

Working subdirectory (Optional)
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Run as Powershell script (Optional, Windows only)
Check the Run as Powershell script checkbox to run the script with Powershell instead of cmd.exe which interprets . bat files. The inline editor supports Powershell syntax.

5. Select **Save**.



Xcode

Prerequisites

- **Apple Xcode 4 –** or later version
- **Certificates and provisioning profiles** – You must install all required developer certificates and provisioning profiles on every machine that Bamboo will use to run your build. See the App Distribution Guide for more information.
- **Bamboo Xcode support plugin** – The latest Xcode plugin installed in your Bamboo server.
- **ios-sim** (*optional* when building Mac applications) – a command line utility used to launch the iOS Simulator from the command line. If you have homebrew installed, you can install it by running `brew install ios-sim`. For other installation methods, see the ios-sim website.
- **Cocoapods** (*optional* if you do not have a `Podfile` in your project) – Cocoapods is the library dependency manager for Mac OS X. In order for Bamboo to install dependencies from your Podfile (if you have created one), Bamboo will need it installed on all systems where the build should run.

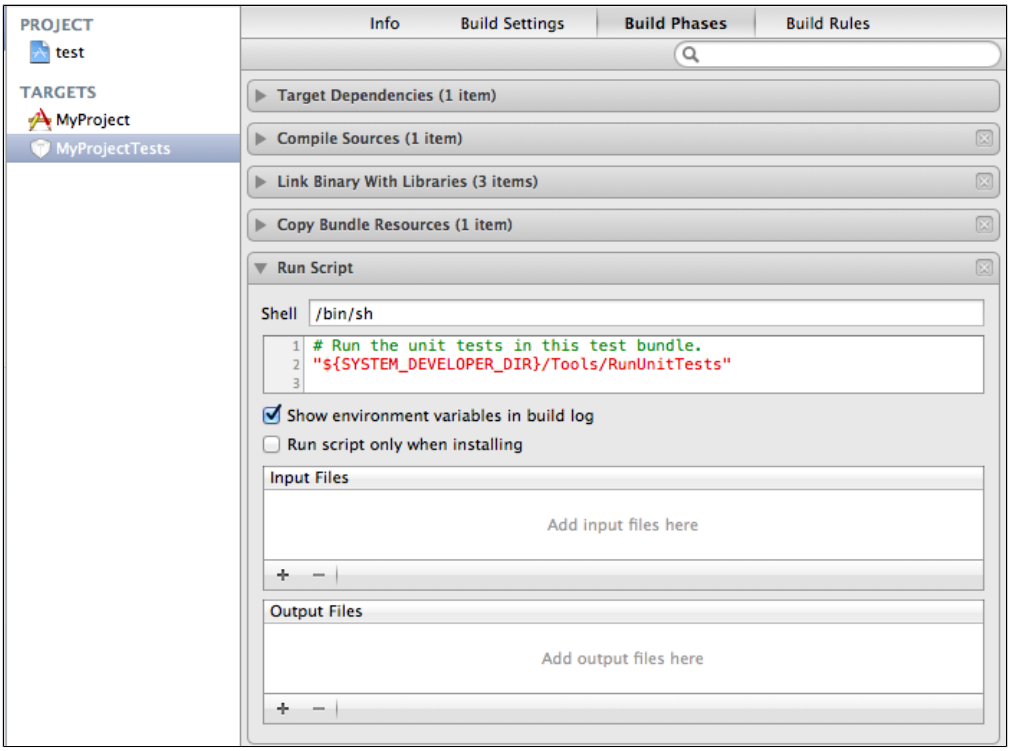
Testing iOS applications

To have tests automatically run on the iOS Simulator and reported within Bamboo you must make some changes to your Xcode project's test bundles and add the Xcode build task to your Job within Bamboo.

Configuring your Xcode project automated simulator tests

Without modifications, Apple does not support running unit tests in the simulator using the `xcodebuild` terminal utility which Bamboo uses to automate builds and tests.

Using ios-sim and a small modification to the RunUnitTests script phase in the test bundle it's possible to overcome this limitation.



Change the content of the script to:

```
if [ "$RUN_UNIT_TEST_WITH_IOS_SIM" = "YES" ]; then
test_bundle_path="$BUILT_PRODUCTS_DIR/$PRODUCT_NAME.$WRAPPER_EXTENSION"
ios-sim launch "$(dirname "$TEST_HOST")" --setenv DYLD_INSERT_LIBRARIES=/../../Library/PrivateFrameworks
/IDEBundleInjection.framework/IDEBundleInjection --setenv XCInjectBundle="$test_bundle_path" --setenv
XCInjectBundleInto="$TEST_HOST" --args -SenTest All "$test_bundle_path"
echo "Finished running tests with ios-sim"
else
"${SYSTEM_DEVELOPER_DIR}/Tools/RunUnitTests"
fi
```

Configuring the Xcode task for testing

To configure a Xcode to test an iOS project task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing Xcode task, or select **Add task** > **Xcode** to create a new task.
3. Complete the following settings:

   Task description
   A description of the task, which is displayed in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

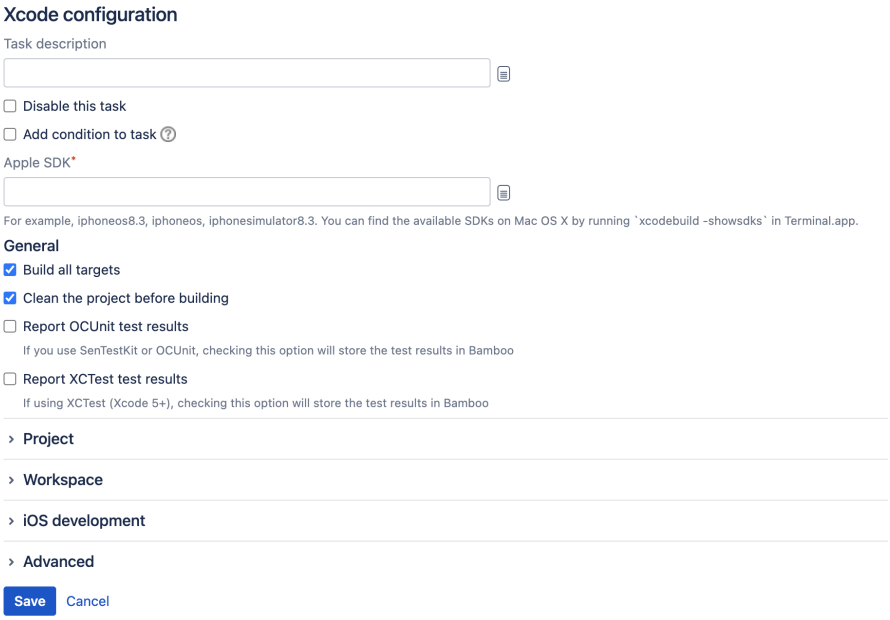
   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Apple SDK
   The Apple SDK to target during the build.

   Report test results
   Report and store any OCUnit/SenTestKit results run during the build.

   Run tests in iOS simulator
   Provides the RUN_UNIT_TEST_WITH_IOS_SIM variable used in the custom build phase to run the unit tests on the simulator.

4. Select **Save**.



Updating the available SDKs when Xcode is upgraded

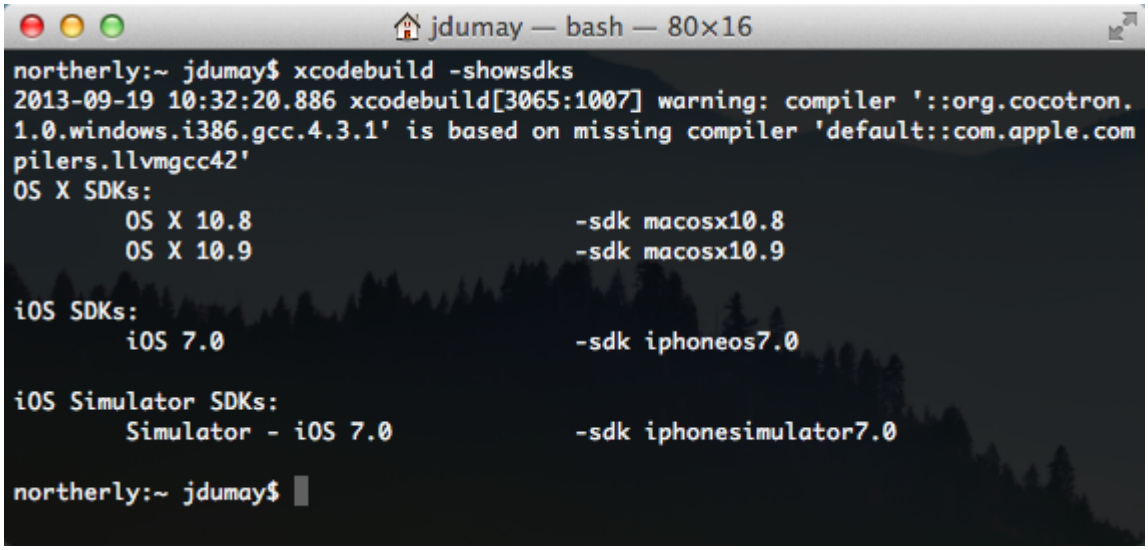
When you upgrade Xcode you may need to update Bamboo with the correct SDK information.

If you use local agents:

1. Log in as an administrator.
2. In the upper-right corner of the screen, select **Administration** > **Overview**.
3. Under **Build resources**, select **Server capabilities**.
4. Select **Detect server capabilities**.

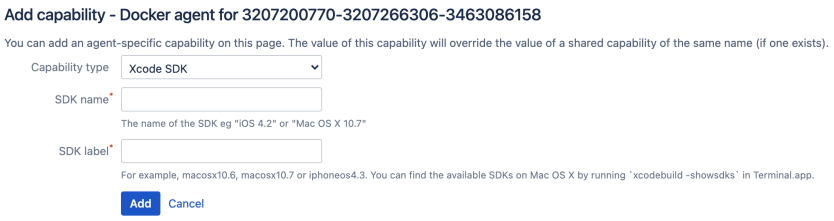
If you use remote agents:

1. Run `xcode-build -showsdks` from the command line.



2. Log in as an administrator.
3. In the upper-right corner of the screen, select **Administration** > **Overview**.
4. Under **Build resources**, select **Agents**.
5. On the **Agents summary** page, pick the agent you wish for the new SDK capability to be present on.
6. Select **Add capability** and pick **Xcode SDK** from the **Capability type** field.
7. Set a name for the SDK (e.g. OS X 10.9).

8. Set an SDK label (e.g. macosx10.9).



Configuring a test task

Test tasks in Bamboo parse test data, and may run tests, using a particular testing framework.

Please note:

- Java builder tasks in Bamboo (e.g. Maven) parse test information as part of the task. You do not need to configure a test task, if you have specified that test results will be produced as part of the builder task. However, you can configure a builder task to not produce test results and use a test task to parse the test data instead. For example, you may want to set up one JUnit Parser task to parse test data for a number of Maven tasks after they have executed.
- .Net builder tasks in Bamboo (e.g. NAnt) **do not** parse test information as part of the task. You must configure a test task (e.g. NUnit Parser), if you want test results from the builder task to be parsed.

Related pages:

- Configuring a builder task

See the following pages for more information on configuring specific test tasks:

- JUnit Parser
- MBUnit Parser
- MSTest Parser
- MSTest Runner
- NUnit Parser
- NUnit Runner
- PHPUnit
- TestNG

Community test task plugins

There are numerous test task plugins available on the Atlassian Marketplace. These plugins are unsupported by Atlassian for the time being but the source code has been made freely available.

Bamboo plugin	Testing framework	Languages and Platforms	Supported by Atlassian?	Source code	Issue tracking adding official support
Bamboo Xcode Task	OCUnit	Objective-C, Apple iOS, Cocoa and Mac OS X	NO	Available on Bitbucket	~~BAM-6149~~ - Provide official support for the Bamboo Xcode plugin CLOSED
Bamboo Ruby Plugin	RSpec	Ruby	NO	Available on Github	⚠ BAM-12328 - Jira project doesn't exist or you don't have permission to view it.

Bamboo CppUnit Task	CppUnit	C++	NO	Available on Bitbucket	BAM-7839 - Support the CppUnit task plugin GATHERING INTEREST

JUnit Parser

This page describes how to configure a Bamboo task to parse JUnit test results.

Because TestNG uses the JUnit XML format, the JUnit Parser task is also able to parse TestNG test results.

Before you begin:

- Java builder tasks in Bamboo (e.g. Maven) parse test information as part of the task. You do not need to configure a test task, if you have specified that test results will be produced as part of the builder task.

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

Atlassian blogs:

- So you want to run tests in parallel... now what?

To configure a JUnit Parser task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing JUnit Parser task, or select **Add task** > **JUnit Parser** to create a new task.
3. Update the task settings:

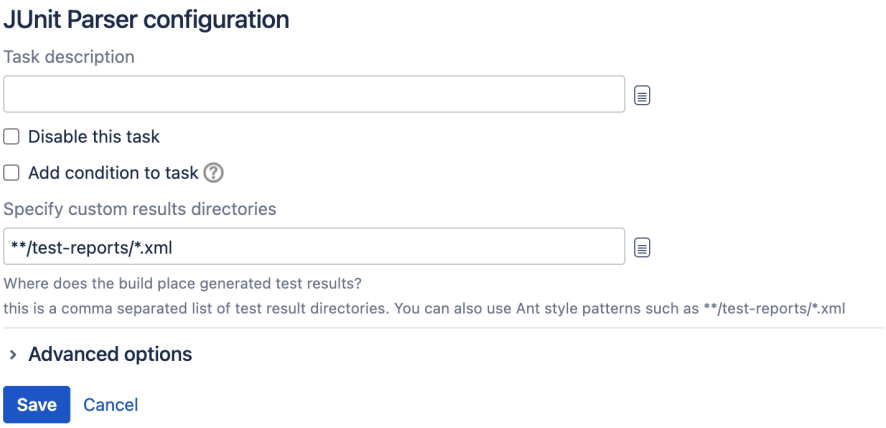
   Task description
   Enter a description of the task, for display in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Specify custom results directories
   Enter the name of the test results directory (or multiple directories, separated by commas). You can also use Ant-style patterns such as `**/test-reports/*.xml`/ where the base directory is the "working directory" – this can be found at the start of your build log. Do not specify an absolute path.
   For jobs that use CVS, the job build's root directory is `<bamboo-home>/xml-data/build-dir/JOB_KEY/<cvs-module>`.

4. Select **Save**.



MBUnit Parser

This page describes how to configure a Bamboo task to parse MBUnit test results.

Before you begin:

- .NET builder tasks in Bamboo (e.g. NAnt) do not parse test information as part of the task. You must configure a test task (e.g. MBUnit Parser), if you want test results from the builder task to be parsed.

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

To configure a MBUnit Parser task:

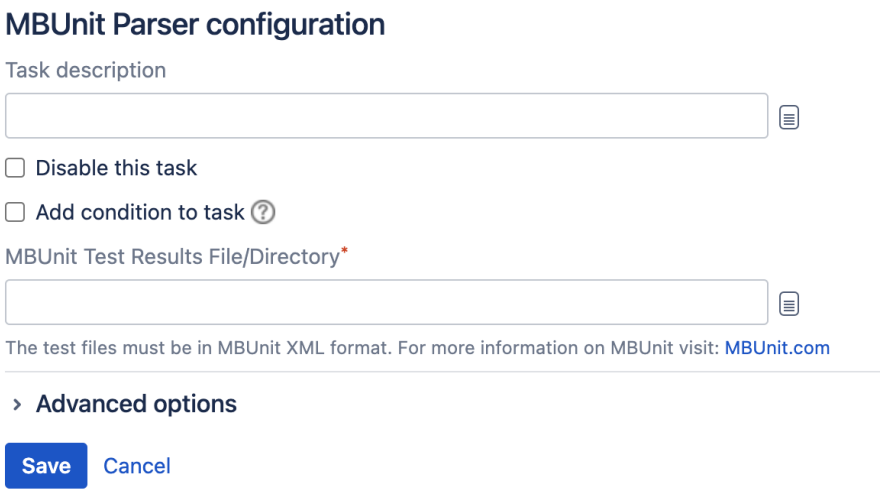
1. Navigate to the **Tasks** configuration tab (this will be the default job if creating a new plan).
2. Select the name of an existing MBUnit Parser task, or select **Add task** > **MBUnit Parser** to create a new task.
3. Update the task settings:

   Task description
   Enter a description of the task, for display in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   MBUnit Test Results File/Directory
   Enter the name of the test results file. The test file must be in MBUnit XML format. For more information on MBUnit, see http://www.mbunit.com/.
4. Select **Save**.



MSTest Parser

This page describes how to configure a Bamboo task to parse MSTest results.

.NET builder tasks in Bamboo (for example NAnt) do not parse test information as part of the task. To have the test results parsed, you need to configure a test task such as MSTest Parser.

Note that each test results file must have a unique name. You can use Bamboo variables to achieve this. Here is a customer-supplied example that includes the revision and build numbers in the name of the test file:

```
<Project-Test-Subfolder>\TestResults\<Project>TestResults-Rev_${bamboo.repository.revision.number}-
Build_${bamboo.buildNumber}.trx
```

To configure a MSTest Parser task:

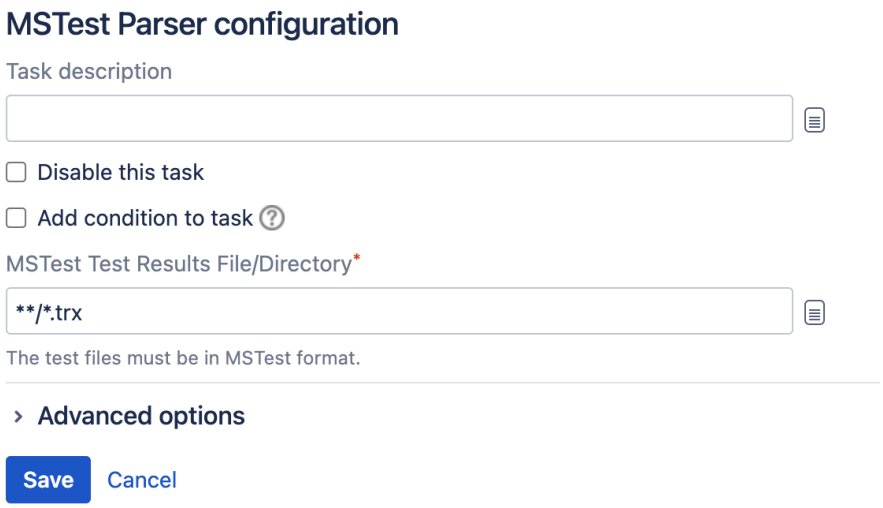
1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing MSTest Parser task, or select **Add task** > **MSTest Parser** to create a new task.
3. Update the task settings:

   Task description
   Enter a description of the task, for display in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   MSTest Test Results File/Directory
   Enter the name of the test results file. The test file must be in MSTest format. For more information on MSTest, see this MSDN page.
4. Select **Save**.



MSTest Runner

This page describes how to configure a Bamboo MSTest Runner task. The MSTest Runner task runs and parses tests for .NET builds.

Before you begin:

- .NET builder tasks in Bamboo (e.g. NAnt) do not parse test information as part of the task. You must configure a test task (e.g. MSTest Parser), if you want test results from the builder task to be parsed.

- If Bamboo is running as a Windows service, ensure that the Service is running as a local user instead of a System User (Bamboo will install itself as the SYSTEM user on Windows).

To configure a Bamboo MSTest Runner task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default Job if creating a new plan).
2. Select the name of an existing MSTest Runner task, or select **Add task** > **MSTest Runner** to create a new task.
3. Update the task settings:

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

Task description
A description of the task, for display in Bamboo.

Disable this task
Check, or clear, to selectively run this task.

Add condition to task
Make task run only when a certain condition is met.
You can find conditions on Atlassian Marketplace or implement your own.

Executable
The MSTest Runner executable that you wish to use for this task (e.g. "Visual Studio 2010"). The executable that you select will become one of the task's capability requirements (and hence, one of the job's requirements). For details, please see Configuring a job's requirements.

Specifically for MSTest, we recommend that the executable be defined with the Visual Studio IDE folder path. Example:

```
C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\
```

This will allow Bamboo to find the necessary resources.

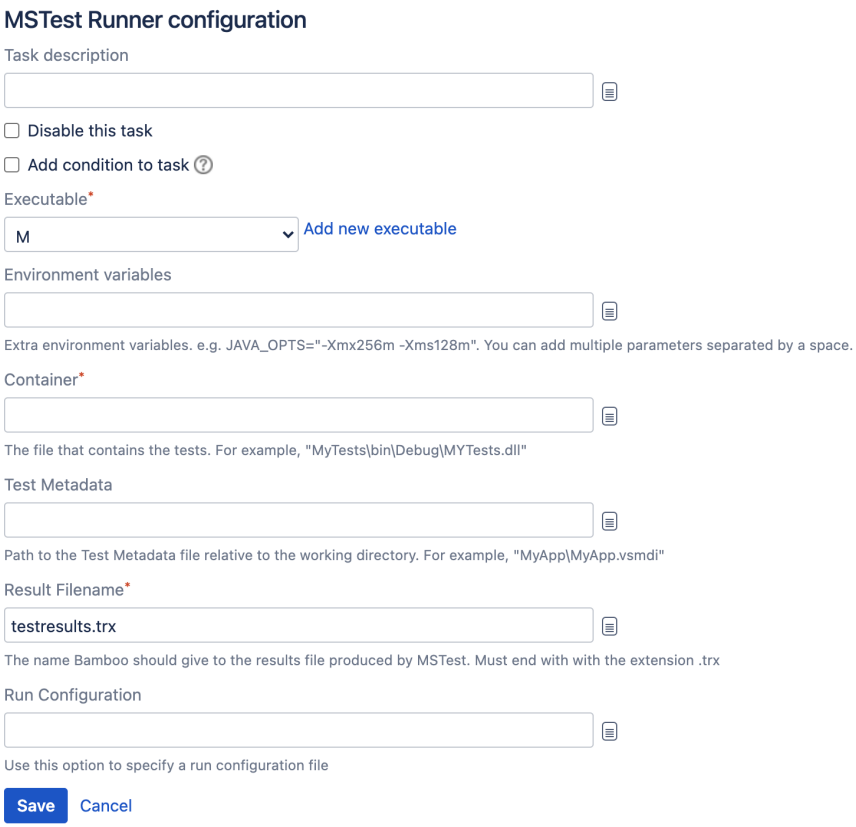
Environment variables
Any extra environment variables you want to pass to your build. e.g. JAVA_OPTS="-Xmx256m -Xms128m".

Container
The test container, i.e. the file that contains the tests you want to run. For example, `tests.dll`. The value of this field is passed to the MSTest.exe as the `/testcontainer` parameter. See MSTest.exe Command-Line Options (MSDN).

Test Metadata
The path to the Test Metadata file relative to the working directory. For example, "MyApp\MyApp.vsmdi"

Result Filename
The file that you want to save the test results to. For example, `testResults.trx`. The value of this field is passed to the MSTest.exe as the `/resultsfile` parameter. See MSTest.exe Command-Line Options (MSDN).

Run Configuration
The run configuration that you want to use. For example, `localtestrun.Testrunconfig`. The value of this field is passed to the MSTest.exe as the `/runconfig` parameter. See MSTest.exe Command-Line Options (MSDN).



NUnit Parser

This page describes how to configure a Bamboo NUnit Parser task.

Before you begin:

- .NET builder tasks in Bamboo (e.g. NAnt) do not parse test information as part of the task. You must configure a test task (e.g. MSTest Parser, NUnit Parser), if you want test results from the builder task to be parsed.

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

To configure a NUnit Parser task:

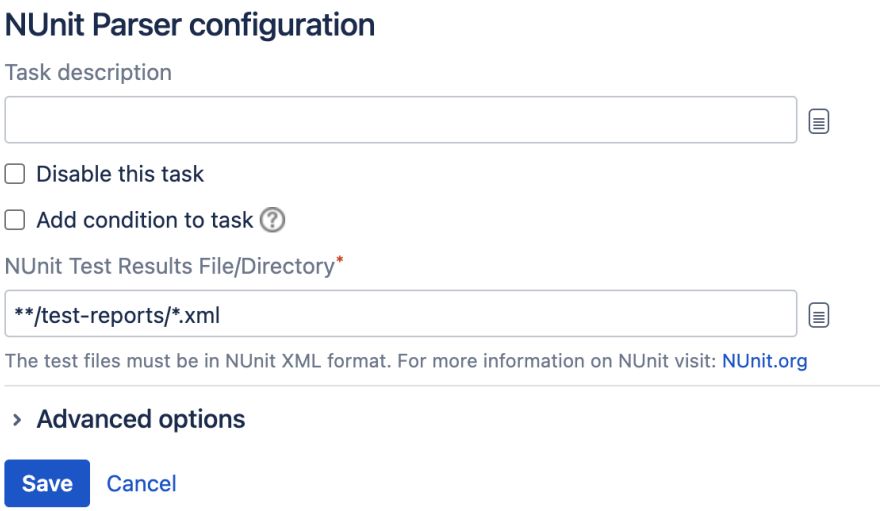
1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing NUnit Parser task, or select **Add task** > **NUnit Parser** to create a new task.
3. Update the task settings:

   Task description
   Enter a description of the task, for display in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   NUnit Test Results File/Directory
   Enter the name of the test results file/directory. The test files must be in NUnit XML format. For more information on NUnit, see http://www.nunit.org/.
4. Select **Save**.



NUnit Runner

This page describes how to configure a Bamboo task to run NUnit tests, and then parse the test results.

Before you begin:

- .NET builder tasks in Bamboo (e.g. NAnt) do not parse test information as part of the task. You must configure a test task (e.g. MSTest Parser, NUnit Parser), if you want test results from the builder task to be parsed.

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

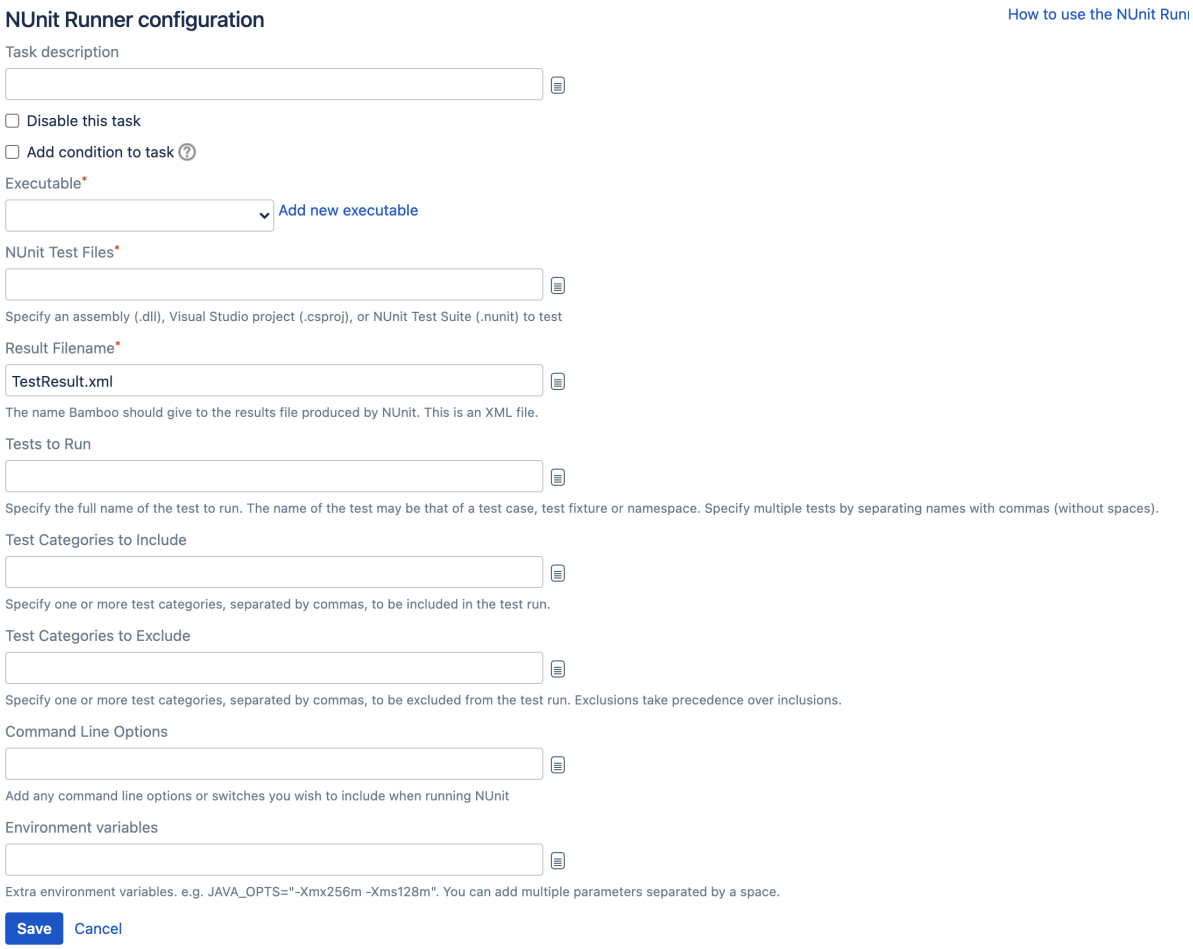
To configure a NUnit Runner task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing NUnit Runner task, or select **Add task** > **NUnit Runner** to create a new task.
3. Update the task settings:

Task Description	A description of the task, which gets displayed in Bamboo.
Disable this task	Check, or clear, to selectively run this task.
Add condition to task	Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
Executable	The NUnit Runner executable that is available to perform the task. The executable that you select will become one of the task's (and so, the job's) requirements. You can add other executables, if required.
NUnit Test Files	The name of an assembly (.dll), Visual Studio project (.csproj), or NUnit Test Suite (.nunit) to test. See http://www.nunit.org/.
Result Filename	The name to be used for the XML results file.
Tests to Run	The name of the test case, test fixture or namespace to run.
Test Categories to Include	Specify one or more test categories, separated by commas, to be included in the test run.
Test Categories to Exclude	Specify one or more test categories, separated by commas, to be excluded from the test run. Exclusions take precedence over inclusions.
Command Line Options	Specify any command line options or switches you wish to include when running NUnit.
Environment variables	Any extra environment variables you want to pass to your build. e.g. JAVA_OPTS="-Xmx256m -Xms128m".

4. Select **Save**.

For more information on NUnit, see http://www.nunit.org/.



PHPUnit

This page describes how to configure a PHPUnit task.

Before you begin:

- To use this task, you will need to install PHPUnit and reference the path to your PHP command-line interpreter, (e.g. `/usr/bin/phpunit` on Ubuntu).

Related pages:

- Configuring tasks
- Configuring jobs

To configure a PHPUnit task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing task, or select **Add task** > **PHPUnit** (or another option, such as **PHPUnit 3.3.X**) to create a new task.

3. Update the task settings:

Task description
Enter a description of the task, for display in Bamboo.

Disable this task
Check, or clear, to selectively run this task.

Add condition to task
Make task run only when a certain condition is met.
You can find conditions on Atlassian Marketplace or implement your own.

Executable
Select the PHPUnit executable that you wish to configure for this task (e.g. "PHPUnit 3.3.x" or "PHPUnit"). The executable that you select will become one of the task's capability requirements (and hence, one of the job's requirements). For details, please see Configuring a job's requirements.

Arguments
Type the name of the directory/files that will be analyzed recursively by PHPUnit. The default value is "." (i.e. the working subdirectory, if specified). You must specify at least one argument.

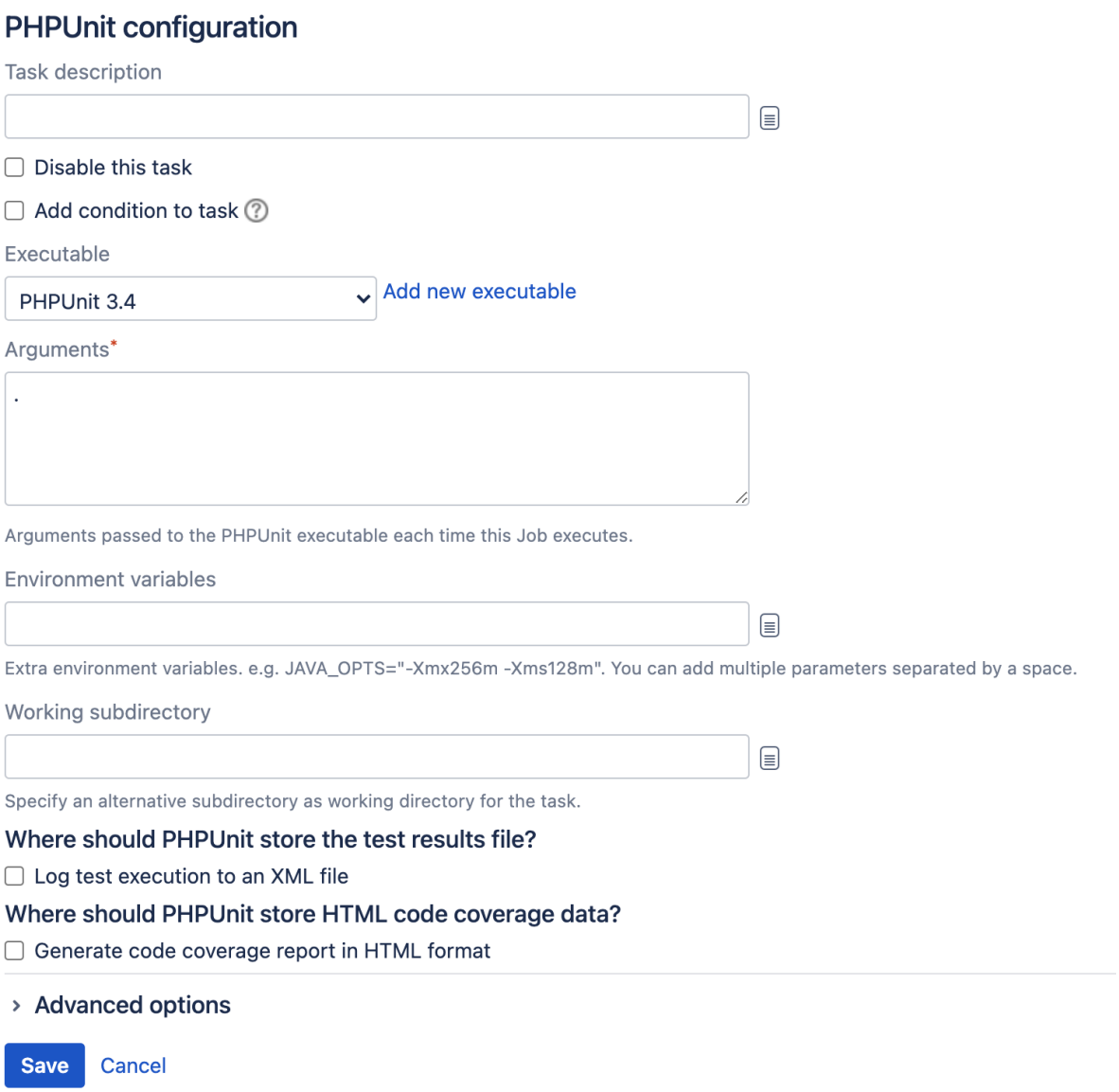
Environment variables (Optional)
Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Using global, plan or build-specific variables).
Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g ANT_OPTS="-Xms200m -Xmx700m").

Working subdirectory (Optional)
An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Log test execution to XML file
Select if you want PHPUnit to record test results in JUnit format. This format is also used by TestNG.
Test Result File — the relative location, and name, of the file to record PHPUnit test results.

Generate code coverage report in HTML format
Select if you want PHPUnit to generate code coverage data in HTML format (e.g. for PHPUnit HTML Code Coverage reports).
HTML Code Coverage Directory — the relative location of the directory to store the code coverage report.

4. Select **Save**.



TestNG

This page describes how to configure a Bamboo task to parse TestNG test results.

Before you begin:

- Java builder tasks in Bamboo (e.g. Maven) parse test information as part of the task. You do not need to configure a test task, if you have specified that test results will be produced as part of the builder task.

To configure a TestNG Parser task:

Related pages:

- Configuring tasks
- Configuring jobs
- Configuring a test task

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing TestNG task, or select **Add task** > **TestNG** to create a new task.
3. Update the task settings:

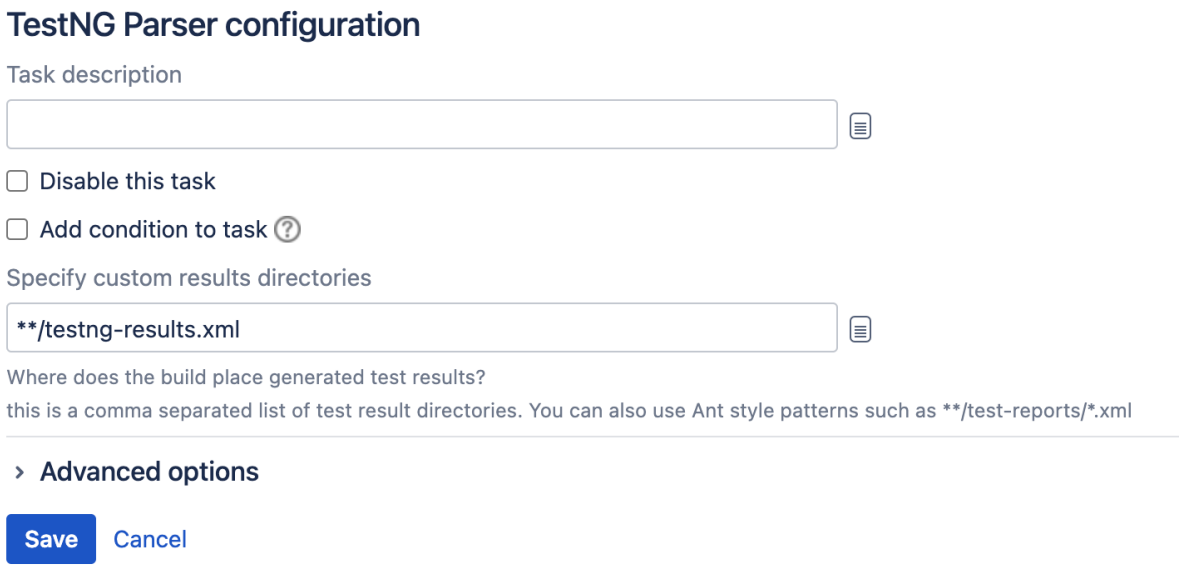
   Task description
   Enter a description of the task, for display in Bamboo.

   Disable this task
   Check, or clear, to selectively run this task.

   Add condition to task
   Make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.

   Specify custom results directories
   Enter the name of the test results directory (or multiple directories, separated by commas). You can also use Ant-style patterns such as `*/test-reports/.xml`. Please specify file path relative to your job build's root directory. Do not specify an absolute path. For jobs that use CVS, the job build's root directory is `<bamboo-home>/xml-data/build-dir/JOB_KEY/<cvs-module>`.

4. Select **Save**.



Configuring a variables task

Variables tasks in Bamboo allow you to:

- pass a value between stages.
- pass a value from a plan to a deployment project.
- read variables from a file using a 'key=value' format.
- print to file the current values of the available variables in your build.

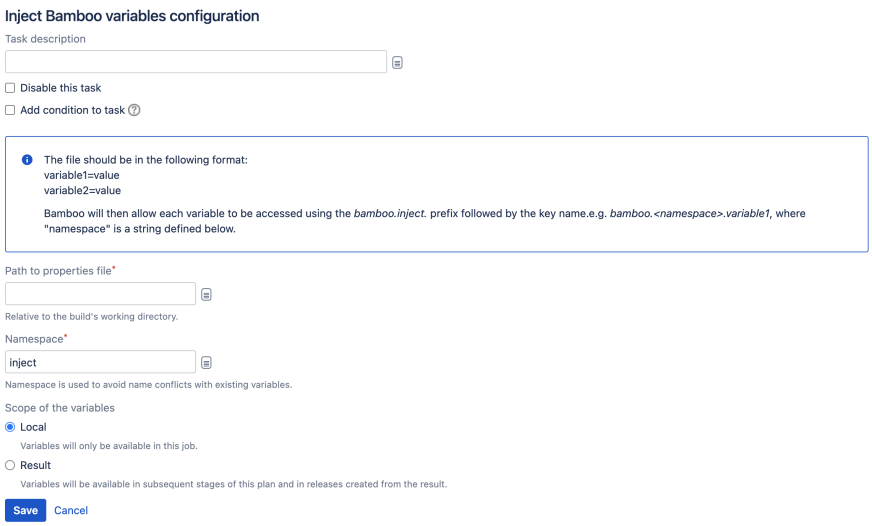
Inject Bamboo variables task

The Inject Bamboo variables task allows you to read the values for variables from a file, and create those variables in your build plan.

The file should use a 'key=value' format. Note that starting from Bamboo version 5.14, you must provide relative paths to the property file.

You can choose if those variables should have a local scope (in which case they cease to exist when the job finishes) or result scope (in which case they are persisted and passed into subsequent stages or related deployment releases).

See Configuring tasks for help in creating a task.

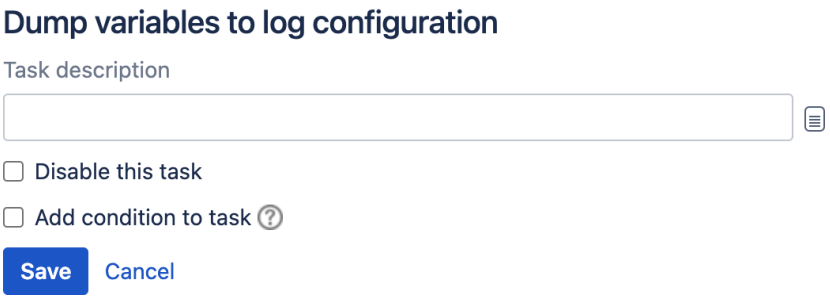


A note regarding using Injected variables with release names: while they can be used for versioning they cannot currently be configured to auto-increment.

Dump variables to log task

The Dump variables to log task simply writes out the current values of all variables used in the build.

See Configuring tasks for help on creating a task.



FAQ

Q. What happens if the same key is used twice?

A. The last assignment will prevail. If you set the scope to local variable with the same key as an existing result variable, the value of the result variable will be restored when the job finishes.

Q. What if I manually set a variable with the same key as a result variable?

A. Same as above – the last assignment wins.

Q. Can I manually override a result variable in a subsequent manual stage?

A. Yes.

Q. What if two jobs in the same stage create the same variable?

A. The variable will exist but it is undefined which value will ultimately be assigned to it.

Q. Is restarting builds, re-running failed jobs or continuing from a manual stage supported?

A. Of course! One caveat though: if you restart a build which has an associated deployment release, the variable in the release will not be refreshed. We're working on that...

Configuring a deployment task

Deployment tasks in Bamboo allow you to set up plans that can manage the continuous deployment and delivery of your application.

See the following pages for more information on configuring specific deployment tasks in Bamboo:

- Using Tomcat with Bamboo for continuous deployment
- Using the SCP task in Bamboo
- Using the SSH task in Bamboo
- Using the AWS CodeDeploy task

Using the SCP task in Bamboo

You can use the Bamboo SCP task to upload files from Bamboo directly to a remote server as part of a Bamboo job. The SCP task is able to copy multiple files and preserves the directory structure for the copied files.

See Configuring a deployment task for an overview of Bamboo deployment tasks.

Related pages:

- Configuring a deployment task
- Using the SSH task in Bamboo

To configure an SCP task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing SCP task, or click **Add task** > **SCP Task** to create a new task.
3. Complete the following settings:

Setting	Description
Task description	Helps you identify the purpose of the task.
Disable this task	Check, or clear, to selectively run this task.
Add condition to task	Make the task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
Hosts	Comma-separated list of hostnames or IP addresses of the remote servers to which the files will be copied.
Verify remote host fingerprint on connect	Enter the host fingerprint to be verified. See below for more details.
Port	The port number of the remote host that is used for the SSH connection. The default value is 22.
Username	The username to use to connect to the remote host.
Authentication Type	**Password** – the password associated with **Username**. **Key without passphrase** – browse to the SSH private key with which to authenticate with the remote host. **Key with passphrase** – browse to the SSH private key, and supply the passphrase, to use to authenticate with the remote host.
Local Path	The local path (relative to the Bamboo working directory) to the files you want to copy. Use commas to separate files and directories. You can also use Ant-style pattern matching to include multiple files, such as `**/target/*.jar`.
Remote Path	The path to the destination directory on the remote server.

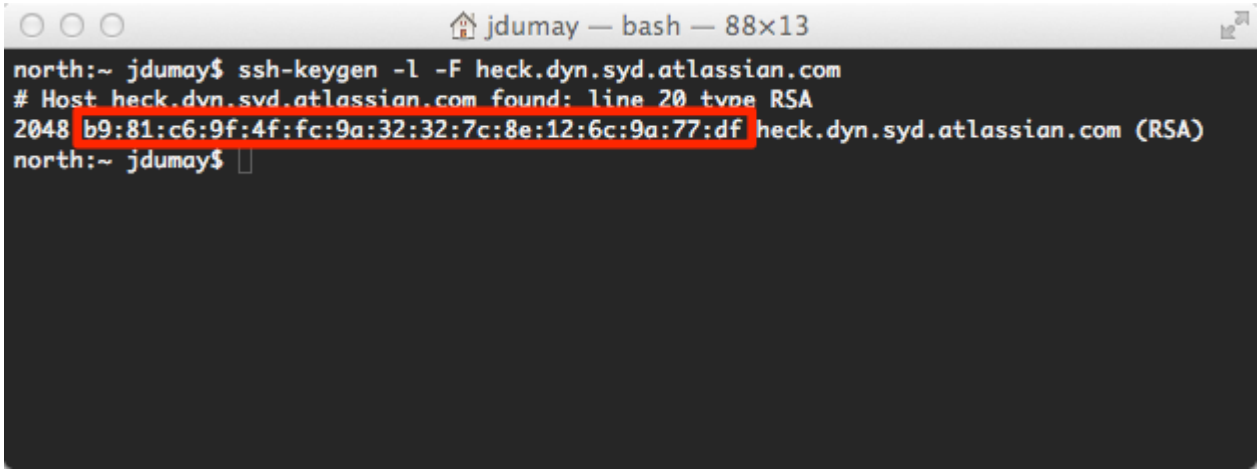
4. Select **Save**.

Host fingerprint

You can determine the fingerprint for a host by running:

```
ssh-keygen -l -F <HOSTNAME>
```

The fingerprint is the part of the response shown in the screenshot below:



Using the SSH task in Bamboo

On this page:

Related pages:

- Configuring a deployment task
- Using the SCP task in Bamboo

You can use the Bamboo SSH task to execute a SSH command on a remote computer as part of a Bamboo job.

You can use the SSH task to do such things as:

- Calling database migration scripts
- Starting and stopping services
- Anything you can run on the command line on a remote machine

See Configuring a deployment task for an overview of Bamboo deployment tasks.

To configure an SSH task:

1. Navigate to the **Tasks** configuration tab for the job (this will be the default job if creating a new plan).
2. Select the name of an existing SSH task, or select **Add task** > **SSH Task** to create a new task.
3. Complete SSH task configuration settings:

Setting	Description
Task description	Optional description for the configured task.
Disable this task	Prevent the task from running.

Add condition to task	Run the task only when a certain condition is met.
	• **Condition type** (for SSH tasks, **Variable** is the only available option)
	• **Variable name** (the name of the Bamboo variable to base the condition on)
	• **Condition** (the conditional keyword):
	  ◦ **exists**
	  ◦ **not exists**
	  ◦ **equals** (exact value)
	  ◦ **not equals** (exact value)
	  ◦ **matches** (regular expression to match variable value)

Hosts	Comma-separated fully qualified domain names or IP addresses of the remote hosts.
Authentication type	Choose whether Bamboo should authenticate with the host using a **Username and password** or **SSH private key**, and then either provide custom credentials or select shared credentials.
SSH command	The shell command to execute on the remote host. You can only enter a single command here. ⓘ You can configure a remote host to accept build-specific variables sent from Bamboo as environment variables for use in shell scripts run by the SSH task.

4. If you want Bamboo to verify the remote host fingerprint on connection, under **Advanced options**, select **Verify remote host fingerprint on connect**, and then paste the host fingerprint.

You can determine the fingerprint for a host by running the following command from a Unix terminal, where <HOSTNAME> is the fully-qualified domain name or IP address of the remote host:

```
ssh-keygen -l -F <HOSTNAME>
```

For example:

```
ssh-keygen -l -F myhost.acme.com
```

5. Select **Save** .

Configuring a remote host to receive environment variables from Bamboo

Bamboo SSH tasks can send build-specific variables to a remote host as environment variables for use during script execution.

> If the remote host doesn't allow Bamboo to set environment variables, build-specific variables won't be available for use in shell scripts run by SSH tasks and references to Bamboo build variables (`${bamboo_*}`) will resolve to empty lines.

Related content:

Using the SSH task in Bamboo

Bamboo variables

To allow Bamboo to set environment variables on the remote host:

1. Append the following lines to the hosts's `/etc/ssh/sshd_config` file:

```
# Allow Bamboo SSH task to pass build variables
AcceptEnv bamboo_*
```

2. Restart the SSH daemon on the remote host:
   - On Debian-based Linux distributions, run:

```
sudo systemctl restart ssh
```

   - On Red Hat-based Linux distributions, run:

```
sudo systemctl restart sshd
```

You can now reference build-specific variables in your shell scripts using the following syntax:

```
${bamboo_*}
```

For example:

```
${bamboo_buildNumber}
```

Using Tomcat with Bamboo for continuous deployment

You can use Bamboo to deploy and manage your Java web application with Tomcat 6 or 7, without having to directly interact with Maven, Ant or write special scripts.

Bamboo provides tasks that use the HTTP-based scripting interface to the Tomcat Manager application that ships with Tomcat. You can use the Bamboo tasks to perform the following Tomcat operations:

- Deploy an application to a Tomcat instance
- Start an application in a Tomcat instance
- Stop an application in a Tomcat instance
- Reload an application to a Tomcat instance
- Undeploy an application from a Tomcat instance

Each of these tasks run as part of a Bamboo job.

On this page :

Related pages:

- Configuring a deployment task

Atlassian blogs:

- Continuous deployment with Bamboo and Tomcat

Setting up Tomcat

You will need to prepare the Tomcat server before Bamboo can manage and deploy applications to it.

1. Download the Tomcat 7 distribution and unzip it on your file system.
2. Add a new Tomcat user for Bamboo to use the Tomcat Application Manager by adding the following line in `conf/tomcat-users.xml` between the `<tomcat-users>` tags:

   ```
   <user username="bamboo" password="bamboo" roles="manager-script,manager-gui"/>
   ```

3. Start Tomcat by running `bin/startup.sh` on Linux or Mac, or `bin/startup.bat` on Windows.
4. Test this setup by browsing to http://localhost:8080/manager and using the username and password you configured in the step above. You should see the Tomcat Web Application Manager page, and a list of the running applications on your instance.

For more information about the Tomcat Application Manager and its authentication and authorization configuration see the Tomcat documentation.

Deploying an application from Bamboo

You use Tomcat deployment tasks in the context of a job in a build plan in Bamboo. This plan should generate a deployable artifact, such as a WAR file. To deploy the artifact, you add a Tomcat deploy task to the plan, as follows:

1. Navigate to the task configuration for the job (this will be the default job if you are creating a new plan).
2. Select **Add task** > **Deploy Tomcat Application**.

3. Configure the Tomcat task settings, as described below.
4. Select **Save**.
5. To deploy the application, simply run the plan.

You can check that the deployment has been successful by:

1. Navigating to the logs for the job. Towards the end you should see something like:

```
> Deploying application with war file ‘target/tomcat-test-0.1.war’ to context ‘/myapp’ to server
[http://localhost:8080/manager/](http://localhost:8080/manager/)
> Application was successfully deployed.
```

   This indicates that Bamboo completed the task successfully.
2. Now, browse to the expected address for your application. You should see the welcome page.

Configuring the Tomcat tasks

The Tomcat Deploy, Start, Stop, Undeploy and Reload tasks each make use of some or all of the following configuration settings:

Task description
To help you to identify the task.

Disable this task
Check, or clear, to selectively run this task.

Add condition to task
Make task run only when a certain condition is met.
You can find conditions on Atlassian Marketplace or implement your own.

Tomcat Manager URL
The URL for the Tomcat Manager e.g. http://localhost:8080/manager/

Target Tomcat server is version 6.x
Select this if deploying to a Tomcat 6.x server.

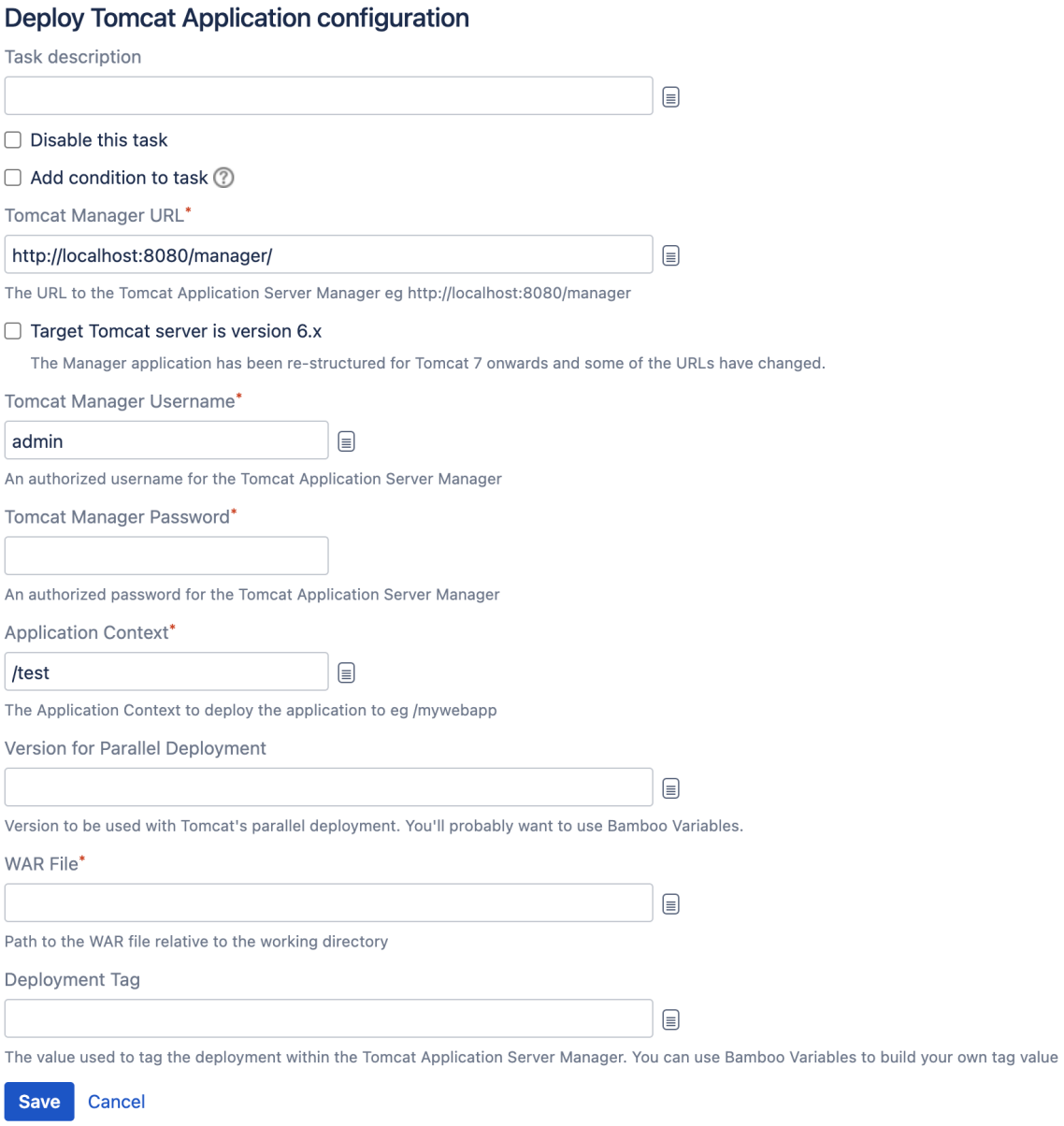
Tomcat Manager Username and Password
These should match the credentials set in `conf/tomcat-users.xml` when you configured Tomcat, as described above.

Application Context
Specifies where the application should sit on the Tomcat server once deployed.

WAR File
The path to the WAR file, relative to the Bamboo working directory, for example “target/tomcat-test-0.1.war”

Deployment Tag
The value used to tag the deployment within the Tomcat Manager. You can use Bamboo variables to build the tag value.

For example, using the value ${bamboo.buildResultKey} will tag the deployment with the build number of the build that was used to deploy the application.



Using the AWS CodeDeploy task

With the AWS CodeDeploy task for Bamboo you can deploy applications to EC2 instances automatically, reliably, and rapidly. Additionally, AWS CodeDeploy keeps track of the whole deployment process.

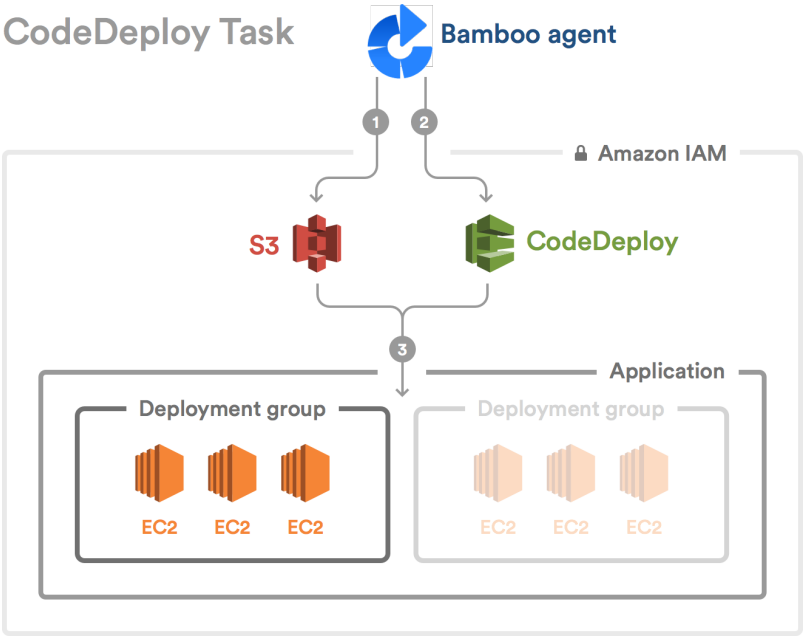
On this page:

Overview

The AWS CodeDeploy task compresses the specified directory with an AppSpec file into a .zip file, uploads the file to Amazon S3, and starts the deployment according to the configuration provided in the CodeDeploy application.

Further reading

- What is AWS CodeDeploy?
- AWS CodeDeploy deployments



Before you begin

There are several requirements that must be met before you can start using the AWS CodeDeploy task. In short, you must configure the following in your AWS Management Console:

- an EC2 instance with a tagged deployment group
- a CodeDeploy application
- an IAM user
- an S3 bucket

For more guidelines about your AWS configuration, see AWS CodeDeploy configuration for Bamboo.

The content that you want to be zipped and deployed requires a specific structure. For more information, see Preparing files for deployment.

Adding an AWS CodeDeploy task to a Bamboo plan

To use the CodeDeploy task:

1. Go to the plan configuration.

2. Select **Add task**.
3. Select **AWS CodeDeploy**.
4. Provide the following details:

Field	Description
Task description	A short description of the task.
Disable this task	Select the check box to skip this task in the build.
Add condition to task	Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
AWS credentials	You can select existing AWS credentials from the list or add new AWS credentials. The newly created AWS credentials are added to the shared credentials list in Bamboo. To make existing AWS credentials available for selection within the AWS CodeDeploy task in Bamboo, add them to Shared credentials.
Region	A region in which the application is deployed.
Deployable content directory	Location of the directory that contains the deployable content and an AppSpec file. By default, it is the root build directory. The content of the directory is compressed into a .zip file and sent to Amazon S3 bucket for deployment. For more information, see Preparing files for deployment.
Amazon S3 bucket	The name of an S3 bucket from which the deployable content (your app and the AppSpec file) is deployed. Start typing to open a selection list of the existing S3 buckets that are available for the AWS credentials provided in the task configuration. For more information, see Amazon S3 bucket.
Application name	The name of the CodeDeploy application that you created in the AWS management console. For more information, see AWS CodeDeploy application.
Deployment group	Start typing to open a list of deployment groups available for the Application name specified in the previous step.

AWS CodeDeploy configuration for Bamboo

The infrastructure setup is described in detail by AWS. For more information, see Getting Started with AWS CodeDeploy.

Atlassian provides guidelines for the following:

- IAM user
- AWS CodeDeploy application
- Amazon S3 bucket

IAM user

We recommend creating a dedicated CodeDeploy IAM user or group.

The following policy gives full permissions to Amazon S3 buckets, CodeDeploy application and deployment group:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:*",
                "s3:*"
            ],
            "Resource": "*"
        }
    ]
}
```

For more examples of policies, see Bucket Policy Examples.

Further reading

- AWS CodeDeploy User Access Permissions Reference
- Create IAM Instance Profile and Service roles

To make existing AWS credentials available for selection within the AWS CodeDeploy task in Bamboo, add them to Shared credentials.

AWS CodeDeploy application

Each CodeDeploy application holds information about the deployment configuration.

For more information, see Create an Application with AWS CodeDeploy.

Amazon S3 bucket

An Amazon S3 bucket must exist. We recommend creating a dedicated CodeDeploy S3 bucket that is located in the same region as the instances to which you want to deploy the application.

For more information about how to grant access to S3 buckets, see IAM user.

Preparing files for deployment

The deployable content that is compressed and sent to an Amazon S3 bucket is called *a revision* and it consists of the application and an AppSpec (Application Specification) file.

Further reading

- AWS CodeDeploy Application Specification Files
- Prepare a Revision for AWS CodeDeploy
- Add an AppSpec File

Examples of revisions

- simple
- advanced

Pattern matching reference

Bamboo supports a powerful type of regular expression for matching files and directories (as with pattern matching in Apache Ant).

These expressions use the following wildcards:

?	Matches one character (any character except path separators)
*	Matches zero or more characters (not including path separators)
**	Matches zero or more *path segments*.

Remember that Ant globs match *paths*, not just simple filenames.

- If the pattern does not start with a path separator i.e. / or \, then the pattern is considered to start with `/**/`.
- If the pattern ends with `/` then `**` is automatically appended.
- A pattern can contain any number of wild cards.

Also see the Ant documentation.

Examples

***.txt**	Matches `/foo.txt` and `/bar.txt` but not `/foo.txt` or `/bar.txt`
/*.txt	Matches `/foo.txt` but not `/bar/foo.txt`
dir1/file.txt	Matches `/dir1/file.txt`, `/dir3/dir1/file.txt` and `/dir3/dir2/dir1/file.txt`
****/dir1/file.txt**	Matches `/dir1/file.txt`, `/dir3/dir1/file.txt` and `/dir3/dir2/dir1/file.txt`
//dir1/file.txt**	Matches `/dir1/file.txt`, `/dir3/dir1/file.txt` and `/dir3/dir2/dir1/file.txt`
/dir3//dir1/file.txt**	Matches `/dir3/dir1/file.txt` and `/dir3/dir2/dir1/file.txt` but not `/dir3/file.txt`,`/dir1/file.txt`
/dir1/**	Matches all files under `/dir1/`

Configuring the Docker task in Bamboo

The Docker task in Atlassian Bamboo allows you to use Docker images and containers in your Bamboo builds and deployments.

Before you begin

- Make sure you have Docker installed. We advise to use the most recent version.
- Define a Docker capability in Bamboo. See Defining a new Docker capability
- If you're using Bamboo on Windows, you can't run Docker commands directly from the Windows command line. To use Docker tasks with Bamboo Windows, run Docker Machine.

The Docker task supports the following Docker actions:

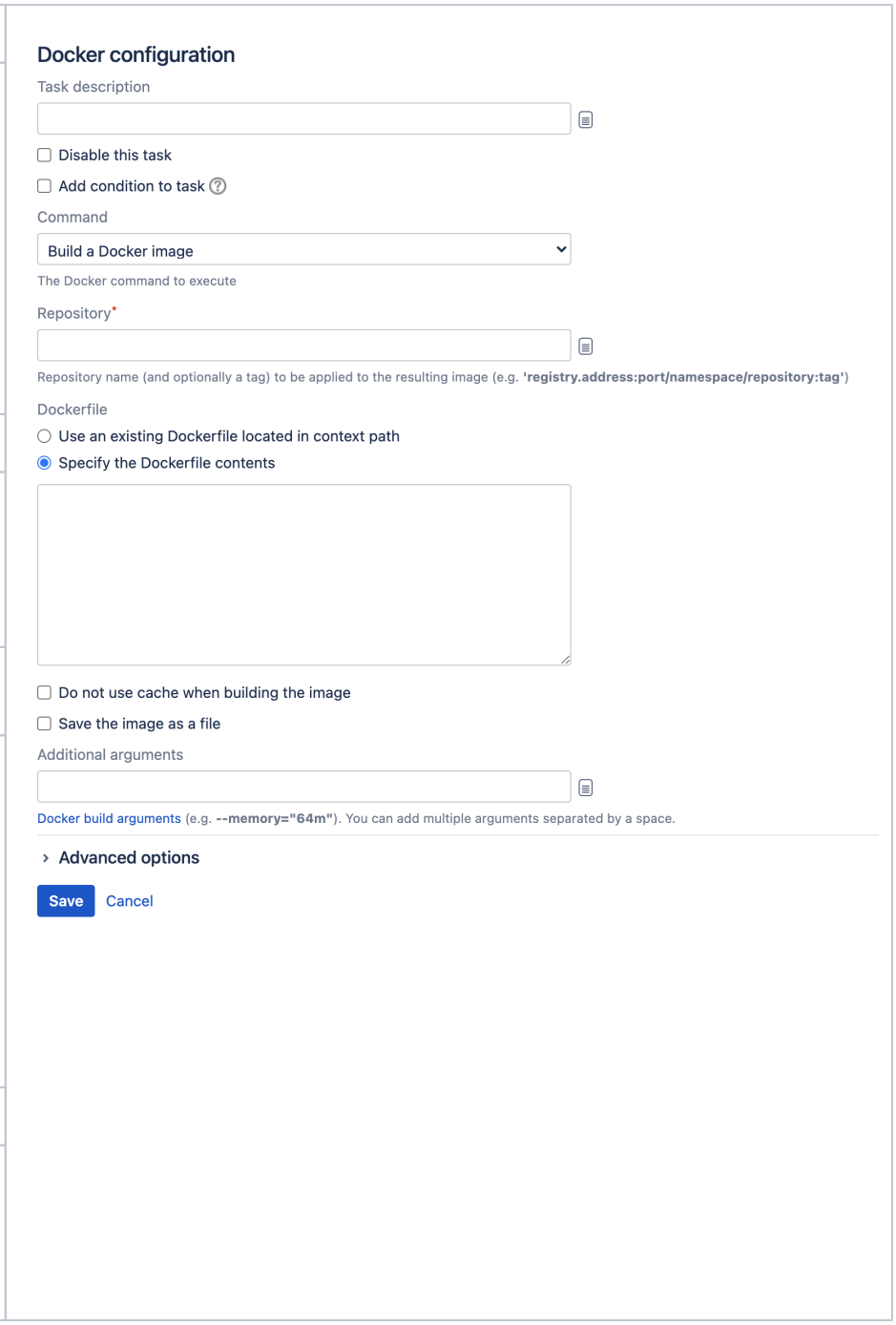
- Build a Docker image
- Run a Docker container
- Push a Docker repository to a Docker registry

Build a Docker image

Builds a Docker image based on the specified Dockerfile. The Dockerfile may be provided as an existing file in the task's working directory or defined in the task configuration.The image is stored in Docker's local image installation directory and can be used by subsequent Docker tasks in the job. You can optionally save the image to a file in the working directory which can then be packaged as a build artifact.

To build a Docker image in Bamboo:

1. Create a new Docker task for the relevant job. See Configuring tasks.
2. Add a **Task description**.
3. Use the **Disable this task** checkbox to control whether the task gets run.
4. Use the **Add condition to task** checkbox to make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
5. Select the **Build a Docker image** command and complete the settings. See more information about the settings below:

Repository	
The repository name (and optionally a tag) to be applied to the resulting image, following this pattern: `registry.address:port/namespace/repository:tag` Only `repository` is mandatory.	
Dockerfile	
Use either an existing Dockerfile (located in the working directory for the task), or specify the contents of the Dockerfile.	
Do not use cache when building the image	
By default, Docker will reuse a cached build during the next build. See the Docker documentation. Select **Do not use cache...** t o ensure that the new image will include changes since the last build. Note that this may incur a performance penalty.	
Save the image as a file	
Specify the directory location and file name. Optionally configure a job artifact to pass it to next stages and deployments.	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working subdirectory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Save your changes!

Run a Docker container

Starts a Docker container based on the specified image.

By default, the task's working directory is mounted and used as the Docker container's working directory, but you can specify your own settings.

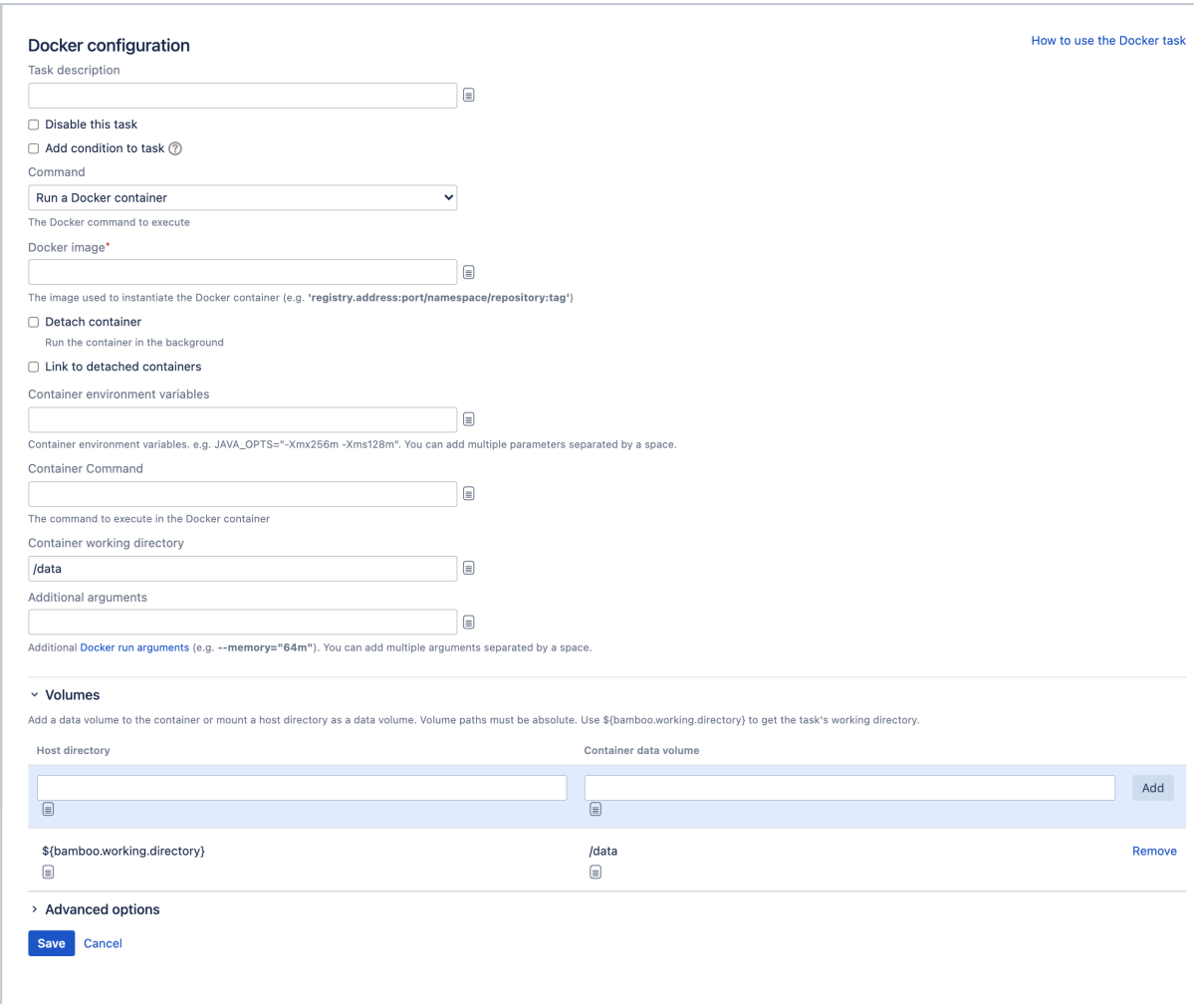
By default, the container is removed on completion of the task, but you can select **Detach container** to have the container continue to run after a deployment project completes. Containers can be linked to detached containers started by preceding tasks in a job by selecting the Link to detached containers option.

Note that a non-detached container that fails to start will not be removed when the Bamboo task completes. See this KB article for more details.

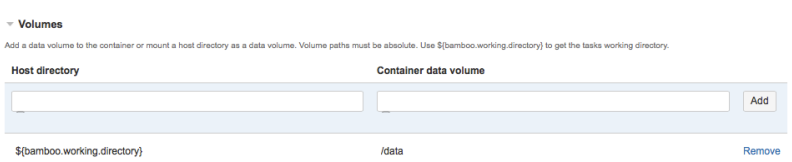
To run a Docker container in Bamboo:

1. Create a new Docker task for the relevant job. See Configuring tasks for details.
2. Add a **Task description** to help remind you why you created the task.
3. Use the **Disable this task** checkbox to control whether the task gets run.
4. Use the **Add condition to the task** checkbox to make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
5. Select the **Run a Docker container** command and complete the settings. See more information about the settings below:

Docker image	The image you want to use to instantiate the Docker container.
Detach container	



Allows you to run the container in the background, after a deployment project completes.



Specify a **Container name** that isn't used by other containers in this job.

Select **Add port mapping** to specify mappings that bind ports inside the container to ports on the host.

Wait for service to start	Allows you to specify how long Bamboo should wait for the service to become available. You need to specify a pattern for the URL that Bamboo should check, and a timeout period.
Link to detached containers	

Allows you to link containers to detached containers started by preceding tasks in a job.	
Container environment variables	

Allows you to specify parameters to pass to the container, for example `JAVA_OPTS="-Xmx256m -Xms128m"`. Separate multiple parameters with spaces. Parameters with spaces must be quoted.	
Container command	
The command to run in the Docker container.	

Container working directory	The working directory for the container.
Additional arguments	Additional Docker run options. Argument string that have value which co

```
n
t
ai
n
s
p
a
c
e
s
r
e
q
ui
r
e
t
h
e
e
n
ti
re

a
r
g
u
m
e
n
t
st
ri
n
g
b
e
c
o
n
t
ai
n
e
d
w
it
hi
n
q
u
o
t
e
s,
t
h
a
t
is
,
t
h
```

e
p
a
r
a
m
e
t
e
r
m
a
rk
e
r,
t
h
e
p
a
r
a
m
e
t
e
r,
t
h
e
e
q
u
al
s
si
g
n
,
a
n
d
t
h
e
v
al
u
e
c
o
n
t
ai
ni
n
g
s
p
a
c
e
s.
F

```
or example: "--health-cmd=mysqladmin ping || exit 1"
```

Volumes

You can mount additional host directories as data volumes inside the container.	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working subdirectory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

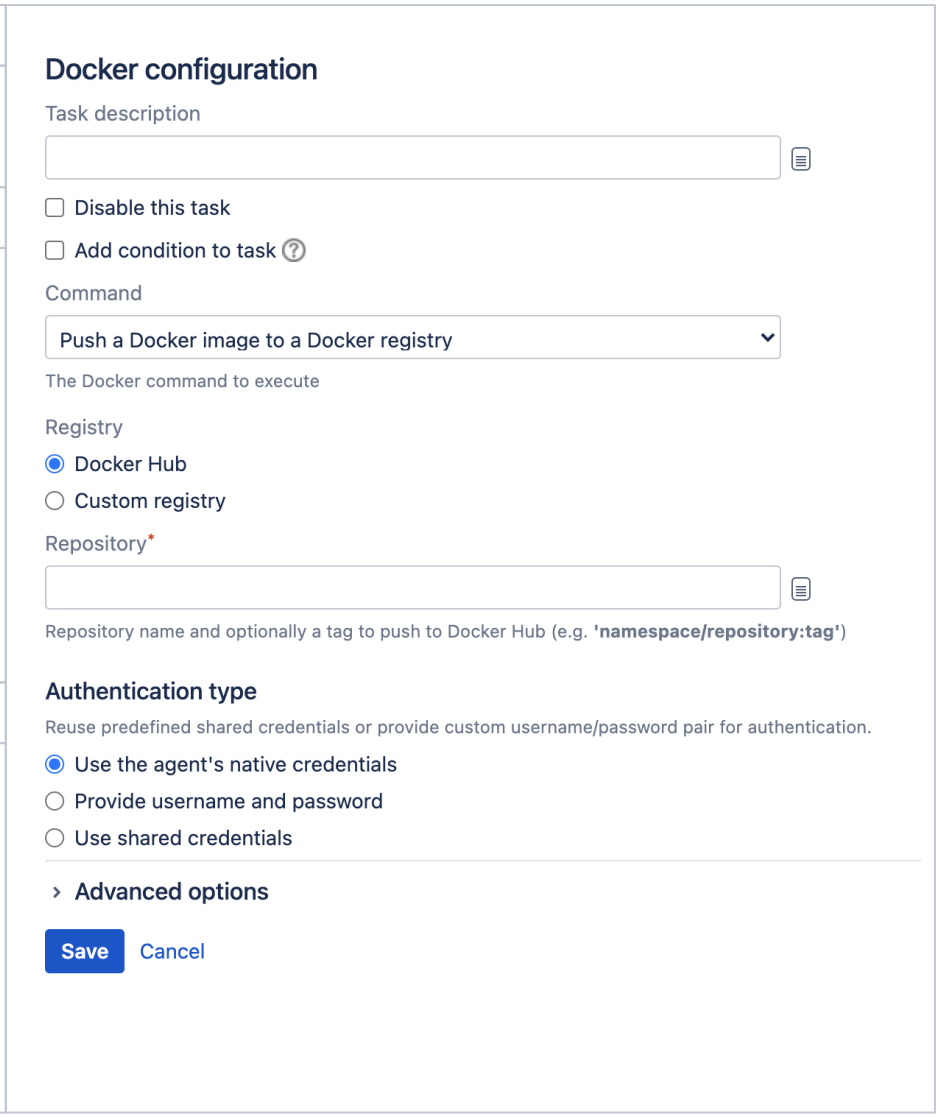
Save your changes!

Push a Docker image to a Docker registry

Pushes a Docker image to a Docker registry. This may be the central Docker Hub registry or a custom registry.

To push a Docker repository from Bamboo to a registry:

1. Create a new **Docker** task for the relevant job. See Configuring tasks for details.
2. Add a **Task description** to help remind you why you created the task.
3. Use the **Disable this task** checkbox to control whether the task gets run.
4. Use the **Add condition to the task** checkbox to make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.
5. Select the **Push a Docker image to a Docker registry** command and complete the settings. See more information about the settings below:

Registry	
Select to push to either Docker Hub or a custom registry.	
Repository	
For Docker Hub you must specify the repository name and optionally a tag, for example: `namespace/repository:tag`For a custom registry you must specify the registry address, repository name and optionally a tag, for example: `registry.address:port/namespace/repository:tag`	
Authentication type	
Select how you'd like to authenticate: • use agents' `~/.docker/config.json` file • use username and password • select predefined shared credentials from the drop-down list	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working subdirectory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Save your changes!

Advanced authentication

The push and pull tasks allow you to define a username and password for authentication purposes.

If the other tasks require authentication, or if you want to share docker credentials between all builds for certain agents, it's possible to create the docker configuration file `~/.docker/config.json` on the agent itself.

> Bamboo will only use the contents of `~/.docker/config.json` if you've selected **Use the agent's native credentials** as the preferred authentication type for the Docker task. Otherwise, if present, Bamboo will copy the contents of the original configuration file to a temporary location each time the `docker` command is invoked so it can be adjusted to the provided credentials.

The following is an example of the contents of `~/.docker/config.json`:

```
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "ZG9ja2VyaHVidXNlcjpkb2NrZXJodWJwYXNz"
                },
                "https://index.example.com": {
                    "auth": "XhhbXBsZXVzZXI6ZXhhbXBsZXBhc3M="
                }
        }
}
```

When using the push and pull tasks, leave the authentication fields empty in order to use `~/.docker/config.json` instead.

Troubleshooting

No space left on device

Docker stores its images in a local image installation directory. Over time this directory may grow to consume all of the available disk space. When this occurs you should remove unused images by running the `docker rmi` command.

The following Docker issues affecting disk space may provide further information:

- Device-mapper does not release free space from removed images
- Graph deletes are non-atomic, db refs deleted without deleting on-disk entities

Permission denied on files created within a Docker container

Docker runs processes inside containers as the root user. This means files created on mounted volumes are owned by the root user and not by the user running the Docker command (the bamboo agent user). This may cause an issue if a subsequent task requires access to those files on the host.

Docker plans to allow mapping between container and host users in the future. Until then, you can work around this issue by changing the owner of the files in the mounted volume to the host user:

- Supply the host user's id and group id to the container by setting the following environment variables in the Docker run task configuration:
  - `HOST_UID=$UID`
  - `HOST_GID=$GID`
- Run a script inside the container to change the owner of the files in the mounted volume:

```
chown -R $HOST_UID:$HOST_GID /<path_to_mounted_volume>
```

Permission denied when running Docker

When attempting to run a Docker container you may see a permission denied issue:

```
2015/02/10 06:35:31 Post http:///var/run/docker.sock/build?rm=1&t=docker-toy-demo: dial unix /var/run
/docker.sock: permission denied
```

The solution is to add the Bamboo user agent to the Docker group on the agent.

Getting execution errors for valid docker files or unable to start docker container

Example build output:

```
 Driver devicemapper failed to get image rootfs
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158: Error mounting '/dev/mapper/docker-202:16-
17252355-511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158' on '/mnt/docker/devicemapper/mnt
/511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158': invalid argument
simple    12-Feb-2015 12:12:14    Failing task since return code of [/usr/bin/docker build --no-cache=true
--tag="docker.atlassian.io/dk:9.3" /home/bamboo/bamboo-agent-home/xml-data/build-dir/dkr-build-JOB1] was 1
while expected 0
error    12-Feb-2015 12:12:14    Error occurred while running Task 'Build docker image(5)' of type com.
atlassian.bamboo.plugins.bamboo-docker-plugin:task.docker.cli.
error    12-Feb-2015 12:12:14    com.atlassian.bamboo.task.TaskException: Failed to execute task
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.service.BuildService.execute
(BuildService.java:53)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.tasks.cli.DockerCliTask.execute
(DockerCliTask.java:60)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.task.TaskExecutorImpl$3.call(TaskExecutorImpl.
java:281)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.task.TaskExecutorImpl$3.call(TaskExecutorImpl.
java:278)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.task.TaskExecutorImpl.
executeTaskWithPrePostActions(TaskExecutorImpl.java:198)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.task.TaskExecutorImpl.executeTasks
(TaskExecutorImpl.java:278)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.task.TaskExecutorImpl.execute(TaskExecutorImpl.
java:105)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.build.pipeline.tasks.ExecuteBuildTask.call
(ExecuteBuildTask.java:75)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.v2.build.agent.DefaultBuildAgent.build
(DefaultBuildAgent.java:188)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.v2.build.agent.BuildAgentControllerImpl.
waitAndPerformBuild(BuildAgentControllerImpl.java:112)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.v2.build.agent.DefaultBuildAgent$1.run
(DefaultBuildAgent.java:110)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.utils.BambooRunnables$1.run(BambooRunnables.
java:49)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.security.ImpersonationHelper.runWith
(ImpersonationHelper.java:31)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.security.ImpersonationHelper.
runWithSystemAuthority(ImpersonationHelper.java:20)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.security.ImpersonationHelper$1.run
(ImpersonationHelper.java:52)
error    12-Feb-2015 12:12:14        at java.lang.Thread.run(Thread.java:745)
error    12-Feb-2015 12:12:14    Caused by: com.atlassian.bamboo.plugins.docker.client.DockerException:
Error running Docker build command
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.client.DockerCmd.build
(DockerCmd.java:149)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.service.BuildService.execute
(BuildService.java:40)
error    12-Feb-2015 12:12:14        ... 15 more
error    12-Feb-2015 12:12:14    Caused by: com.atlassian.utils.process.ProcessException: Error executing
/usr/bin/docker build --no-cache=true --tag="docker.atlassian.io/dk:9.3" /home/bamboo/bamboo-agent-home/xml-
data/build-dir/DDT-REP-JOB1
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.process.
DockerTaskProcessService.execute(DockerTaskProcessService.java:57)
error    12-Feb-2015 12:12:14        at com.atlassian.bamboo.plugins.docker.client.DockerCmd.build
(DockerCmd.java:145)
error    12-Feb-2015 12:12:14        ... 16 more
```

If the agent consistently fails executing docker run commands, either when building an image or running an instance there is a risk that you've run into https://github.com/docker/docker/issues/4036. To help diagnose this you can SSH to the agent and look at the kernel messages by running:

```
dmesg
```

There are several possible messages that indicate this problem. Some of those are listed here:

```
[83471099.881879] JBD2: no valid journal superblock found
[83471099.881883] EXT4-fs (dm-2): error loading journal
[88401612.723018] EXT4-fs (dm-1): warning: mounting fs with errors, running e2fsck is recommended
[88401612.724764] EXT4-fs (dm-1): mounted filesystem with ordered data mode. Opts: discard
[88401612.744549] EXT4-fs error (device dm-1): ext4_lookup:1448: inode #2: comm docker: deleted inode
referenced: 131073
```

There is a big risk that the device mapper is corrupt. This means that you need to stop Docker and remove the files used by devicemapper, then restart Docker. If running on an elastic agent, terminating the agent and starting a new one is also a viable option.

To stop Docker and remove the files, run the following:

```
sudo -i
#stop the docker daemon
service docker stop
#remove the broken devicemapper files
rm -rf /var/lib/docker
service docker start
```


The location of the devicemapper files may differ from the example above. Run the following to find the exact path:

```
docker info
```

Building a Docker image in Bamboo

In Bamboo, you can build a Docker image based on the specified Dockerfile. The Dockerfile may be provided as an existing file in the task's working directory or defined in the task configuration.

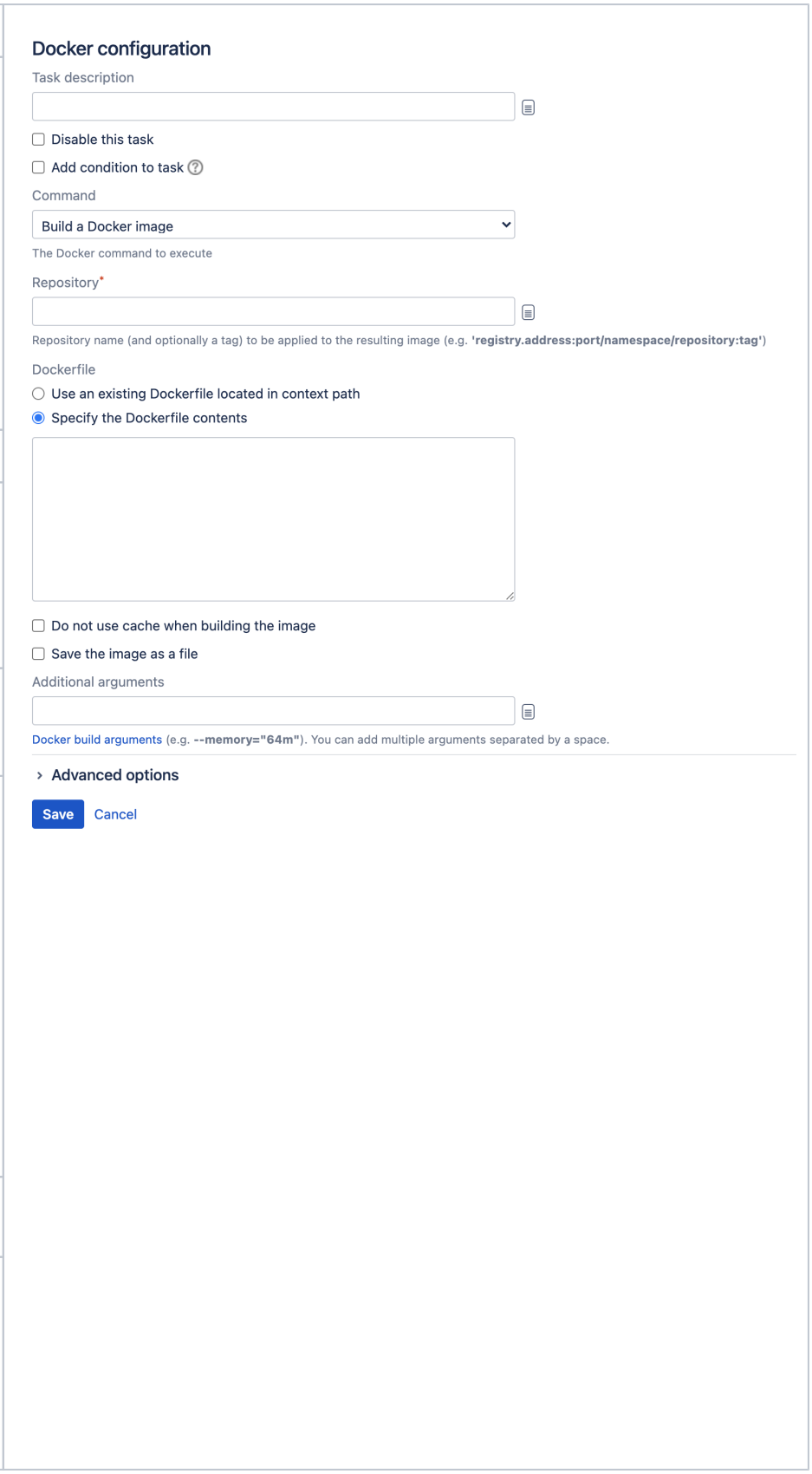
The image is stored in Docker's local image installation directory and can be used by subsequent Docker tasks in the job. You can optionally save the image to a file in the working directory which can then be packaged as a build artifact.

Before you begin

- Make sure you have Docker installed. We advise to use the most recent version.
- Define a Docker capability in Bamboo. See Defining a new Docker capability
- If you're using Bamboo on Windows, you can't run Docker commands directly from the Windows command line. To use Docker tasks with Bamboo Windows, run Docker Machine.

To build a Docker image in Bamboo:

1. In the job configuration screen, select **Add task**.
2. Search for the Docker task type and select it.
3. (*optional*) For future reference, add a **Task description**.
4. (*optional*) Use the **Disable this task** checkbox to control whether your task gets run.
5. (*optional*) Use the **Add condition to task** checkbox to make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.
6. From the **Repository** drop-down list, select the **Build a Docker image**.
7. Complete the task settings.

Repository	
The repository name (and optionally a tag) to be applied to the resulting image, following this pattern: `registry.address:port/namespace/repository:tag` Only `repository` is mandatory.	
Dockerfile	
Use either an existing Dockerfile (located in the working directory for the task), or specify the contents of the Dockerfile.	
Do not use cache when building the image	
By default, Docker will reuse a cached build during the next build. See the Docker documentation. Select **Do not use cache...** to ensure that the new image will include changes since the last build. Note that this may incur a performance penalty.	
Save the image as a file	
Specify the directory location and file name. Optionally configure a a job artifact to pass it to next stages and deployments.	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working sub directory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

8. Select **Save**.

Pulling a Docker image from a registry

You can pull a Docker image from a Docker registry. This may be the central Docker Hub registry or a custom registry.

Before you begin

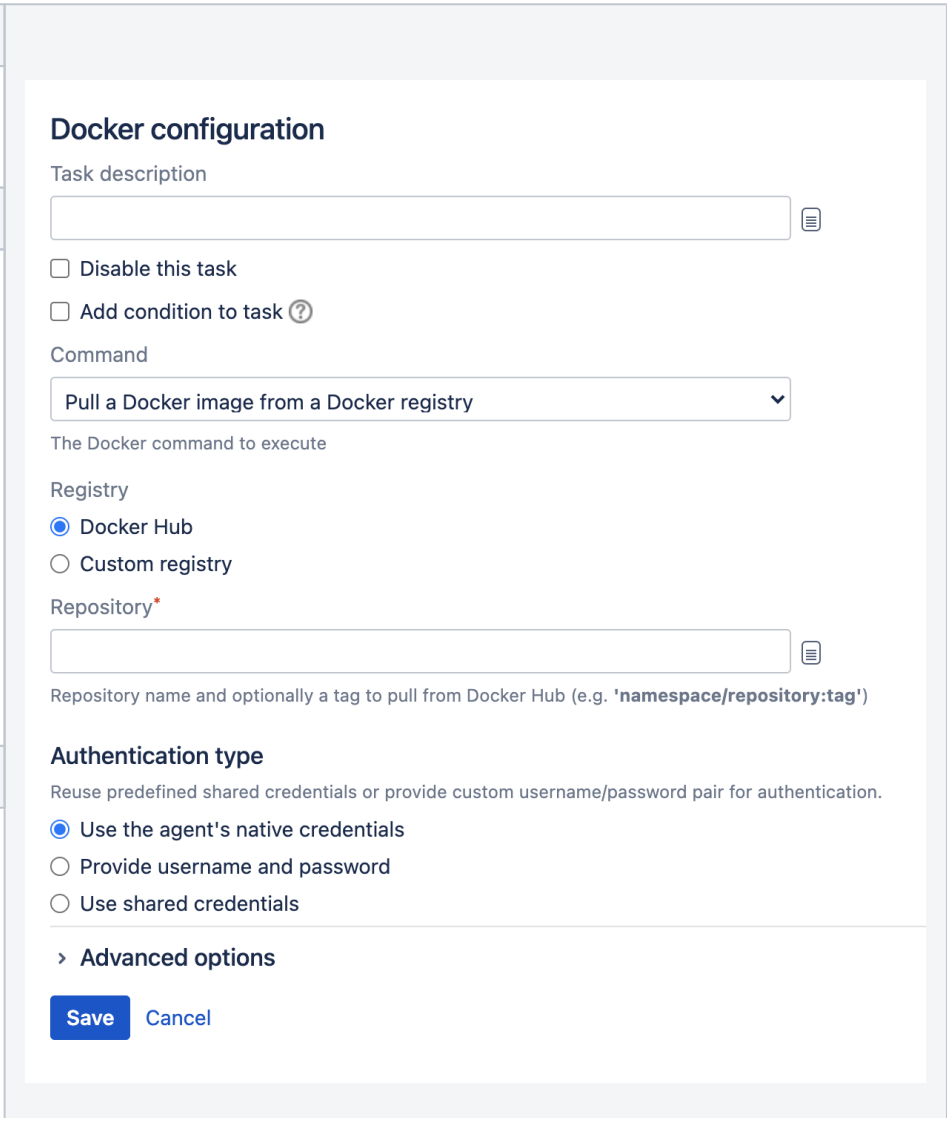
- Make sure you have Docker installed. We advise to use the most recent version.
- Define a Docker capability in Bamboo. See Defining a new Docker capability
- If you're using Bamboo on Windows, you can't run Docker commands directly from the Windows command line. To use Docker tasks with Bamboo Windows, run Docker Machine.

To pull a Docker image from a registry

> ⚠ Authentication tokens are valid only for 12 hours. Generate a new token using a script task before pulling any image from the registry.

1. In the job configuration screen, select **Add task**.
2. Search for the Docker tasks type and select it.
3. (*optional*) For future reference, add a **Task description**.
4. (*optional*) Use the **Disable this task** checkbox to control whether your task is run.
5. (*optional*) Use the **Add condition to task** checkbox to make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.
6. From the **Repository** drop-down list, select **Pull a Docker image from a Docker registry**.
7. Complete the settings:

Registry	Select to push to either Docker Hub or a custom registry.
Repository	For Docker Hub you must specify the repository name and optionally a tag, for example: `namespace/repository:tag`For a custom registry you must specify the registry address, repository name and optionally a tag, for example: `registry.address:port/namespace/repository:tag`
Authentication type	



Select how you'd like to authenticate: • use agents' `docker cfg` file • use username and password • select predefined shared credentials from the drop-down list	
Advanced options *(optional)*	
Environment variables Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").	
Working subdirectory An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory	

8. Select **Save.**

Pushing a Docker image to a registry

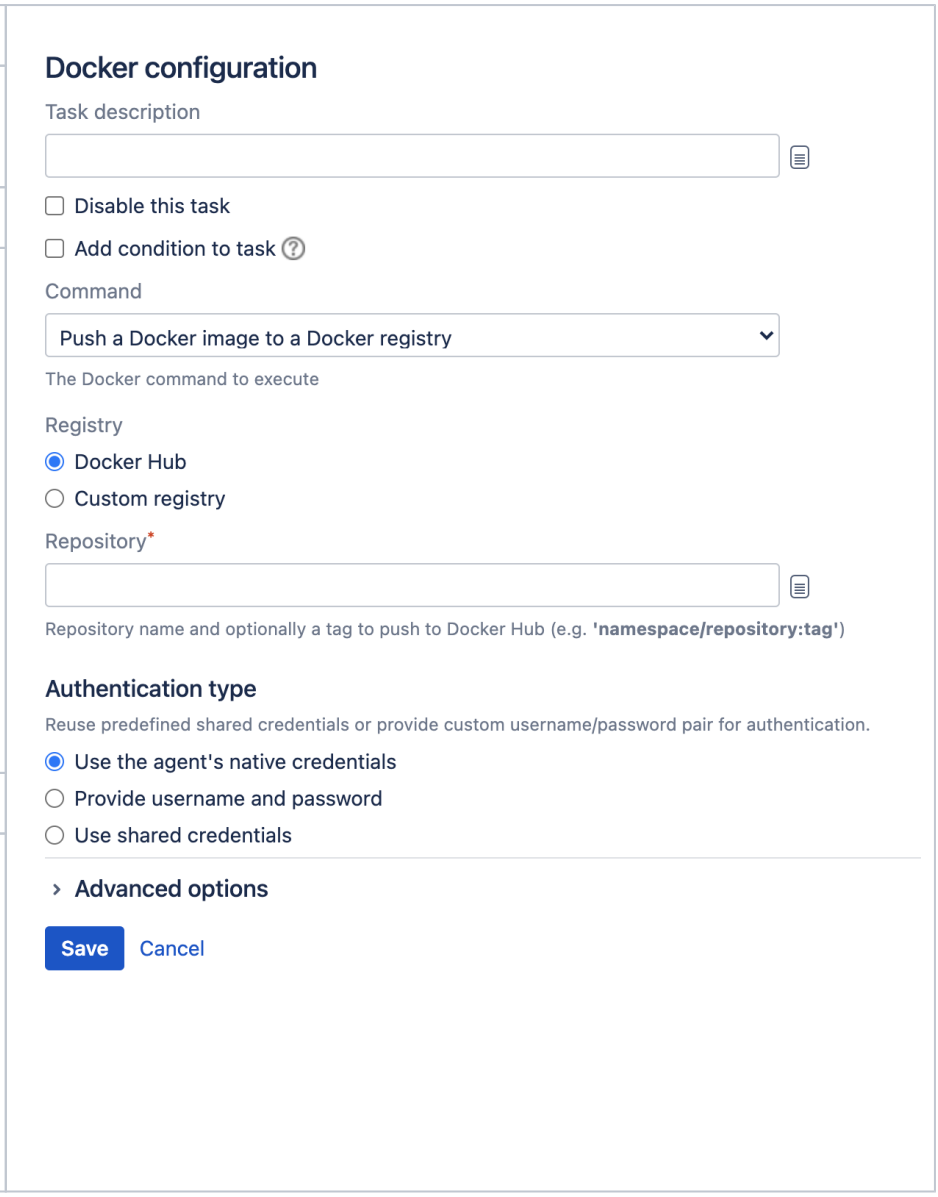
You can push a Docker image to a Docker registry. This may be the central Docker Hub registry or a custom registry.

Before you begin

- Make sure you have Docker installed. We advise to use the most recent version.
- Define a Docker capability in Bamboo. See Defining a new Docker capability
- If you're using Bamboo on Windows, you can't run Docker commands directly from the Windows command line. To use Docker tasks with Bamboo Windows, run Docker Machine.

To push a Docker repository from Bamboo to a registry:

1. In the job configuration screen, select **Add task**.
2. Search for the Docker tasks type and select it.
3. (*optional*) For future reference, add a **Task description**.
4. (*optional*) Use the **Disable this task** checkbox to control whether your task gets run.
5. (*optional*) Use the **Add condition to task** checkbox to make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.
6. From the **Repository** drop-down list, select **Push a Docker image to a Docker registry**.
7. Complete the settings:

Registry Select to push to either Docker Hub or a custom registry.	
Repository For Docker Hub you must specify the repository name and optionally a tag, for example: `namespace/repository:tag` For a custom registry you must specify the registry address, repository name and optionally a tag, for example: `registry.address:port/namespace/repository:tag`	
Authentication type Select how you'd like to authenticate: • use agents' dockercfg file • use username and password • select predefined shared credentials from the drop-down list	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working subdirectory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Save your changes!

Running a Docker container in Bamboo

By default, the task's working directory is mounted and used as the Docker container's working directory, but you can specify your own settings.

By default, the container is removed on completion of the task, but you can select **Detach container** to have the container continue to run after a deployment project completes. Containers can be linked to detached containers started by preceding tasks in a job by selecting the Link to detached containers option.

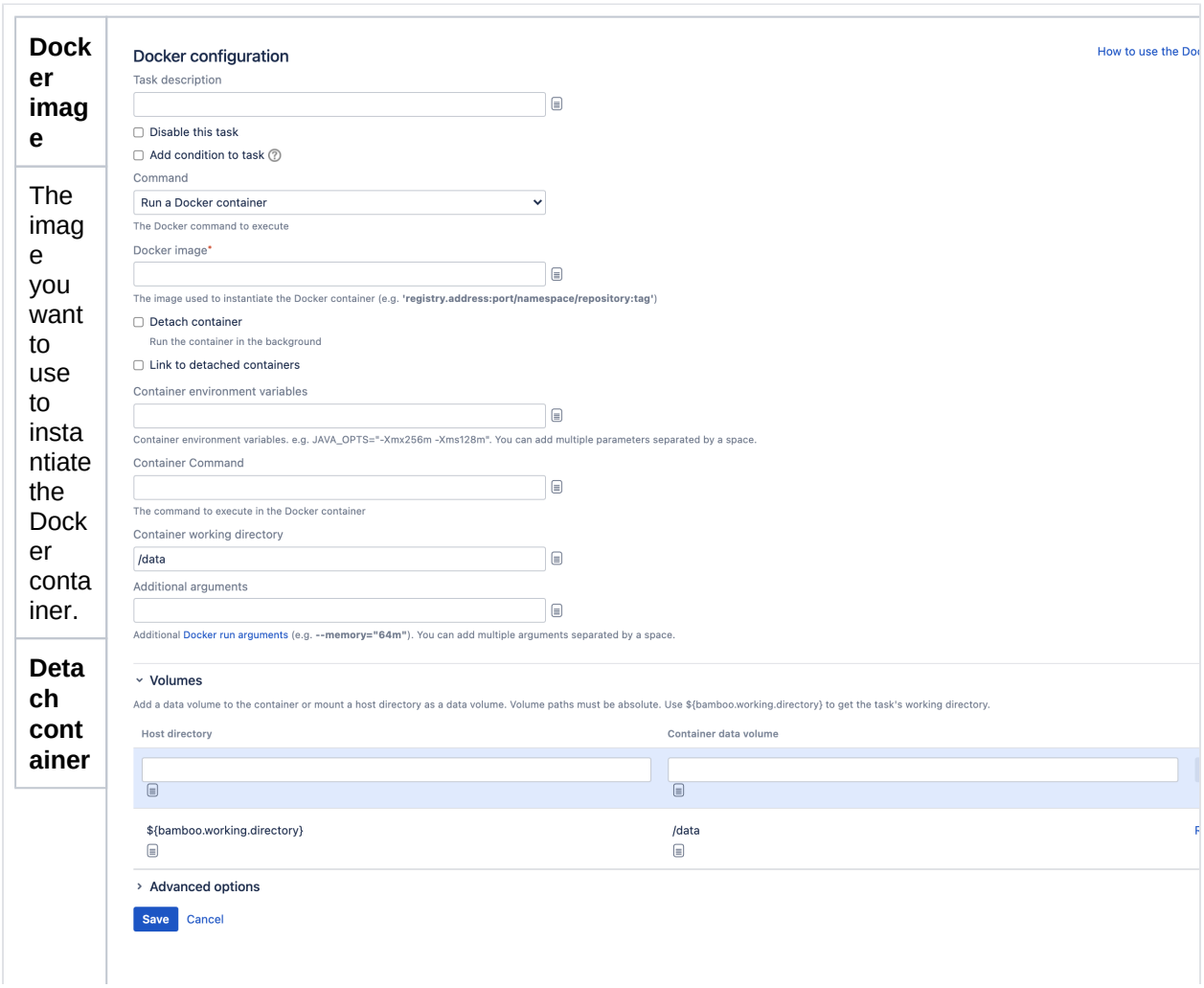
Note that a non-detached container that fails to start will not be removed when the Bamboo task completes. See this KB article for more details.

Before you begin

- Make sure you have Docker installed. We advise to use the most recent version.
- Define a Docker capability in Bamboo. See Defining a new Docker capability
- If you're using Bamboo on Windows, you can't run Docker commands directly from the Windows command line. To use Docker tasks with Bamboo Windows, run Docker Machine.

To run a Docker container in Bamboo:

1. In the job configuration screen, select **Add task**.
2. Search for the Docker tasks type and select it.
3. (*optional*) For future reference, add a **Task description**.
4. (*optional*) Use the **Disable this task** checkbox to control whether your task gets run.
5. (*optional*) Use the **Add condition to task** checkbox to make task run only when a certain condition is met.
   You can find conditions on Atlassian Marketplace or implement your own.
6. Select the **Run a Docker container** command and complete the settings. See more information about the settings below:



Allows you to run the container in the background, after a deployment project completes. Specify a **Container name** that isn't used by other containers in this job. Select **Add port mapping** to specify mappings that bind ports inside the container to ports on the host.

Wait for service to start	Allows you to specify how long Bamboo should wait for the service to become available. You need to specify a pattern for the URL that Bamboo should check, and a timeout period.
Link to detached containers	

Allows you to link containers to detached containers started by preceding tasks in a job.	
Container environment variables	

Allows you to specify parameters to pass to the container, for example `JAVA_OPTS="-Xmx256m -Xms128m"`. Separate multiple parameters with spaces. Parameters with spaces must be quoted.	
Container command	
The command to run in the Docker container.	

Container working directory	The working directory for the container.
Additional arguments	Additional Docker run options. Argument string that have value which co

```
ntainspacesrequirethe

entire

argumentstringbecontainedwithinquotes,thatis,
```

```
the parameter marker, the parameter, the equals sign, and the value containing space
```

s. For example: "--health-cmd=mysqladmin ping || exit 1"

Volumes

You can mount additional host directories as data volumes inside the container.	

If required, specify advanced options:

Environment variables
(Optional) Additional system environment variables that you want to pass to your build. Note that existing environment variables are automatically available to the executable. You can also include Bamboo global or build-specific variables (see Bamboo variables). Multiple variables should be separated with spaces. Parameters with spaces must be quoted (e.g JAVA_OPTS="-Xms200m -Xmx700m").
Working subdirectory
(Optional) An alternative subdirectory, relative to the job's root directory, where Bamboo will run the executable. The root directory contains everything checked out from the job's configured source repository. If you leave this field blank, Bamboo will look for build files in the root directory. This option is useful if your task has a build script in a subdirectory and the executable needs to be run from within that subdirectory.

Save your changes!

Configuring a Source Control task

Source Control tasks is an umbrella term for all Repository tasks available in Bamboo. Source Control tasks allow you to quickly apply your changes to repositories. Source Control tasks work in deployment and build plans and they are fully configurable using Bamboo Specs.

Bamboo provides support for Source Control tasks in the following repositories:

- Bitbucket Cloud
- Bitbucket Server
- Git

  > **Git LFS is supported** for Source Code Checkout, Repository Commit, and Repository Push tasks.

- Mercurial

Here is a list of Source Control tasks available in Bamboo:

Icon	Task	Description
	Source Code Checkout	Task for **checking out** (cloning) a remote repository. You can read more about this task in Checking out code.
	Repository Commit	*Available in Bamboo 6.7.1 and later.* Task capable of **committing and pushing** changes to a remote repository. This task will take all modified files in repository directory, commit them with a given message, and then push them to the remote location. In Bamboo 6.7.1 changes made through **Repository Commit** tasks were recorded by Bamboo and they triggered a plan. in 6.7.2 changes made through this task type are still recorded but they don't trigger any plan. Change detection simply ignores them.

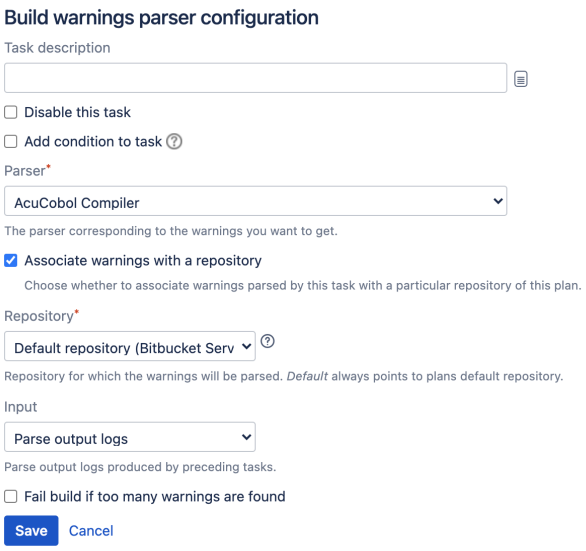
	Repository Push	*Available in Bamboo 6.7.1 and later.* Task capable of **pushing** commits to a remote repository. To be used in place of the VCS Commit task if the commits were already created and only the 'push' part is needed. The concept of push only exists in DVCS repositories. Here are some use examples of the Repository Push task: • pushing commits created by other tasks over which the user has no control, • pushing custom commits (e.g. cryptographically signed or with a custom author), • pushing merge results, • transactional processing through your build (multiple commits with a single push at the end). ⓘ In Bamboo 6.7.1 changes made through **Repository Push** tasks were recorded by Bamboo and they triggered a plan. in 6.7.2 changes made through this task type are still recorded but they don't trigger any plan. Change detection simply ignores them.
	Repository Branch (previously known as: VCS Branching)	Task capable of **creating a branch and pushing** it to a remote repository. This task will create a new branch with specified name from the latest commit in the checkout directory. For Mercurial, it will create a new commit "Creating branch...". ⓘ Cannot be used with the Git implementation embedded in Bamboo. (You need to have set up native Git).
	Repository Tag (previously known as: VCS Tagging)	Task capable of **creating a tag and pushing** it to a remote repository. This task will create a new tag with specified name for the latest commit in the checkout directory. For Mercurial, it will create a new commit "Adding tag...". ⓘ Cannot be used with the Git implementation embedded in Bamboo. (You need to have set up native Git).

Configuring Build warnings parser task

Use Build warnings parser tasks to scan build logs and output files for compiler warning. Warnings are aggregated into a build artifact and the summary of the warnings is displayed in the build result page.

To create a Build warnings parser task:

1. In the job configuration screen, select the **Tasks** tab.
2. Select **Add task**.
3. From the Builder type group, select **Build warnings parser**.
4. Configure the following settings:



Task description
A description of the task, which is displayed in Bamboo.

Disable this task
Check, or clear, to selectively run this task.

Add condition to task
Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.

Parser
Type of parser used by the task. Select the one that matches the compiler (or other tools) used in previous steps of the build.

Associate warnings with a repository
Warnings can be associated with a repository containing your sources. This information can be later used to notify the source control system about the amount and severity of warnings found by this task. By default, the warnings are linked to the default repository of the build.

Repository
Select the correct source repository to associate the warnings with.

Input
Where Bamboo should look for warnings. You can select between parsing the output logs (default) or the file matching *a glob pattern.*

Fail build if too many warnings are found
Select this option of causes build to fail if number of warnings exceed defined threshold.

Sharing artifacts

This page describes how to keep and share artifacts produced by a job, such as reports, websites or .jar files. Bamboo allows artifact sharing between:

- Jobs
- Build plans
- Build plans to deployment environments.

Related pages:

- Viewing a build's artifacts
- Configuring a job's build artifacts
- Pattern matching reference

Atlassian Blogs:

- Boost Your Build Automation with Artifact Sharing

Define an artifact to keep for a job

You can specify which artifacts to keep by setting up an artifact definition for the job. The artifacts will be available after each build of the job.

To set up a new artifact definition:

1. Navigate to the job, as described in Configuring jobs.
2. Select the **Artifacts** tab, and then **Create definition**:
    a. Specify a **Name** for the artifact.
    Names can contain only alphanumeric characters, underscores, dashes, and spaces.
    b. In **Location**, specify the folder relative to the build directory, where the artifact will be located. Don't use the absolute path to the artifact. Wildcards are not supported.
    c. In **Copy pattern(s)**, specify the name (or Ant file copy pattern) of the artifact or artifacts you want to keep relative to **Location**. For example, if you want to keep the latest version of a .jar file, you could specify **Copy pattern** to be `*/.jar` and the **Location** to be `target`.
    d. In **Exclusion pattern(s)**, specify the name (or Ant file copy pattern) of the artifact or artifacts that you want to exclude. For example: `**/*.jar`.
    e. Select the **Shared** check box if you want to share artifacts with other jobs in the plan.
    f. Select the **Required** check box if you want to fail the build if the artifact cannot be published.
    g. Select the **Compression** check box if you want to compress the artifact when it's transferred between an agent and the server.
3. Select **Save**.

Artifacts are copied to a subdirectory (`/JOB_KEY/download-data/`) under your Build Directory folder – see Locating important directories and files.

Sharing artifacts between jobs

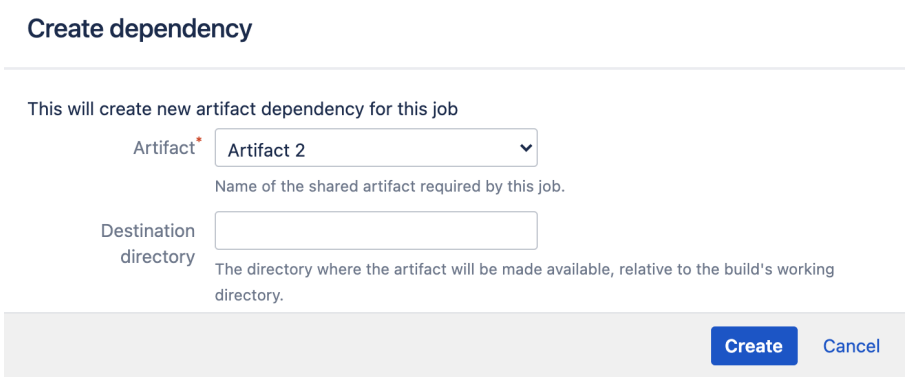
You can share artifacts between jobs in different stages using artifact dependencies. For example, you may want to run acceptance tests on a build, sharing the same WAR from one job to another without rebuilding it each time.

Each time the artifact is shared with a subsequent job, it is copied to the job's agent.

To share an artifact between two jobs in different stages:

1. Navigate to the configuration pages for the job that will produce the artifact, as described in Configuring jobs, and select the **Artifacts** tab (see Configuring a job's build artifacts).
2. Either select **Share** for an existing artifact, or create a new artifact definition, as described above.
3. Navigate to the job in a subsequent stage that will consume the artifact, and select the **Artifacts** tab.

4. Select **Create dependency**, then:
    - Select from the **Artifact** list.
    - Specify the **Destination directory**, then select **Create**.



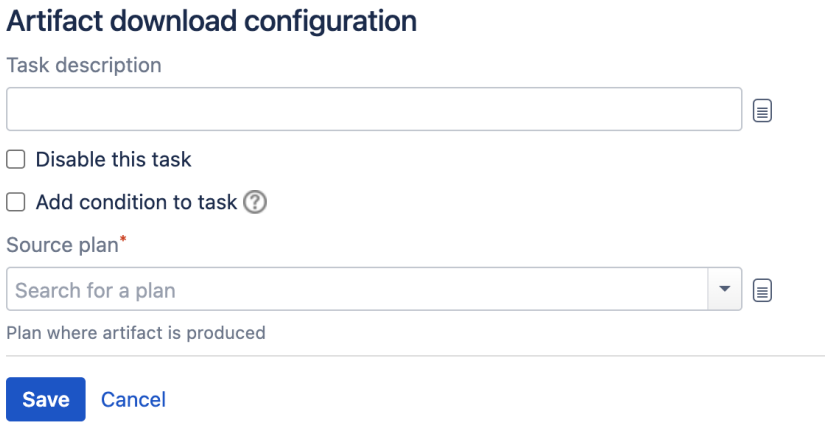
1. The **Artifact** list only shows artifacts from jobs in previous stages that have been marked as shared. This is described in Configuring a job's build artifacts.
2. **Destination directory** is relative to the build directory. Do not use the absolute path to refer to the destination directory.
3. The artifact from the most recent successful build will be used. If there are no successful builds from the artifact-producing plan or the artifacts have expired, the artifact-consuming job will fail.

Sharing artifacts between build plans

You can share artifacts between different build plans, however you need to use the Artifact downloader task to do so. For example, you may want to run acceptance tests on a particular build from a different plan by sharing the same WAR from one plan to another without rebuilding it each time.

To share an artifact between two build plans:

1. Locate the build plan that you wish to associate an artifact with.
    a. Select **Actions** > **Configure plan**.
    b. Select **Stages & jobs** and select a job or create a new job if one does not already exist.
    c. Select the **Tasks** tab for the selected job.
2. Select **Add task** > **Artifact downloader**.



Complete the configuration using the following options:

Field	Description	Optional?
Task description	A brief description of the artifact downloader task.	✓
Disable this task	Select this checkbox to disable the task.	-
Add condition to task	Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.	✓
Source plan	The build plan that is the source of the artifact you need to download.	✗

3. Select **Add another artifact** to add another artifact to the download list. Alternatively, use the grey cross icon to delete an artifact from your configuration.
4. Select **Save**.

1. The **Artifact** dropdown menu only shows artifacts from jobs in previous stages that have been marked as shared. This is described in Configuring a job's build artifacts.
2. **Destination directory** is relative to the build directory. Do not use the absolute path to refer to the destination directory.

Sharing artifacts from a build plan to a deployment environment

You can also share artifacts from a build plan into a deployment environment. For example, you may wish to share a particular build result from a plan with a deployment environment. To do this, you need to add the Artifact downloader task to a deployment environment during or after the environment creation process.

To share an artifact from a build plan to a deployment environment:

1. Open your deployment project. Expand the relevant environment panel, and select **Edit tasks**. Select **Add task** > **Artifact download**.
2. Complete the configuration using the following options:

Field	Description	Optional?
Task description	A brief description of the artifact downloader task.	✓
Disable this task	Select this checkbox to disable the task.	-
Add condition to task	Make task run only when a certain condition is met. You can find conditions on Atlassian Marketplace or implement your own.	✓
Artifact name	Use the dropdown menu to locate the name of the artifact that you want to download.	✗
Destination path	The location of the working directory into which you want the artifact downloaded.	✓

3. Select **Add another artifact** to add another artifact to the download list.
4. Select **Save**.

1. The **Artifact** dropdown menu only shows artifacts from jobs in previous stages that have been marked as shared. This is described in Configuring a job's build artifacts.
2. **Destination directory** is relative to the build directory. Don't use the absolute path to refer to the destination directory.

Working with builds

The following pages contain information on working with your Bamboo builds.

- Working with build results
- Working with comments
- Working with labels
- Quarantining failing tests
- Setting up plan build dependencies
- Viewing test statistics for a job
- Reordering jobs in the build queue
- Stopping an active build

Working with build results

About builds

A build is the execution of either a plan or a job. The execution of a plan is referred to as a plan build and that of a job is a job build.

Related pages:

- Viewing a build result
- Deleting the results of a plan build
- Working with comments
- Working with labels
- Assigning responsibility for build failures
- Configuring build results expiry for a plan

About build results

Every completed build has a *build result*:

- Successful — the code compiled, with or without errors, and all tests completed successfully.
- Failed — either the code did not compile, or at least one test failed.
- Incomplete — the build was not completed, e.g. it may have been stopped manually.

Additionally,

- if the build result is Failed, and the previous build result was Successful, the build is labeled as Broken.
- if the build result is Successful, and the previous build result was Failed, the build is labeled as Fixed.

The latest build result for every plan is listed on the Dashboard. Bamboo can also send notifications and generate RSS feeds about build results.

Viewing a build result

The instructions on this page describe how to view the build results for a plan.

Every completed build has a *build result*:

- Successful — the code compiled, with or without errors, and all tests completed successfully.
- Failed — either the code did not compile, or at least one test failed.
- Incomplete — the build was not completed, e.g. it may have been stopped manually.

Additionally,

- if the build result is Failed, and the previous build result was Successful, the build is labeled as Broken.
- if the build result is Successful, and the previous build result was Failed, the build is labeled as Fixed.

Viewing the most recent build result for a plan

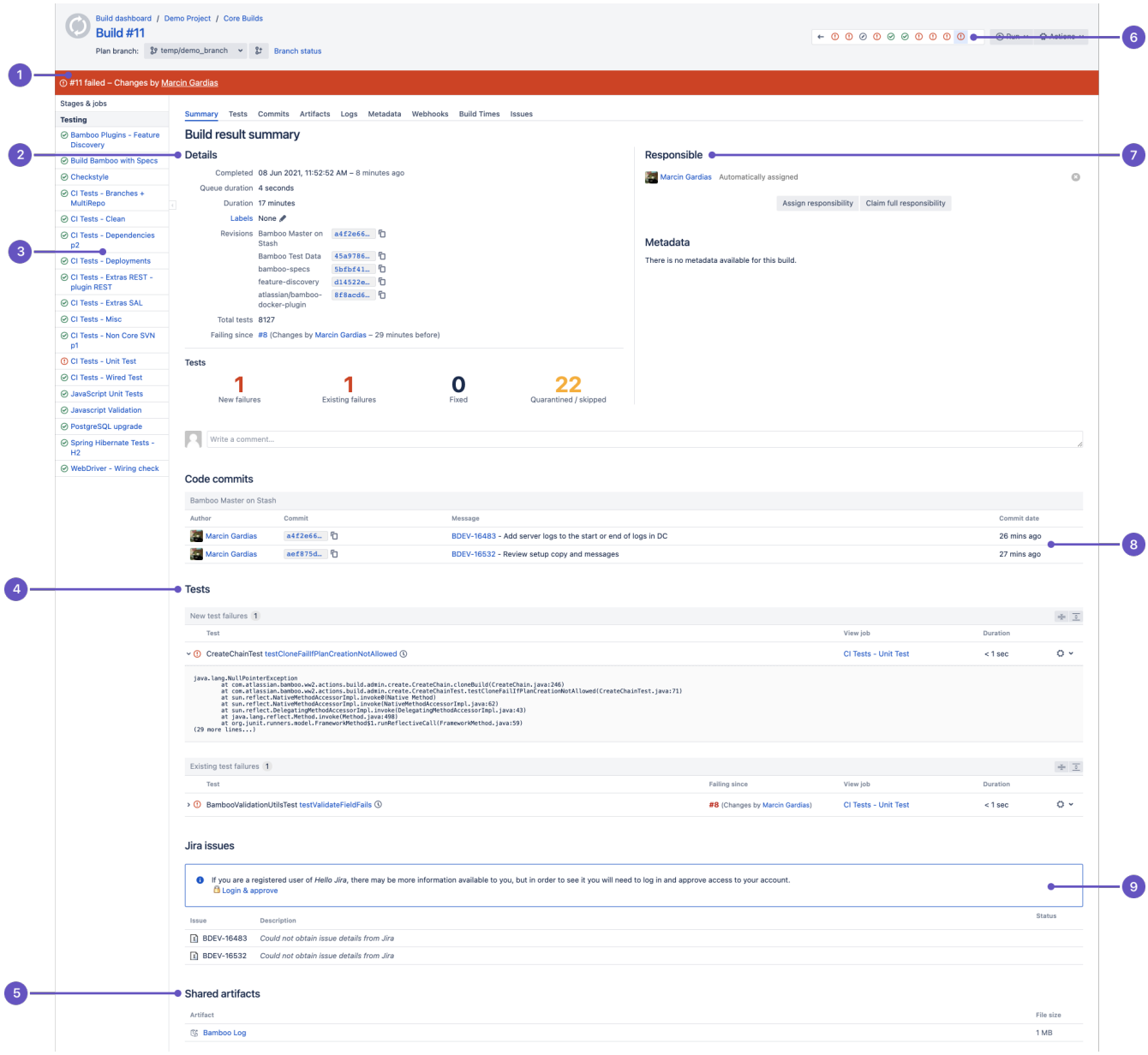
To view the most recent job build result of a plan:

1. From the top navigation bar select **Build** > **All build plans**, and select the build number.

On this page:

Related pages:

- Viewing test results for a build
- Viewing the code changes that triggered a build
- Viewing a build's artifacts
- Viewing a build log
- Viewing the metadata for a build result
- Viewing linked Jira application issues
- Reporting



1. **Status ribbon:** Did the build succeed or fail?
2. **Details:** Scan details of the build easily.
3. **Stages and jobs in the plan:** Scan the success of job builds. Select an icon to see details.
4. **Test summary:** Quickly see how many test are failing and how many were fixed in this build.
5. **Shared artifacts:** See the artifacts shared from this build and download them.
6. **History:** Scan the status of recent builds.
7. **Who is responsible?** Users who commit code are automatically assigned.
8. **Code changes:** See the code changes associated with this build.
9. **Jira issues:** See the JIRA issues related to this build. Select to go to the issue in Jira for details.

Tab	Description

Build summary	Displays a snapshot of the build result. • indicates a successful build. • indicates a build that was not completed. For example, it may have been stopped manually. • indicates a failed build. If a build has failed, you can run the entire build again or rerun just the failed stage.
Tests	Provides details of the build's test results.
Changes	Provides details of the code changes that triggered this build (if applicable).
Artifacts	Shows any artifacts relating to this build.
Logs	Displays a complete build log.
Metadata	Displays any metadata that relates to this build.
Build times	Displays a histogram of build times for jobs, and a list of which agents were used to build each job.
Issues	Provides details of the Jira issues linked to this build (if applicable). Availability depends on Bamboo's configuration.
Clover	Displays the Clover code-coverage that relates to this build (if applicable). The clover tab is located on the job level because a build can have more than one jobs, and each job might have different Clover results or not have clover tab at all. That's why in order to see the Clover tab, you need to drill down to the individual job that contains the clover report.

- You can assign responsibility for a broken build, either to yourself (select **Claim full responsibility**) or to someone else in your team (select **Assign responsibility**).

Viewing all build results for a plan

To view all build results for a plan:

1. From the top navigation bar select **Build** > **All build plans**, and select your plan.
2. The ten most recent builds will be displayed in the Recent history section on the **Plan summary** tab. See Viewing a plan's build information.
3. Select the **History** tab to view all builds for the plan.

Viewing all build results for a job

To view all build results for a job:

1. Navigate to the desired job, as described on Configuring jobs. The ten most recent builds will be displayed in the Recent history section of the **Job summary** tab.
2. Select the **History** tab to view all builds for the job.

Viewing test results for a build

Bamboo provides a convenient summary of all the tests that were run when a particular build was executed — as well as full details of any errors. This is useful when you are investigating what caused a build to fail.

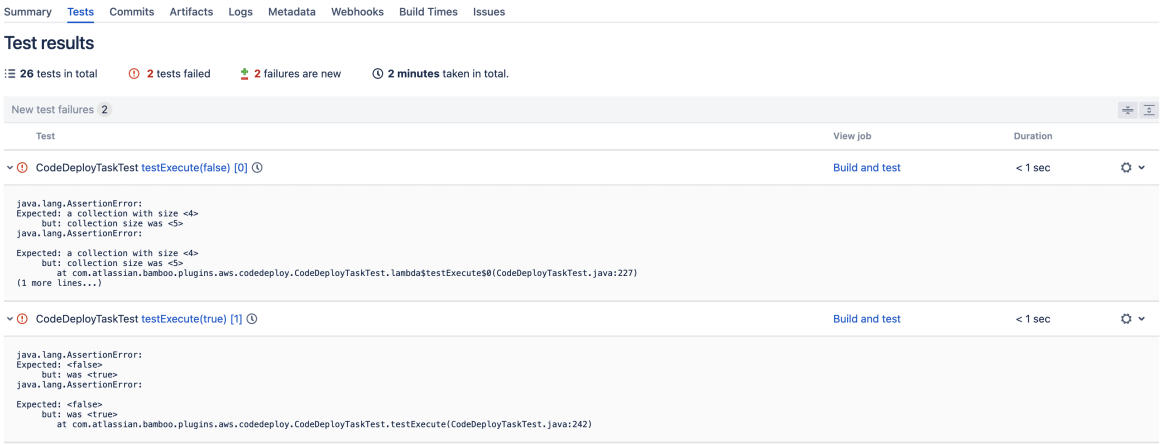
Note that for more meaningful display of test names within Bamboo, the word test is stripped out of test case name names if it occurs at the beginning, and capitals and underscores are treated as word separators.

Related pages:

- Viewing a test's history

To view the test results for a particular build:

1. Navigate to the build results for the plan or job, as described in Viewing a build result, and select the desired build result.
2. Go to the **Tests** tab.
    - Select the test name to see a particular test's results for other builds.



Viewing a test's history

A test's *history* shows you:

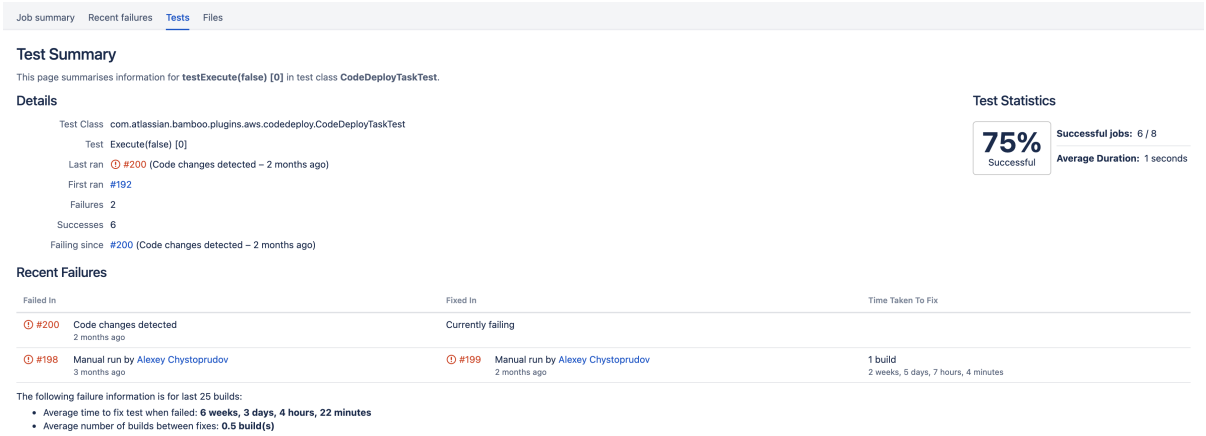
- The occasions when the test has failed. This can be useful when investigating what code changes were related to a failed test (see below).
- The test's average *duration* (running time), and whether the duration is increasing or decreasing across builds.

Related pages:

- Viewing test results for a build

To view a test's history:

1. Navigate to the build results for the plan or job, as described in Viewing a build result, and select the desired build result.
2. Go to the **Tests** tab.
3. Select the name of the test in which you are interested. The test's latest result will be displayed.
4. Select the test's history (🕓) icon.



Viewing the code changes that triggered a build

If a build was triggered by a code change, the updated files will be listed in the build result.

When Atlassian's Fisheye is connected to your Bamboo server, you can view the code changes that triggered a build. When a build fails due to a compilation error or failed test, you can explore the failed build in Fisheye and jump directly into the changeset that broke the build. You can view the history of that changeset to see what the author was trying to fix, take advantage of the the side-by-side diff view to analyze the change and then open the correct files in your IDE.

Related pages:

- Linking to source code repositories
- Triggering builds

To view the code changes that triggered a particular build result:

1. Navigate to the build results for the plan, as described in Viewing a build result, and select the desired build result.
2. Go to the **Changes** tab. A list of updated files will be shown.
    - Select the link to the source file to view the changes.
    - Select the version number to view the entire file.
    - Select the **diffs** link to view the differences between the current and previous version of each file.

Links to individual source-code files will only be available if your Bamboo administrator has connected the plan to the source repository, as specified in the Advanced options on the Repositories tab for the plan. For details, see Integrating Bamboo with Fisheye.

Viewing a build's artifacts

After a build has run, you can view the artifacts that were produced by all of the jobs in the plan. You can also view the latest version of an artifact from the most recent build.

Artifacts are files created by a *job build* (e.g. JAR files). Artifact definitions are used to specify which artifacts to keep from a build and are configured for individual jobs.

Viewing the artifacts for a build

To view a build's artifacts:

1. Go to the build result. See Viewing a build result for instructions.
2. Select the **Artifacts** tab. The artifacts produced by the jobs in the plan will be displayed. The artifact definitions for a job determine which artifacts are kept and which artifacts are shared with other jobs in the plan.

On this page:

Related pages:

- Configuring a job's build artifacts
- Sharing artifacts

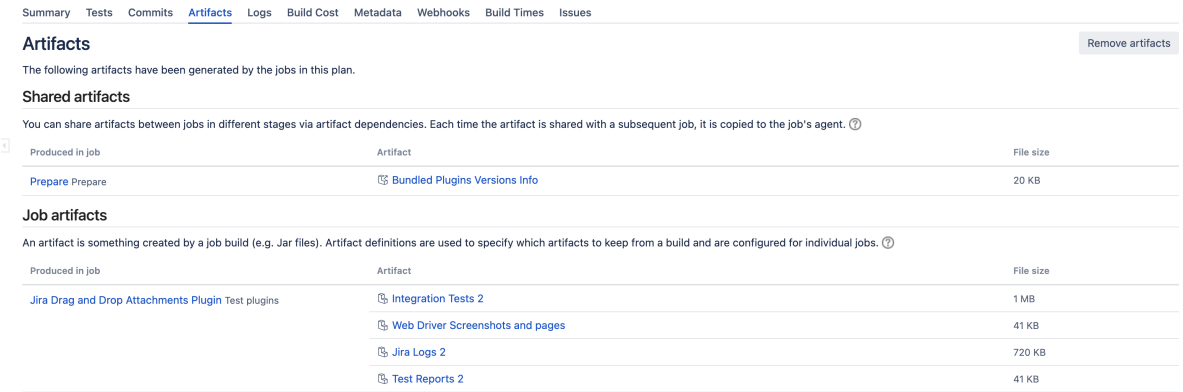
Viewing the latest version of an artifact from the latest build

To view the latest version of an artifact from the most recent build, you can manually edit the build artifact URL to retrieve it.

To view the latest version of an artifact from the most recent build:

1. Copy the URL for the build artifact.
2. Paste the URL for the build artifact in your browser and replace the build number in the URL with '`/latest`'.
    - If you need to log in to view the artifacts, you can append `os_username` and `os_password` parameters to the URL to access the files.

For example, if the URL for your artifact is:
`http://server/bamboo/browse/MYBUILD-254/artifact/logs/sample-log.log`
You would replace '`-254`' with `/latest`:
`http://server/bamboo/browse/MYBUILD/latest/artifact/logs/sample-log.log`



Viewing a build log

Every *build* has a *build log*. A *build log* is a permanent record of all the output generated by compiling the *job*'s source-code and executing the tests.

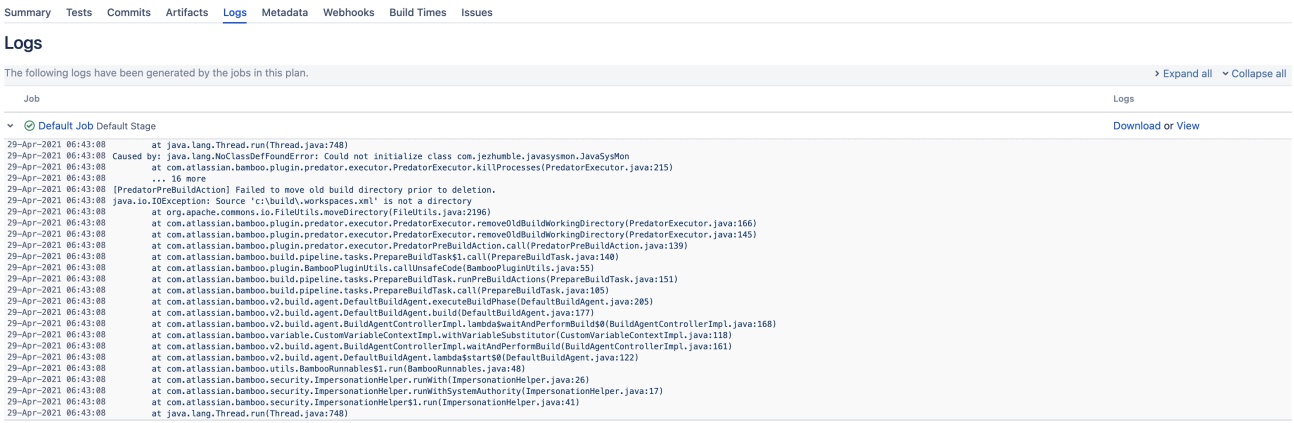
Related pages:

- Working with build results

To limit the nose, starting from version 7.2 Bamboo will be logging less data by default. In return we've introduced a verbose mode which will allow you to turn on logging of additional data, like logs from various VCS and environment variables. You can enable the verbose mode when running a customized plan, or in the deployment screen.

To view a build log:

1. Navigate to the build results for the plan or job, as described in Viewing a build result, and select the desired build result.
2. Select the **Logs** tab.
    - Select **View** for the desired log.
    - Select **Download** to download a text file of the log.



Viewing the metadata for a build result

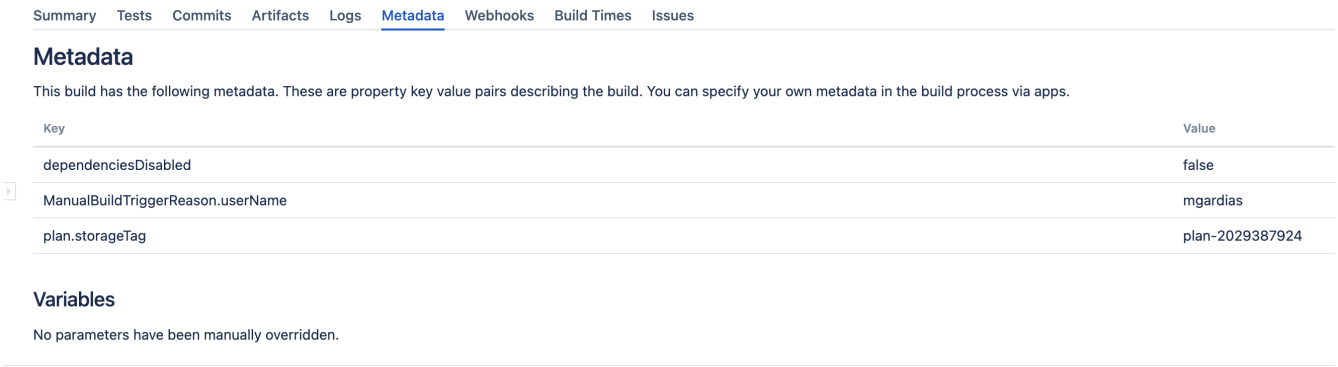
If your source-code repository provides *metadata* (i.e. key-value properties that are used to describe your build) for your build results, Bamboo will display it.

Related pages:

- Working with build results

To view the metadata for a build result:

1. Navigate to the build results for the plan or job, as described in Viewing a build result, and select the desired build result.
2. Select the **Metadata** tab.



Assigning responsibility for build failures

Bamboo automatically alerts the people who are assigned as responsible for a broken build, and lets other members of the team know that someone is looking at the problem. As you investigate the build failure, you can revise who is responsible, or claim all the responsibility for yourself.

People are assigned as being responsible for fixing a broken build in two ways:

- When a build fails, Bamboo automatically assigns all those who committed code to the failing build as responsible.
- You can manually assign people as being responsible.

Bamboo then sends notifications to whoever is assigned. Once the build is successful, Bamboo removes the responsible people from the build – they're off the hook!

Note that notifications need to have been configured first, using the Change of responsibilities **Event** and the Responsible user **Recipient type**. See Configuring notifications for a plan and its jobs for more information.

Related pages:

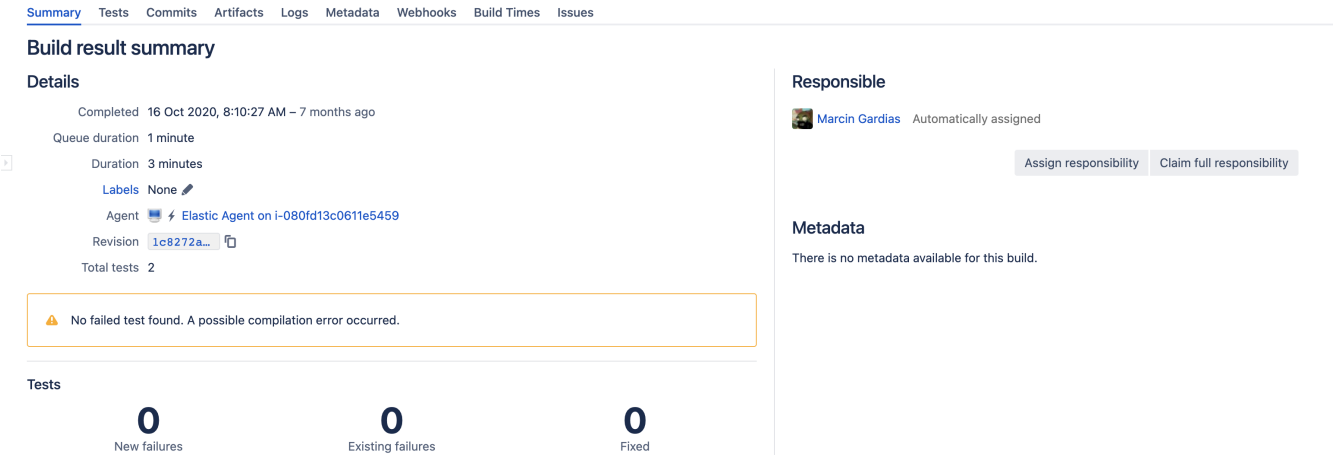
- Working with build results
- Working with comments
- Notifications

To assign responsibility for a broken build manually:

1. Go to the build result summary for a plan.
2. Select **Assign responsibility** to make another member of your team responsible for fixing the build.
3. Select **Claim full responsibility** if you want to fix the build yourself.

People who are responsible for the broken build are displayed on the build result summary.

Broken builds that are assigned to you are displayed on your **My Bamboo** page, available from the top navigation bar.



Configuring build results expiry for a plan

By enabling build expiry for a particular plan (described below), you override the global expiry settings that affect all plans in Bamboo. If you disable build expiry for a plan, that plan's build result data will never be automatically deleted from your Bamboo server.

You can select the build result data that will be kept for a plan and for how long this data will be kept (e.g. for reporting purposes), before Bamboo automatically deletes it.

Note that the build expiry *event* is a global event that runs periodically, regardless of whether you disable or enable build expiry in your plans. When this event occurs, the build results for your plan will be expired according to the settings below, or globally. To configure the global event and global build expiry settings, please refer to Configuring global expiry.

You can also delete the results of a plan build manually — see Deleting the results of a plan build.

Configure the expiry of build results for a plan

Ensure that you back up your build results data before its expiry date is reached.

Configure build expiry as follows:

1. Navigate to the configuration for the desired plan, as described on Configuring plans.
2. Select the **Other** tab.
3. Select the **Override global build expiry configuration** checkbox. Configure expiry using the following settings:

   Do not expire anything for this plan
   Select to disable expiry for all build results and artifacts for this plan – these will never be deleted automatically.

   Build results
   All build results data (including artifacts and build logs) are deleted.

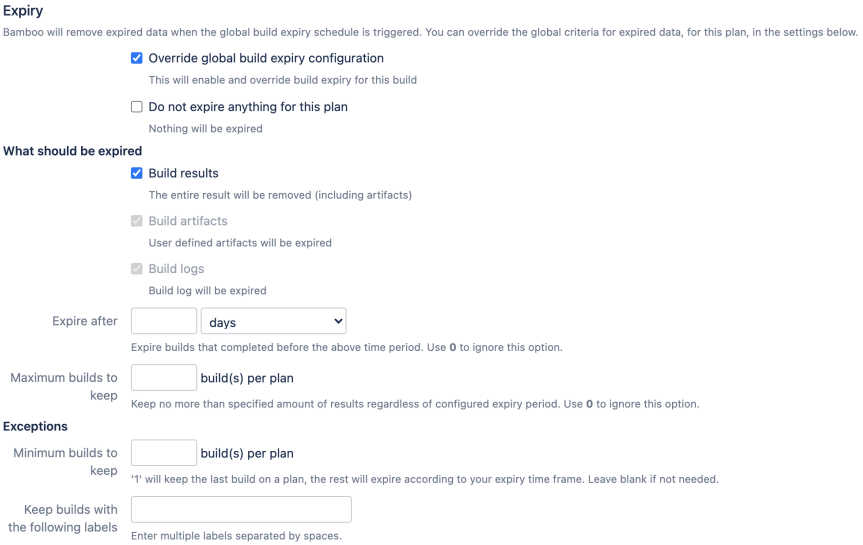
   Build artifacts
   Only user-defined artifacts are deleted from the build results.

   Build logs
   Only build logs are deleted from the build results.

   Expiry after
   Specifies the period (days, weeks or months) for which you want to keep build results.
   E.g. specify 24 months to keep all build results for the last two years.

   Minimum builds to keep
   Specifies the minimum number of build results you want to keep.
   For example, specify 50 to keep the latest 50 build results, even if they are older than the period specified with **Expiry period**.

   Keep builds with the following labels
   Specifies the build labels (not plan labels or job labels) applied to builds for which you want to keep build results, regardless of the **Expiry period** and **Minimum builds to keep** settings. Note that builds can be labeled either manually or automatically.
4. Select **Save**.



Deleting the results of a plan build

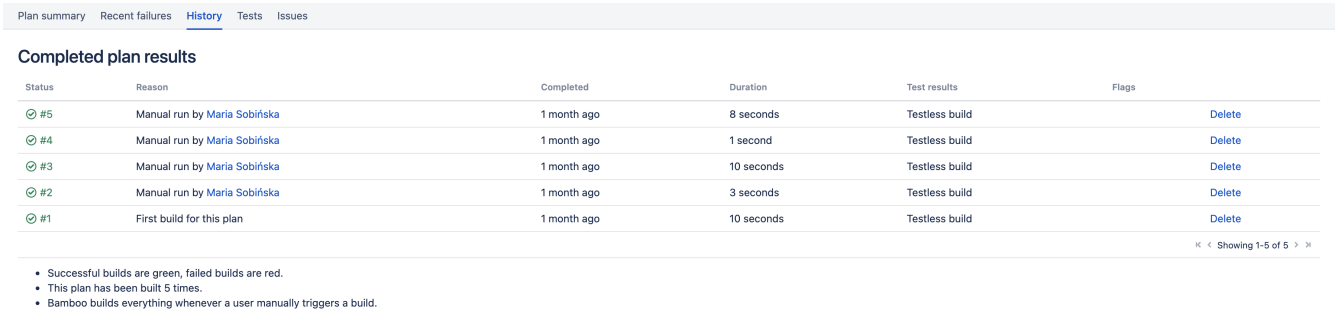
If the results of a plan builds are no longer required, you can completely remove the them from your Bamboo system. The results include all the results of all job builds that were processed as part of an individual plan build (with a specific build number). Note that you can also remove job build result data that reaches a particular age. See Configuring global expiry or Configuring expiry of a plan's job build results for more information.

Before you begin:

- The Edit global permission or Edit plan permission is required to delete plan build results.
- The result of a plan build cannot be deleted if that plan is currently being built. If you need to delete the result of a plan build, stop the plan's build first. Refer to Stopping an active job build for more information.

To delete the result of a plan build:

1. From the top navigation bar select **Builds** > **All build plans**, and select the desired plan.
2. Select the **History** tab. A table of completed plan build results will be displayed, with the most recent builds at the top.
3. Find the desired build result and select **Delete**.
4. Confirm the deletion. The plan build result and any artifacts generated as a result of the plan build's execution will be deleted.



Configuring live logs transmission

Bamboo transmits logs from agents to the server by default. Excessive logging is known to cause serious performance problems, including build result processing timing out, turning build grey. Since Bamboo 7.1, it is possible to turn live log transmission off.

In Bamboo, live logs are enabled by default. When you disable live logs transmission the following happens:

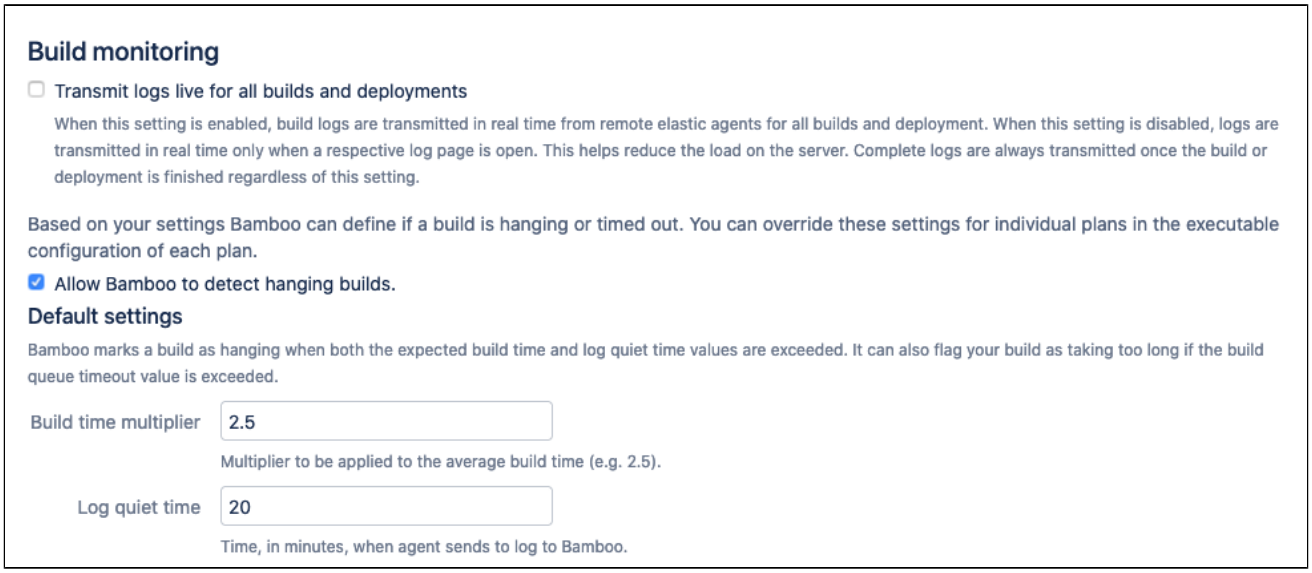
- Build logs are transmitted from an agent only if the build logs page is visible. If the page is closed or ignored, the agent will stop transmitting logs after 30s.
- Build logs *are not immediately* persisted to the disk on Bamboo server. When the build ends, the build logs are transferred using the same method that is used for transferring artifacts between remote agents and Bamboo server.
- Build logs are still persisted to the disk on the agent. If an agent fails to complete the build for any reason, the logs won't be transmitted and they will be available only on the agent file system. Bamboo will not retry to transmit the logs if agent process is abruptly terminated or Build Resiliency feature is not available.
- While the build is in progress, the build logs can have gaps.
- Logs are not available for downloading while the build is in progress.

Data Center only:

- The method of transferring logs as artifacts is used regardless if live logs are on. If the live logs are on, the partial logs are persisted on-the-fly on the server and replaced with the content of the full logs, once the build finishes.
- If agent completes the build but fails to transmit the logs to the server, it will retry the transmission as long as configuration of Build Resiliency allows it.

To disable live logs transmission

1. Go to  > **Plans**.
2. From the left-hand side column, select **Build monitoring**.
3. Clear the **Transmit logs live for all build and deployments** checkbox.



Working with comments

Comments are a useful way to record and share information about builds. There are two types of comments in Bamboo:

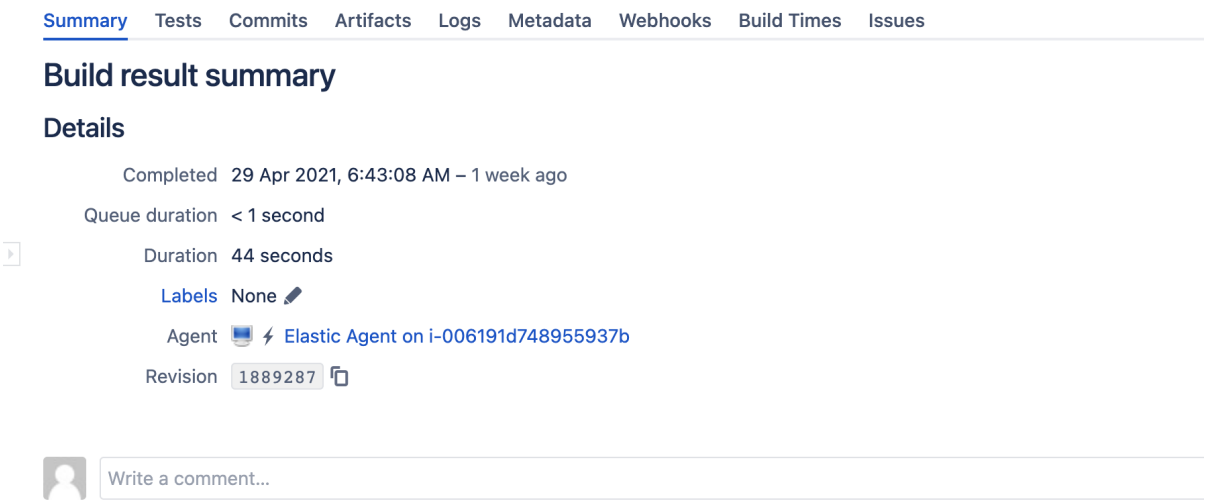
- Comments you make **about a build result** — these are comments that you make about a particular build result.
- Comments you make **when you commit code** — these comments are automatically copied into Bamboo from your source-code repository.

When you include Jira issue keys in your build and commit comments. Bamboo will automatically convert these into hyperlinks to the respective Jira issues, if Bamboo is integrated with Jira. The issue key must be of the default Jira issue key format (that is, two or more uppercase letters (`[A-Z][A-Z]+`), followed by a hyphen and the issue number, for example BAM-123).

Comment on a build result

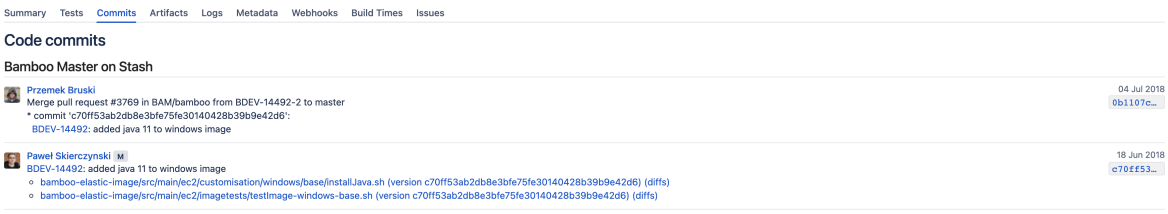
When you are logged in to Bamboo, you can comment on a build result to record relevant information for future reference, and to collaborate with your team. You can see other's comments there too, of course.

Simply navigate to a build result and enter your comment on the **Summary** tab:



Commit comments

If a build was triggered by a code change, the *commit comment* (or *check-in comment*) will be shown on the **Commits** tab of the build result:



You can see more details of the commit on the **Commits** tab of the build summary.

Working with labels

A *label* is a convenient way to tag and group build results that are logically related to each other. Labels can also be used to define RSS feeds and to control build expiry. With Bamboo, you can label your build results in whatever way works best for your team. Labels are not restricted to a particular plan, so you can apply the same label to build results from different plans.

For example, it might not be practical for your QA team to review every build, and you need to know which builds they have reviewed. By using labels such as "qa_passed" and "qa_failed", Bamboo allows them to simply indicate which builds have passed and failed QA.

You can include a Jira issue key in the label, as long as the key is of the default Jira issue key format (that is, two or more uppercase letters (`[A-Z][A-Z]+`), followed by a hyphen and the issue number, for example BAM-123).

Bamboo administrators can also configure automatic labeling of job build results.

Label a build result

You must be logged in to Bamboo before you can label a build result.

To label a particular build result, simply select the pencil icon (), beside **Labels** in the Details section of the Summary tab. You can also label a build result using Instant Messaging. Select the **x** at the right of a label if you need to remove it.

Select **Labels** on the Summary tab to see the other labels that have been used for the plan's builds. Select a label there to see all the projects, plans, and build results where that label is used.

Label a plan

Bamboo allows you to label plans. Labeling a plan allows you to filter the plans displayed on the Build dashboard or Wallboard. You may want to do this if you have set up a large number of plans in your Bamboo instance and want to highlight specific plans for attention.

For example, you may want to label all builds related to the release with a release label. You can then filter your Wallboard during your release, to display only these builds.

You must be logged in to Bamboo before you can label a plan.

Simply go to the plan you want to label and select **Actions** > **Modify plan label**.

See also these Atlassian blog posts:

- Making your Bamboo dashboard quicker and more relevant using plan labels
- Get to know Bamboo's build expiry and labels

Quarantining failing tests

There may be times when you want to prevent a failing test from causing the whole build to fail.

Possible scenarios where this may be useful include:

- You want to build an artifact despite there being a failing test, but can't do this while the plan build is failing.
- In test-driven development (TDD), a test will fail until the functionality is implemented - you want to quarantine all but the relevant tests.
- A test may give unpredictable results, perhaps because of infrastructure issues or dependencies.
- You want to remove a test from a build, but don't want to alter or delete the test source code because doing so could affect another Bamboo plan.

In Bamboo, you can temporarily disconnect any test's results from the plan build results by quarantining the test. The test is still run whenever the plan is built, but the test's results do not affect the plan's build results.

You can always restore a test's results to the build results when required, for example if the test is now passing.

All the quarantined tests for a plan are displayed on the **Quarantined tests** tab of the plan summary. The status bar for each test shows the recent build history of the test.

On this page:

Related pages:

- Working with builds
- Viewing a plan's build information
- Viewing test results for a build
- Viewing a build result
- Configuring plans

Enable test quarantining

To enable test quarantining:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Plans**, select **Quarantine settings**.
3. On the **Quarantine settings** page, select **Enable quarantine**.
4. Select **Save**.

Quarantine a failing test

- You need plan administrator or build permission to quarantine a test.
- Test quarantining must be enabled

To quarantine a test:

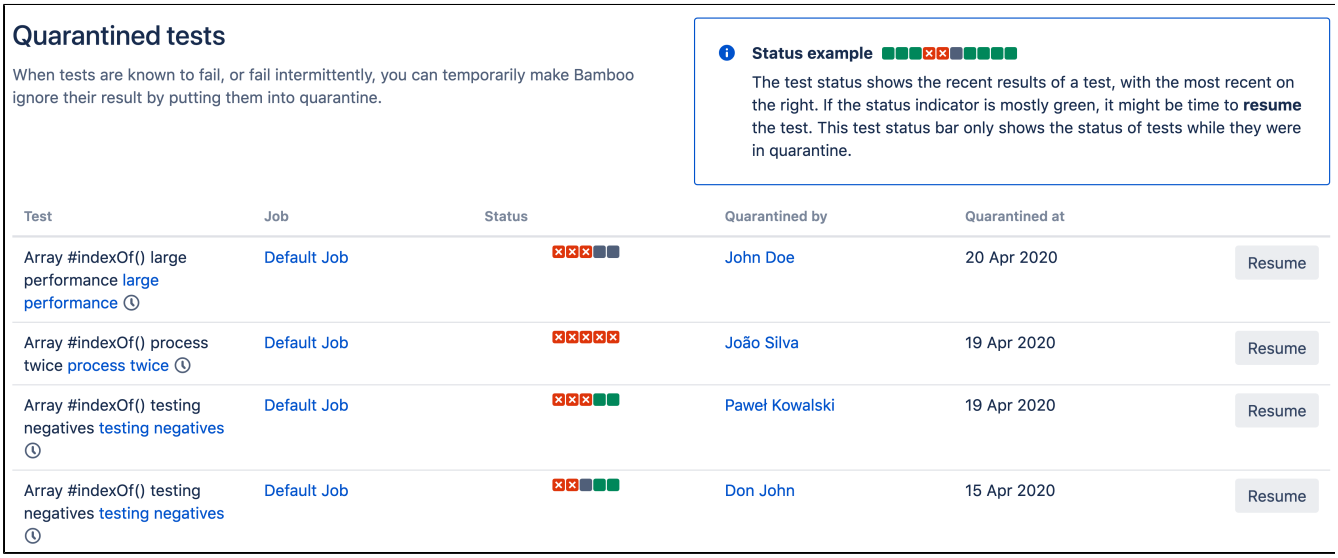
1. Select **Builds** > **All build plans** > **#buildresult** to go to the build result where the test is failing.
2. Select **Quarantine** for the failing test (in the Summary section of the build).

Restore a quarantined test to a build

- You need plan administrator permission to restore a test.
- Test quarantining must be enabled.

To restore a quarantined test to a build:

1. Go to your plan's summary.
2. Select the **Quarantined tests** tab.
3. Select **Unleash** for the test to be restored.



Setting up plan build dependencies

You may want to trigger a plan build when another plan's build has successfully completed. This ensures that changes to any job's source code associated with one plan does not break the build of another dependent plan (known in this context as a *child* plan).

For example, there could be two plans in Bamboo:

1. **Acme – Core** — which contains the core code for an application.
2. **Acme – Plugin** — which contains code for a plugin to the application.

In this scenario, the **Acme – Plugin** plan is a *child* of **Acme – Core**. Any changes to source code associated with the **Acme – Core** plan should trigger a build of **Acme – Plugin**.

On this page:

Triggering dependent plans

To trigger a child plan to build when this plan builds successfully:

1. From the top navigation bar select **Builds** > **All build plans**.
2. Locate the plan in the list and select the edit ( ) icon to display the plan's Configuration pages.
3. Select the **Dependencies** tab.
4. Under **Child plans**, begin typing a plan name in **Search for plan** to select child plans to trigger. You can set multiple plans to be triggered.
5. Select **Save**.

Automatic dependency management with Maven 3

> Automatic Dependency Management only works with the master plan.

Automatic Dependency Management is a feature for users who use Maven 3 and wish for their parent and child dependencies to be set up according to the dependencies in the Maven pom.xml. Every time the plan is run, the Bamboo Automatic Dependencies are updated to reflect any additions or removals of Maven dependencies.

Bamboo Maven auto dependency always lists the dependencies needed and produced by a plan. However, linking the parent-child plans will happen for SNAPSHOT builds only. Effectively, Bamboo searches for the word `SNAPSHOT` in the `pom.xml` file.

To setup automatic dependency management:

1. From the top navigation bar select **Builds** > **All build plans**.
2. Locate the plan in the list and select the edit ( ) icon to display the plan's configuration pages.
3. Locate the job that contains the pom.xml you wish to use to automatically update plan dependencies by analyzing a Maven pom file.
4. Select **Actions** > **Configure job**.
5. Select the **Tasks** tab.

6. Select **Add task** and add the **Maven Dependency Processor** task to the job. For best results, ensure that the task runs last by dragging it to the bottom of the task list. For more information on configuring tasks, see Configuring tasks.

   Override project file
   Optional. The location relative to the working directory or sub-working directory where the project file (pom.xml) is located.

   Working subdirectory
   Optional. The subdirectory from which the Task should look for the project file (pom.xml).

   Alternate location of settings.xml
   Optional. Specify an alternate settings.xml to be used if the Task needs to resolve dependencies from specific Maven repositories.

   Path to Maven local repository
   Optional. Specify a full path to a local Maven repository for the Task to use to resolve dependencies.
7. Select **Save**.
8. Use the Plan Navigator to return to the plan.
9. Select the **Dependencies** tab.
10. Select **Automatic Dependency Management**. You should see the name of the job for which you configured the **Maven Dependency Processor** appear.
11. Select **Save**.

Dependency blocking

Dependency blocking is an advanced feature of dependent build triggering that can be used to manage plan builds with parent build dependencies. This ensures that a "tree" of dependent builds always runs in tree hierarchy order, even if child plan builds are triggered independently of their parents. For more information, see Dependency blocking strategies. Please note, dependency blocking only works when the plan build is triggered because of source repository code updates.

Notes

Build dependencies work together with the trigger configuration of plans to trigger builds of these plans. For example, you can set up Plan A to poll its repository for changes as well as to be dependent on a parent plan (Plan B). In this case, builds of Plan A will be triggered when code changes are detected in its repository *and also when* builds of Plan B complete successfully.
If you want your builds to *only* be triggered by successful parent builds from your build dependencies, don't configure triggering for your child plans at all. See Running a plan build manually.

- If the child build *uses the same source* as the parent build (for example, the Subversion URL is the same), the child build will be forced to check out the same revision of source code as the parent build. This ensures that builds are consistent when triggering one build from another.
- Take care not to create *circular dependencies*, where your child build triggers one of its parent builds. Otherwise your plans may build continuously. See Running a plan build manually.

Dependency blocking strategies

Dependency blocking is an advanced feature of dependent build triggering that can be used to manage the builds of plans that have parent plans. This ensures that a "tree" of dependent builds always runs in tree hierarchy order, even if child plan builds are triggered independently of their parents.

The three dependency blocking strategies are:

Do not block
When triggered by a source code update, the plan will always be built, regardless of any parent plan build dependencies.

Block build if parent builds are queued or in progress
When triggered by a source code update, the plan will *not* be built if its parent plans are building or are waiting in the build queue.

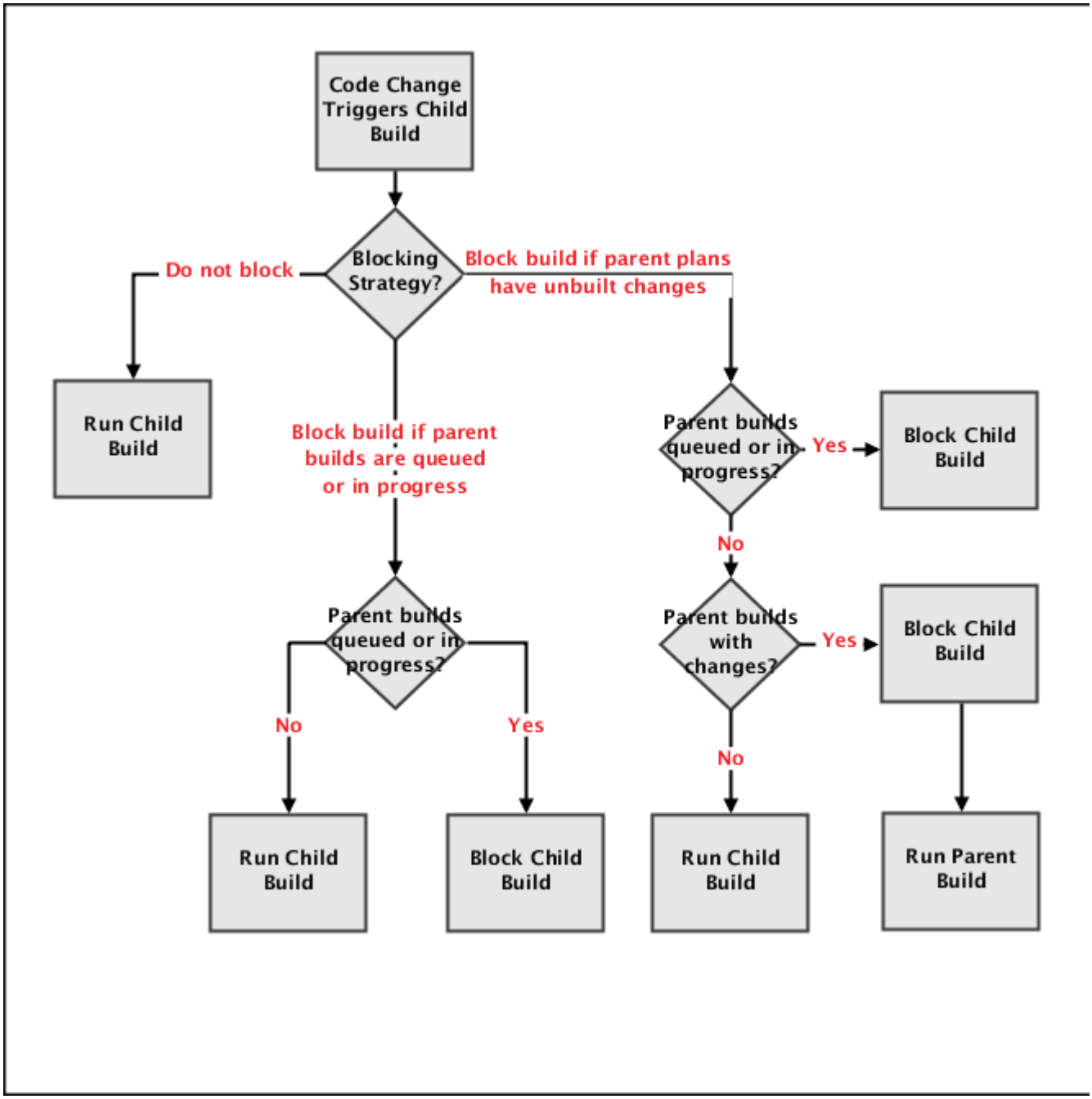
Block build if parent plans have unbuilt changes
When triggered by a source code update, the plan will *not* be built if its parent plans are building, are waiting in the build queue, or have changes.

When Bamboo finds parent plans with source repository changes, those plans will be triggered and your plan will be blocked.

Note that for the *Block build if parent plans have unbuilt changes* option, only the repositories of parent plans that are specified by triggers (that is, by the Repository polling or Repository triggers the build when changes are committed trigger types) are scanned for unbuilt changes; if there are repository changes (for parent plans), then the parent plans are triggered and the current plan is blocked.

⚠ *Dependency blocking only works when the plan uses a trigger configuration based on source code updates (i.e. Repository polling or Repository triggers the build when changes are committed). This feature will not work when a plan uses a trigger configuration based on a schedule or triggered via a parent build (when there are multiple parent plan builds in progress).*

These dependence blocking strategies are illustrated in the flowchart below:



Viewing test statistics for a job

Bamboo provides a summary of test results across all of a job's builds. This helps you to:

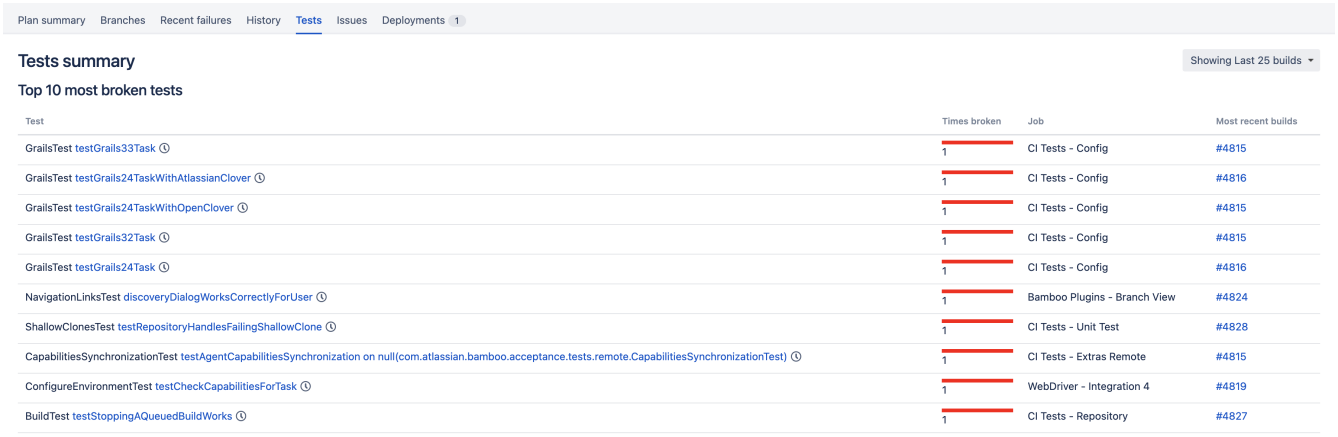
- **Troubleshoot** by identifying which tests fail most frequently, and which tests take longest to fix.
- **Manage your build duration** by identifying the plan's slowest running tests.
- **Ensure quality** by monitoring the number of tests over time: are your test cases growing with your code base?

Related pages:

- Reporting

To view the test statistics for all of a job's builds:

1. Navigate to the desired build result page, as described in Viewing a build result.
2. Select the **Tests** tab.
3. Select the sub-tabs to filter the rest statistics (see screenshots below).
    - To view a test's history, select the test name.



Reordering jobs in the build queue

Bamboo automatically assigns a plan's jobs to the build queue when the plan is triggered and no agents are available to run them. The build queue is displayed on the **Build activity** page under the **Build** tab of the Dashboard.

If you want to prioritize one job build over another in the build queue, you can manually reorder these jobs in the build queue. This will not force a job build to run immediately, but will promote it in the build queue. Your job build will still require an agent (which has the capabilities to meet the job's requirements) to become available. Similarly, you can demote a job build in the build queue if you do not need it to run urgently.

Bamboo administrators can reorder plans in the queue. To do this, use the ↕ icon to move the plan to its new position in the queue.

Stopping an active build

The instructions on this page describe how to stop a plan or job build that is running.

Note that if your Bamboo server runs on Windows, it may only be possible to stop an active build by going to the Windows Task Manager and ending the relevant processes.

To start a building a plan manually, see Running a plan build manually.

Note that if you stop an active build, Final tasks will still run. Artifacts will be shown if they were created before stopping the build.

On this page:

Related pages:

- Running a plan build manually
- Disabling or deleting a plan
- Disabling or deleting a job

Stopping an active plan build

To prevent Bamboo submitting a plan to the build queue, refer to Disabling or deleting a plan.

To stop an active plan build:

1. From the top navigation bar select **Build** > **All build plans**.
2. Select the stop ( ) icon next to the active plan you want to stop.

Stopping an active job build

To prevent Bamboo submitting a job to the build queue, refer to Disabling or deleting a job.

To stop an active job build:

1. From the top navigation bar select **Build** > **All build plans**.
2. Select the name of the plan.
3. Select the stop ( ) icon next to the active job you want to stop (in the Current activity section).

Deployment projects

What are deployment projects?

A deployment project in Bamboo is a container for holding the software project you are deploying: releases that have been built and tested, and the environments to which releases are deployed. Teams typically have QA, staging and production environments.

Why use deployment projects?

Continuous Integration was not designed for Continuous Delivery. Continuous Integration is designed to keep developers informed about the state of the latest code changes.

In Continuous Integration, historical build results (along with information such as issue and commits) are de-emphasized as more changes are made, since only the latest build is important to the developer.

Using a traditional Continuous Integration server for Continuous Delivery is less than ideal because:

- **Deployed builds are difficult to find** – Builds that were deployed only a few days ago are de-emphasized by the system. While this is good for a Continuous Integration workflow, the behavior makes it difficult for team members to identify which builds have been deployed and when, versus which have not. Teams can work around this with a system that uses labeling but it's not immediately obvious how it should work unless team members are trained to use it correctly.
- **Difficult to find what changes were made between deployments** – Build results report commit and issue data between a specific build result and the one immediately before it. There can be many build results between two different deployments. Often the entire history has to be navigated between the two deployments to build a complete view of the changes between them. Sometimes, even other tools have to be used, such as version control systems.
- **Difficult to know what was deployed, and when and where it was deployed** – Builds do not have visibility of where code is deployed or what was previously deployed to an environment.
- **Lack of insight into the QA process** – Given a list of deployment candidates, builds offer no clear way (other than commenting or labeling) for QA to "sign off" on a tested release or mark a release as broken or un-releasable.
- **Poor control over who can deploy** – While it can be controlled by permissions who can run, view or edit a build, they do not give enough fine grained control over which people in the team can deploy to production or other sensitive environments. In essence, if someone has permission to run the build they can deploy the software any time they wish.

To solve these issues Bamboo provides the following concepts:

- **Deployment project** – Represents the software you are deploying (such as a web application), the releases of the software deployed and the environments that they will be deployed to throughout the lifecycle.
- **Environment** – Represents the servers or groups of servers where the software release has been deployed to, and the tasks that are needed for the deployment to work smoothly. Example environments could be named Development, QA, Staging or Production. Environments have permissions that allow fine grained control of who can deploy, edit or view an environment and record the full history of releases deployed to it.
- **Release** – Identifies a snapshot of artifacts and its associated data such as commits, Jira issues and the builds that were used to test it. As a release contains the information of the difference between itself and the release beforehand, it's very easy to see the changes between releases or to show the difference between the software deployed on two different environments. Releases also track what environments they have been deployed to.

How do deployment projects work?

Consider the following diagram:

On this page:

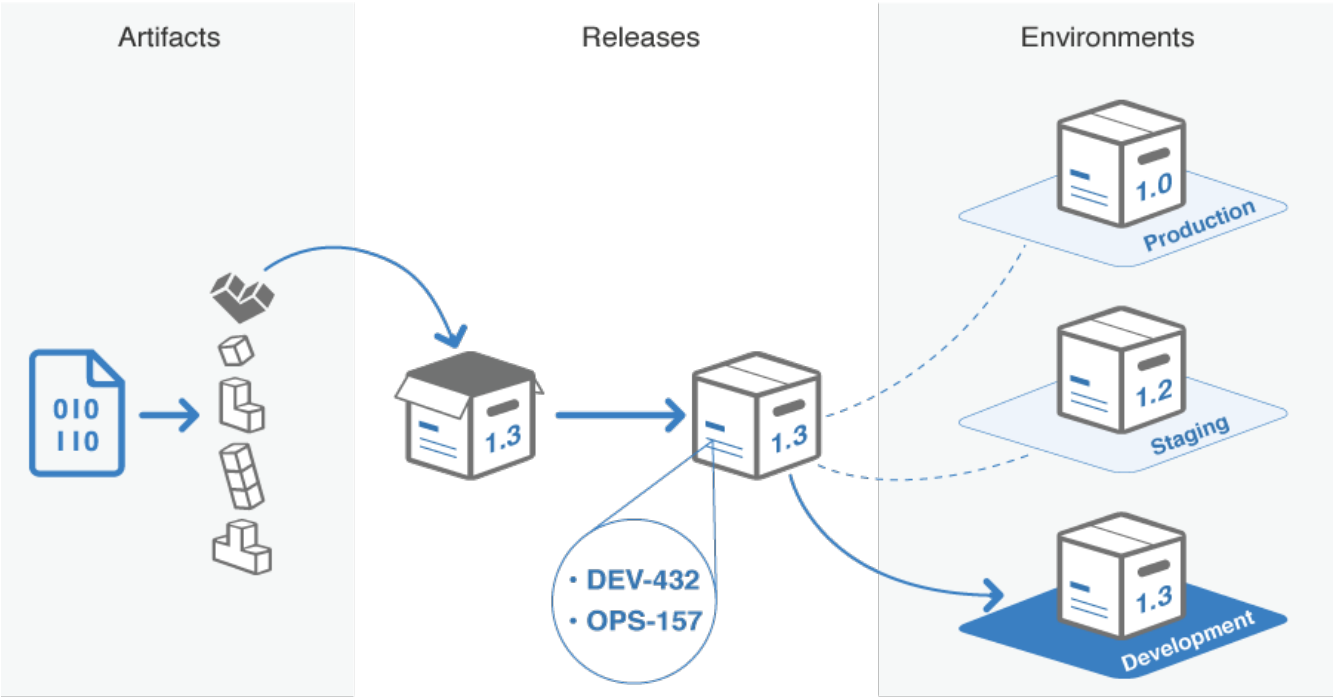
Related pages:

- Understanding deployment releases
- Deployment projects workflow
- A sample deployment project
- Creating and configuring a deployment project
- Creating a deployment environment
- Managing deployment projects
- Manually starting a deployment
- Deployments from branches

What is Continuous Delivery?

Continuous Delivery is the practice where all changes made to a software project are automatically built, tested and made ready for deployment to users. In practice, once the project has been built and tested it is "staged" somewhere where it can be manually verified and then made available to users.

Unlike Continuous Deployment (the process where code changes are automatically built, tested and deployed without human intervention), typically there is a decision made by a human being to whether or not the software is of sufficient quality or if it is the correct time for the business to make the software available to its users.



Artifacts

Create and test deployable artifacts with build plans. Ensure any artifacts you wish to deploy with Bamboo are flagged as "shared" to make them available to the deployment instructions of the environment.

Releases

Any artifact that has been successfully tested can be used to create a release; you can create as many releases as you like. Bamboo will add other metadata such as related commits and Jira issues to each release which enable reporting and tracking as it moves through your environments.

Environments

Environments in Bamboo reflect the development, testing and production environments in your IT infrastructure – hostnames and authentication credentials for each environment reside at the task level inside your deployment jobs. At any point in time, you will be able to see which release is running in each environment, which release it replaced, when it was deployed and who deployed it. You will also be able to see any associated Jira issues.

Understanding deployment releases

Key to getting the most out of deployment projects is understanding what releases are, and how you should be using them.

It is also important to understand the difference and relationship between *artifacts* - the results of a build plan - and *releases* - a snapshot of artifacts at a specific time that can be deployed somewhere.

On this page:

What are artifacts?

When the continuous integration process is triggered by a developer committing code, the first stage of the process compiles the code, runs tests and then assembles the code into binaries. These assembled binaries are known as *artifacts*. The build process can produce build artifacts at any stage of the build that can then be shared with other builds or deployment projects.

Since Bamboo manages artifacts, any artifacts that are needed by builds or deployments are automatically transferred by Bamboo to a remote server as needed, so long as that build or deployment project declares that it needs the artifact to complete its work.

For more information, see Sharing artifacts.

What are deployment releases?

Releases are used to track exactly what software was deployed to an environment. In essence, a release is a snapshot of any number of artifacts that will be used in the deployment process and their associated metadata, such as Jira issues, code changes and any test metadata that might be relevant to what is being deployed.

A release is created from the result of a single build. When you view a release, you can see all the code changes, Jira issues and other metadata that were used when making the artifact for that build. This information can be used for purposes such as release notes, quality control and infrastructure planning, and allows you to compare any two releases to see the changes between them.

Why use releases?

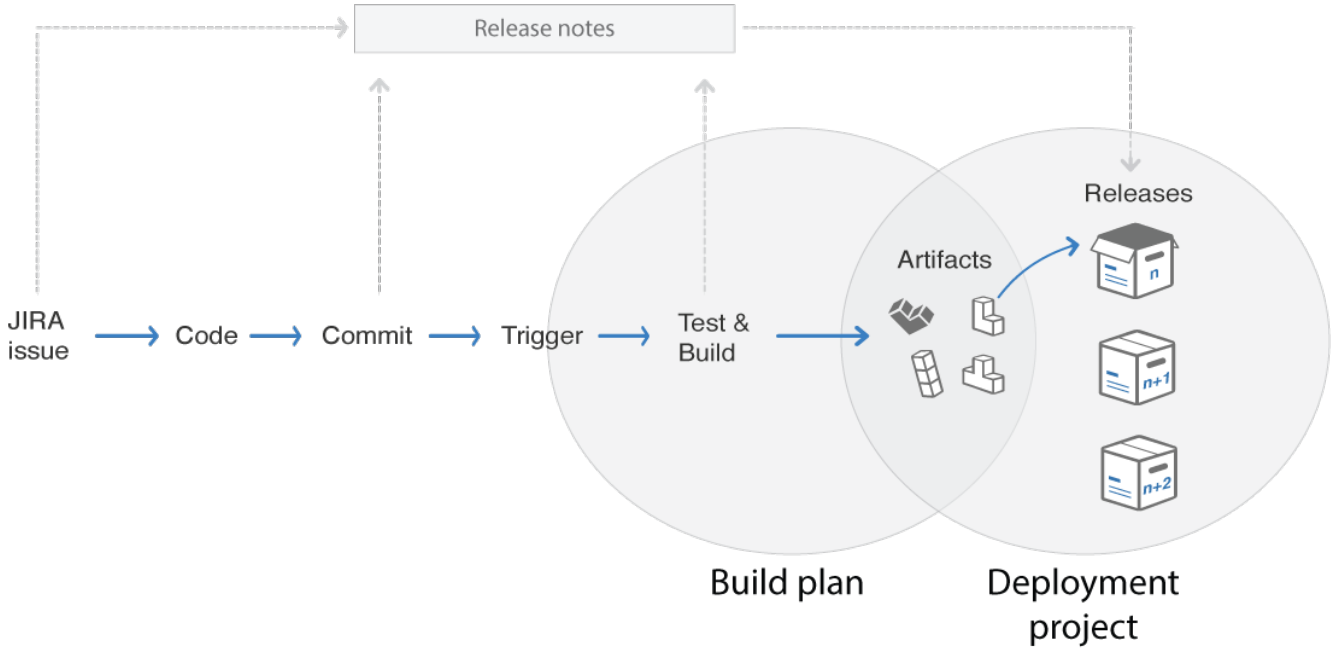
In Bamboo, releases are tracked against environments, which represent a server or group of servers that you wish to deploy your software to. Because each environment can only host a single active release at any one time, Bamboo gives a unique release name to the software being deployed. By checking the environments for our project, we can quickly identify:

- Where releases have been deployed
- Which release is currently deployed
- The release deployment history
- The release deployment status

Another key feature of releases is that as well as providing a deployable snapshot of your artifacts, they also collate the Jira issues, commit record and test & build metadata for the specific series of changes associated with the release. This enables much smoother reporting and tracking as the release moves through your environments, and allows you to easily track changes between releases.

How artifacts and deployment releases work together

The relationship between artifacts and releases shows the hand-over point between Bamboo builds and Bamboo deployments.



As the diagram shows, a developer who is responding to Jira issues, commits a code change and triggers a build. This build produces a number of artifacts. In a deployment, these artifacts are assembled into a release, and the Jira issue, commits and test/build metadata are added. This release then gets a unique identification name which serves as an identifier throughout the system. You can define the unique identifier according to your needs using the release naming system.

Once a release has been created, it is now ready to be deployed to an environment.

The next step

The next step is to examine and understand the deployment project workflow. Learn more about the deployment project workflow.

Deployment projects workflow

Deployment projects are an important feature of the continuous deployment philosophy. Identifying and understanding the key configuration steps for a deployment project will help you to gain a better insight into how a deployment project functions.

On this page:

Deployment project prerequisites

There are a number of prerequisites that must be in place before you can start using deployment projects. The prerequisites are:

1. A build plan
2. Artifacts to deploy (these are produced by the build plan and shared)

Step 1: Create a new deployment project

Creating the deployment project is the first step. Here we will give the project a name and a description, but most importantly we associate the deployment project with an existing Bamboo build plan. This is why we must have a build plan available to associate with our new deployment project.

Learn more about creating a deployment project here.

Step 2: Decide on a release naming scheme

The next step is to configure the release naming scheme for the deployment project. The release naming scheme will define how Bamboo names the releases that you create from your build artifacts for deployment. You can use either a simple release naming scheme, or a scheme that uses global or plan variables already defined in Bamboo.

Learn more about release naming schemes here.

Step 3: Decide who can view and edit the project

You need to decide who can view and edit the deployment project: This is done using the permission scheme. You can add or remove individuals or groups from the scheme, and give them access to either view and/or edit the project.

Learn more about the permissions scheme here.

Step 4: Create a deployment environment

The next step is to create a deployment environment. A deployment environment represents the servers or groups of servers where the software has been deployed, and any tasks needed for the deployment to go smoothly. You can call the deployment environment anything you like, though typical names are QA, Staging and Production.

Learn more about creating a deployment environment here.

Step 5: Customize your deployment environment

Once you have created your deployment environment, you need to set it up to reflect the needs of your project. You can control most aspects of the deployment environment, including:

- **Tasks** - Run executable tasks during the deployment process, for example downloading a needed artifact from a different plan
- **Triggers** - Decide which events or schedule points will trigger off deployment of your project to an environment
- **Permissions** - Decide who can view and edit your deployment environment
- **Agents** - Control which agents you will use to support your deployment process
- **Notifications** - Create a notification scheme to keep you informed about your deployment progress
- **Variables** - Assign variables for your deployment projects

Step 6: Start deploying!

Once you have set up your deployment project, you're ready to start the deployment process.

A sample deployment project

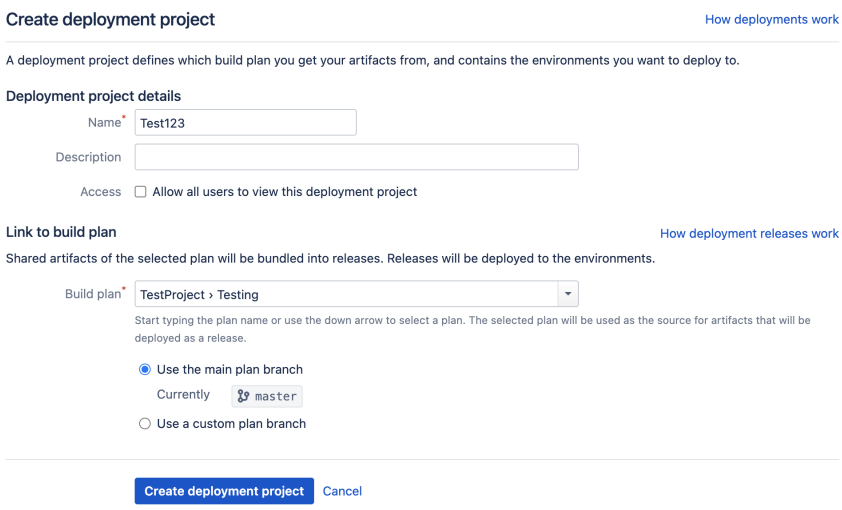
On this page we will examine a sample deployment project, and work through the steps required to get a deployment project up and running.

On this page:

Step 1: Create a deployment project

The first step in creating a deployment project is to associate the project with an existing build plan. You can do it at the same time as creating the deployment project. To create a new deployment project, and associate an existing build plan with it:

1. Select **Create** > **Create deployment project** in the dropdown menu from the top navigation bar.
2. Complete the **Name** and **Description** fields as required.
3. Select an existing build plan in **Build plan**. Bamboo will identify any relevant build plans in the menu.
4. If your build plan has a plan branch, Bamboo will detect it and offer an additional field for completion.
5. Select **Save deployment project**. Your deployment project will be created, and will automatically be associated with the build plan you selected above.

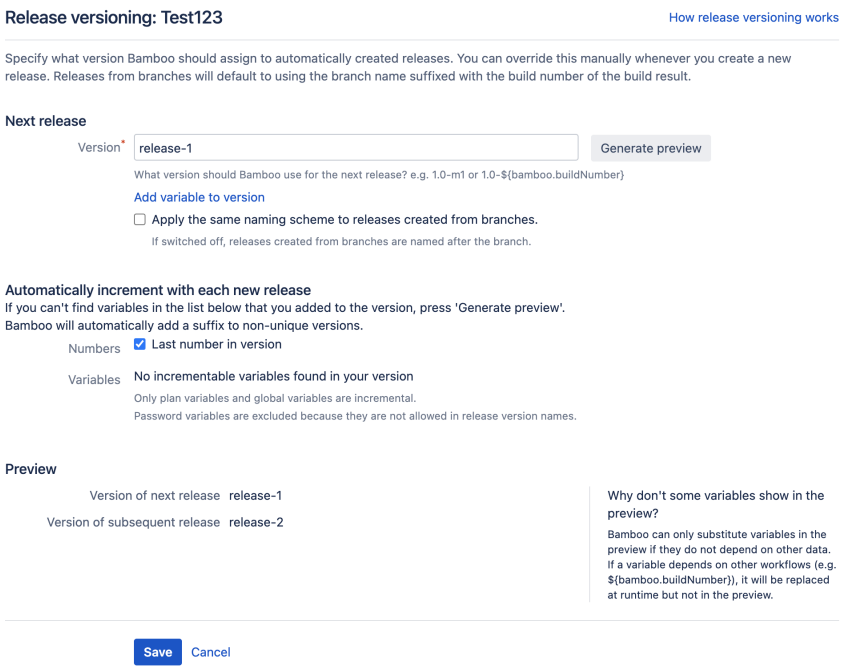


Step 2: Define the release naming scheme

The next step is to provide a version naming strategy for the deployment project. This will define how the deployment project will ascribe names to current and subsequent artifact bundles that it generates. See Naming versions for deployment releases for more information. To configure your version naming scheme:

1. From the deployment project configuration screen, select **Release versioning**.

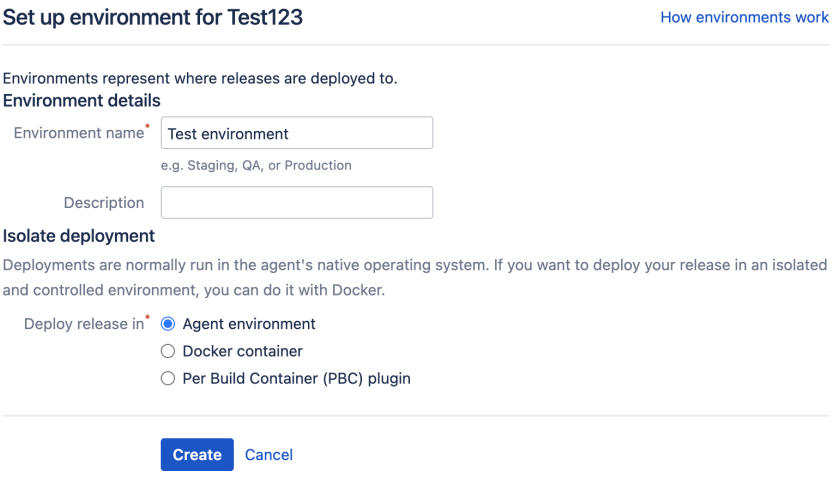
2. Complete the required fields according to your naming scheme. In this example we can see that a simple naming scheme has been adopted - the next name will be release-1, and the subsequent release-2.
3. Select **Save**.



Step 3: Create a deployment environment

Once we have defined our naming scheme, we need to create a deployment environment for the artifact(s) to be deployed into. Typically, deployment environments include Test, Staging, QA, and Production, however there's no limit to creating useful deployment environments. Let's look at how it's done:

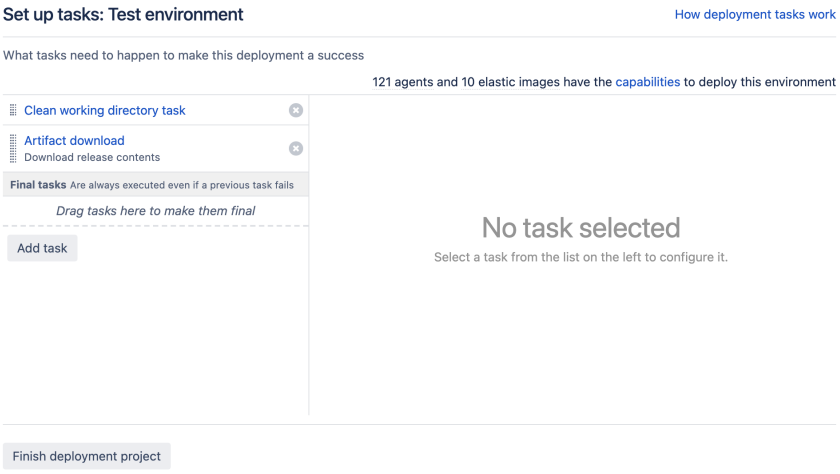
1. From the deployment project configuration screen, select **Add environment**.
2. Enter the name of the deployment environment, and a brief description.
3. Select **Create**.



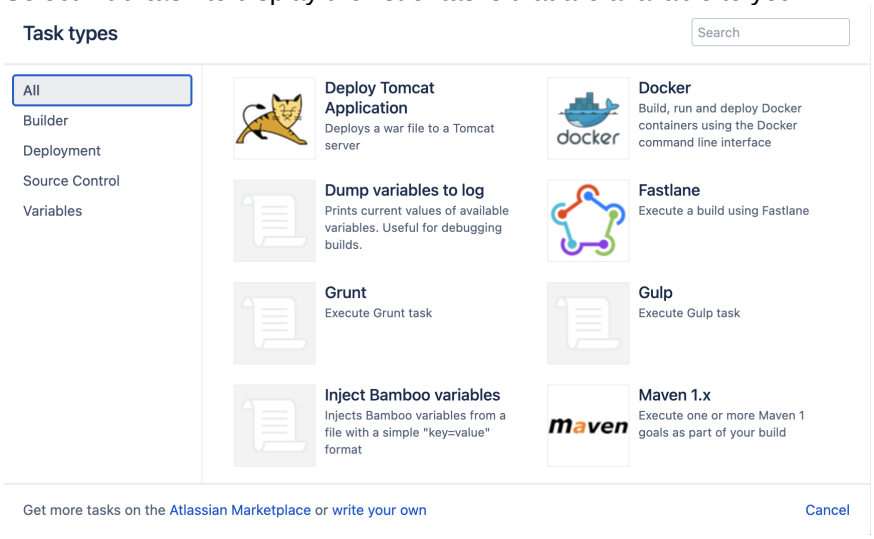
Step 4: Add some environment tasks

Tasks are activities that the deployment project will perform in order to run. These could be checking out some code from a repository, downloading an artifact from a server, or running a script. Let's have a look at how to add a couple of tasks to the deployment environment:

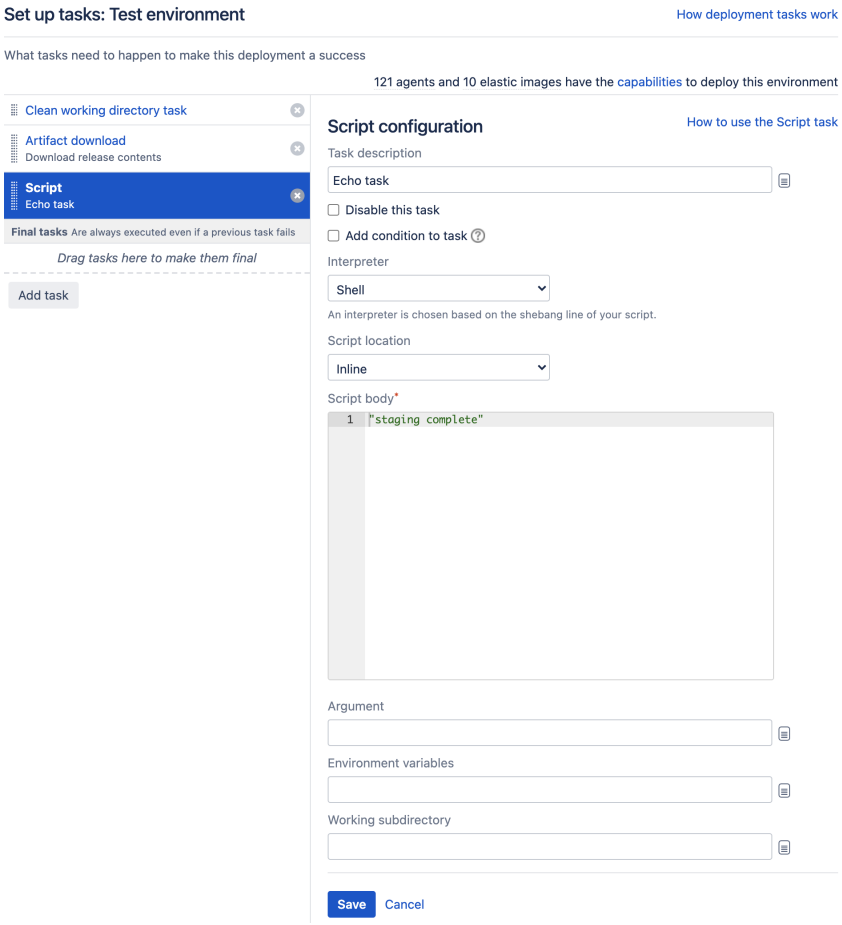
1. From the deployment project configuration screen, select **Set up tasks**:



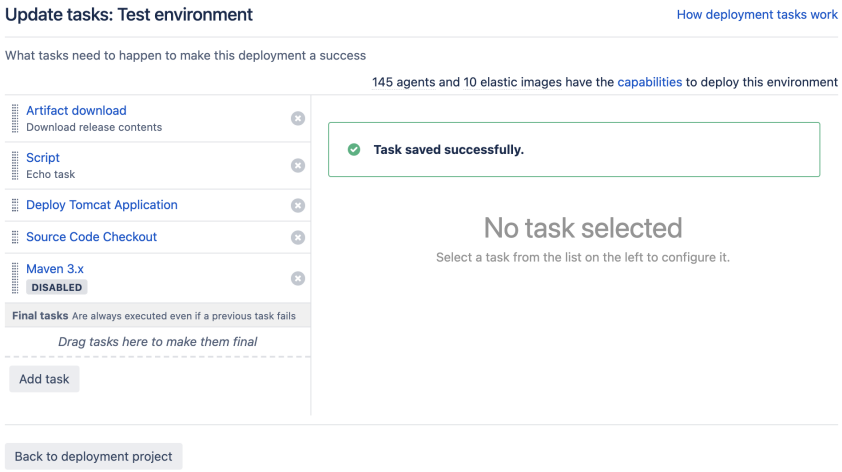
2. Select **Add task** to display the list of tasks that are available to you:



In this example, we will add a simple script task to run as part of our build. Selecting the task we wish to add adds it to the set up tasks screen, and allows us to configure the individual task:



3. Select **Save** to save the individual task configuration, and then on **Finish deployment project** to complete configuration of the script task for the deployment environment. In reality, we would require a number of tasks, not least one to obtain an artifact for use in the deployment. The following task configuration for a production environment includes an artifact download, DB change script, a Tomcat deployment, source code checkout and a Maven 3.x task:



Step 5: Let's deploy!

Our sample deployment project now has all of the elements required to run. We can trigger the deployment project manually by selecting the appropriate deploy ( ) icon on the **Deployment projects** page.

Step 6: Additional deployment environment options

But deployments don't end here. This simple example is just a snapshot of how a deployment project is configured and works. Bamboo deployment projects feature a host of additional features to help you manage your development and deployment processes. These include:

- **Automated triggering** - Select to automatically deploy after a successful build plan completes, or at a scheduled time.
- **Agents** - Assign specific agents, elastic agents, or image configurations to execute the deployment for the environment.
- **Variables** - Incorporate variables for use when deploying versions to environments.
- **Permissions** - Define what users are allowed to view, edit, and deploy in the environment.
- **Notifications** - Define who and how notifications about events for the environment are made.

Creating and configuring a deployment project

Creating a deployment project from a plan is easy with Bamboo.

A deployment is a container that holds:

- Environments that represent the physical environments, such as QA, Staging, and Production.
- Releases that represent the actual software artifacts being deployed - these include the issues and commits which make up the release.

To create a new deployment project you need to:

1. Provide a name and a description that represents your project.
2. Associate the project with a build plan. The build plan will produce the artifacts you will snapshot into a release and deploy to the environment. Associating the deployment project with a build plan tells the deployment project which set of artifacts to use for the deployment.

If you are using plan branches, you will also need to associate the deployment with the plan branch. The plan branch represents a build for a branch within the version control system that inherits the configuration defined by the parent plan. Any new branch created is automatically built and tested using the same build configuration as the parent. When the plan branch build succeeds, it can be merged back into the master.

Learn more about Deployments from branches.

On this page:

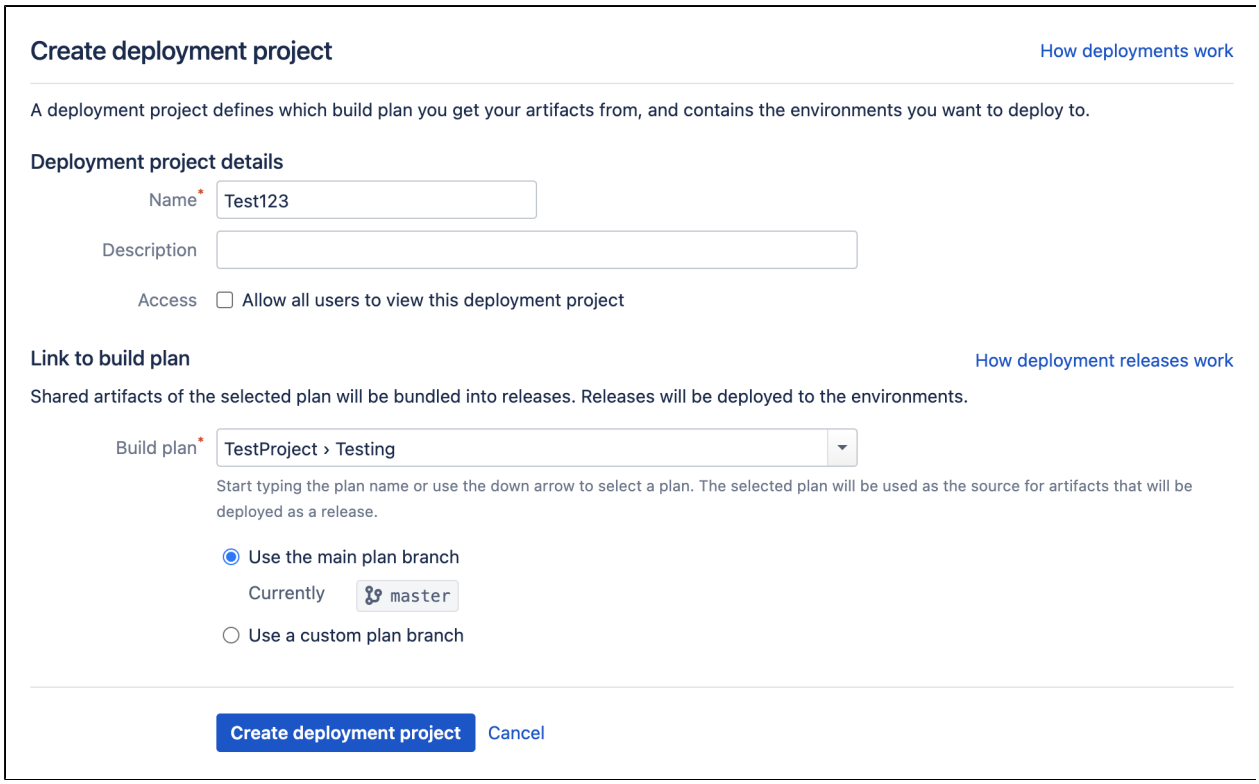
Related pages:

- Naming versions for deployment releases
- Deployments from branches

Creating a new deployment project

To create a new deployment project:

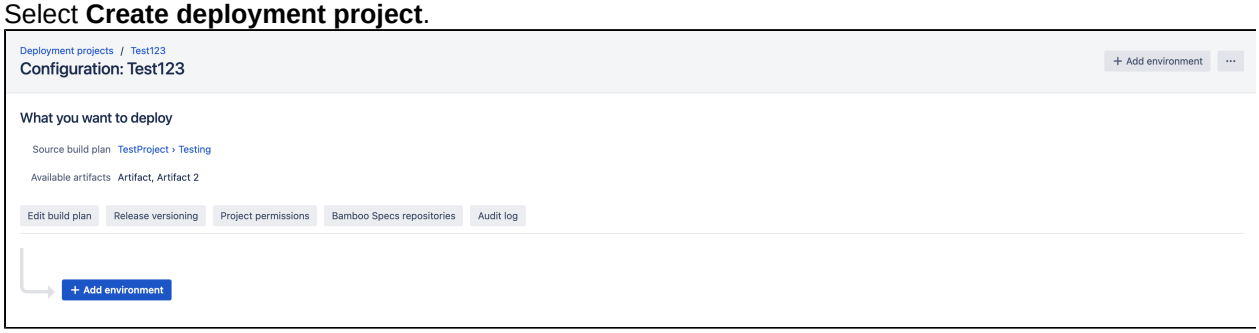
1. Select **Create** > **Create deployment project** in the menu from the top navigation bar. If your build plan has a plan branch, Bamboo will detect it and offer an additional field for completion.



2. Complete the Create deployment project screen using the following fields:

Field	Description	Optional?
Name	The name of your deployment project.	✗
Description	A brief description of your deployment project.	✓
Access	Select the checkbox to make the deployment project visible to all users.	✓
Build plan	The name of the plan you wish to associate with the deployment project. This field identifies the source of your deployment artifacts.	✗
Use the main plan branch/Use a custom plan branch	The plan branch that you wish to deploy. Bamboo will auto detect available plan branches for you.	✗

3. Select **Create deployment project**.



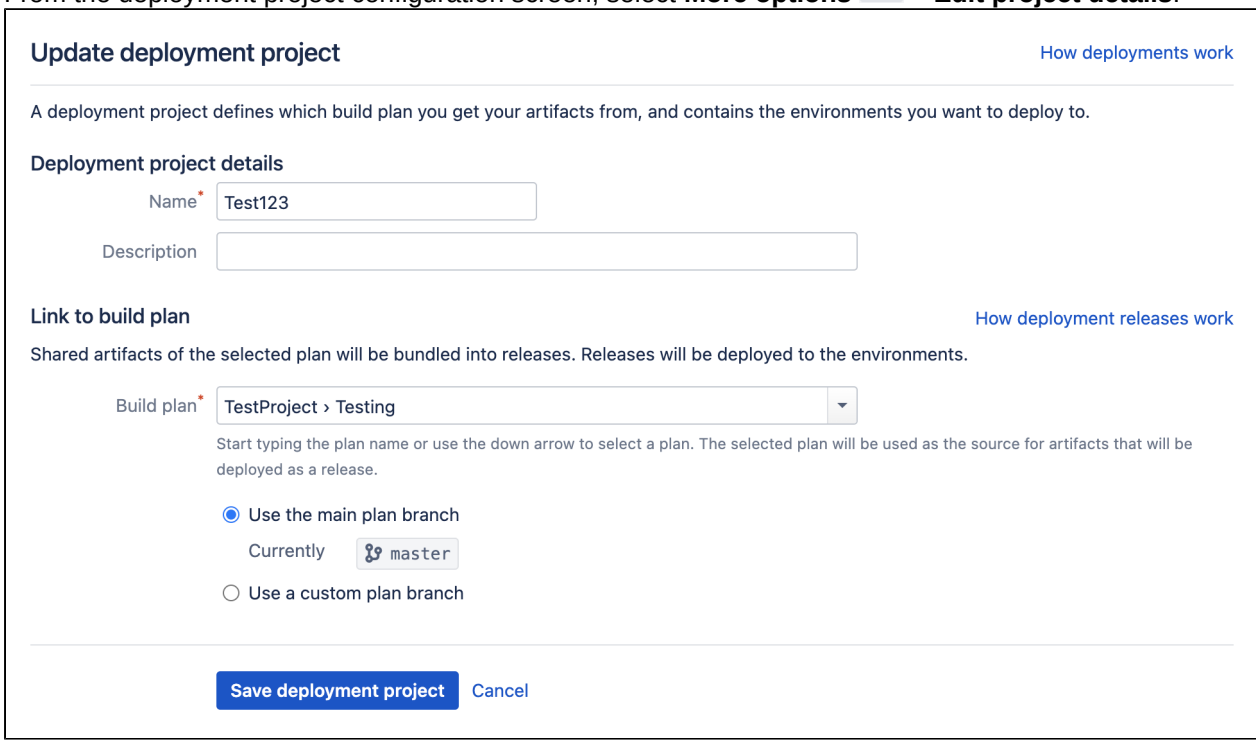
Your deployment project has been created with the build plan relation, name, and description you specified. It is now ready for configuration.

Editing the details of an existing deployment project

Bamboo allows you to edit the details of an existing deployment project.

To edit the details of an existing deployment project:

1. From the deployment project configuration screen, select **More options** ··· > **Edit project details**.



2. Complete the Update deployment project screen using the following fields:

Field	Description	Optional?
Name	The name of your deployment project.	✗
Description	A brief description of your deployment project.	✓
Build plan	The name of the plan you wish to associate with the deployment project. ***Hint:*** This field identifies the source of your deployment artifacts.	✗
Use the main plan branch/Use a custom plan branch	The plan branch that you wish to deploy. This option will only display if your plan has a valid branch, as described above.	✗

3. Select **Save deployment project**.

Configuring release naming

Bamboo's release naming configuration allows you to control:

- What Bamboo will call the next release the deployment project generates
- Automatic incrementing of the release number each time a new release is created
- Automatic incrementing of the release number as specified by a global variable each time a new release is created

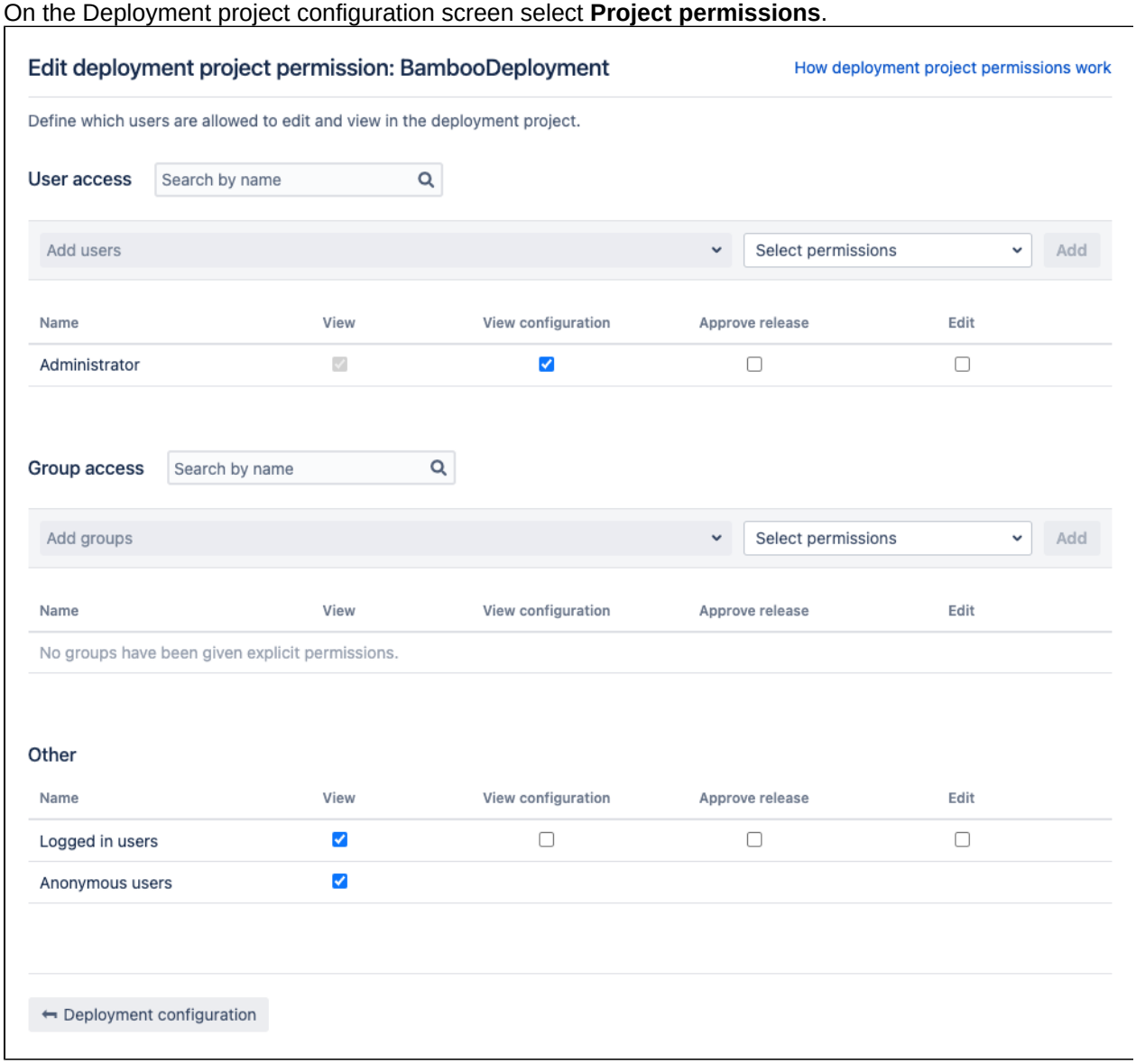
See Naming versions for deployment releases for more information.

Configuring deployment project permissions

Bamboo gives you control over who has permission to view, view the configuration, approve a release, and edit your deployment project.

To configure your permission strategy:

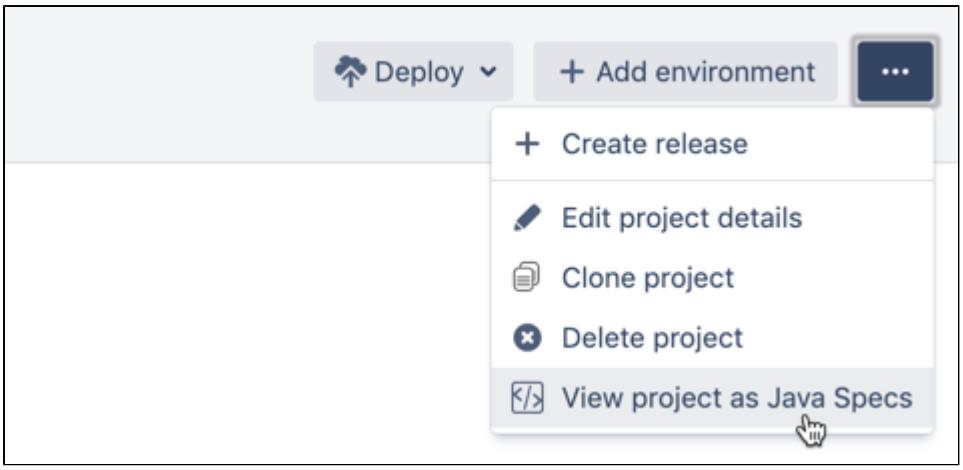
1. On the Deployment project configuration screen select **Project permissions**.



2. Select **Add user** or **Add Group** to search for and add users or groups.
3. Check the relevant **View**, **View configuration** (Bamboo Data Center only), **Approve release**, and **Edit** permission boxes to assign your desired permission scheme, and select **Add**.

Viewing a Bamboo deployment project as Java or YAML Specs

Bamboo instance administrators can view the deployment configuration as Java Specs in **Deployment project configuration** > ··· > **View Project as Java Specs** or **View Project as YAML Specs**.



Naming versions for deployment releases

Bamboo provides a range of options that allow you to control your release naming scheme. You can specify how Bamboo handles release versioning, and control automatic incrementing between releases.

Bamboo allows you to use:

- Simple incremental numbering
- Advanced numbering based upon Bamboo variables

Bamboo also allows you to manually override automatic release settings when you create a new release.

On this page:

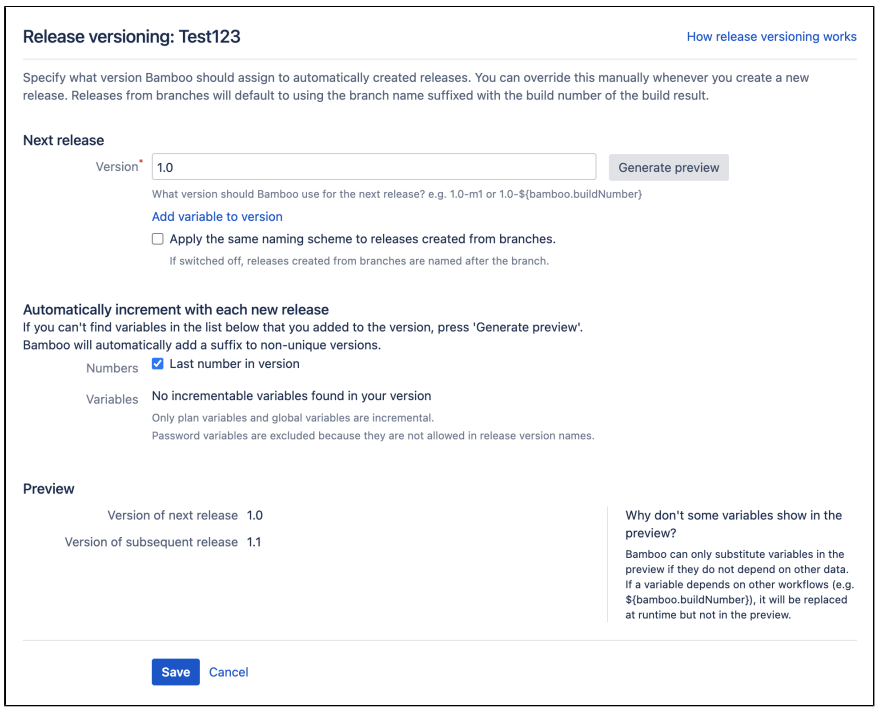
Simple release versioning

Simple release versioning allows you to specify a starting release number, for example, 1.0, which Bamboo will automatically increment. When using simple release versioning, Bamboo will increment the final number in the release name. For example:

Release name	Incremented release name
1	2
1.1	1.2
1.11	1.12
1.0.1	1.0.2

To configure simple release naming:

1. Select **Release versioning** on the Deployment project configuration screen.



2. Complete the fields using the following data:

Field	Description	Optional?
Next release	The identification for the next release name that Bamboo will create. In simple release naming, you should use something straightforward like 1.0.	✗
	If you want the naming scheme for the release to be applied to branches, select **Apply the same naming scheme to releases created from branches**.	✓
Automatically increment with each new release	Select the **Numbers** checkbox to automatically increment the release number according to the Next release field, as defined above. If you leave this box unchecked, no release number incrementing will occur.	✓
Preview	This field allows you to preview what the next release name will look like. To view the preview, select the **Generate preview** button. **Note:** In some cases, a preview may not be available.	-

3. Select **Save**.

Release versioning using variables

Release versioning using variables allows you to develop more complex naming schemes, based upon variables set up within Bamboo. You can use global, plan, and build variables in your releasing scheme.

> For security reasons, you can't use password variables in version names.

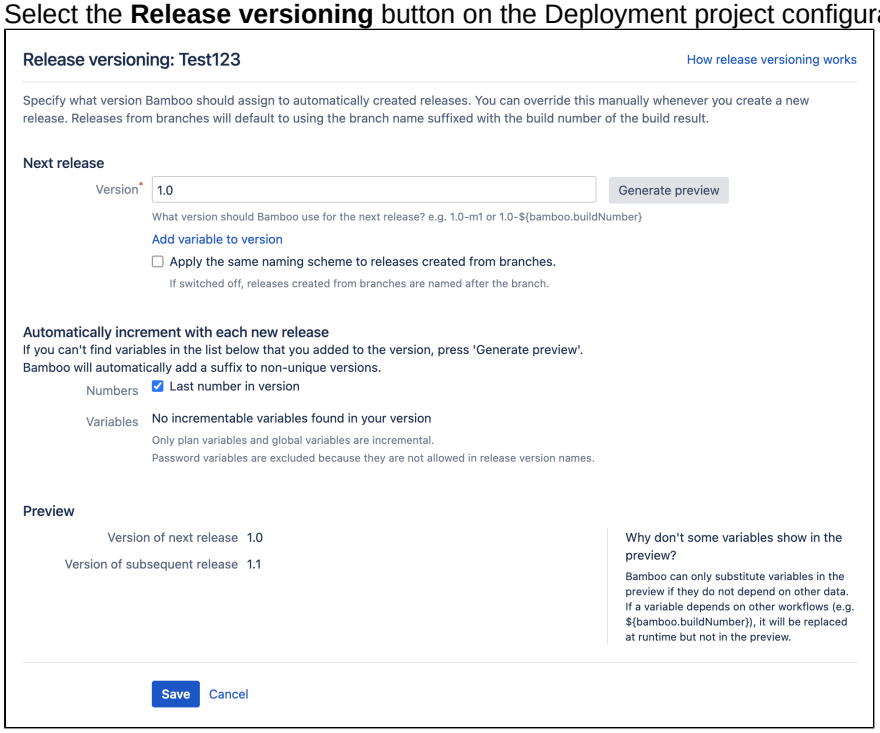
Example

You may have a plan variable called "planvar" with a value of "m6". By including this variable key within the **Next release** field, Bamboo will automatically add the variable value to the next release name, and increment it accordingly:

Variable key	Variable value	Next release	Next release version	Subsequent release version
planvar	m6	1.0-${bamboo.planvar}	1.0-m6	1.1-m7

To configure release naming using variables:

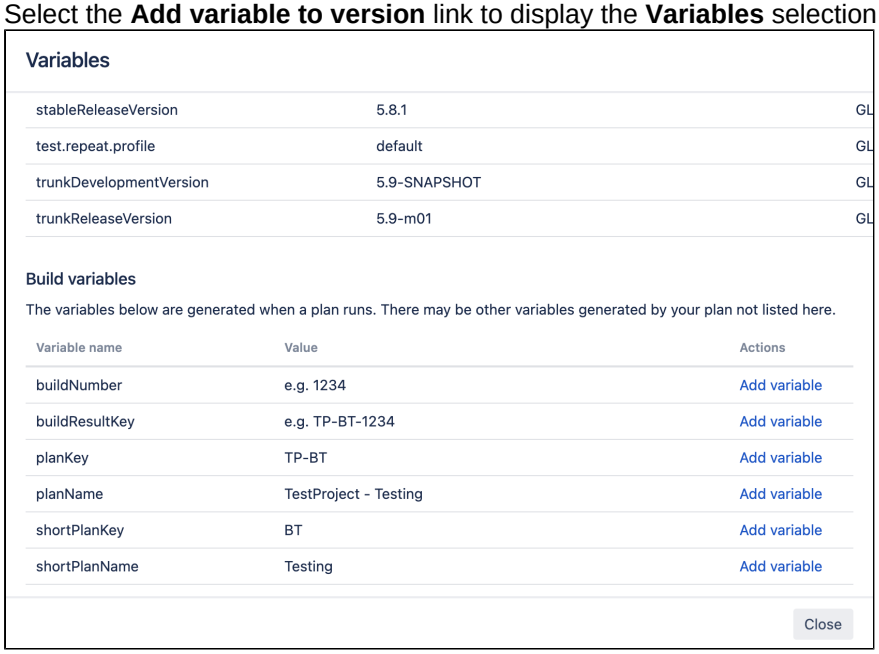
1. Select the **Release versioning** button on the Deployment project configuration screen.



2. Complete the **Version** field using the following data:

Field	Description	Optional?
Version	The identification for the next release name that Bamboo will create. In simple release versioning, you should use something straightforward like 1.0.	✗

3. Select the **Add variable to version** link to display the **Variables** selection screen:



4. Select **Add variable** to include the variable in your release naming scheme. Then select **Close** to return to the **Release versioning** screen.
5. Complete the remaining fields using the following data:

Field	Description	Optional?
Automatically increment with each new release	1. Select the **Numbers** checkbox to automatically increment the release number according to the **Next release** field, as defined above. If you leave this box unchecked, then no release number incrementing will occur. 2. Select the **Variables** checkbox to automatically increment selected variable (s) when a new release is created. If matching plan variable exists, its value is incremented, otherwise matching global variable is incremented. Additionally, if a release is created from a branch while the **Apply the same naming scheme to releases create from branches** option is on, the branch variable is incremented (as long as it exists). Please note, injected variable values (from the Inject Variables task) cannot be automatically incremented using this option.	✓
Preview	This field allows you to preview what the next release name will look like. To view the preview, select the **Generate preview** button. **Note:** In some cases, a preview may not be available.	-

6. Select **Save**.

Example release versioning schemes

Bamboo also allows you to use combinations of simple and variable release naming. The following table provides examples of combined naming schemes and demonstrates how careful control of the **Numbers** and **Variables** checkboxes can be used to customize your scheme.

Naming scheme	Next release field	Numbers checkbox	Variables checkbox	Variable value	Next release name	Subsequent release name
Static naming	1.0	✗	✗	-	1.0	1.0
Naturally unique variable	1.0-${bamboo.buildNumber}	✗	✗	13	1.0-13	1.0-13
Number incrementing	1.0	✓	✗	-	1.0	1.1
Number incrementing + static variable	1.0-${bamboo.appName}	✓	✗	${bamboo.appName} Plan or Global variable: Awesome	1.0-Awesome	1.1-Awesome
Variable incrementing	1.0-${bamboo.milestone}	✗	✓	${bamboo.milestone} Plan or Global variable: m6	1.0-m6	1.0-m7
Number and variable incrementing	1.0-b${bamboo.appNumber}	✓	✓	${bamboo.appNumber} User defined variable: 1567	1.0-b1567	1.1-b1568

Creating a deployment environment

Once you have created and configured your new deployment project, you can create environments for it to deploy to. Bamboo allows you to create multiple deployment environments and also allows you to manage:

- Environment details such as the name, description, and deployment prerequisites
- Tasks
- Triggers
- Permissions
- Agents
- Notifications
- Variables

To create a new deployment environment you will need to:

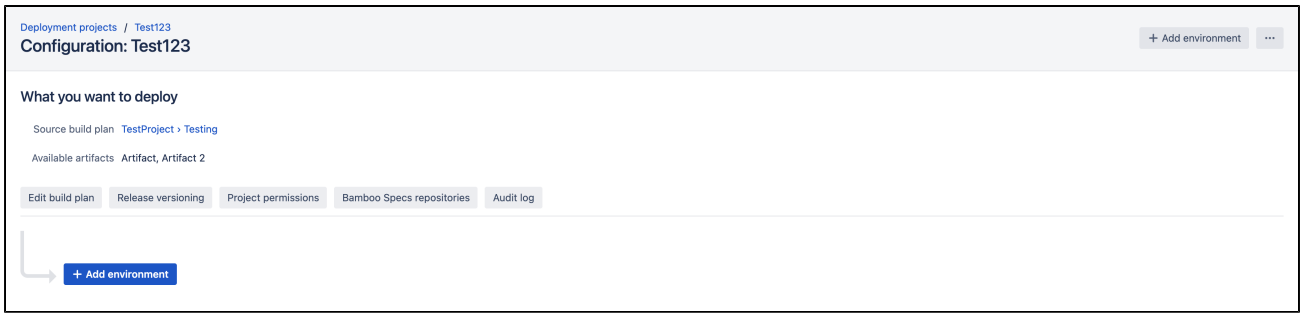
1. Provide a name that represents your environment e.g. "Test" or "Production".
2. Provide a description of the function of your environment.
3. Decide on the release approval policy the environment should use.

On this page:

Related pages:

- Tasks for deployment environments
- Triggers for deployment environments
- Agents for deployment environments
- Notifications for deployment environments
- Variables for deployment environments
- Permissions for deployment environments
- Requirements for deployment environments
- Release approval policy for deployment environments

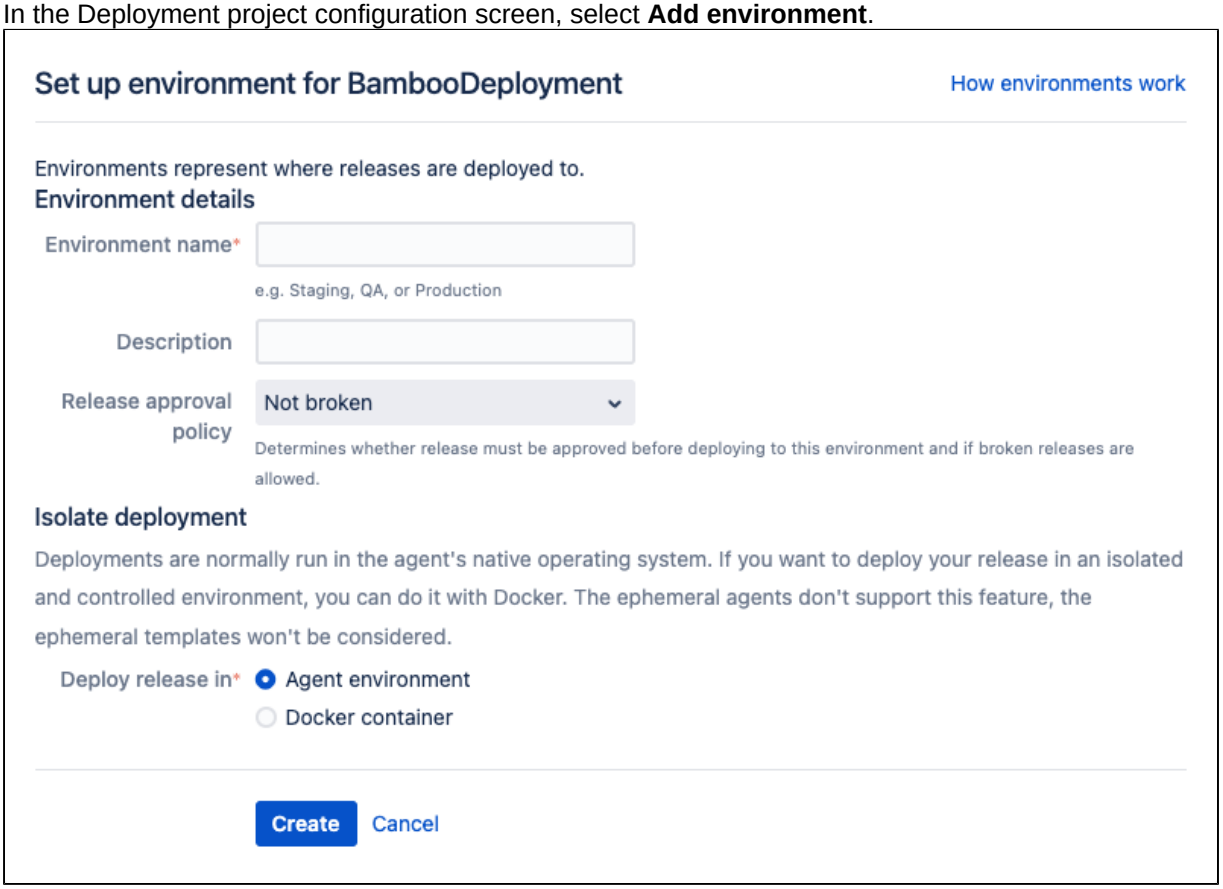
Deployment environments are added from the Deployment project configuration screen:



Creating a new deployment environment

1. From the top navigation bar, select **Deploy** > **All deployment projects**.
2. Select the edit icon for the deployment project you want to edit.

3. In the Deployment project configuration screen, select **Add environment**.



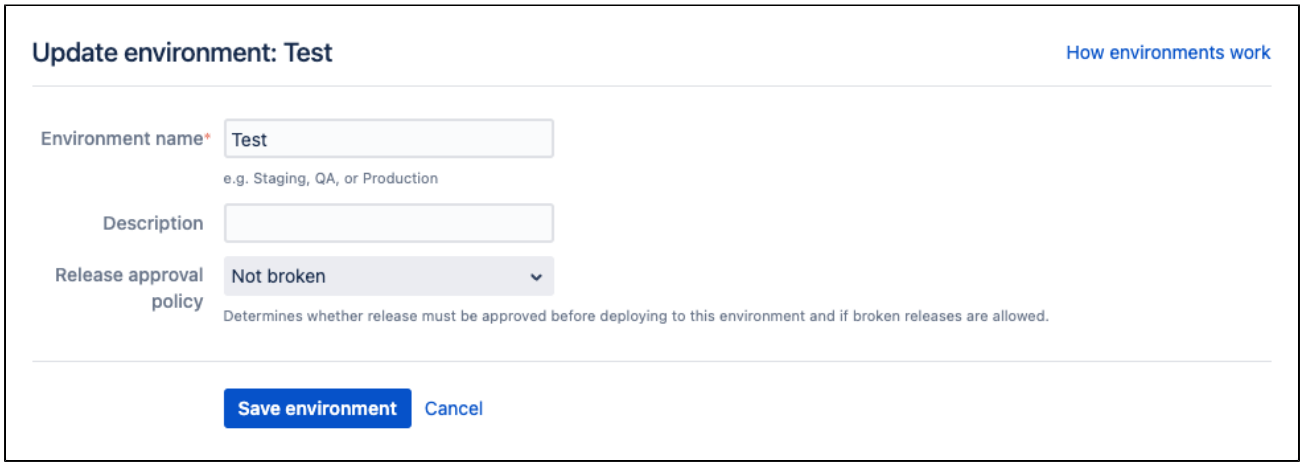
4. Provide your deployment environment details.
5. Select if you want to run your deployment in the agent environment or in a docker container. See Docker Runner.
6. Select **Create**.

> Your newly created environment appears with a red exclamation mark because no tasks have been defined for this environment yet.

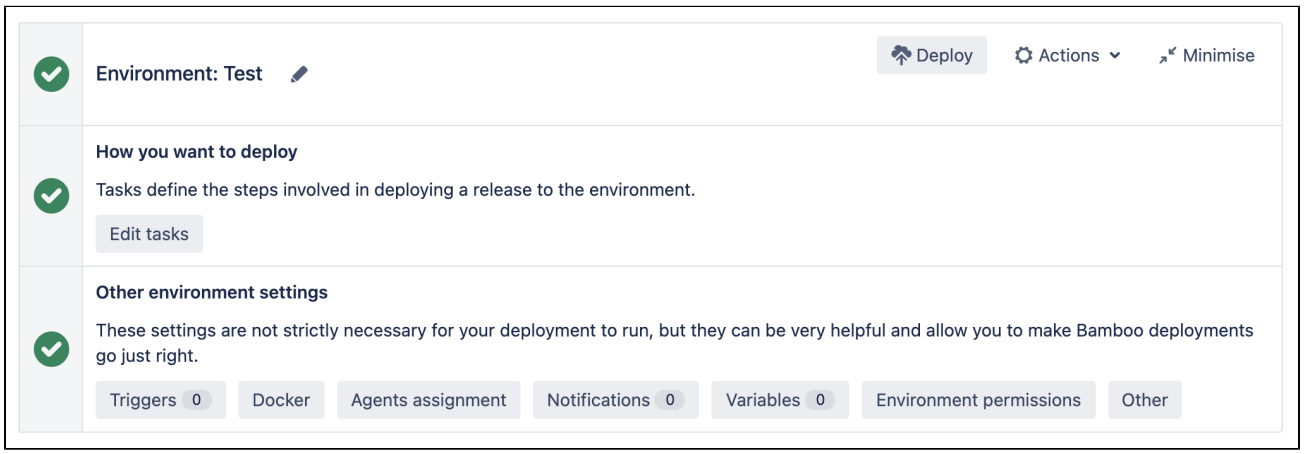
Learn more about configuring tasks for deployment environments
Learn more about the release approval policy for deployment environments

Using the deployment environment panel

All deployment environments are managed from the Deployment project configuration screen. By default, when the screen loads, each environment panel is displayed in its collapsed state. Select **Edit** to expand the deployment environment panel:



When expanded, the environment panel shows three separate sub-panels:



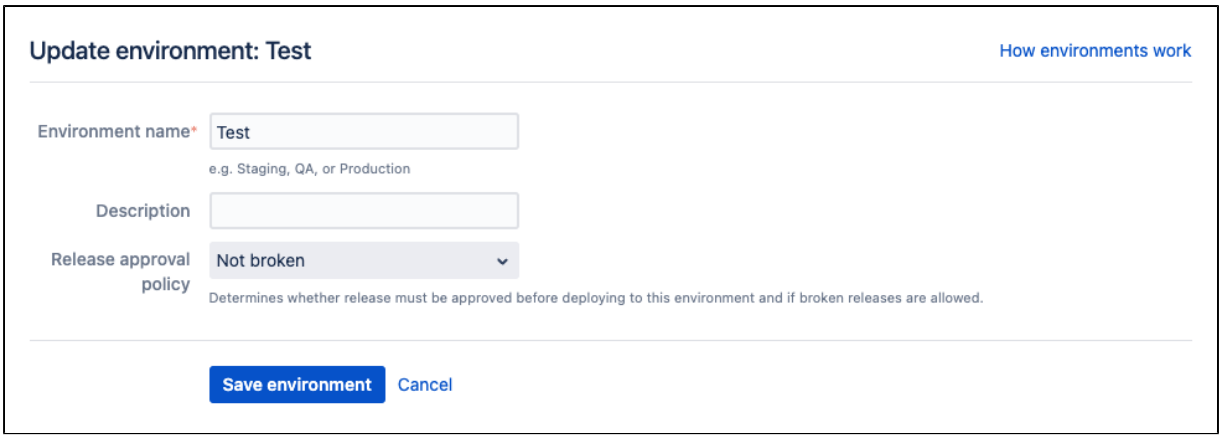
The three sub-panels provide the following functionality:

Sub-panel	Functionality	Description
Environment	• Deploy • Actions • Minimize	• Manually deploys to the environment • Allows the user to View, Delete, or Move down the environment • Minimize the environment panel back to its collapsed state
How you want to deploy	• Edit tasks	Allows the user to edit the tasks associated with the environment
Other environment settings	• Triggers • Docker • Agents assignment • Notifications • Variables • Environment permissions • Other	A set of optional settings that make your Bamboo deployments run more smoothly

Editing environment details

Bamboo allows you to change both the environment name and the description. To edit these details:

1. Expand the environment panel and select the edit ( ) icon next to the environment name. The Update environment screen will display:



2. Complete the Update environment screen using the following fields:

Field	Description	Optional?
Environment name	The name of the environment	✗
Description	A brief description of your environment	✓
Release approval policy	The required release approval state before the environment can be deployed	✗

3. Select **Save environment** to save your changes.

Tasks for deployment environments

Once you have created and configured your new deployment project and deployment environments, you can set up associated tasks for the deployment process. Bamboo allows you to execute a range of different tasks upon deployment including:

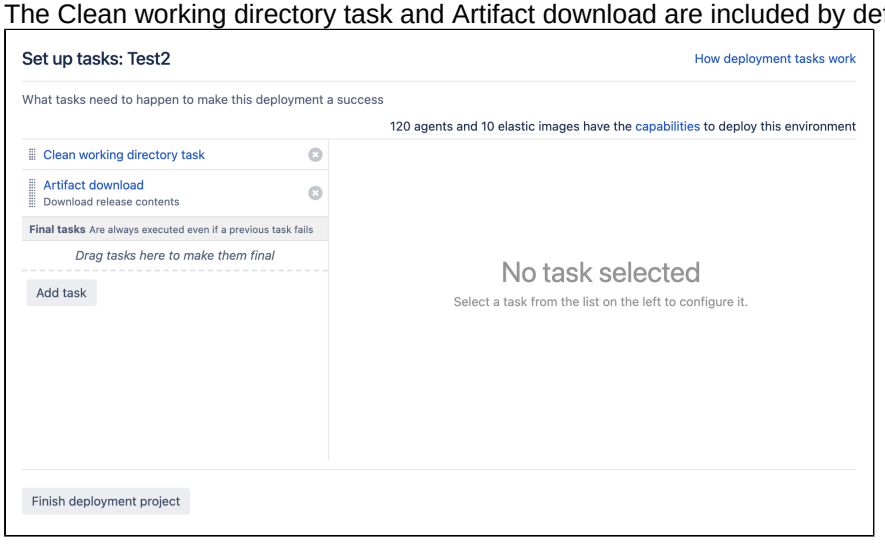
- Bash and other shell commands
- Bespoke written scripts
- SCP, SSH, and Artifact handler tasks
- Ant executables
- Maven 1.x, 2.x, and 3.x executables
- Tomcat executables

On this page:

Add an environment task

You can add tasks to a deployment environment either while you create the environment or afterwards. You can modify tasks any time after creating them.

1. Open your deployment project and expand the relevant environment panel.
2. Select **Set up tasks** (under **How you want to deploy**).
   The Clean working directory task and Artifact download are included by default:



3. Select **Add task** and find the desired task. Only tasks applicable to the deployment environment will be available for selection.
4. Configure the task according to the needs of your deployment project. Different tasks will have different requirements.

> Remember that capability and requirement matching is still in effect for deployment environments. If your task does not have the right capabilities it will not be executed, even if the relationship has been defined.

5. Select **Save** when you have finished.
6. Select **Finish deployment project** to return to the deployment project page.

Some useful deployment tasks

Deploying with Tomcat

You can use Bamboo to deploy and manage your Java web application with Tomcat 6 or 7, without having to directly interact with Maven, Ant, or write special scripts.

See Using Tomcat with Bamboo for continuous deployment.

Copying and moving files with SCP

You can use the Bamboo SCP task to upload files from Bamboo directly to a remote server as part of a Bamboo job. The SCP task is able to copy multiple files and preserves the directory structure for the copied files.

See Using the SCP task in Bamboo.

Deploying ASP.NET applications with MSDeploy

You can use Bamboo to deploy your ASP.NET web application by using a Script task to run `msdeploy.exe`. The MSDeploy command-line syntax is available at: http://technet.microsoft.com/en-us/library/dd569106(v=ws.10).aspx

Assign a final task

Once all of your tasks have been configured, you may assign some or all of them to be Final Tasks. Final Tasks are always executed at the end of the build.

1. Open your deployment project and expand the relevant environment panel.
2. Select **Edit tasks** (under **How you want to deploy)**.
3. To make a task final, simply drag the task below the **Final tasks** bar.
4. Select **Finish deployment project** to return to the deployment project page.

Triggers for deployment environments

Use deployment triggers for automatic management of how and when Bamboo starts deployment projects.

Deployments can be triggered automatically based on:

- a successful build of a plan branch
- a successful deployment to some other environment
- a successful build of a plan stage
- a schedule (specific time and date, at an interval, or Crone-based)

Note: You can also start deployments manually.

When an automatic deployment starts, Bamboo creates a new release based on the latest successful build of the plan branch that is defined in the deployment trigger. If there is only one branch in the plan, it is selected by default.

On this page:

Configuring Bamboo deployment triggers

You can customize the deployment by specifying the details of each trigger. Triggers configuration lives in the environment settings of a deployment project.

If you don't know where to find the trigger configuration in Bamboo, see How to find deployment triggers configuration in Bamboo.

Common parameters

The list of parameters that are the same for all deployment triggers.

Trigger description (Optional)
A meaningful name of a trigger by which you can identify the trigger in the GUI.

Disable this trigger (Optional)
Select the check box to ignore the trigger in deployments.

Trigger-specific parameters

The list of parameters that are specific for each deployment trigger type.

After successful build plan

The trigger starts a deployment after a successful build of the specified plan branch. If there is only one existing plan branch, it is selected by default and the branch selection options are hidden.

Branch to trigger this deployment (Required)
Specifies the branch that must be successfully built before the deployment.

You can specify the following:

- **use main plan branch**: displays the name of the plan branch set as the main plan branch
- **use a custom plan branch**: displays a selection list that is pre-filled with all branches in the deployment project

After successful deployment

Starts a deployment after a successful deployment on another environment.

Triggering environment (Required)
Specifies the environment on which a successful deployment must be performed to start the new deployment.

After successful stage

Starts a deployment after a successful build of the specified stage of a plan. If there is only one existing stage, it is selected by default.

Plan stage to trigger this deployment (Required)
Specifies the stage on which a successful deployment must be performed to start the new deployment.

Scheduled

Starts a deployment according to a customized schedule with artifacts from a specific branch.

Schedule (Required)

Select the edit ( ) icon to open the schedule editor. You can select from:

- Daily
- Days per week
- Days per month
- Cron expression

and provide the further details in the fields displayed following to the selection.

Branch to provide artifacts for this deployment (Required)

Specifies the branch from which Bamboo provides artifacts for the deployment.

You can specify the following:

- **use main plan branch**: displays the name of the plan branch set as the main plan branch
- **use a custom plan branch**: displays a selection list that is pre-filled with all branches in the deployment project

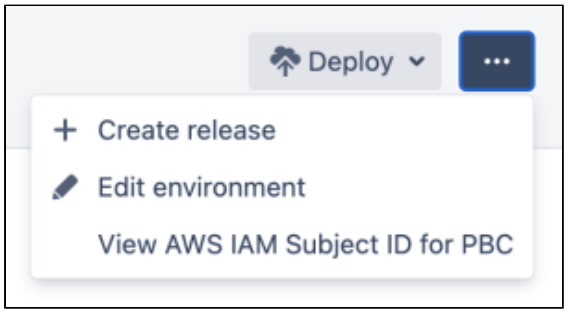
How to find deployment triggers configuration in Bamboo

Deployment triggers are set as part of the of the environment configuration for a deployment project.

To get to the environment configuration details view:

1. From the top navigation bar select **Deploy** > **All deployment projects**.
2. Select the name of an environment to display the environment details view.

3. Select ··· > **Edit environment**:



4. Once you are in the environment edit view, a list of all existing environments is displayed with expanded information about the environment that you want to edit.

   If you want to edit an environment that is different from the expanded one, you can select **Edit** next to the name of the environment.
5. In the **Other environment settings** section, select **Triggers.**
6. In the **Edit triggers** you can add, remove, or configure triggers.

Agents for deployment environments

Bamboo offers a range of optional settings to make your deployment project function more smoothly. Bamboo allows you to assign specific agents, elastic agents, or image configurations to execute the deployment for the environment.

Important Note

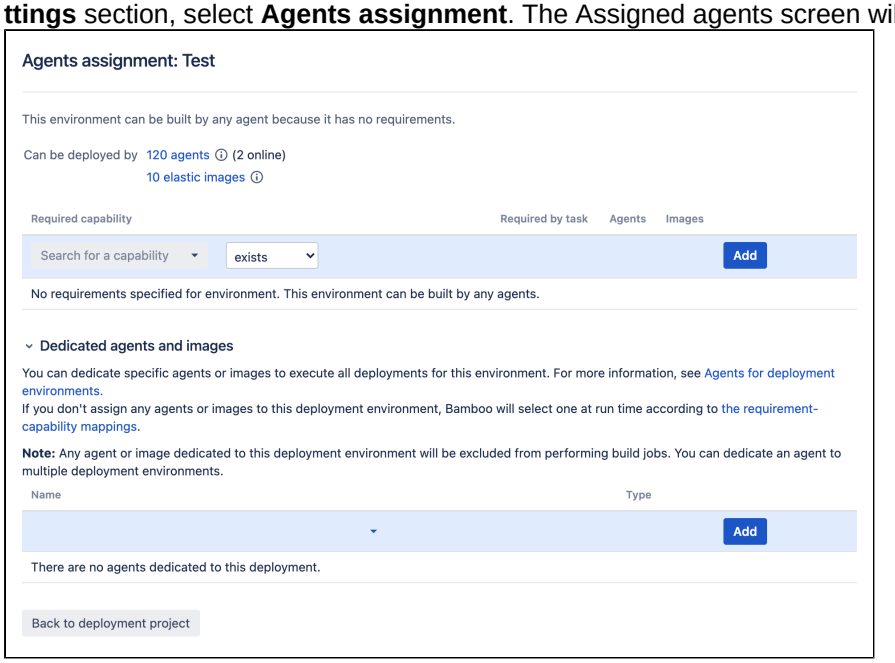
- Assigning agents to deployment tasks may reduce your build capacity. When an agent is assigned, no other builds or deployments can run on it unless they are also explicitly assigned to use that agent or image configuration.
- Starting from Bamboo version 7.1, by default only global administrators are allowed to add and remove agents assignments. You can changes that by selecting the **Allow users to dedicate agents to deployments** option in Security Settings.

Configuring deployment agents

Deployment environment agents are configured as part of the Other settings section of the environment panel.

To configure your deployment agent:

1. Open your deployment project and expand the relevant environment panel. In the **Other environment settings** section, select **Agents assignment**. The Assigned agents screen will display:



2. Enter an agent name, or use the drop down menu to select an appropriate agent.

   Only agents applicable to the deployment environment will be available for selection.

   Remember that capability and requirement matching still applies for deployment environments. If your agent does not have the right capabilities it will not be assigned at runtime even if the relationship has been defined.

3. Select **Add** to save your agent scheme.
4. You can remove an unwanted agent by selecting the associated cross on the right hand side of the screen.

Notifications for deployment environments

Bamboo offers a range of optional settings to make your deployment project function more smoothly. Notifications allow you to assign a specific notification scheme to events triggered by the deployment environment. Notification events include start and finish of a deployment, and may be delivered by any of:

- User or group notification
- Email
- Hipchat
- Instant Messaging

To set up a notification you will need to:

1. Select a triggering event
2. Configure a mechanism for delivering notifications

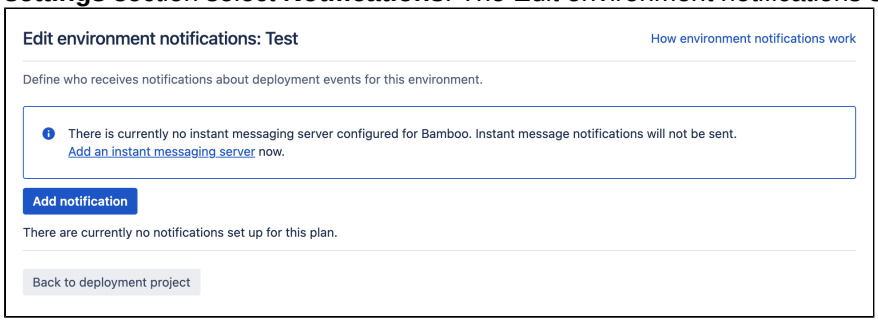
On this page:

Configuring deployment notifications

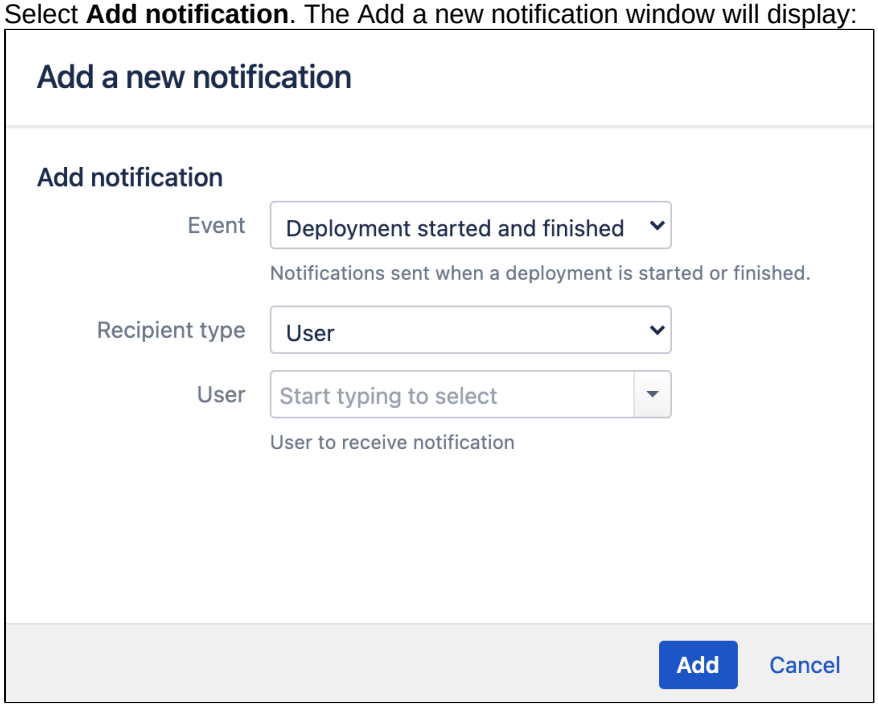
Deployment environment notifications are configured as part of the **Other environment settings** section of the environment panel.

To configure your deployment notifications:

1. Open your deployment project and expand the relevant environment panel. In the **Other environment settings** section select **Notifications**. The Edit environment notifications screen will display:



2. Select **Add notification**. The Add a new notification window will display:



3. Selecting the event to trigger the notification:

   Deployment started and finished
   Notification is issued when a deployment is started and finished

   Deployment finished
   Notification is issued only when deployment is finished

   And configuring the notification delivery system:

Recipient Type	Data requirements
User	Username of the user
Hipchat	Hipchat API token, Hipchat room name, Room participants notification
Group	Groupname
Email Address	Email address
IM Address	Instant messaging address

   Note: If you have not done so, you may need to set up an IM server for IM notifications to work correctly.
4. Select **Add**.

5. You can edit or remove individual notifications.

Variables for deployment environments

Deployment variables

Bamboo manages a number of standard reserved variables that are available when deploying a project.

Variables later in the following list override the previous ones in case of repeating names:

- global variables
- project variables of the plan linked to the deployment project
- plan variables of the plan linked to the deployment project
- release variables as defined below
- user variables defined at the environment level
- the autogenerated variables in the following table:

Variable	Description
`bamboo.agentId`	The id of the agent that the deployment is executed on.
`bamboo.agentWorkingDirectory`	The path to the working directory on the agent. This is not the same as the Bamboo working directory.
`bamboo.build.working.directory`	The path to the working directory for Bamboo. This is used by both the build plan and the deployment project.
`bamboo.deploy.environment`	The name of the environment that the release is to be deployed to.
`bamboo.deploy.project`	The name of the deployment project.
`bamboo.deploy.rollback`	True if the release being deployed is older than the release being replaced.

`bamboo.deploy.release`
 	
`bamboo.deploy.version`
	The name of the release that is being deployed. Either `.release` or `.version` can be used. Both return the name of the release being deployed.

`bamboo.deploy.release.previous`
 	
`bamboo.deploy.version.previous`
	The name of the release that is being replaced (if available). Either `.release` or `.version` can be used. Both return the name of the release being replaced.

`bamboo.resultsUrl`	The URL to the screen in Bamboo that displays build results.

On this page:

Related pages:

- Bamboo variables
- Defining global variables
- Defining plan variables
- Running a plan build manually

`bamboo.triggerReason.key`	The trigger key that caused the deployment to launch. For example: `com.atlassian.bamboo.plugin.system.triggerReason:ManualBuildTriggerReason`

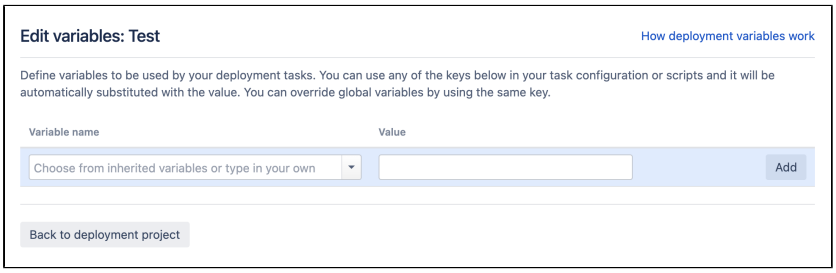
For Bamboo variables to do with build plans, and releases, see Bamboo variables.

Configuring variables for deployment environments

Deployment environment variables are configured as part of the Other environment settings section of the environment panel.

To configure an environment variable:

1. Open your deployment project and expand the relevant environment panel. In the Other environment settings section select **Variables**.
2. Enter a valid key and value into the relevant fields in the Variables screen.
3. Select **Add** to add the variable scheme.



4. You can remove unwanted variables by selecting the relevant cross icon next to variables.
5. Select **Back to deployment project** to return.

Permissions for deployment environments

Bamboo offers a range of optional settings to make your deployment project function more smoothly. Deployment environment permissions allow you to configure which groups or individuals can view, edit, or deploy a project.

> Note that the global Bamboo permissions still take precedence. Where a user has environment permissions enabled but project permissions disabled, they will still be unable to access a deployment environment. Please see Bamboo permissions and Creating and configuring a deployment project for more information on managing deployment project permissions.

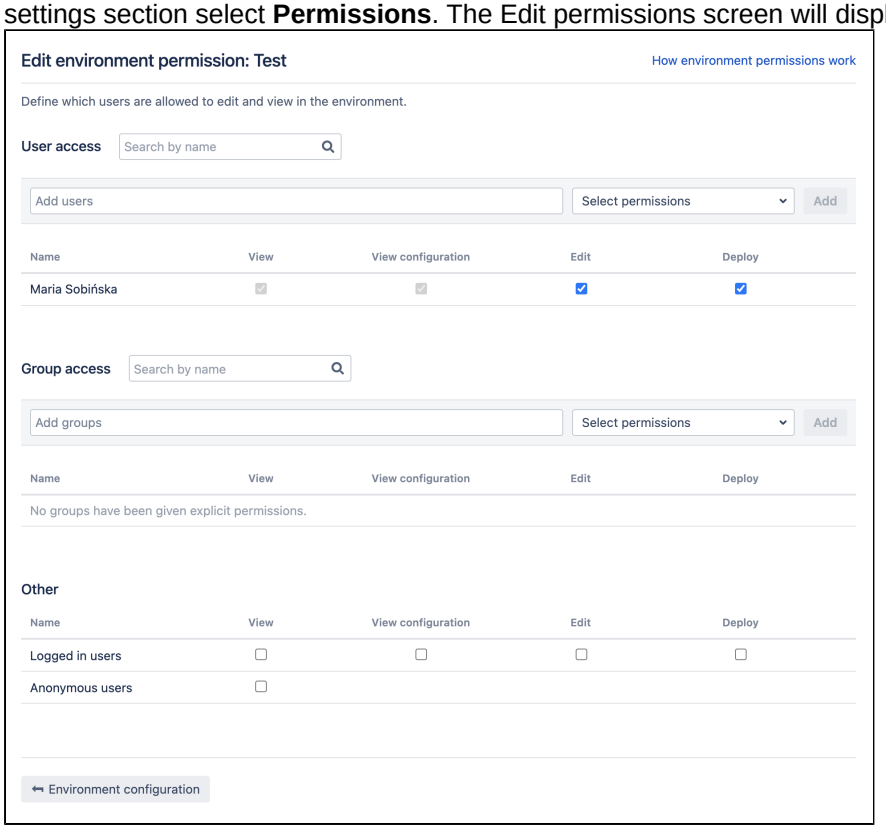
On this page:

Configuring deployment environment permissions

Deployment environment permissions are configured as part of the Other environment settings section of the environment panel.

To configure your permission strategy:

1. Open your deployment project and expand the relevant environment panel. In the Other environment settings section select **Permissions**. The Edit permissions screen will display:



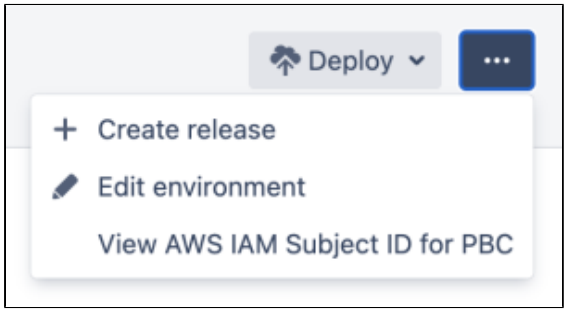
2. Select **Add users** or **Add groups** to search for and add users or groups.
3. Select the relevant View, View configuration (Bamboo Data Center only), Edit, or Deploy permission checkboxes to assign your desired permission scheme.
4. Select **Save** to save your permission scheme.

Requirements for deployment environments

Specify requirements for deployment environments to route the deployment plan execution to agents with matching capabilities. This way, agents that aren't executing dedicated deployments will be available for other jobs.

To view and manage requirements for deployment environments:

1. From the top navigation bar select **Deploy** > **All deployment projects**.
2. Select the name of an environment to display the environment details view.
3. Select ··· > **Edit environment**.

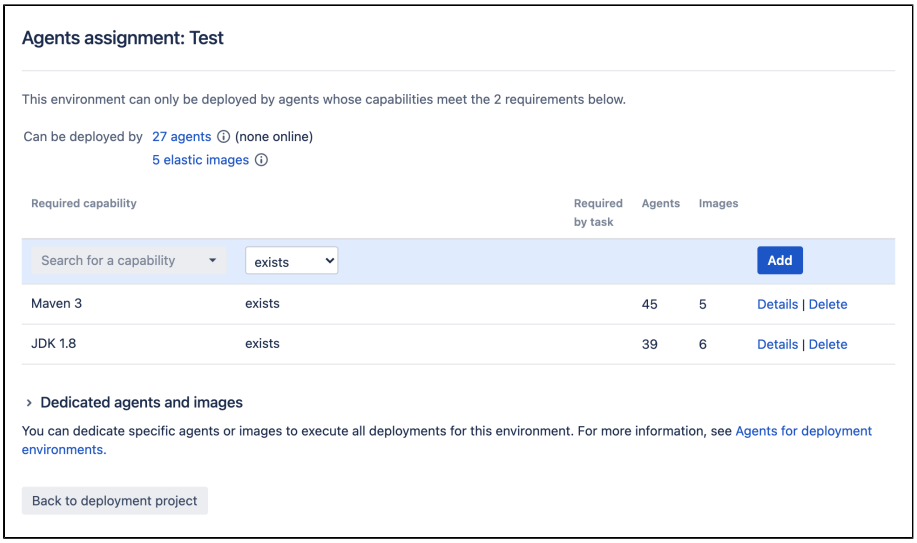


4. Once you are in the environment edit view, a list of all existing environments is displayed with expanded information about the environment that you want to edit.

   If you want to edit an environment that is different from the expanded one, you can select **Edit** next to the name of the environment.
5. To manage and view existing requirements, select **Agents assignment**.

In the **Agents assignment** view you can:

- add and remove capabilities required from an agent to deploy an environment
- check which agents and/or elastic images meet the requirements of your deployment environment
- view capabilities that are required based on the requirements of the deployment tasks specified for the environment

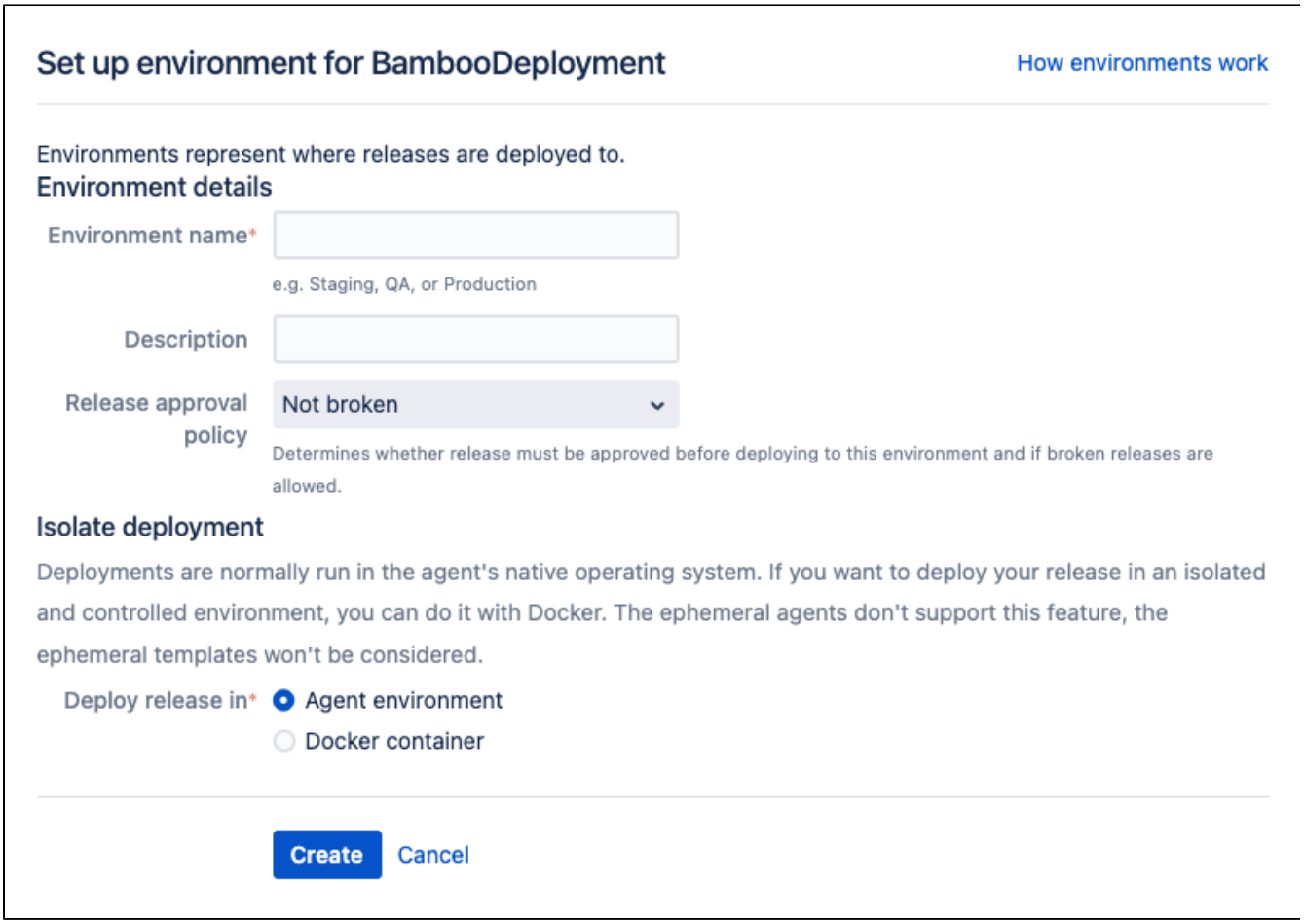


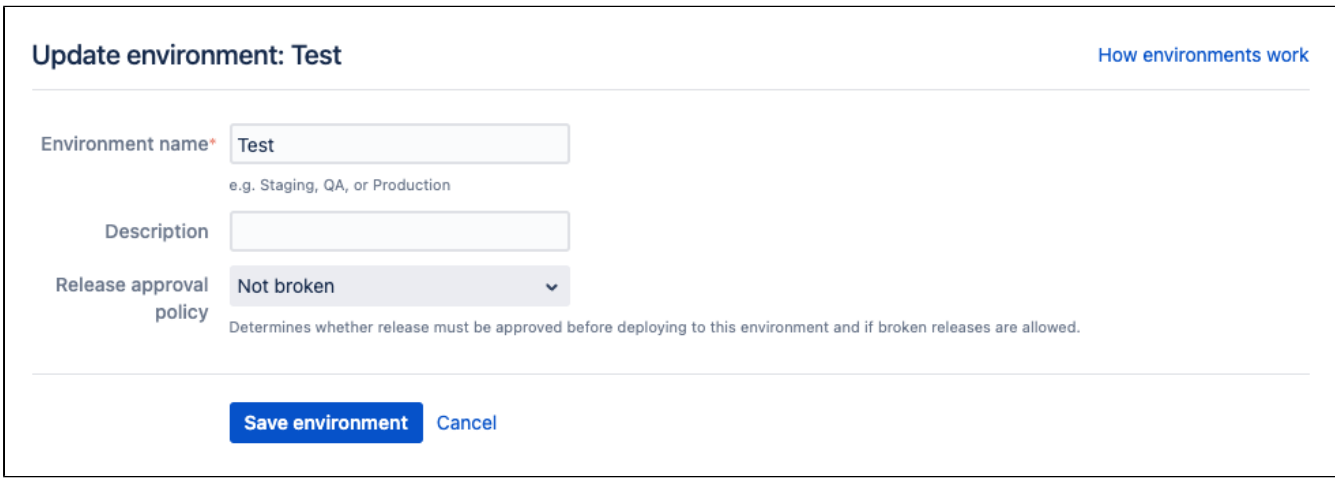
Release approval policy for deployment environments

Every environment has its own **Release approval policy** setting that controls whether a release must be approved before it can be deployed to an environment and if broken releases are allowed. This setting is configurable through both the Bamboo web interface and Bamboo Specs.

Set the release approval policy in Bamboo

In the Bamboo web interface, you can define the release approval policy when setting up a new deployment environment or when editing the settings of an existing deployment environment. Learn how to create or edit a deployment environment in Bamboo





The following release approval policies are available:

- **Not broken** — the release can't be marked as broken. A broken release mark has a higher priority than any number of approvals. This is the default choice.
- **Approved** — the release must be approved at least once. However, no number of approvals can override a broken release mark.
- **None** — there are no requirements at all. Any release can be deployed to such an environment.

The release approval policy applies to both deployments started manually and deployments triggered automatically. Because of that, the deployment won't start if the release approval policy requirements are not met at the time of the trigger occurring.

Set the release approval policy in YAML Specs

To set the release approval policy for an environment in Bamboo YAML Specs, to the environment definition field, add a `release-approval-prerequisite` key with one of the following values:

- `not-broken`
- `approved`
- `none`

For example:

```
version: 2
# ...
QA:
  release-approval-prerequisite: approved
  # ...
```

Set the release approval policy in Java Specs

To set the release approval policy for an environment in Bamboo Java Specs, to the environment definition, add the `releaseApprovalPrerequsitie` property with one of the following values:

- `Environment.ReleaseApprovalPrerequisite.APPROVED`
- `Environment.ReleaseApprovalPrerequisite.NOT_BROKEN`
- `Environment.ReleaseApprovalPrerequisite.NONE`

For example:

```
Environment environment = new Environment("QA")
    .releaseApprovalPrerequisite(Environment.ReleaseApprovalPrerequisite.APPROVED);
```

Managing deployment projects

Bamboo makes it easy to monitor and manage your deployment projects.

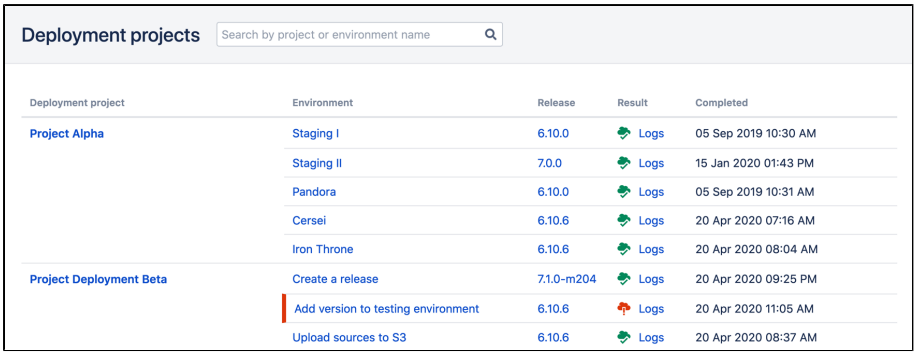
A single dashboard allows you to monitor deployment environments, deployment status, releases, and time stamps. It also allows you to edit and deploy your projects.

On this page:

Manage deployment projects

Deployment projects are viewed and managed from the All deployment projects screen. Think of this as a dashboard for all your deployment projects.

Select **Deploy** > **All deployment projects** from the top navigation bar:



The project list includes the following useful information:

Name
The name of the deployment project.

Environment
The environment the release was deployed to.

Release
The release artifact that Bamboo deployed, or attempted to deploy, to that environment.

Result
The result of the deployment, and a link to the associated logs.

Completed
The time and date stamp of the deployment, or the time spent deploying so far.

Actions
Actions you can perform: *Edit* and *Deploy*.

Broken deployments are indicated by a vertical red line beside the environment name and a red deployment icon.

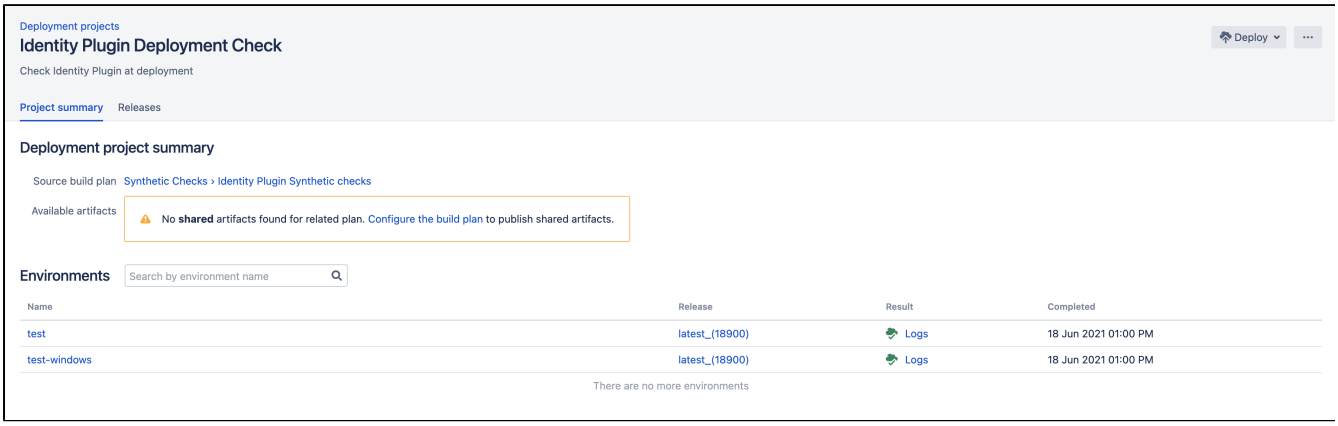
View a particular deployment project

You can drill down into an individual deployment project from the All deployment projects screen (described above) by selecting the name of a project. You can check on the following:

- Associated environments
- Release history
- Project artifacts details

Project summary

The **Project summary** tab shows the status of the environments associated with the deployment project:



Details include:

Environment
The environment the release was deployed to.

Release
The release artifact that Bamboo deployed, or attempted to deploy, to that environment.

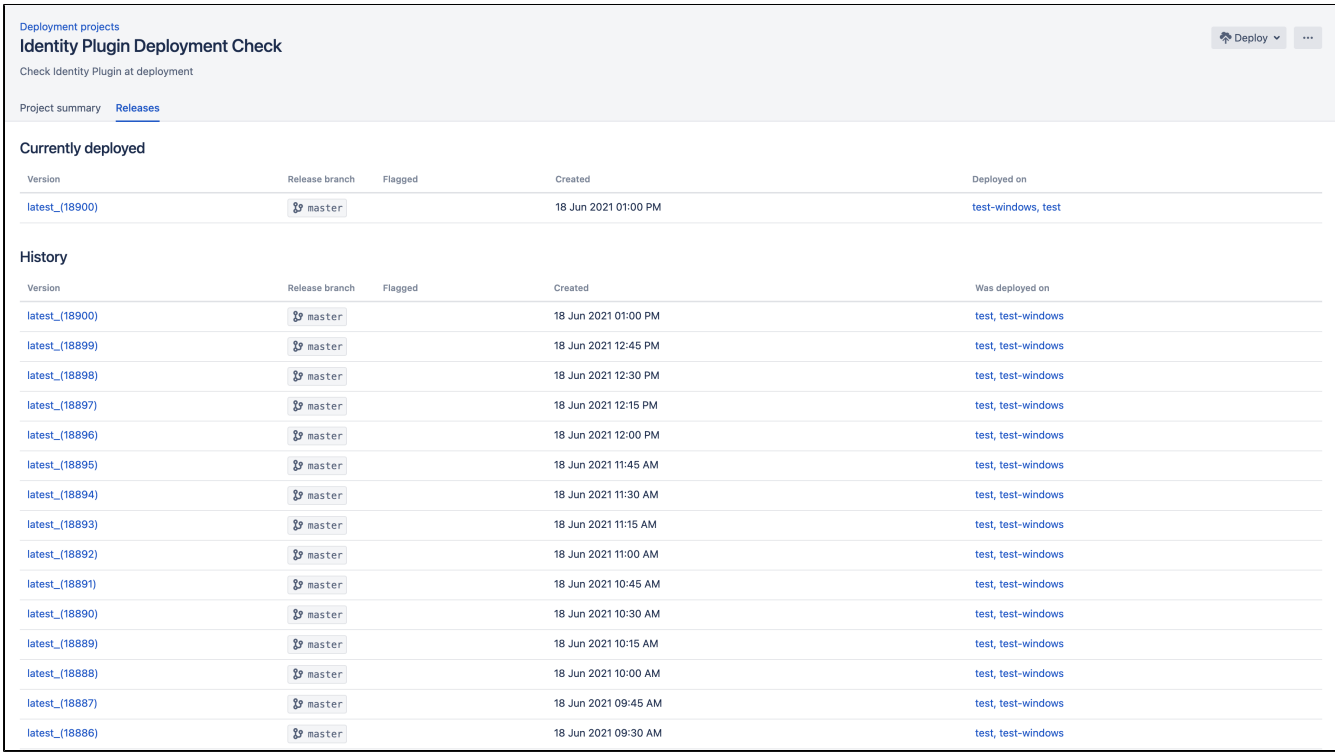
Result
The result of the deployment, and a link to the associated logs.

Completed
The time and date stamp for the deployment, or the time spent deploying so far.

Actions
Actions you can perform: *Edit* and *Deploy*.

Releases

The **Releases** tab provides details of the currently deployed release and the history of previous releases associated with the deployment project.



Release details include:

Version
The name of the release artifact.

Release branch
The branch the release was derived from.

Flagged
Any flags that have been applied to the release. Values are *Broken* and *Approved*. Neutral flags remain blank.

Created
The time and date stamp for when the release was created.

Deployed on
The environment the release was deployed to.

Manually starting a deployment

Bamboo can start deployments either by automated triggers, or by starting the process manually.

Manually executing the deployment gives you the ability to start the process at your convenience, without having to wait for a scheduled event or trigger to take place.

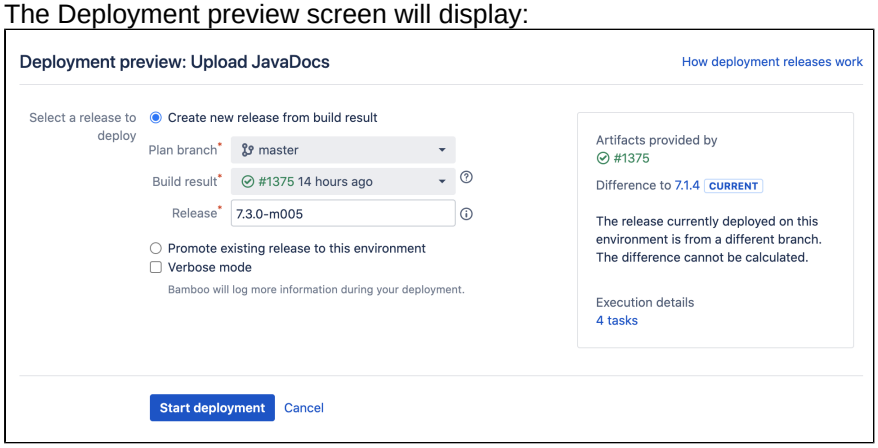
On this page:

Manually starting a deployment

Deployment projects can be viewed and managed from the All deployment projects screen. Think of this as a dashboard view of all of your deployment projects. You can also start deployments from this screen.

To manually start a deployment:

1. From the All deployment projects screen, select the deploy icon next to the desired deployment project. The Deployment preview screen will display:



   The deployment preview screen comprises settings, preview, and information. Bamboo will attempt to display a preview or information to reflect the choices made on the settings side.

2. Using the radio buttons, decide if you wish to create a new release from a build result or promote an existing release to the deployment environment.
   - If creating a new release from a build result:
     a. Select **Create new release from build result**.
     b. Select the **Plan branch** you wish to use.
     c. Select the Build result you wish to use.
        You can only select results from successful builds, and since (and including) the last release created on this particular branch.
     d. Check that the name of your release is correct.
        If you need more information about where the default name comes from, select ⓘ.
   - If promoting an existing release:
     a. Select **Promote existing release to this environment**.
     b. Select the **Plan branch** you wish to promote (optional).
     c. Select the release that you wish to promote.
3. Select **Verbose mode** if you want Bamboo to log more information during your deployment.
4. Select **Start deployment**.

Deployments from branches

What are branch deployments?

Branching is an important tool in your development process, as it offers a very powerful way to let developers work in isolation on different aspects of a software project.

Plan branches represent a build for a branch in the version control system. The plan branch inherits all of the configuration defined by the parent plan, and any new branch created is automatically built and tested using the same build configuration as the parent. When the plan branch build succeeds, it can be automatically or manually merged back into master.

Branch deployments extend plan branches by allowing users to create a deployment release from any plan branch.

Learn more about branching strategies: Bamboo Best Practice - Branching and DVCS.

On this page:

Related pages:

- Understanding deployment releases
- Creating and configuring a deployment project
- Manually starting a deployment
- Triggers for deployment environments

Why should we use branch deployments?

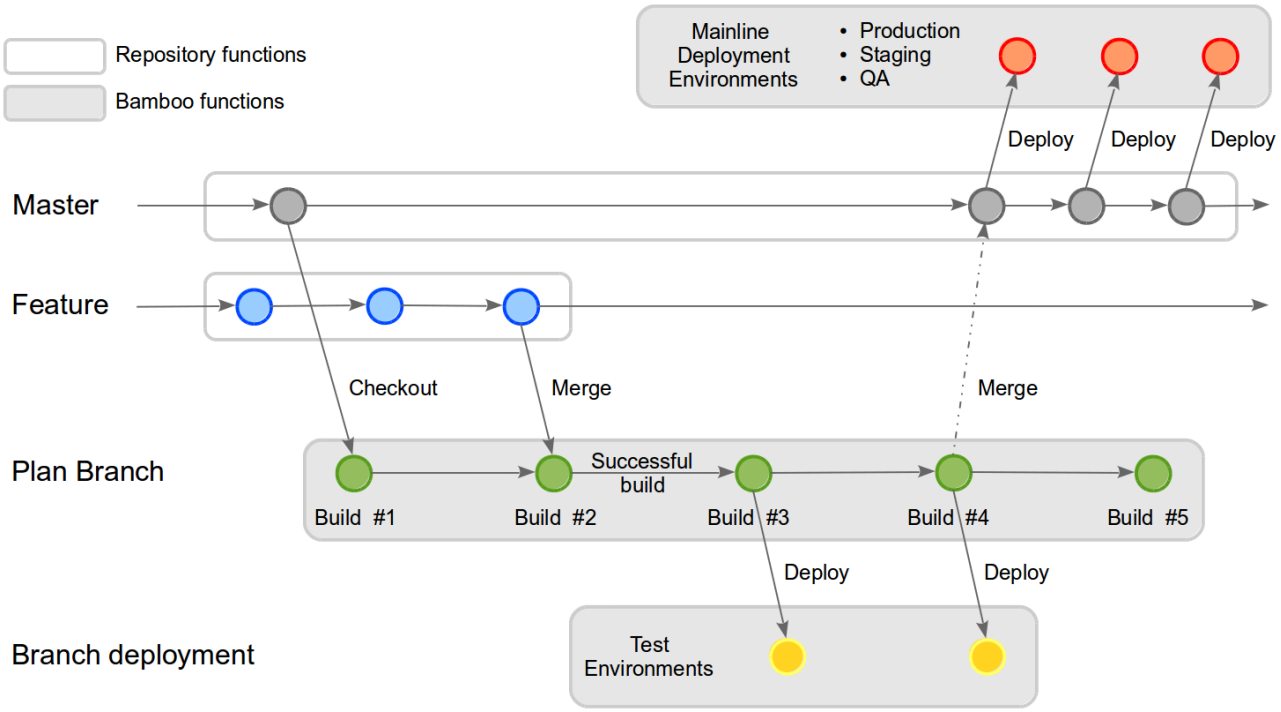
Bamboo deployments allow a plan branch to be deployed to a non-critical test environment before the feature code is merged back to master. This means that the feature code can be thoroughly tested and evaluated in a real server environment before the developer merges back the changes to master.

Developers should consider using branch deployments whenever they want to keep their in-progress development code separate from the master code, but want to test it within a deployable environment.

Learn more about deployment releases and how deployment releases work.

The following diagram shows a typical deployment branch example.

1. The developer creates a new branch off of the master and a plan branch is automatically created for the new branch in Bamboo
2. The developer commits code against the branch and the plan branch automatically builds the changes
3. Following a successful build, they then deploy the results of builds #3 and #4 into a test environment for thorough testing
4. When satisfied that all of the tests have been passed, the developer manually merges their feature branch back into master
5. Now that the changes are in master – sporting the new feature – a new release can be created and deployed to the mainline environments (e.g. QA, Staging, and Production)



Branch deployment use cases

Branch deployments should only ever be triggered into safe testing environments - they should never be triggered into production-like environments such as Staging, QA, or Production.

Learn more about Creating and configuring a deployment project.

There are two typical strategies for managing branch deployments:

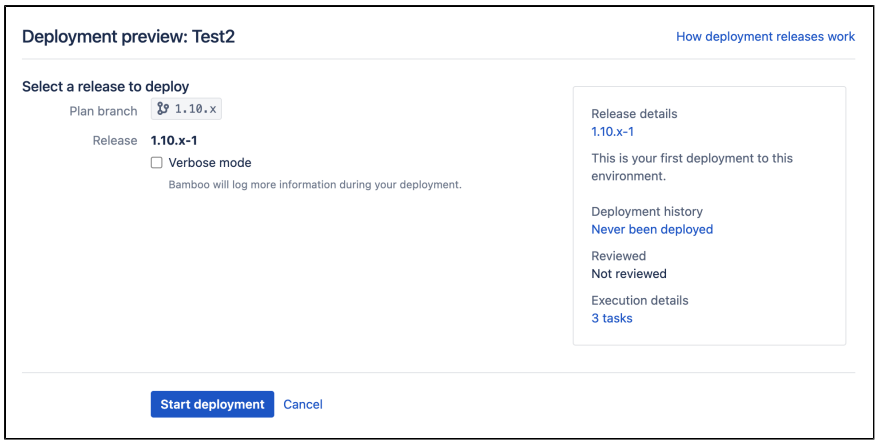
1. Manual branch deployment
2. Automated branch deployment

Let's examine each strategy in more detail.

Manual branch deployment

Let's assume a developer is using a plan branch to work on a new feature for a product. They reach a point in development where the new code needs testing in a server environment.

1. The developer successfully builds and tests the code using Bamboo. Let's call this Build #1.
2. When a successful build occurs, it's ready to deploy by creating a new release for Build #1 and deploying it to the testing environment.



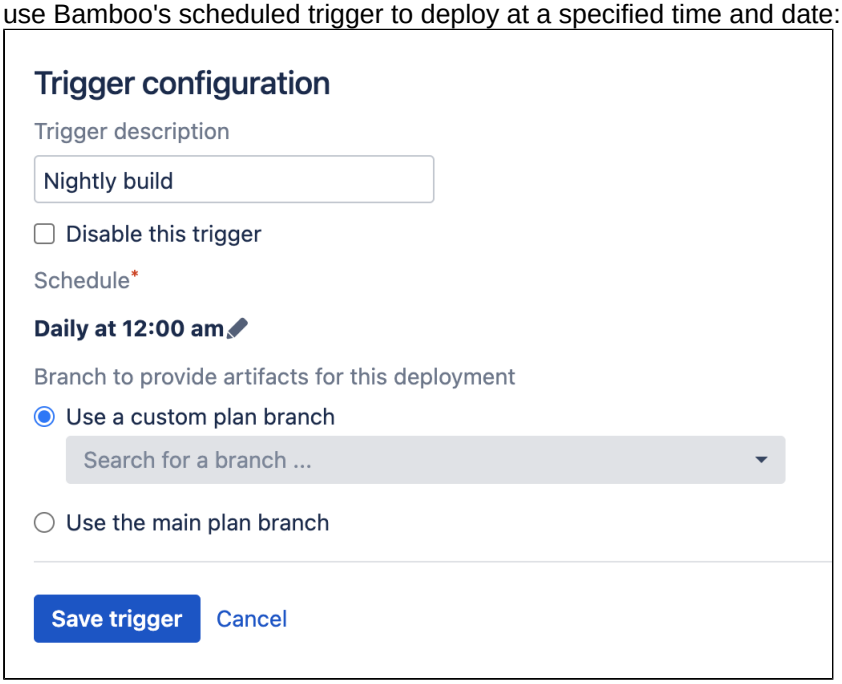
3. When deployed, the developer thoroughly tests their new code. When satisfied that all of the tests have been passed, the developer can merge the changes back into master.

Learn more about Manually starting a deployment.

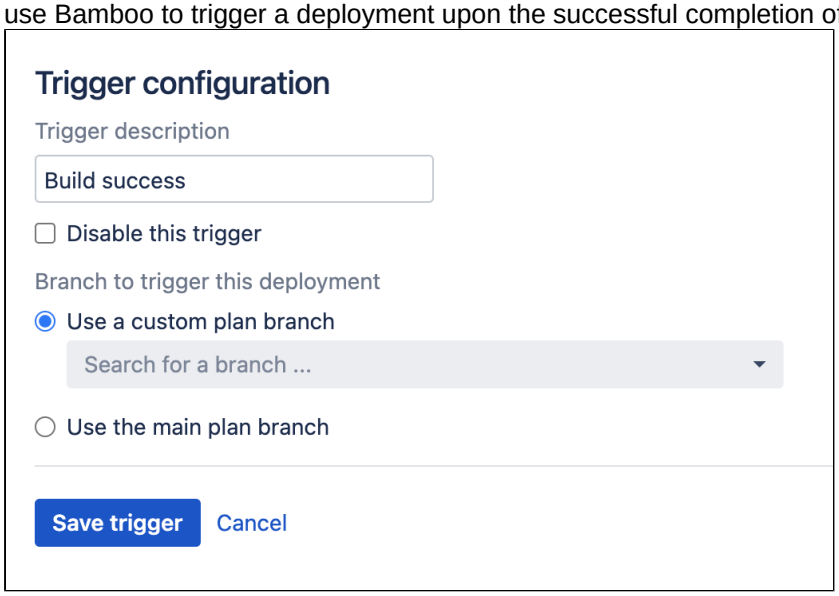
Automated branch deployment

Let's consider another developer who is also using a plan branch to work on a new feature for the product. They decide to automate the branch deployment so that it isn't triggered manually.

1. The developer successfully builds the code, including the new code they have been working on. Let's call this Build #2.
2. The developer doesn't want to deploy manually, so they use Bamboo's automated triggering to set up a strategy to deploy the plan branch into a deployment test environment. Two options are available:
    a. use Bamboo's scheduled trigger to deploy at a specified time and date:



b. use Bamboo to trigger a deployment upon the successful completion of a build plan:



3. The developer sets up the triggering strategy to best match working practices. Once the trigger is reached, the plan branch build is deployed to the test environment.
4. When the plan branch build is deployed, the developer thoroughly tests their new code. When satisfied that all of the tests have been passed, the developer can merge the changes back into master.

Remember: Plan branch code should only be merged back to master AFTER testing of the branch feature code is complete AND successful.

Learn more about Triggers for deployment environments.

Getting feedback

Getting immediate feedback about build results is the essence of continuous integration. Furthermore, getting reports on activity of your development team can give you deep insights into your process efficiencies and schedule risks.

Notifications

Bamboo can send notifications to your team about the success or failure of their builds in a number of ways:

- The Wallboard
- Email
- RSS feeds

- Instant messaging

Reports

Bamboo provides various reports about the build activity of your development team:

- Summary statistics for all users
- Build results for an author
- Comparison charts for authors
- Comparison charts for plans
- Clover code-coverage for a job
- Clover code-coverage for a build

Notifications

Bamboo can send notifications about build results so that you can find out immediately about the success or failure of your builds.

You can get notifications in different ways:

Bamboo Wallboard
Show build results on a dedicated monitor.

See Displaying the wallboard.

Email (e.g. GMail)
Get build results in your inbox.

See Configuring notifications.

Instant messaging (e.g. Hipchat, Google Talk)
Send notifications to your dev chat room.

See Configuring notifications.

RSS feeds
Get aggregated key information about your builds.

See Subscribing to RSS feeds.

See also Changing your notification preferences.

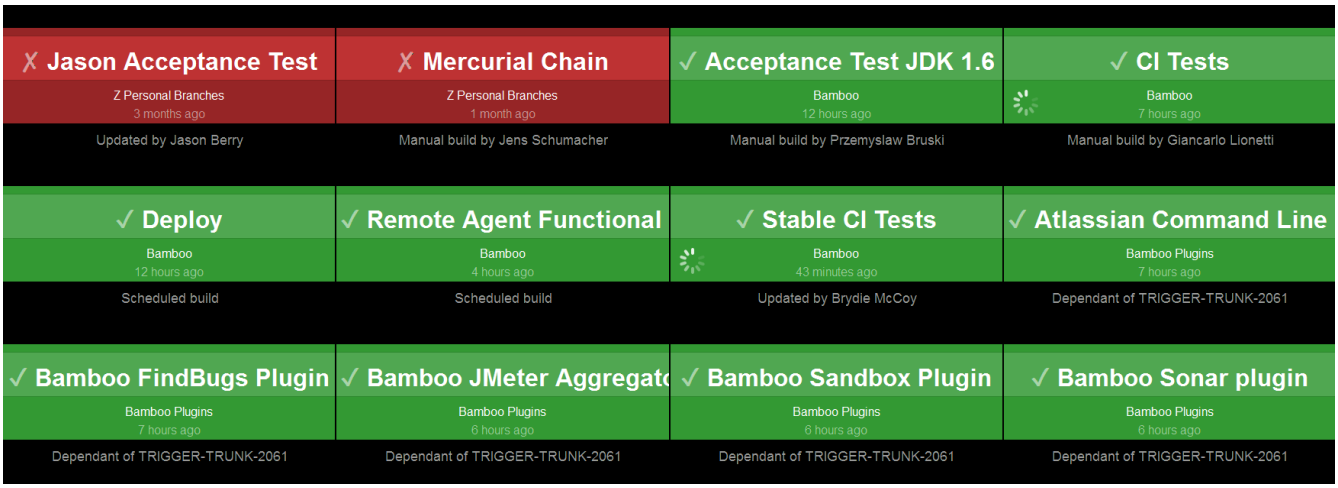
Displaying the wallboard

A development team can benefit from setting up a dedicated monitor to display Bamboo's latest build results using the Bamboo wallboard.

The Bamboo wallboard can display the latest results for:

- all plans that you have permission to see.
- just your favorite plans.
- plans filtered by plan label.

The branches wallboard displays the status of all the branches and the plan that the branches belong to.



On this page:

Related pages:

- Getting feedback
- Using the Bamboo Dashboard

How do I do that?

Log in to Bamboo, if necessary.

Go to **My Bamboo**.

Task	Action	Notes
All plans	**Wallboard** > **All plans**	Alternatively, use the following URL in your browser, replacing '`bambooserver`' with the real name of your Bamboo server: `http://bambooserver:8080/bamboo/telemetry.action`
Favorite plans	**Wallboard** > **Favorite plans**	Only users who have logged in to Bamboo can specify and access favorites. Alternatively, use the following URL in your browser, replacing '`bambooserver`' with the real name of your Bamboo server: `http://bambooserver:8080/bamboo/telemetry.action?filter=favourites`

Filtered plans	**Wallboard** > **Filtered plans**	You need to have set up a plan filter first. See Using the Bamboo Dashboard.

Notes

- You will only be able to display those plans that you have permission to see.
- Once you are viewing the wallboard in your browser window, set your browser to full screen mode to make the wallboard fill your entire screen. (Use **F11** for common browsers on Windows and UNIX/Linux-based systems and **Shift+Cmd+F** for Firefox on Mac OS X.)
- If you are going to display the wallboard permanently, you may want to ask your Bamboo administrator to create a user who has only a limited set of permissions.
- If your wallboard is displayed on a touchscreen (such as an iPad) or its content can be accessed with a human interface device, such as a mouse, then touching or selecting a build result on the wallboard shows more information about that build.

Configuring notifications for a plan and its jobs

Notifications in Bamboo are triggered by a range of events involving a plan and its jobs, including build completion, build outcomes, and comments being posted against build results. You can configure whether notifications are sent for a particular event for each plan and job, and to whom they are sent.

Bamboo users can choose whether to receive their notifications via email, IM, both, or neither. In general, recipie nts do not require Bamboo user accounts .

Adding notifications for a plan or job

Before you begin:

- You must have the Edit permission for a plan to add or remove notifications for it.
- You need to configure Bamboo's SMTP email and/or instant messaging capabilities before Bamboo can send notifications. If you have not configured either or both of these, a note will display on the page prompting you to set up the appropriate server(s):
    - To configure an email server for Bamboo, select **Add an Email Server** in the note and enter the email server details in the window that displays. See Configuring Bamboo to send SMTP Email for more information.
    - To configure an instant messaging server for Bamboo, select **Add an Instant Messaging Server** i n the note and enter the instant messaging server details in the window that displays. See Configur ing Bamboo to use Instant Messaging for more information.

To add a notification for a plan or its jobs:

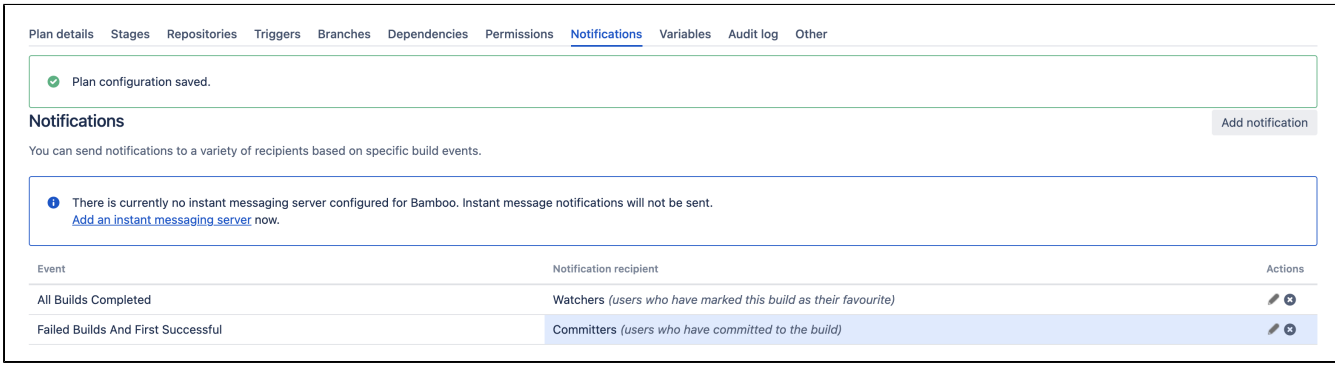
1. Navigate to the configuration for the desired plan, as described on Configuring plans.
2. Go to the **Notifications** tab, and select **Add notification**.
3. Set up a new notification in the Add build notification section as follows:
    a. **Event**
    Select the event type you want to be notified about. Refer to the list of events (below) for details.
    b. **Recipient type**
        i. **User** — Enter the username of the appropriate Bamboo user, or select the icon to select from a list of users.
        ii. **Group** — Enter the name of the appropriate Bamboo group(s) .
        iii. **Email address** — You can use email to send notifications to a person who is not a Bamboo user. Enter the appropriate email address. Note that:
            - If you specify the email address of an existing Bamboo user, the user will receive notifications even if they have elected not to receive notifications in their user preferences.
        iv. **IM address** — This is useful if you need to send Instant Messenger (IM) notifications to a person who is not a Bamboo user. Type the appropriate IM address. Note that:
            - If you specify a broadcast address (eg. project-x@broadcast.chat.mycompany.com), Bamboo will not know the context of related IM responses.
            - If you specify the IM address of an existing Bamboo user, the user will receive notifications even if they have elected not to receive notifications in their user preferences.
        v. **Responsible users** — The Bamboo users who have been assigned as being responsible for a broken build. See Assigning responsibility for build failures .
        vi. **Committers** — The Bamboo users who have committed code to a particular build since build was last checked out by Bamboo.

Committers are notified based on the notification preferences of a Bamboo user associated with the commit's author and email, also known as source repository alias. If there's no user linked to the source repository alias, Bamboo will not be able to send notifications.

vii. **Watchers** — The Bamboo users who have marked this plan as one of their favorites .

viii. **Webhooks** — Select the webhook you want to use for notification. To create new webhooks, see Using webhooks.

4. Select **Add**, then configure further notifications if required.



Notification events

Plan Events

All Builds Completed
Bamboo will send a notification whenever the plan build finishes, regardless of the plan build's result. This notification is recommended for any plans whose latest build activity is critical for people to be informed about.
✅ This is a good plan-based notification to use if you are new to Bamboo. You can change it to a less obtrusive notification option as you become more confident with continuous integration and Bamboo's build processes.

Change of Build Status
Bamboo will send a notification only when there has been a change in the status of the plan's build activity over consecutive plan builds — for example, only whenever a plan's latest build changes from successful to failed or vice versa (i.e. *fixed*) or if the plan's latest build results contain new test failures.
✅ This notification option is less obtrusive than the other plan notifications mentioned above.

Failed Builds And First Successful
Bamboo will send a notification whenever:
- a build of this plan fails.
- the plan is *fixed* (that is, the plan's latest build is successful *and* the previous plan build failed).
✅ This notification is generally suitable for the majority of plans.

After X Failed Builds
This notification allows you to specify the **Number Of Failures** (i.e. number of failed builds of this plan), after which Bamboo will send a notification.
✅ This notification option minimizes the number of messages sent by Bamboo if the plan's builds fail on a frequent basis. You can also use this event to escalate plan build problems, for example, to notify a manager when a plan build fails five times.

Comment Added
Bamboo will send a notification whenever a comment is added to a build result . The email notification will contain all comments against the plan build, whereas IM notifications will only contain the comment that triggered this notification event.
✅ This notification can help improve collaboration between team members. Be aware that you will not receive notifications for any comments that you post yourself.

Change of Responsibilities
Bamboo will send a notification whenever someone is added to, or removed from, the list of those responsible for a broken build.
✅ This notification can help improve collaboration between team members.

Job Events

All Jobs Completed
Bamboo will send a notification whenever a job build of the plan finishes, regardless of the job build's result. This notification is recommended if the latest build activity of all jobs in this plan are critical for people to be informed about.
✅ This is a good job-based notification to use if you are new to Bamboo. You can change it to a less obtrusive notification option as you become more confident with continuous integration and Bamboo's build processes.

Change of Job Status
Bamboo will send a notification only when there has been a change in build activity status of the jobs within this plan over consecutive plan builds — for example, only whenever the latest build of any job in this plan changes from successful to failed or vice versa (i.e. *fixed*).
✅ This notification option is less obtrusive than the other job notifications mentioned above.

Failed Jobs And First Successful
Bamboo will send a notification whenever:
- a build of this job fails.
- the job is *fixed* (that is, the job's latest build is successful *and* the previous job build failed).

First Failed Job For Plan
If multiple jobs fail in a plan, Bamboo will only send a notification for the first failing job detected by the Bamboo system.
✅ This is a less obtrusive notification option that informs about a failing job (and hence, plan) in the shortest possible time.

Job Error
Bamboo will send a notification whenever an error occurs in one of the plan's job build processes (i.e. the activities that Bamboo performs to run a job build). This event is not related to failures of the actual build itself (see the **Failed Jobs And First Successful** and **Failed Builds And First Successful** events above). For example, a notification will be sent if Bamboo encounters an error when connecting to the repository, or detecting changes.

Job Hung
Bamboo will send a notification whenever it determines that one of the plan's job builds has hung, according to the hung job build criteria (read more about configuring your hung job build settings).
✅ Use this notification to ensure that the relevant people are informed when a job build becomes unresponsive.

Job Queue Timeout
Bamboo will send a notification whenever one of the plan's job builds has been waiting in the queue for longer than the build queue timeout criteria (read more about configuring your job's Build Queue Timeout settings).
✅ Use this notification to ensure that the relevant people are informed when a job build is stuck in the build queue for too long.

Job Queued Without Capable Agents
Bamboo will send a notification whenever one of the plan's job builds is queued and there are no agents capable of building it.
✅ Use this notification to ensure that people are notified when changes to agents adversely affect your job's builds.

Removing notifications from a plan or job

You must have the Edit permission for a plan, to add or remove notifications for it.

1. Navigate to the configuration for the desired Plan, as described on Configuring plans.
2. Go to the **Notifications** tab.
3. Select **Remove** for each of the notifications that you wish to remove.

Configuring Bamboo to send SMTP Email

Bamboo can send email notifications about its build results. There are two steps to setting this up:

1. Configure Bamboo to send SMTP email (see below).
2. Configure a plan to send SMTP email notifications about build results (see Configuring notifications for a plan and its jobs).

On this page:

Related pages:

- Configuring notifications for a plan and its jobs

Configuring Bamboo to send SMTP email

To configure Bamboo to send SMTP email:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Communication**, select **Mail server**.
3. Edit the mail server settings as necessary:

   Name
   A display name for the email address e.g. SMTP Server.

   From address
   The email address from which Bamboo notifications will be sent.

   Subject prefix
   The text (if any) which will be added to the start of the email subject line. For example [Bamboo] will result in emails with subjects like:

   - [Bamboo] TEST build 1,001 has FAILED (77 tests failed, no failures were new) : Change made by jsmith
   - [Bamboo] TEST build 1,002 was SUCCESSFUL (with 77 tests) : Change made by jsmith

   Email settings
   Select either **SMTP** or **JNDI**. See the Notes about JNDI below.

   SMTP server
   The address of the email server that Bamboo will use to send notifications e.g. mail.myserver.com.

   Username
   The login name of the account that Bamboo will use to login to the SMTP server.

   JNDI Location
   Depends on your application server, and on the location of the mail resource within the JNDI tree you specify. E.g. java:comp/env/mail/BambooMailServer.
4. Enter a test email address in the **Test recipient address** box.
5. Select **Test**, and verify that a test email is received.
6. Select **Save**.

Configuring email notifications for Gmail

Gmail.com uses TLS (SSL). A JNDI connector needs to be configured. Unfortunately Bamboo does not yet support JNDI with TLS.

To enable Gmail as your mail server:

1. Install Bamboo.
2. Make sure that the following files are copied to and exists only in <Bamboo-Install> `/lib`:
   - `javax.mail-X.X.X.jar`
   - `javax.mail-api-X.X.X.jar` (only if you're using Bamboo 6.0 or later)

   ⚠
   - If the `javax.mail-X.X.X.jar` and `javax.mail-api-X.X.X.jar` files don't exist in the <Bamboo-Install> `/lib` directory, you must move the `javax` files installed at `<Bamboo-Install>/atlassian-bamboo/WEB-INF/lib` to <Bamboo-Install> `/lib`.
   - If the `javax.mail-X.X.X.jar` and `javax.mail-api-X.X.X.jar` files already exists in the <Bamboo-Install> `/lib` directory, simply delete the `javax` files shipped with Bamboo in `<Bamboo-Install>/atlassian-bamboo/WEB-INF/lib`.

     In Bamboo 5.9 the `mail-X.X.jar` and `activation-X.X.jar` files were included in the download archive. Starting with Bamboo 5.10:

     - the `mail-X.X.jar` file has been renamed to `javax.mail-X.X.X.jar` and must be moved to <Bamboo-Install> `/lib`
     - the `activation.jar` file is not required
3. Add the following configuration to your BambooInstall/conf/server.xml file:

```
<Context path="/bamboo" docBase="${catalina.base}/atlassian-bamboo" reloadable="true">
          <Resource name="mail/GmailSmtpServer"
            auth="Container"
            type="javax.mail.Session"
            mail.smtp.host="smtp.gmail.com"
            mail.smtp.port="465"
            mail.smtp.auth="true"
            mail.smtp.user="<your-name>@gmail.com"
            password="<your-pw>"
            mail.smtp.starttls.enable="true"
            mail.smtp.socketFactory.class="javax.net.ssl.SSLSocketFactory"
          />
</Context>
```

Configure Bamboo to use a **JNDI Location** of `java:comp/env/mail/GmailSmtpServer`. Note that the JNDI Location is case sensitive and must match the resource name specified in `server.xml`.

Notes

You can use a mail session as an alternative to specifying mail details directly in Bamboo. You configure the mail session in your application server (e.g. in the server.xml file — see Locating important directories and files), and then use JNDI to look up the preconfigured mail session. JNDI has the following advantages:

- **Centralized management** - mail details are configured in the same place as database details, and may be configured through your application server administration tools.
- **Better security** - mail details are not available to Bamboo administrators through the Bamboo interface, and aren't stored in Bamboo backup files.
- **More SMTP options** - e.g. SSL. If you want to use SMTP over SSL you will need to use JNDI.

Configuring Bamboo to use Instant Messaging

Bamboo can send Instant Messaging (IM) notifications about its build results. There are two steps to setting this up:

1. Configure Bamboo to use Instant Messaging (see below).
2. Configure a plan to send IM notifications about its build results (see Configuring notifications for a plan and its jobs).

Please note, Bamboo supports XMPP protocol for messaging. This means Bamboo can be used with Gtalk or your enterprise XMPP server.

Related pages:

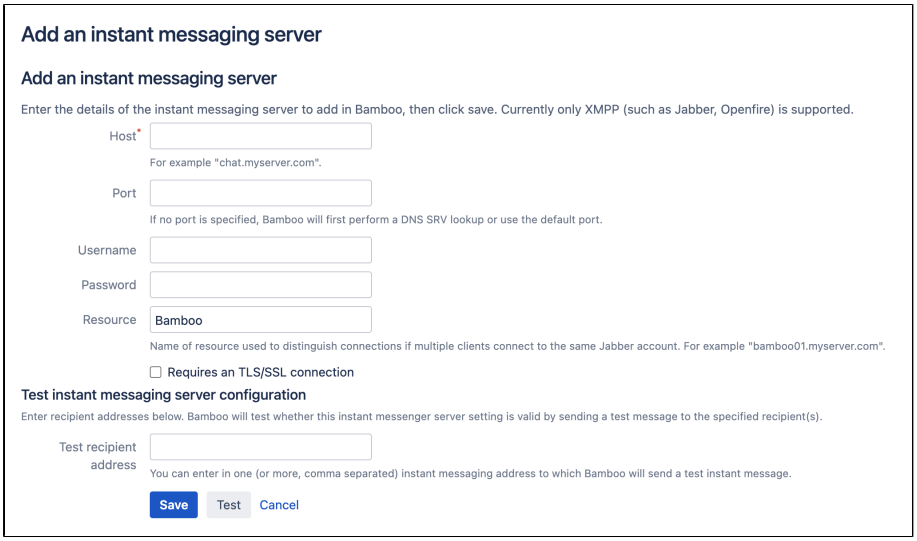
- Configuring notifications for a plan and its jobs
- Configuring Bamboo to use Google Talk for Instant Messaging

To configure Bamboo to use Instant Messaging:

1. From the top navigation bar select  > **Communication** > **IM server**.
2. Enter the following details:

Setting	Notes
Host	The address of your IM server (for example chat.atlassian.com).
Port	The TCP port that your organization uses for IM traffic (or leave this field blank to have Bamboo either perform a DNS lookup or use the default port).
Username	The login name of the IM account from which Bamboo notifications will be sent.
Change password	Select this if you have specified a username different from the currently logged-in user.
Resource	An identifying name for the connection if multiple clients use the same jabber account.
Requires a TLS /SSL connection	Select this if your IM server uses SSL.
Force legacy SSL	
Test recipient address	You can test this configuration by entering an address and selecting **Test**.

3. Select **Save**.



Configuring Bamboo to use Google Talk for Instant Messaging

If your Bamboo server has access to the internet, it can use Google Talk to send IM notifications about build results.

Related pages:

- Configuring notifications for a plan and its jobs
- Working with Instant Messenger (IM) Notifications

Before you begin:

- Google Talk does not allow IM messages to be received unless the receiver has approved the sender. Please ensure that the Gmail user specified below is approved by each Google Talk recipient. That is, ensure that the Host and Username have previously sent messages to each other via Google Talk.
- The Google Talk service is hosted at talk.google.com. The default port is 5222. (Note: be aware that your firewall might be blocking traffic to this port.)
- TLS is required.
- The only supported authentication mechanism is SASL PLAIN. For additional information, please see: http://code.google.com/apis/talk/open_communications.html

To configure Bamboo to use Google Talk for Instant Messaging:

1. From the top navigation bar select  > **Communication** > **IM server**.
2. Select **Edit**. Modify the settings as required.

   Hos
   Type talk.google.com. (If your IM Server uses an @googlemail.com account, type googlemail.com.)

   Port
   Leave blank. Bamboo will perform a DNS lookup to figure out which port to use.

   Username
   Type the login name of the Google account from which IM notifications will be sent. Starting with Bamboo 3.4, you need to include the domain name, e.g. *atlassianbamboo@gmail.com,* NOT *atlassianbamboo*.

   Change password
   Select this if you have specified a username different from the currently logged-in user.

   Resource
   An identifying name for the connection if multiple clients use the same jabber account.

   Requires a TLS/SSL connection
   Select this.

   Force legacy SSL

   Test recipient address
   You can test this configuration by entering an address and selecting **Test.**

3. Select **Save**.

Modifying notification templates

If you want to customize the layout and content of your Bamboo notifications, you can customize the templates for each of the notification types (i.e. HTML email, text email, instant message) and events (e.g. Build Commented). The notification templates are written in Freemarker.

Some content in notifications can also be configured via system properties, such as the number of log lines to include in email notifications that display log information.

⚠ Changes to notification templates only take effect after a Bamboo restart.

On this page:

Related pages:

- Configuring Bamboo to use Instant Messaging

Modifying a notification template

To modify a notification template:

1. Locate the default notification templates in **`WEB-INF/classes/notification-templates/`**
2. Copy the notification template that you wish to modify into the **`templates/notification-templates`** folder of your Bamboo home directory, e.g. `HOME/templates/notification-templates`
   ℹ *The filename will have formatted as:* ***<event name><notification type>.ftl****, e.g.*`AfterXFailedHTMLEmail.ftl`
3. Modify the copied template, as desired. Please see the section on Working with Freemarker below for tips on updating templates.
4. Save your changes to the template. You need to restart your Bamboo server for the template changes to take effect.

Working with Freemarker

The Bamboo notification templates are written in Freemarker. The Freemarker engine allows for dynamic content generation based on the Freemarker markup tags that are used in templates. This document does not describe the Freemarker language in detail. Please see the Freemarker Online Manual for full information on using this markup language.

Generating content via Freemarker involves merging a template (*.ftl file) with a context map. You can access any data in the context map from within the template using the Freemarker markup. For the notifications we have provided a specific subset of Bamboo objects that you can access. For example,

```
[#if buildSummary.successful]
${buildSummary.buildResultKey} was successful.
```

If you had a successful Bamboo build with build result, BAM-1234-1, the above markup would return the following text in your notification:
`BAM-1234-1 was successful.`

You can find more information on working with Freemarker, including Bamboo objects available from Freemarker templates, tips on writing Freemarker templates and examples in the Freemarker and notification templates document.

Configuring notifications content via system properties

The following system properties can be configured to control some of the content that is included in notifications (e.g. the number of log lines to include in email notifications that display log information). For instructions on how to configure a system property, please refer to the Starting Bamboo page.

Before you begin:
The system properties below do not add content to notifications. You still need to ensure that your notification templates contain the relevant entities to display the content. For example, changing the `bamboo.notifications.logLinesToInclude` property will not add log information to your notifications. It only modifies the number of log lines displayed in notification templates that already include logs.

System Property	Description	Default Value
`bamboo.notifications.logLinesToInclude`	Specifies the number of log lines to include in email notifications that display log information.	100

Notes

- **Bamboo does not validate notification templates**. If you have incorrectly formatted the markup text in the template, Bamboo will still use the template to send out notifications. If this happens, your users may receive notifications with unreadable or missing information, as well as error messages. Errors will also be posted to your logs.

Freemarker and notification templates

Notification templates in Bamboo can be modified to customize the format and content of your notifications. The templates are written in Freemarker. This page is intended to complement the Modifying notification templates page and contains information on the Bamboo objects available from Freemarker templates, tips on writing Freemarker templates and examples.

⚠ Changes to notification templates only take effect after a Bamboo restart.

On this page:

Related pages:

- Configuring Bamboo to use Instant Messaging
- Modifying notification templates

Accessing Bamboo data

Each individual notification has a different subset of data that can be accessed from the Freemarker templates. You can find information on the objects available in our javadocs below.

- Build Completed Notification ("All Completed Builds" and "Failed and First Success")
- After X Failed Builds Notification
- Build Commented Notification
- Build Hung Notification
- Build Error Notification

Special considerations when working with Freemarker

Never assume data exists

Unfortunately Freemarker is not very forgiving if data does not exist or is null. Hence, you will need to check whether information exists before adding it to a page. The sample code below shows how you can validate for non-existent data.

```
[#if issue.jiraIssueDetails.summary?has_content][/#if]
[#if issue.jiraIssueDetails.summary??][/#if]
${issue.jiraIssueDetails.summary?if_exists}
${issue.jiraIssueDetails.summary!}
```

Check the encoding of your information

Freemarker has built-in utilities for escaping special characters. These could be characters that you deliberately do not want to be interpreted as HTML, or data that could potentially contain malicious content. The sample code below shows how you can escape characters in Freemarker.

```
${commit.comment?html} // for data to be encoded to be displayed as html
${commit.author?url} // for data to be encoded for a url
```

You can find more information on these utilities in the official Freemarker documentation.

Use white space carefully

When editing text email content and instant message content, you need to be very careful with spacing and line breaks. Any spaces and line breaks that you have entered in the Freemarker template will also exist in the evaluated content. Freemarker provides you with some utilities to remove white space, so that you can still retain some formatting in the templates.

More information can be found the official Freemarker documentation.

Freemarker examples

Below are some raw examples of additional information that you can add to your emails.

Please note, these examples are intended to demonstrate the use of Freemarker and how to access Bamboo objects. You will need to modify these examples to include your desired formatting and make it work with your data.

List files in a commit

```
[#if buildSummary.commits.size() > 0]
    [#list buildSummary.commits as commit]
         [#if commit_index gte 3][#break][/#if] //only shows 3 commits
         Author:   <a href="[@ui.displayAuthorOrProfileLink commit.author/]">
                        ${commit.author.fullName?html}
                   </a>
                   <br/>
         Comment:  ${commit.comment?html}
                   <br/>
         [#if commit.guessChangeSetId()?has_content]
         Revision: ${commit.guessChangeSetId()?html}
         <br/>
         [/#if]

         [#if commit.files?has_content]
         Files:
              [#list commit.files as file]
                   ${file.cleanName} [#if file.revision?has_content](${file.revision})
                   <br/>
              [/#list]
         [/#if]
     [/#list]
[#else]
     This build does not have any commits.
[/#if]
```

Provide test error details

```
[#list buildResults.testResults.newFailedTests.values() as testResultClass]
   [#list testResultClass.testResults as testResult]
         <a href="${baseUrl}${fn.getViewTestClassResultUrl(build.key, buildResults.buildNumber,
testResultClass.name)}">
              ${testResultClass.shortName?html}
         </a> :
         <a href="${baseUrl}${fn.getViewTestCaseHistoryUrl(buildSummary.buildResultKey, testResult.
className,
                   testResult.actualMethodName)}">
              ${testResult.methodName?html}
         </a>
         <br/>
         [#if testResult.errors?has_content]
              [#list testResult.errors as error]
                        <pre>${error.errorMessage}</pre>  // a <pre/> tag is required to reserve formatting
of error
               [/#list]
         [/#if]
    [/#list]
[/#list]
```

Working with Instant Messenger (IM) notifications

Bamboo can send you notifications about build results for a particular plan. Each plan's recipients are specified by a Bamboo administrator, but you can choose whether you would like to receive your Bamboo notifications via email and/or an instant messenger (IM) service.

As well as receiving IM notifications, you can interact with Bamboo using IM, as described on this page.

On this page:

Related pages:

- Changing your notification preferences
- Configuring Bamboo to use Instant Messaging
- Getting feedback

Labeling a build result using IM

To label a build result using IM:
In your Instant Messenger client, type your comment in the following format:

```
label [build key] <labels>
```

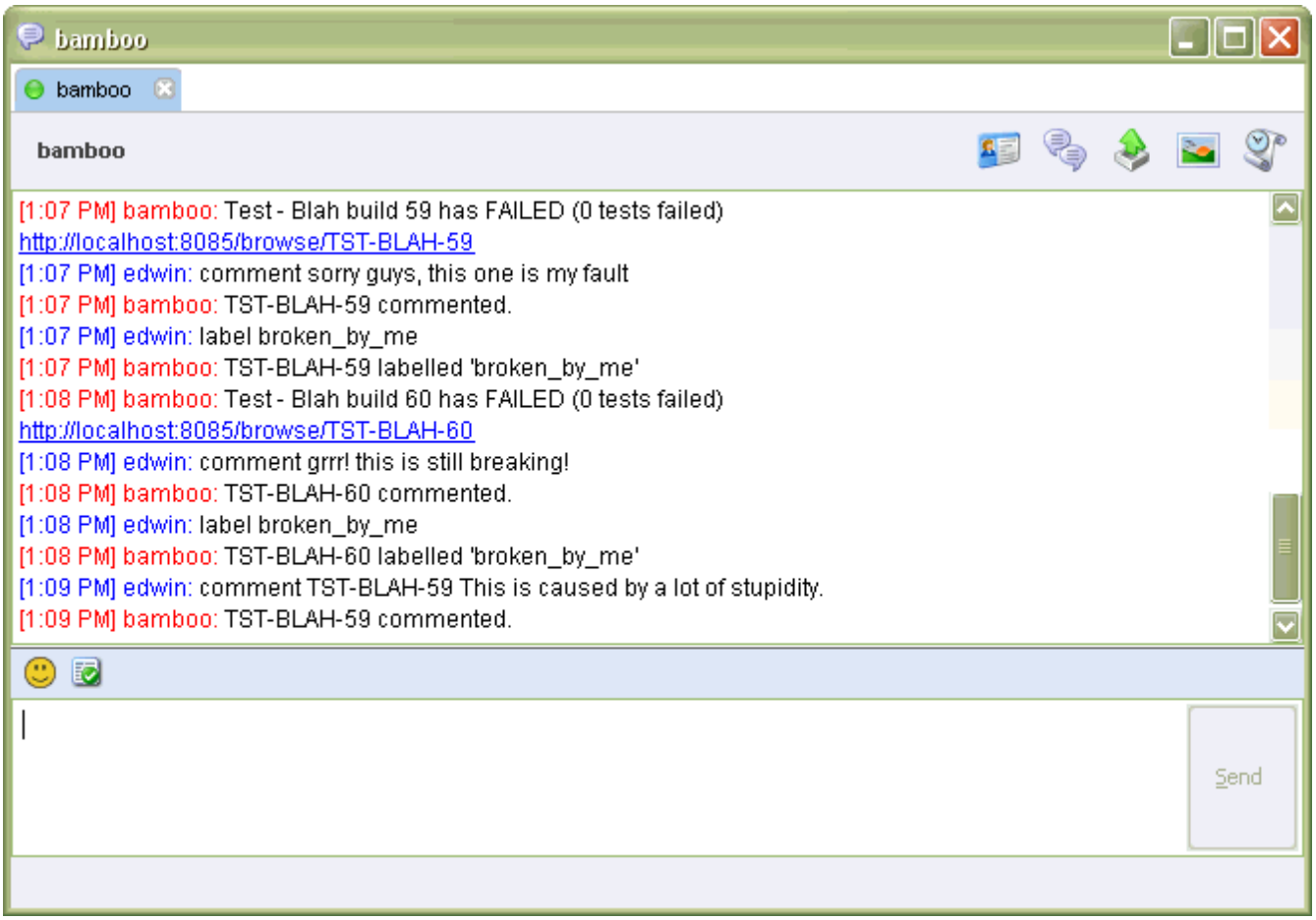
Entering a build key is optional. If none is specified, Bamboo will look up the last time it corresponded with you and the build that was in context. The context gets updated when you specify a build key in your command, and when Bamboo sends you a notification about a particular build.

Commenting about a build result using IM

To comment on a build result using IM:
In your Instant Messenger client, type your comment in the following format:

```
comment [build key] <comment message>
```

Entering a build key is optional. If none is specified, Bamboo will look up the last time it corresponded with you and the build that was in context. The context gets updated when you specify a build key in your command, and when Bamboo sends you a notification about a particular build.



Subscribing to RSS feeds

Bamboo aggregates key information about your builds into RSS feeds. You can subscribe to these feeds using any feed reader.

RSS feed scope	Options	Set up
All plans	• All build results • Failed build results	1. Go to the **Build** > **All build plans**. 2. Locate the RSS icon at the bottom of the screen. 3. Select either **all builds** or **all failed builds**, and copy the URL. 4. Paste the URL into your RSS reader.
A specific plan	• All build results • Failed build results	1. Go to the desired plan. 2. Locate the RSS icon at the bottom of the screen. 3. Select either **all builds** or **all failed builds**, and copy the URL. 4. Paste the URL into your RSS reader.
Build results with a particular label		1. Go to the **Build** > **All build plans**. 2. Go to any plan that has a label (this may involve trial and error). 3. Select any label, near the top of the screen. 4. Select **All labels**. 5. Select the label of interest. 6. Locate the RSS icon at the bottom of the screen. 7. Select **Feed for builds labeled** and copy the URL. 8. Paste the URL into your RSS reader.

System level notifications

System level notifications in Bamboo are triggered by a small range of system level events. This means that you don't need to configure these notifications for each plan you are running, as they are applied globally across the Bamboo platform.

Bamboo users can choose whether to receive their notifications via email, IM, both or neither. In general, recipie nts do not require Bamboo user accounts.

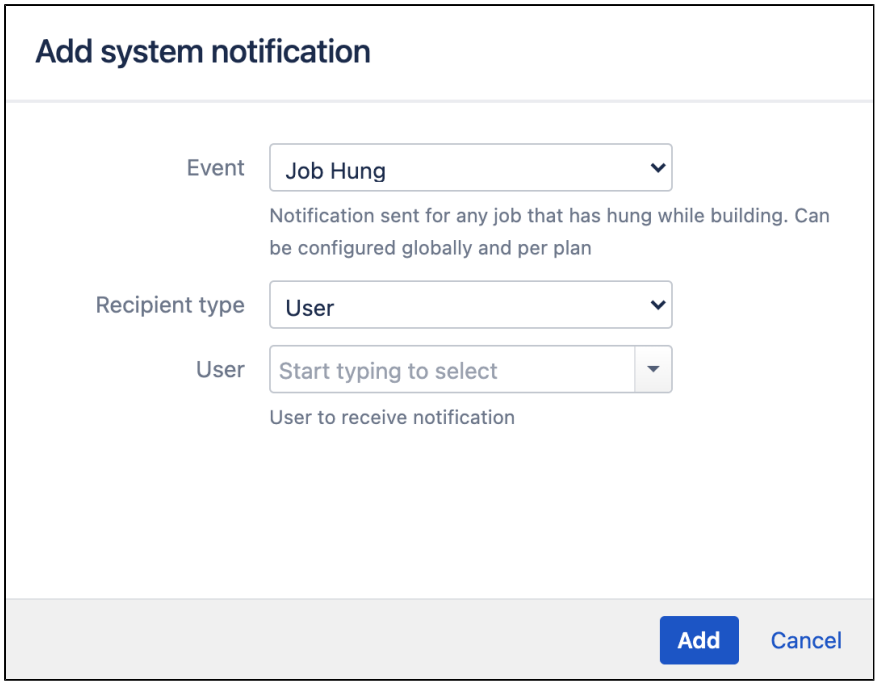
On this page:

Related pages:

- Notifications
- Configuring plans
- Changing your notification preferences

Add system level notifications

1. From the top navigation bar select  > **Communication** > **System notifications**.
2. Select **Add notification**.



a. Complete the Event and Recipient type fields using the following table:

Setting	Notes
Event	**Agent online –** Bamboo will send a notification whenever an agent goes online.
	Agent offline – Bamboo will send a notification whenever an agent goes offline. Notifications include data on 3 last executed jobs.

	Job Hung — Bamboo will send a notification whenever it determines that one of the plan's job builds has hung, according to the hung job build criteria (read more about configuring your hung job build settings). ✅ Use this notification to ensure that the relevant people are informed when a job build becomes unresponsive.
	Job queued without capable agents — Bamboo will send a notification whenever one of the plan's job builds is queued and there are no agents capable of building it. ✅ Use this notification to ensure that people are notified when changes to agents adversely affect your job's builds.
	Job queue timeout —Bamboo will send a notification whenever one of the plan's job builds has been waiting in the queue for longer than the build queue timeout criteria (read more about configuring your job's Build Queue Timeout settings). ✅ Use this notification to ensure that the relevant people are informed when a job build is stuck in the build queue for too long.
Recipient type	**User** — Enter the username of the appropriate Bamboo user, or select the icon to select from a list of users.
	Group — Enter the name of the appropriate Bamboo group(s).
	Email address — You can use email to send notifications to a person who is not a Bamboo user. Type the appropriate email address. Note that: • If you specify the email address of an existing Bamboo user, the user will receive notifications even if they have elected not to receive notifications in their user preferences.
	IM address— This is useful if you need to send Instant Messenger (IM) notifications to a person who is not a Bamboo user. Type the appropriate IM address. Note that: • If you specify a broadcast address (eg. project-x@broadcast.chat.mycompany.com), Bamboo will not know the context of related IM responses. • If you specify the IM address of an existing Bamboo user, the user will receive notifications even if they have elected not to receive notifications in their user preferences.
	Responsible users — The Bamboo users who have been assigned as being responsible for a broken build. See Assigning responsibility for build failures.
	Committers — The Bamboo users who have committed code to a particular build since build was last checked out by Bamboo.
	Watchers — The Bamboo users who have marked this plan as one of their favorites.

3. Select **Add** to confirm your configuration.

Change system notifications

1. From the top navigation bar select ⚙ > **Communication** > **System notifications**.
2. Select:

- The pencil symbol in the Actions column to edit the notification.
- The cross symbol in the Actions column to remove the notification.

Using webhooks

Bamboo webhooks allow you to send selected real-time information about Bamboo to third-party apps. For example, you can display Bamboo build status in your team's chatroom, or signal an alarm in case a plan fails.

By default, Bamboo comes with two webhook templates that you can use:

- Build webhook

  Method: POST

  Payload:

```
{
  "uuid": "${bamboo.webhook.uuid}",
  "timestamp": "${bamboo.webhook.timestamp}",
  "notification": "${bamboo.webhook.notification.description}",
  "webhook": {
    "webhookTemplatedId": "${bamboo.webhook.template.id}",
    "webhookTemplatedName": "${bamboo.webhook.template.name}"
  },
  "build": {
    "buildResultKey": "${bamboo.buildResultKey}",
    "status": "${bamboo.buildState}",
    "buildPlanName": "${bamboo.buildPlanName}",
    "startedAt": "${bamboo.date.started}",
    "finishedAt": "${bamboo.date.finished}",
    "triggerReason": "${bamboo.trigger.reason}",
    "triggerSentence": "${bamboo.trigger.name.for.sentence}"
  }
}
```

  Headers:

```
content-type:application/json
```

- Deployment webhook

  Method: POST

  Payload:

```
{
  "uuid" : "${bamboo.webhook.uuid}",
  "timestamp" : "${bamboo.webhook.timestamp}",
  "notification" : "${bamboo.webhook.notification.description}",
  "webhook" : {
    "webhookTemplatedId" : "${bamboo.webhook.template.id}",
    "webhookTemplatedName" : "${bamboo.webhook.template.name}"
  },
  "deployment" : {
    "deploymentResultId" : "${bamboo.deploy.result.id}",
    "status" : "${bamboo.buildState}",
    "deploymentProjectId" : "${bamboo.deploy.project.id}",
    "environmentId" : "${bamboo.deploy.environment.id}",
    "environmentName": "${bamboo.deploy.environment.name}",
    "deploymentVersionId" : "${bamboo.deploy.version.id}",
    "deploymentVersionName" : "${bamboo.deploy.version.name}",

    "startedAt" : "${bamboo.date.started}",
    "finishedAt" : "${bamboo.date.finished}",
    "agentId" : "${bamboo.agentId}",

    "triggerReason" : "${bamboo.trigger.reason}",
    "triggerSentence" : "${bamboo.trigger.name.for.sentence}"
  }
}
```

Headers:

```
content-type:application/json
```

You can modify the existing templates to suit your needs, or create your own templates, which can later be defined as notifications.

If you're using webhooks in your deployments, you can check their status in the deployment details screen. Once you select the webhook details, you will see a table with all webhooks responses related to the current deployment. If you can't see the table, it means that no webhooks have been sent.

The table is visible for all users who can see build results. The table contains basic information about the webhook response: its status code, status, URL, event name, template name, and send and receive time. The plan admin has permissions to view webhook details: response header and body. If any secret variables are used in the URL, body, or header, they will be hidden.

Before you begin

Only Bamboo administrators can add, edit, and delete webhook templates.

To create a webhook template:

1. Select > **Communication** > **Webhook templates**.
2. Select **Add template**.
3. Provide the following details for your new webhook template:
    - Name
    - HTTP Method
    - Payload
    - Headers

    > You can use Bamboo variables when editing webhook payload and headers.
4. Select **Add**.

Your webhook template is ready to use. You can define it as a notification type in your plan and jobs configuration, see Configuring notifications for a plan and its jobs.

To edit a webhook template:

1. Select > **Communication** > **Webhook templates**.
2. Next to the webhook template you want to change, select the **Edit** button.
3. Make your changes to the webhook template.
4. Select **Save**.

To delete a webhook template:

1. Select > **Communication** > **Webhook templates**.
2. Next to the webhook template you want to delete, select the **Delete** button.

Reporting

You are able to get reports about various kinds of activity in Bamboo:

Summary statistics for all users
A list of summary build statistics for all Bamboo users, showing such things as the number of builds triggered, broken and fixed.

See Viewing build statistics for all users.

Build results for an author
Build results summaries for someone who has committed code to projects in Bamboo, including the last 10 builds, the last 10 broken and the last 10 fixed.

See Viewing build results for an author.

Comparison charts for authors
Create comparison charts of build activity for selected authors.

See Generating reports on selected authors.

Comparison charts for plans
Create comparison charts of build results for selected plans.

See Generating reports across multiple plans.

Clover code-coverage reports

See Viewing the Clover code-coverage for a plan.

See Viewing the Clover code-coverage for a build.

Viewing build statistics for all users

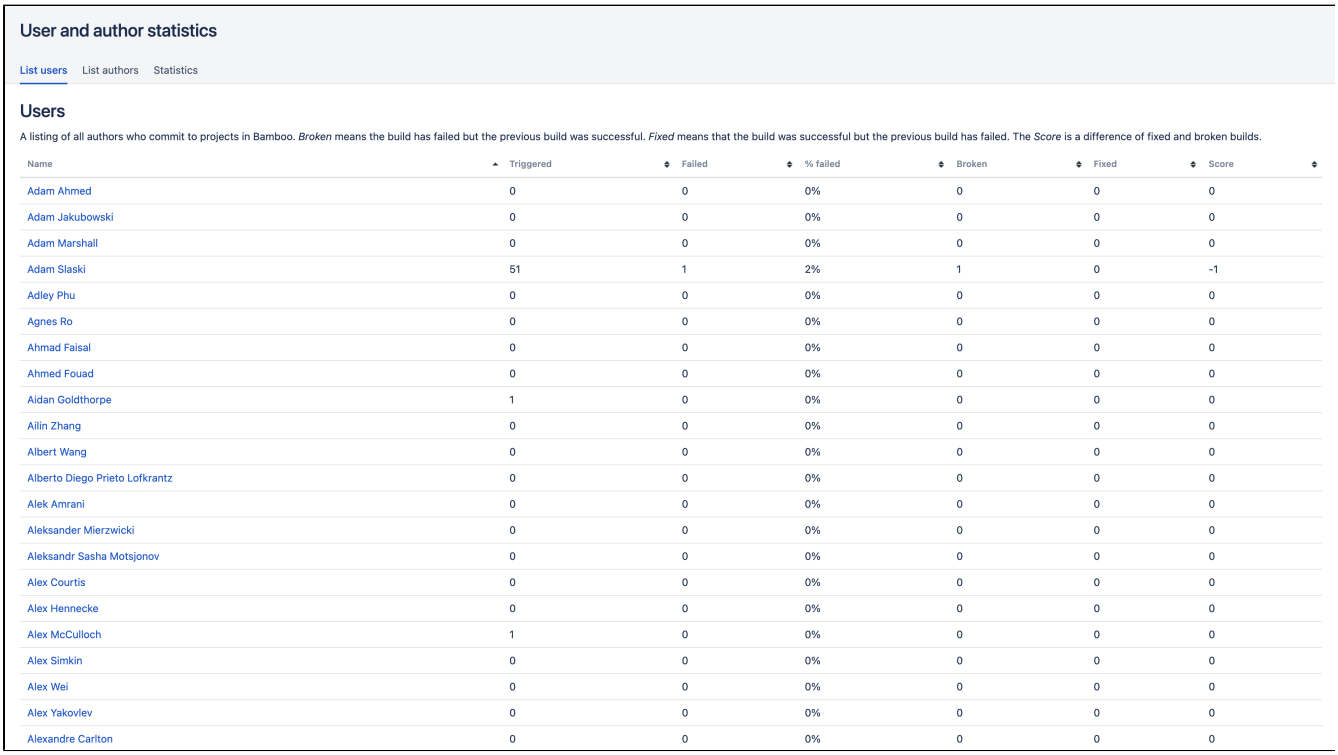
The build statistics summary gives you an overview of the activity of Bamboo users.

To view summary statistics for all users:

1. From the top navigation bar select **Reports** > **Authors**.
2. Select the **List users** tab.

Related pages:

- Reporting
- Viewing build results for an author



User and author statistics

List users | List authors | Statistics

Users

A listing of all authors who commit to projects in Bamboo. *Broken* means the build has failed but the previous build was successful. *Fixed* means that the build was successful but the previous build has failed. The *Score* is a difference of fixed and broken builds.

Name	Triggered	Failed	% failed	Broken	Fixed	Score
Adam Ahmed	0	0	0%	0	0	0
Adam Jakubowski	0	0	0%	0	0	0
Adam Marshall	0	0	0%	0	0	0
Adam Slaski	51	1	2%	1	0	-1
Adley Phu	0	0	0%	0	0	0
Agnes Ro	0	0	0%	0	0	0
Ahmad Faisal	0	0	0%	0	0	0
Ahmed Fouad	0	0	0%	0	0	0
Aidan Goldthorpe	1	0	0%	0	0	0
Ailin Zhang	0	0	0%	0	0	0
Albert Wang	0	0	0%	0	0	0
Alberto Diego Prieto Lofkrantz	0	0	0%	0	0	0
Alek Amrani	0	0	0%	0	0	0
Aleksander Mierzwicki	0	0	0%	0	0	0
Aleksandr Sasha Motsjonov	0	0	0%	0	0	0
Alex Courtis	0	0	0%	0	0	0
Alex Hennecke	0	0	0%	0	0	0
Alex McCulloch	1	0	0%	0	0	0
Alex Simkin	0	0	0%	0	0	0
Alex Wei	0	0	0%	0	0	0
Alex Yakovlev	0	0	0%	0	0	0
Alexandre Carlton	0	0	0%	0	0	0

Viewing build results for an author

An author's source-code repository login must have been associated with their Bamboo user profile before you can see their build results in Bamboo.

To view build results for a particular author:

1. From the top navigation bar select **Reports** > **Authors**.
2. Select the **List authors** tab.
3. Select an author's name to see statistics and recent build results for the author:

   User details
   The author's user details.

   Builds summary
   A statistical summary of all the author's builds.

   Last 10 builds
   The last 10 builds that were triggered by this author.

   Last 10 broken
   The last 10 builds that were triggered by this author, where the build failed and the previous build for the same plan was successful.

   Last 10 fixed
   The last 10 builds that were triggered by this author, where the build was successful and the previous build for the same plan failed.

Related pages:

- Viewing build statistics for all users

Generating reports on selected authors

An *author* is any person who checks in code to a repository that is associated with a Bamboo plan. An author need not be a Bamboo user.

Generating a report on selected authors

To generate a report on selected authors:

1. From the top navigation bar select **Reports** > **Authors**.
2. Select the **Statistics** tab.
3. Set the report parameters:

   Report
   Select from the available reports, shown below. Additional reports may have been added through custom plugins.

   Authors
   Select the authors on whom you want to report. Use the <Ctrl> key to select multiple authors.

   Group by
   Select the time scale for the horizontal axis.

4. Select **Submit**.

On this page:

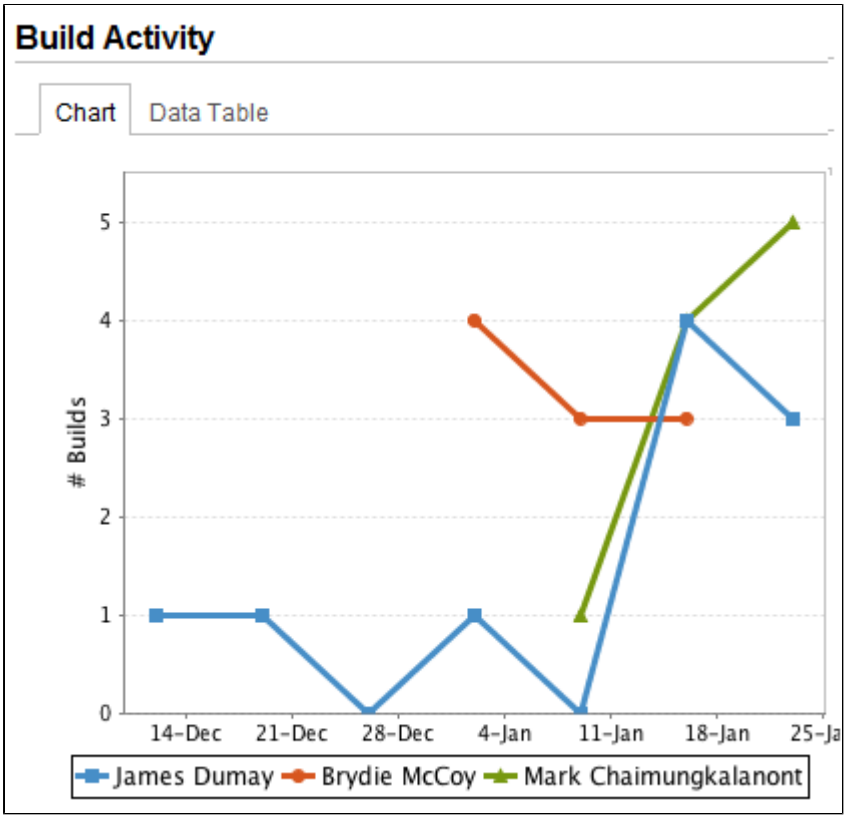
Related pages:

- Viewing build results for an author
- Getting feedback
- Notifications

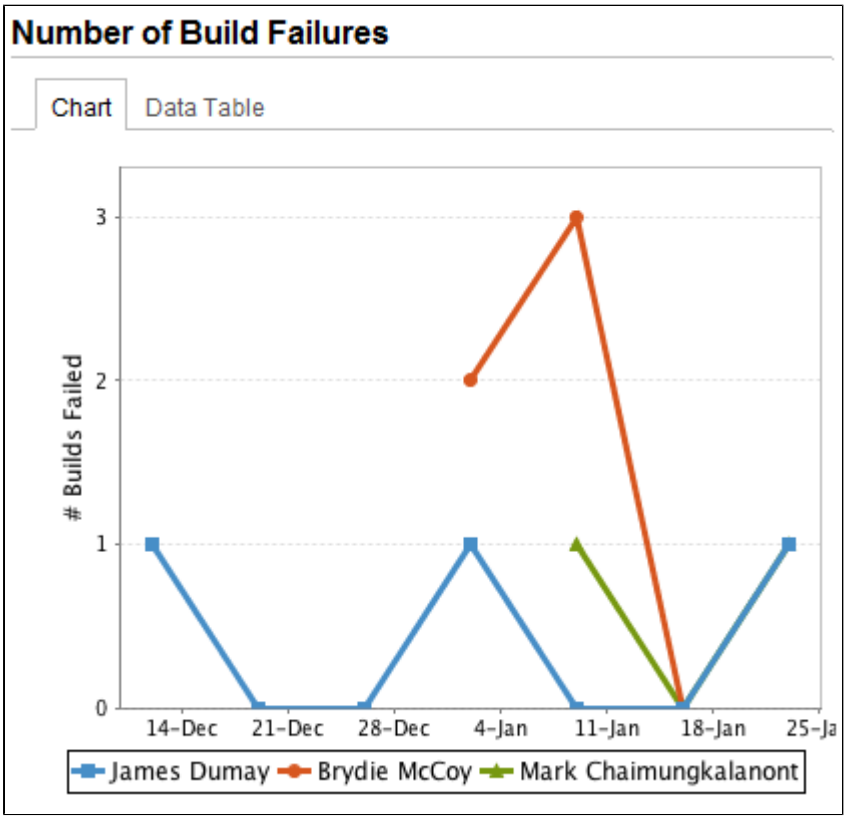
Selected author report types

The following standard report types are available.

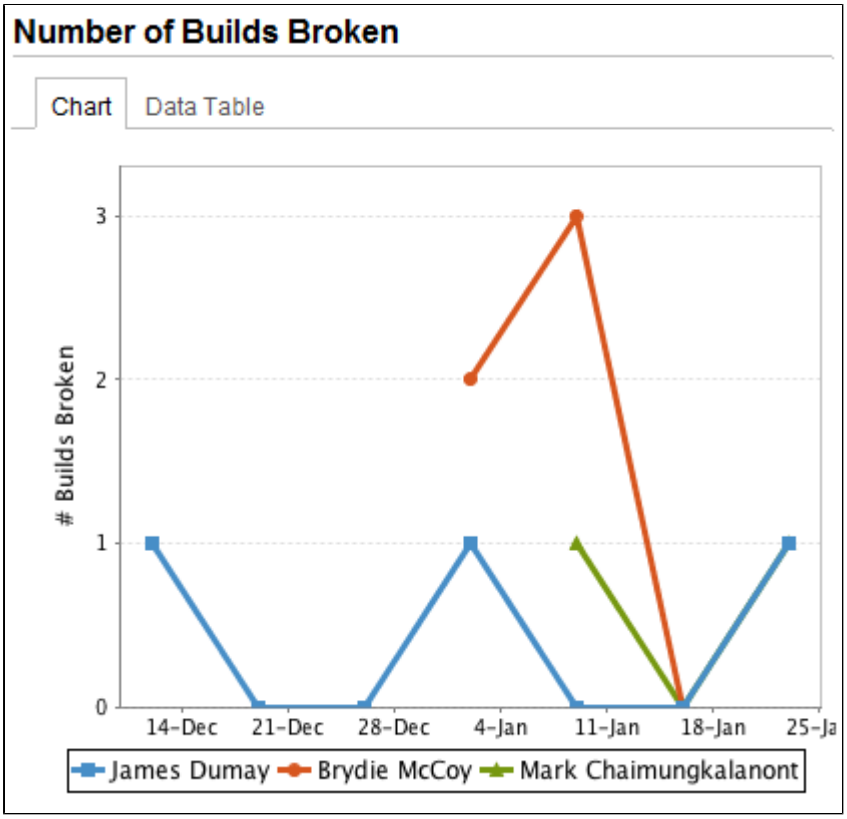
Build activity



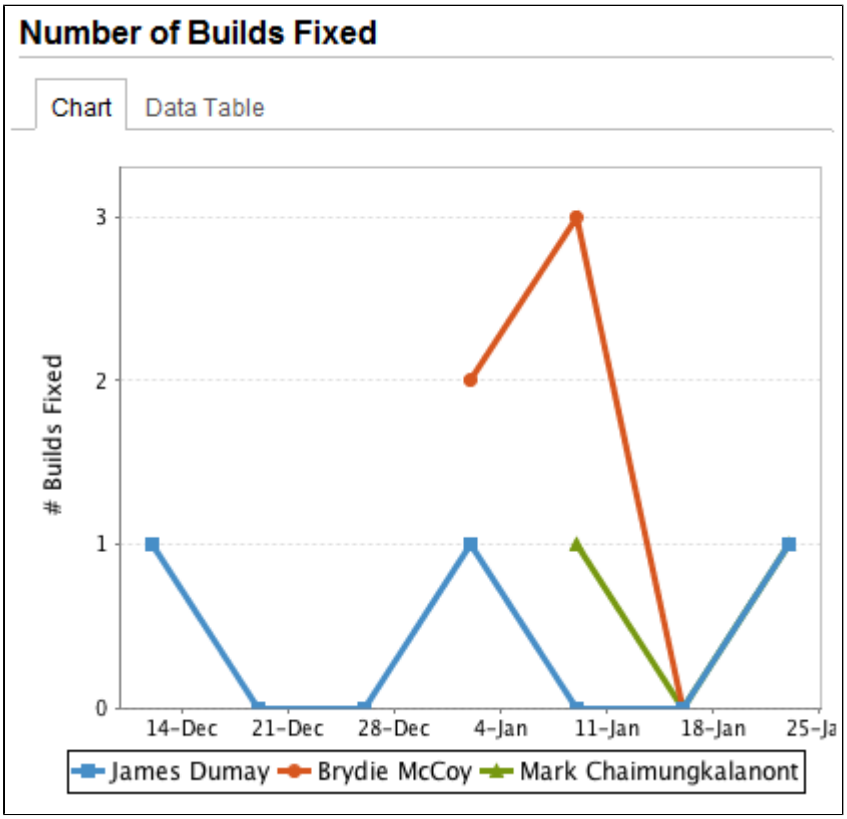
Number of build failures



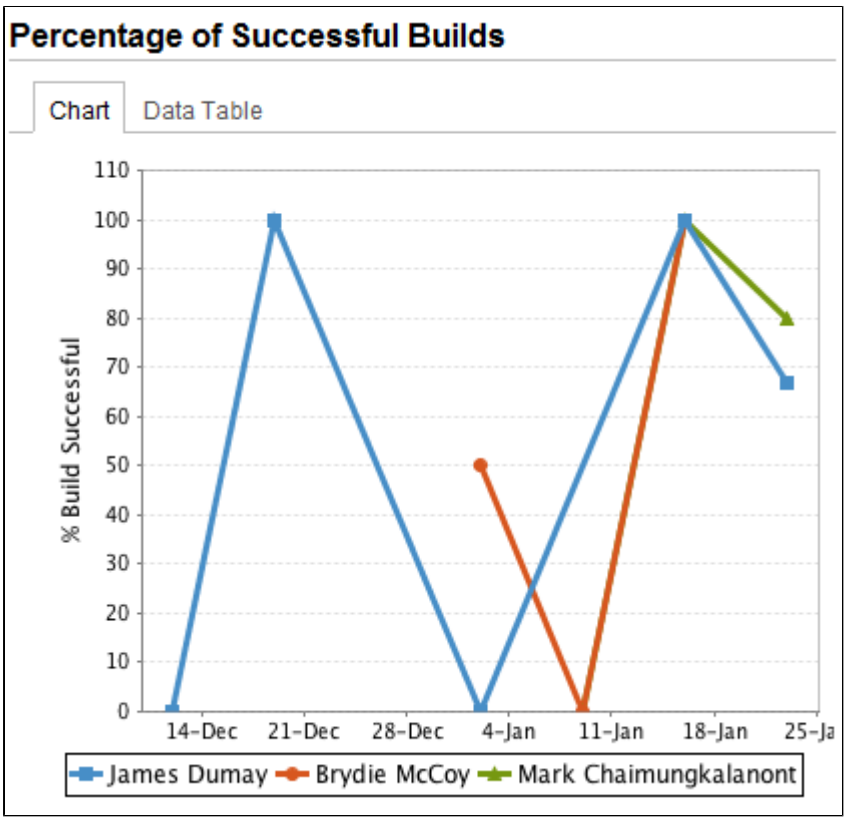
Number of builds broken



Number of builds fixed



Percentage of successful builds



Generating reports across multiple plans

Bamboo provides a report generator that enables you to compare build statistics across one or more plans, using a variety of different metrics.

Generating plan reports

To report on build statistics per plan:

1. From the top navigation bar select **Reports** > **Reports**.
2. Set the report parameters:

   Report
   Select from the available reports, shown below. Additional reports may have been added through custom plugins.

   Build plans
   Select the plans on which you want to report. You can use the **<Ctrl>** key to select multiple plans.

   Group by
   Select the time scale for the horizontal axis.

   Date filter
   Select the time period on which to report. Use **Select range** to set a custom range.

3. Select **Submit**.

On this page:

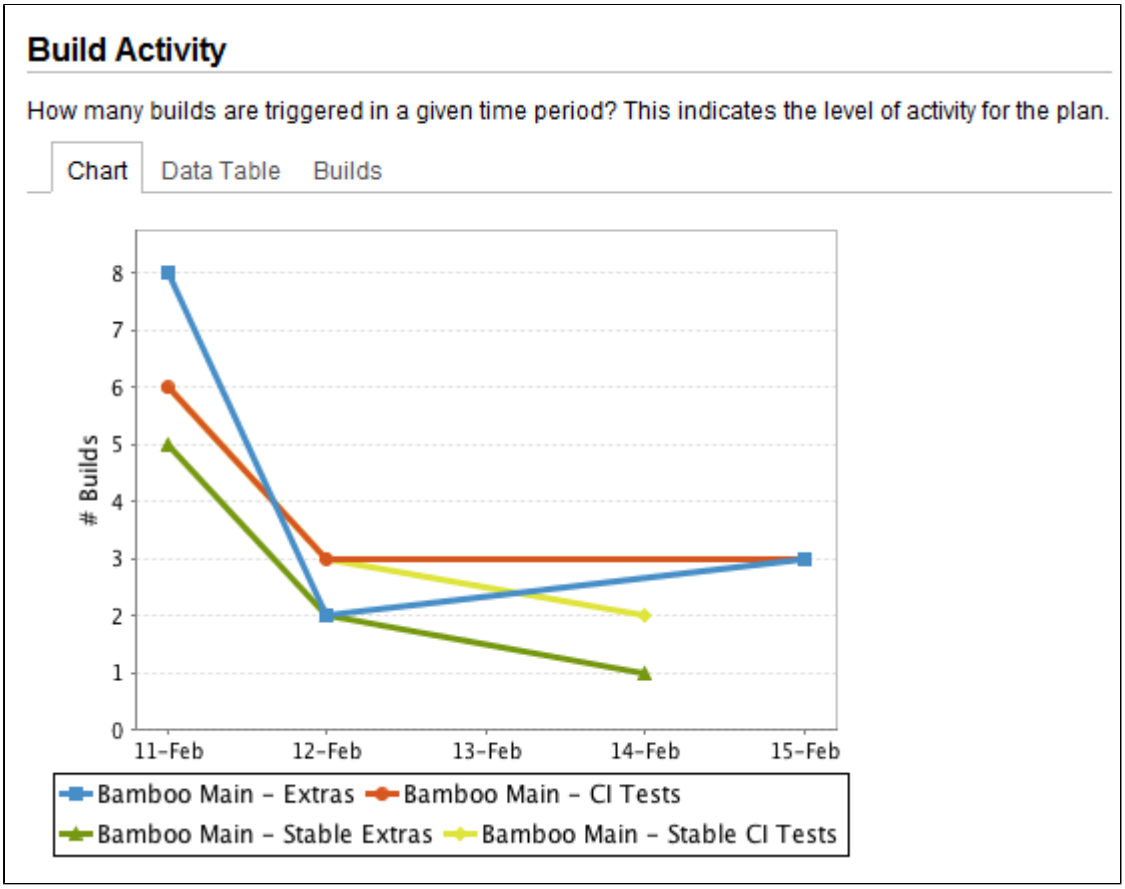
Related pages:

- Generating reports on selected authors
- Viewing build results for an author
- Getting feedback

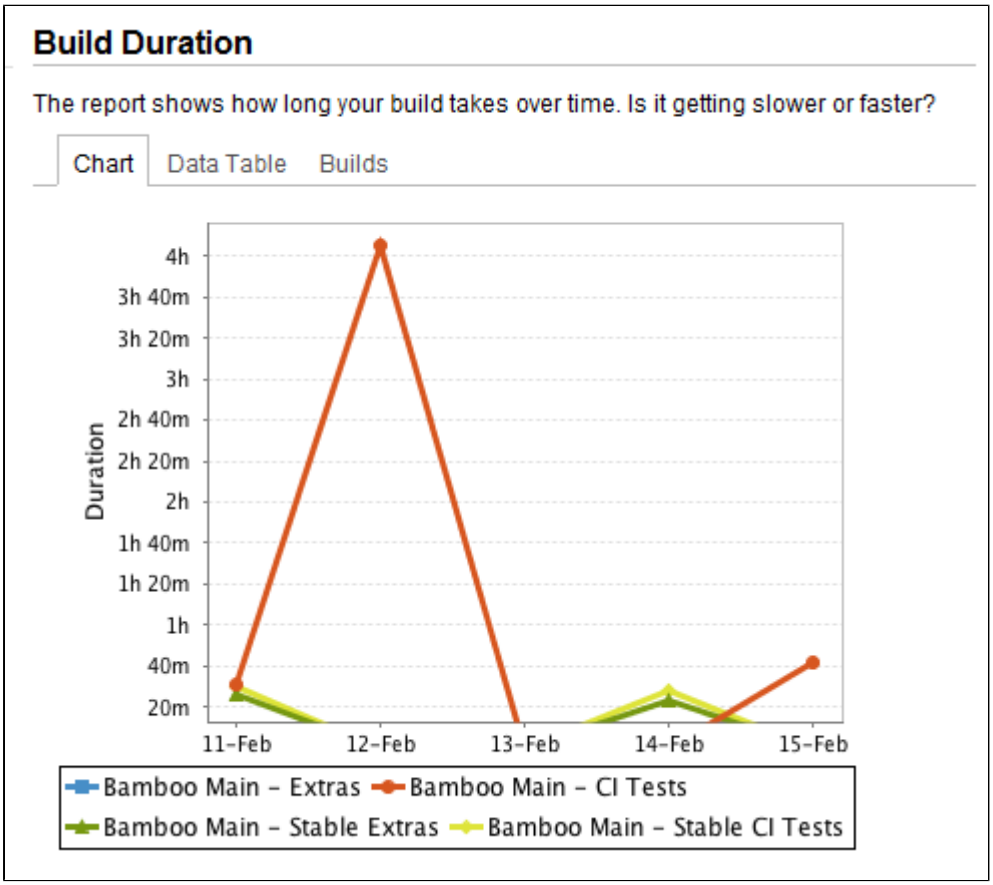
Plan report types

Some of the standard plan report types are illustrated below.

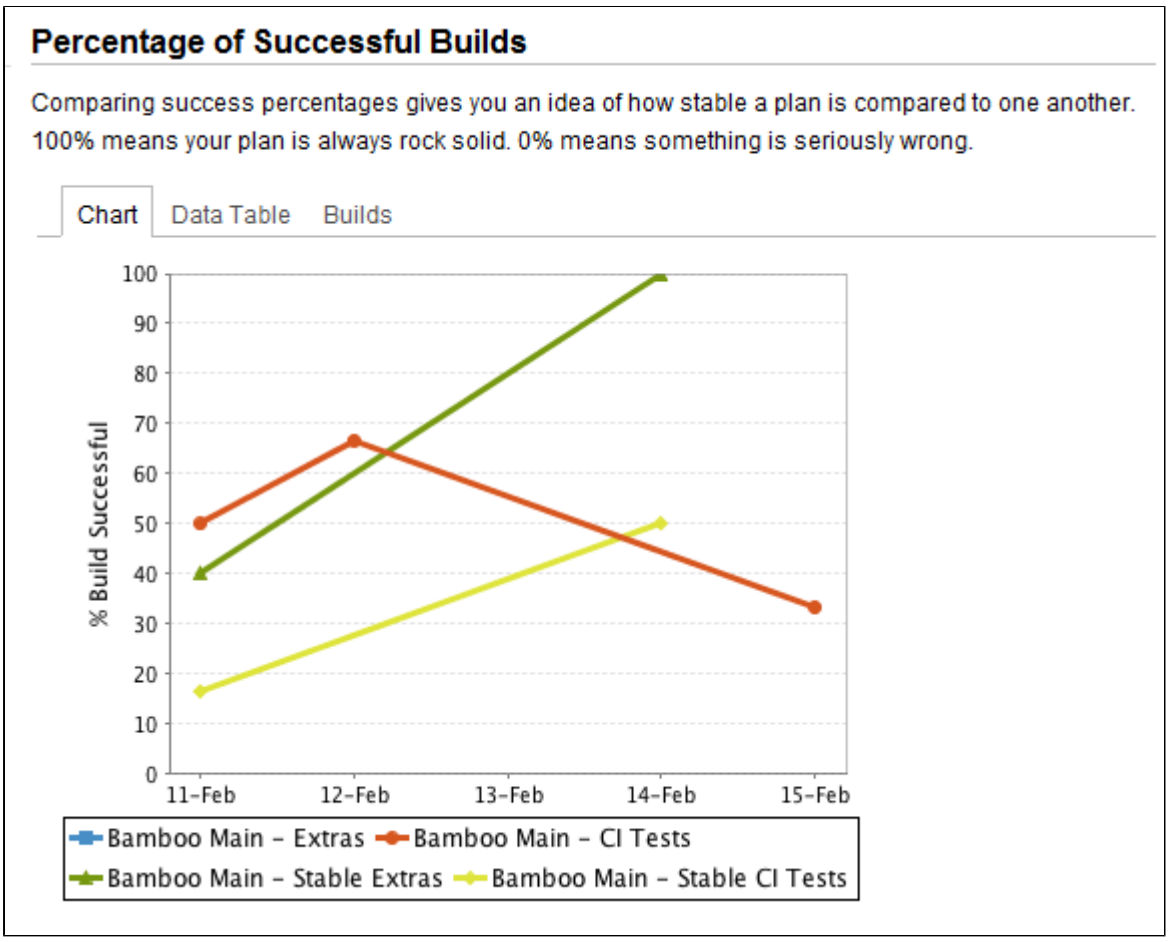
Build activity



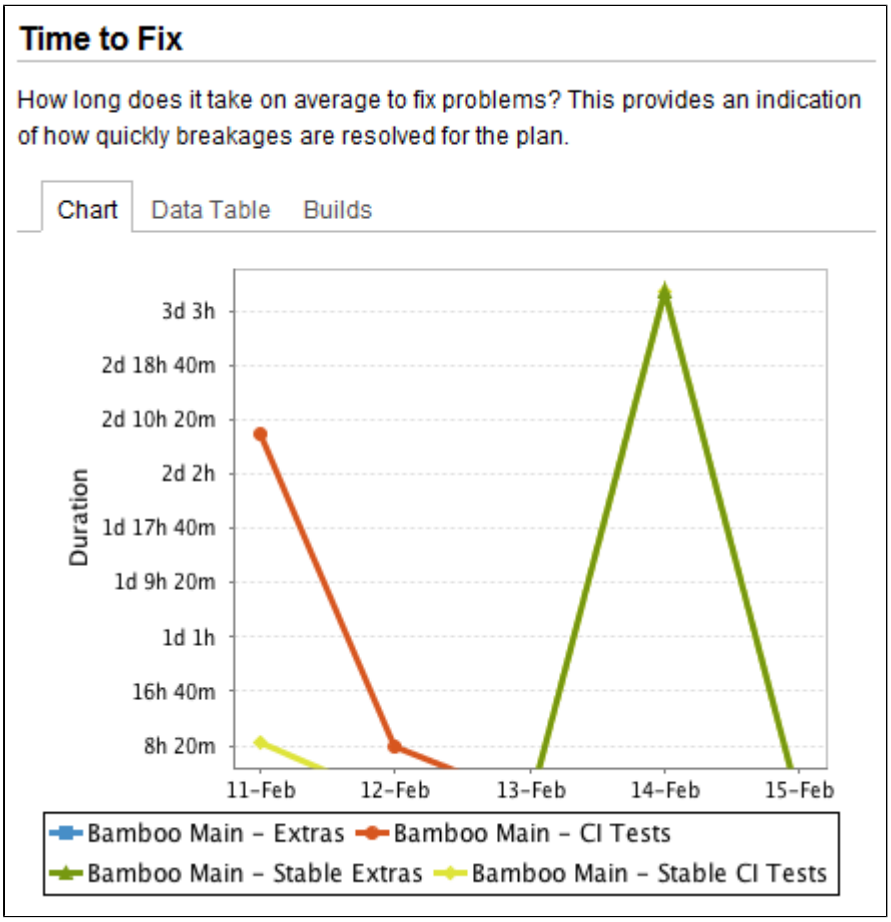
Build duration



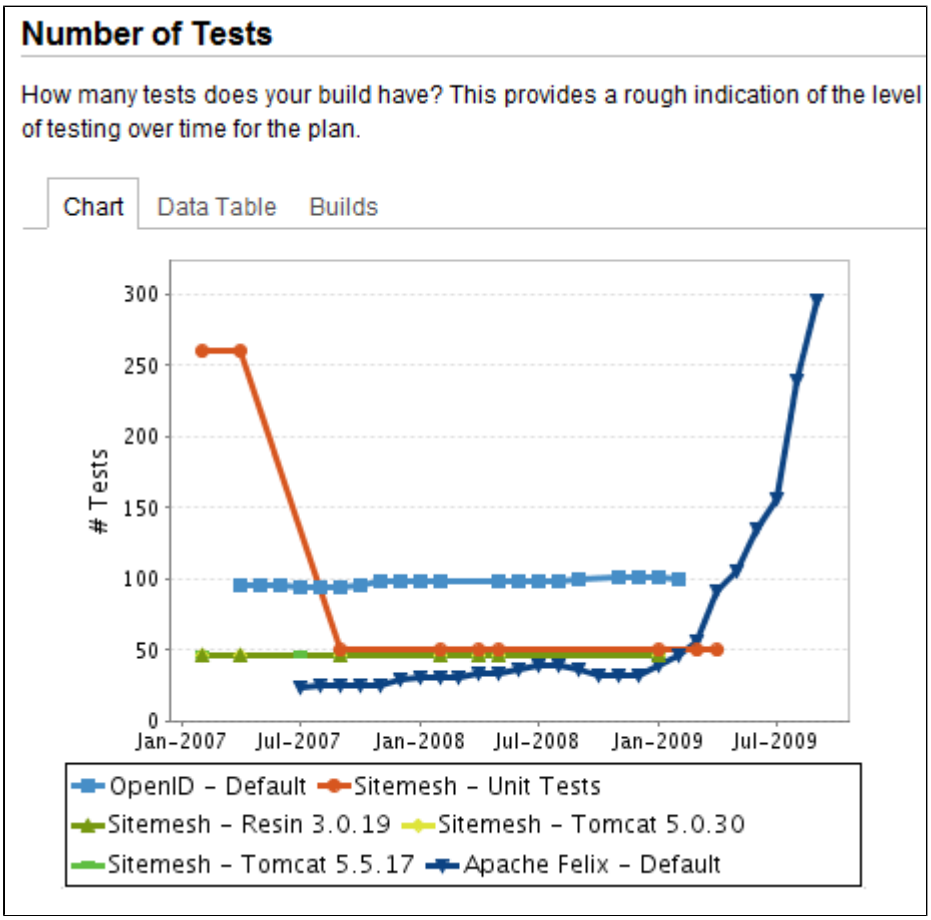
Percentage of successful builds



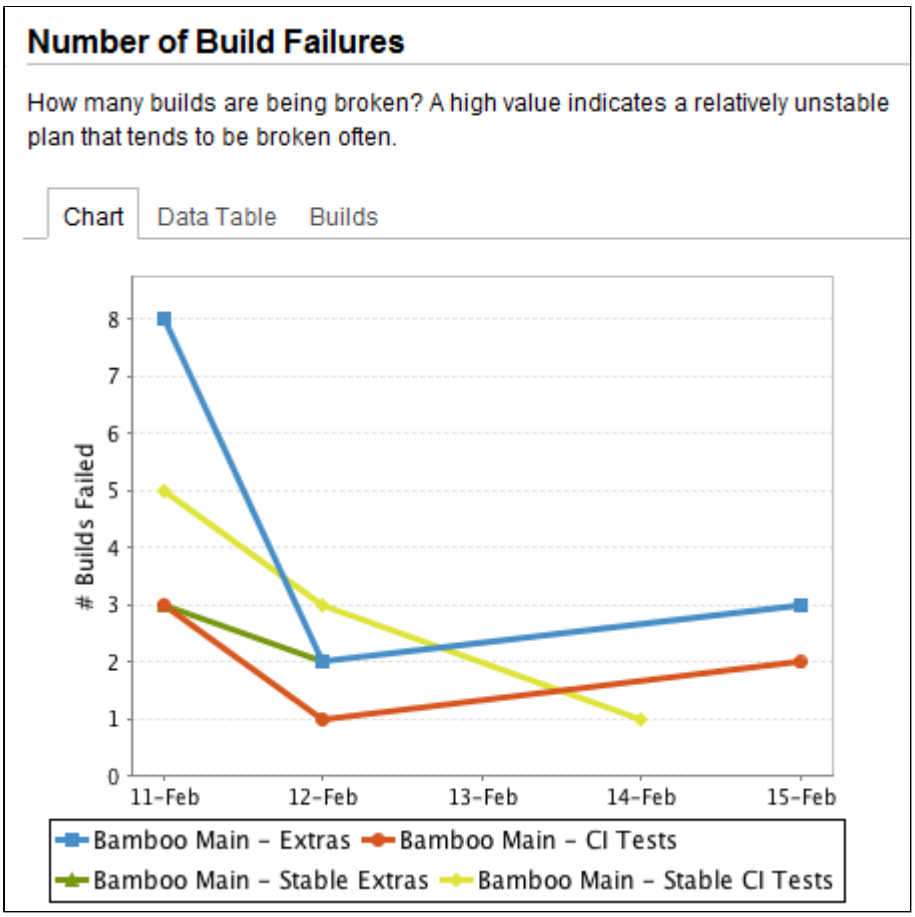
Time to fix



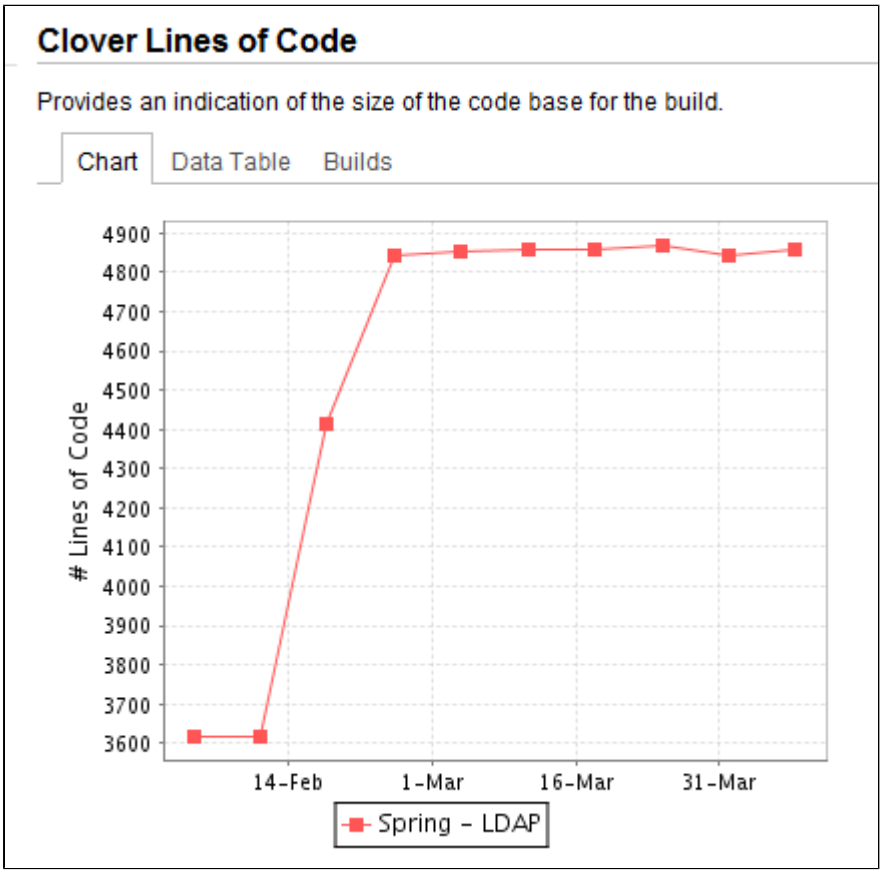
Number of tests



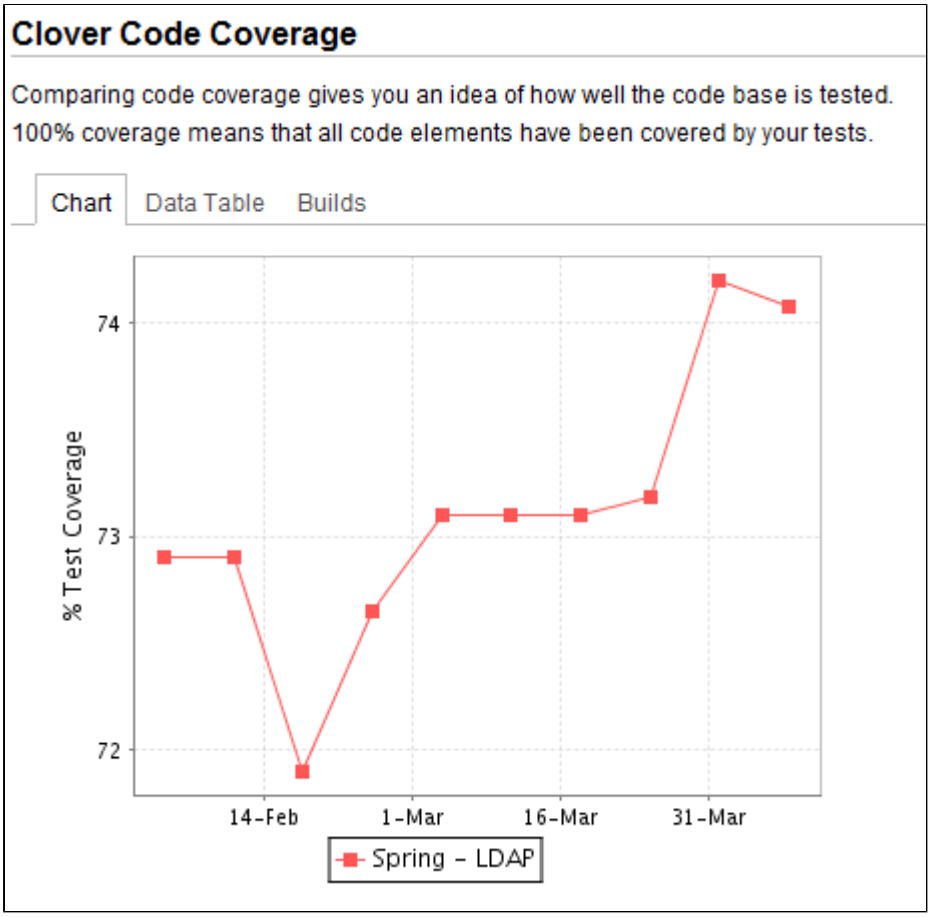
Number of build failures



Clover lines of code



Clover code coverage



Viewing the Clover code-coverage for a plan

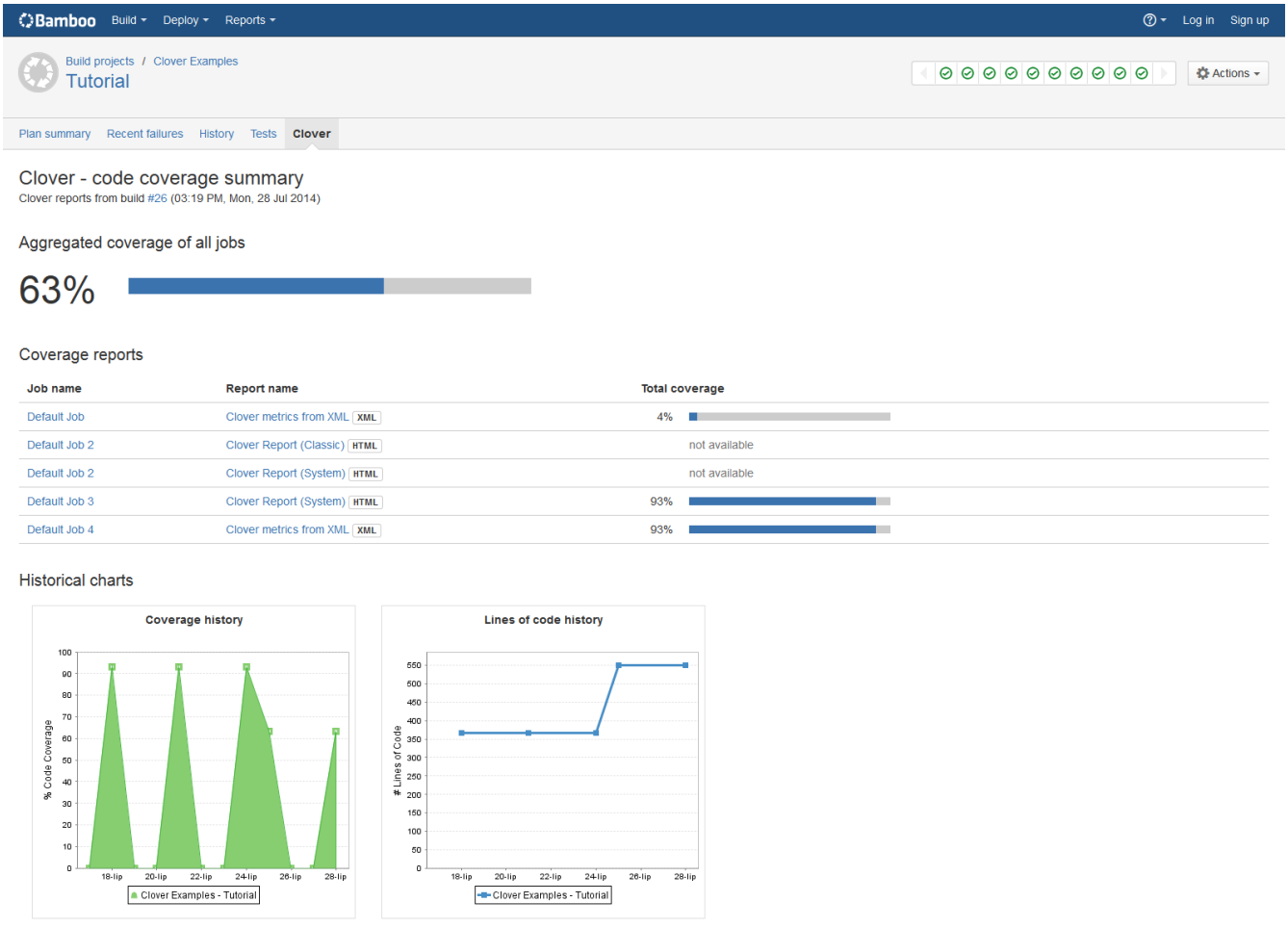
If you use Atlassian's Clover and your job specifies a Clover directory (see Enabling Clover for Bamboo), you will be able to view the Clover coverage summary for the plan.

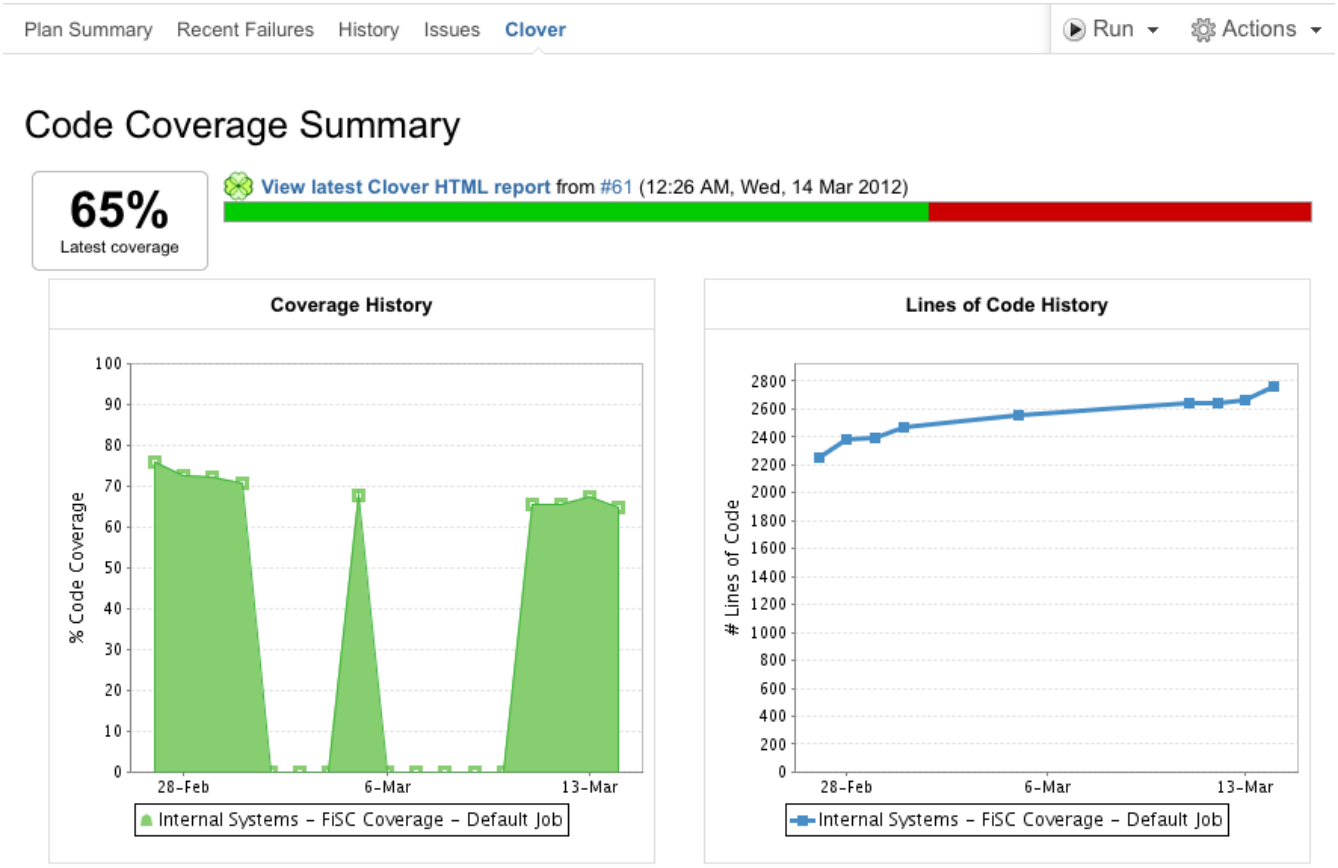
Related pages:

- Enabling Clover for Bamboo
- Viewing the Clover code-coverage for a build
- Generating reports across multiple plans

To view the Clover coverage summary for a plan:

1. Go to the desired plan.
2. Select the **Clover** tab. The Code coverage summary information includes:
    - Latest coverage from the most recent build as a percentage and bar representation (aggregated results from all Clover-enabled jobs).
    - A link to detailed HTML reports.
    - Coverage History chart showing changes in percentage Code Coverage over time.
    - Lines of Code History chart showing changes in LOC over time.





Note:

1. Charts are only generated when build results from at least a 2-day span are available. Where shorter time spans are available, the user will receive a warning stating "Insufficient data in range to draw the chart."
2. Where your plan contains multiple jobs with Clover, then Code Coverage and Lines of Code values are aggregated from all these jobs.
3. *Bamboo 5.6.0 and older:* If your plan contains multiple jobs with Clover, the **View latest Clover HTML report** link will point to the default job only. In order to see other reports, you must go to the specific job summary, as described in Viewing the Clover code-coverage for a build.

Viewing the Clover code-coverage for a build

If your organization uses the Atlassian Clover code-coverage tool, Bamboo can record code-coverage details (i. e. the percentage of code covered by tests) for each build result.

This is only available if the build's plan specifies a Clover directory (for details please refer to the Enabling Clover for Bamboo).

Bamboo also provides data on code-coverage trends for a plan over a period of time. For details see the *Related pages*.

Related pages:

- Working with build results
- Enabling Clover for Bamboo
- Generating reports across multiple plans

Clover HTML report for a job

Where Clover generates an HTML report (created by default in automatic integration), you can examine the report in the build job summary page. To view the report:

1. Go to the plan summary.
2. Select the relevant build number.
3. Select the appropriate job.
4. Select the **Clover** tab to open the report. If a job produces more than one report, a list is shown and you can switch between them.

The **Clover** tab is not available on the Build summary page - you must navigate to the Job summary. This is because your build may contain multiple jobs, each of which may have its own Clover report.

Clover statistics report for a job

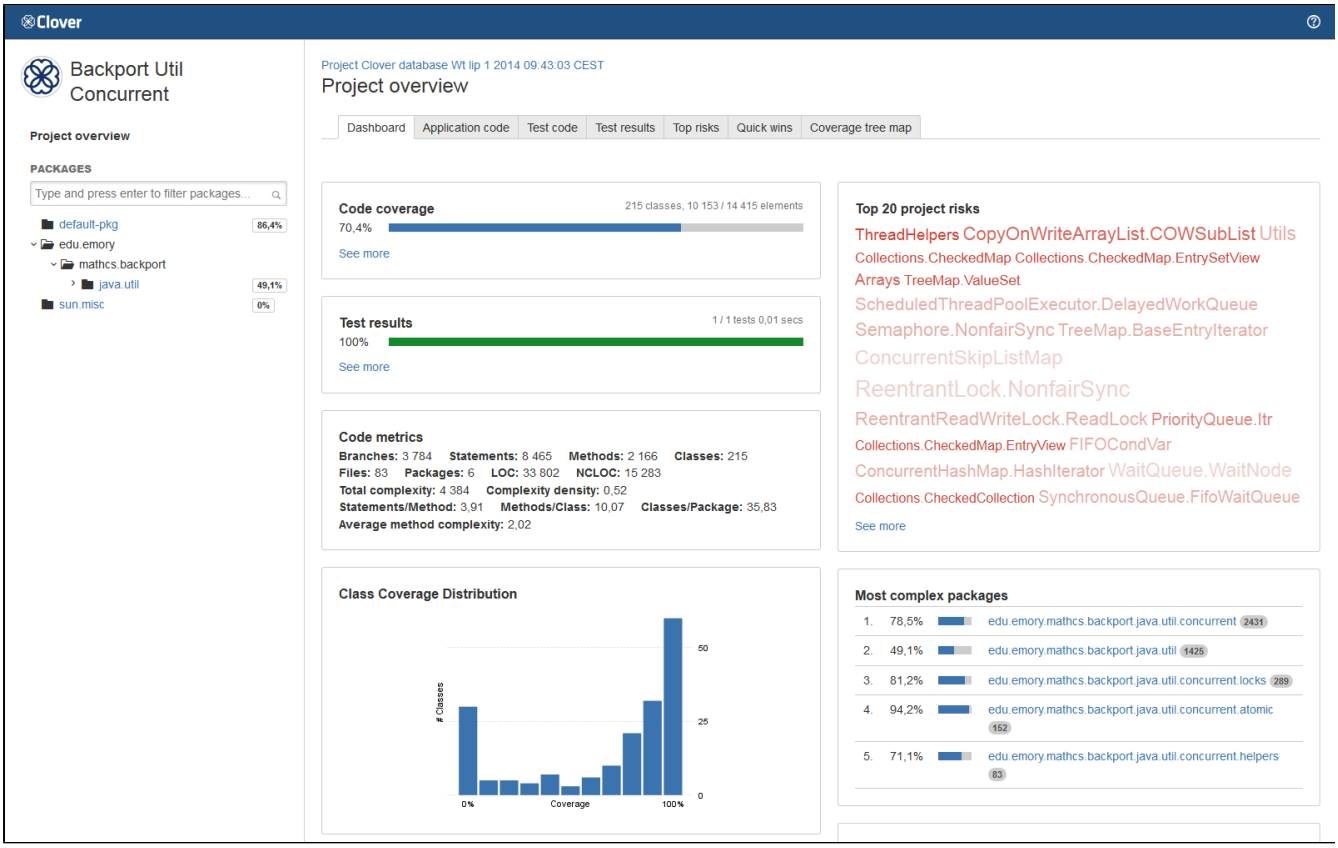
If your build generates a Clover XML report but not the HTML report, then the Clover Report artifact is not available on the **Artifacts** tab, however the build job summary page will contain a few code coverage statistics:

1. Go to the plan summary.
2. Select the relevant build number.
3. Select the appropriate job.
4. Select the **Clover** tab to open the report.

⚠ TIP: This usually happens for manual Clover integration. In case you want to see full Clover report, configure it as described on Enabling Clover for Bamboo page.

References

The content of the Clover HTML report is discussed in detail on the Clover Documentation Home - 4. Understanding Reports page. For completeness, an example Clover Code Coverage HTML report is shown below.



Troubleshooting

The Clover tab shows the directory listing instead of the HTML report

Please check which artifact handler you use. The Amazon S3 Artifact Handler serves files on a one-by-one basis, instead of exposing all files as a static website. To change this, open **Configure plan** and on the **Other** ta b select the **Use custom artifact handler settings** checkbox. Then select **Server-Local Artifact Handler** for shared and non-shared artifacts, and finally re-run the build. See this bug report: CLOV-1560.

Integrating Bamboo with other applications

You can use application links to integrate Bamboo with a number of Atlassian products as well as external, third-party applications by means of OAuth 2.0. Take a look at the following sections to decide which type of integration is best for your needs.

Atlassian applications

You can integrate Bamboo with the following Atlassian applications:

When Bamboo is integrated with a Jira application, you can:

- See Bamboo development activity in Jira applications. Learn more...
- View detailed Bamboo build result information
- View detailed Bamboo deployment information
- Run a Bamboo build when releasing a Jira Software Server version
- Have Bamboo automatically link a plan branch with an issue
- View the Jira application issues linked to a build result
- View the Bamboo builds that relate to a Jira application project or version

When Bamboo is integrated with Bitbucket Server:

- Bamboo will automatically run a build when changes are pushed to the Bitbucket repository, without needing to configure polling.
- Bamboo will automatically update plan branches when a developer pushes a new branch to the repository (or deletes a branch).
- You can click through to Bitbucket to see the commit diff for all files that are part of the changeset.
- Bitbucket commits that are part of a build are displayed in Bamboo.
- Build results are notified to Bitbucket (and displayed for the associated commits and pull requests).

When Bamboo is integrated with Confluence, you can add the following Bamboo gadgets to a Confluence wiki page:

- Bamboo Charts
- Bamboo Plan Summary Chart
- Bamboo Plan Status

When Bamboo is integrated with Fisheye, you can:

- view the code changes that triggered a build
- explore a failed build in Fisheye and jump directly into the changeset that broke the build
- view the history of the changeset to see what the author was trying to fix
- analyze the change using the side-by-side diff view
- open the relevant files in your IDE.

When Bamboo is integrated with Clover, you can:

- View code-coverage details (i.e. the percentage of code covered by tests) for each build result
- View code-coverage trends for a job over a period of time
- View the code-coverage summary for the job.

Also, don't forget to check out the big list of Atlassian gadgets.

External applications

You can also connect Bamboo to external applications using OAuth 2.0. For more information, see Linking to another application.

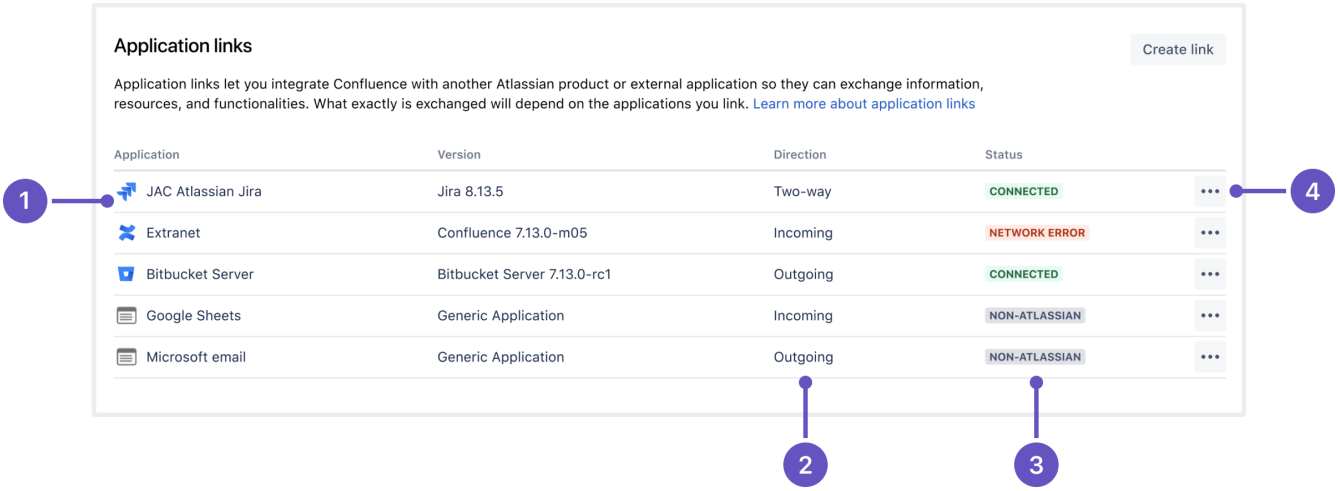
Linking to another application

Application links is a bundled plugin that allows you to link Bamboo to other Atlassian products or external applications so that they can exchange information or share access to certain resources or functionalities.

You can also link Bamboo to external applications using OAuth 2.0. These integrations are typically used for internal integrations and require that your application is compatible with application links.

View application links

You can view your applications links on the **Application links** administration page. Here's what it looks like:



1. **Application** — name of the linked application and its version. External applications always appear as **Generic application**.
2. **Direction** — communication direction, either **Incoming**, **Outgoing**, or **Two-way**. For Atlassian products, you should configure two-way communication, but some external applications won't need it.
3. **Status** — connection status. For external applications, it always appears as **Non-Atlassian**.
4. **Actions** — actions you can perform on your links, such as edit or delete. For OAuth 2.0 connections, you can additionally view your OAuth credentials.

To go to the **Application links** administration page:

1. From the top navigation bar, select **Administration** (⚙) > **Overview**.
2. On the **Bamboo administration** page, select **Application links** from the left menu.

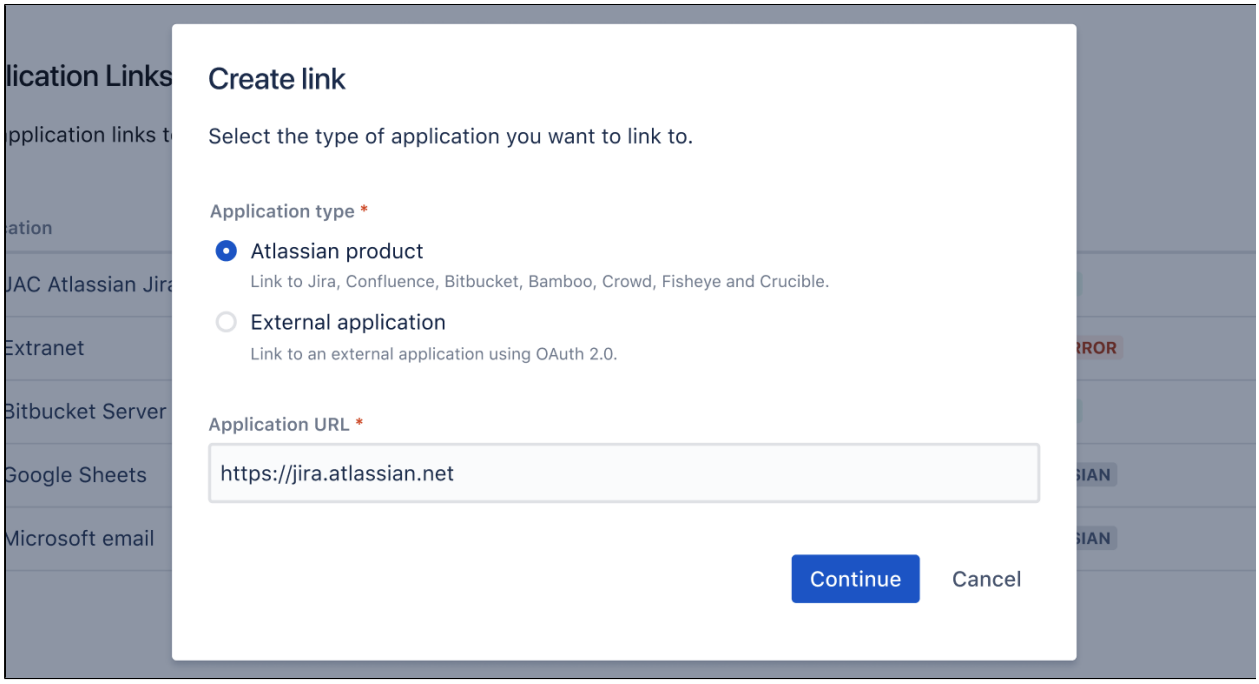
Link to Atlassian products using OAuth 1.0

When you link Bamboo to other Atlassian products, the communication happens over OAuth 1.0.

To link to other Atlassian products using OAuth 1.0:

1. On the **Application links** administration page, select **Create link**.
2. Select **Atlassian product** as the link type.
3. Enter the **Application URL** of your Atlassian product or external application.
4. Follow the steps in the wizard.

   You'll be redirected between Bamboo and the product you're linking to to authorize the two-way connection.



For more information about integrating Bamboo with other Atlassian products, see:

- Integrating Bamboo with JIRA applications
- Integrating Bamboo with Confluence
- Integrating Bamboo with Fisheye
- Integrating Bamboo with Bitbucket Data Center

Link to external applications using OAuth 2.0

You can link Bamboo to external applications using OAuth 2.0 in both directions, either making Bamboo act as a client (outgoing link) or provider (incoming link).

Configure Bamboo as an OAuth 2.0 provider (incoming link)

In this scenario, Bamboo acts as an OAuth provider, allowing the external application to access its data.

For more information, see Configuring an incoming link.

Configure Bamboo as an OAuth 2.0 client (outgoing link)

In this scenario, Bamboo acts as an OAuth client, requesting data from the external application.

> ⚠ While Application links allow you to configure an outgoing link, Bamboo does not yet make use of this functionality. We're planning to add support for this in a future release of Bamboo.

Configuring an incoming link

When you configure an incoming link with an external application, Bamboo acts as the OAuth provider, allowing the external application access to Bamboo data. To learn more about the type of links and additional details, see Linking to another application.

Before you begin

- If you're creating an OAuth 2.0 integration and want to use Bamboo as the provider, you can find the details of our OAuth 2.0 implementation in Bamboo OAuth 2.0 provider API.
- You can configure additional details using OAuth 2.0 provider system properties.

Steps

To create and configure an incoming link:

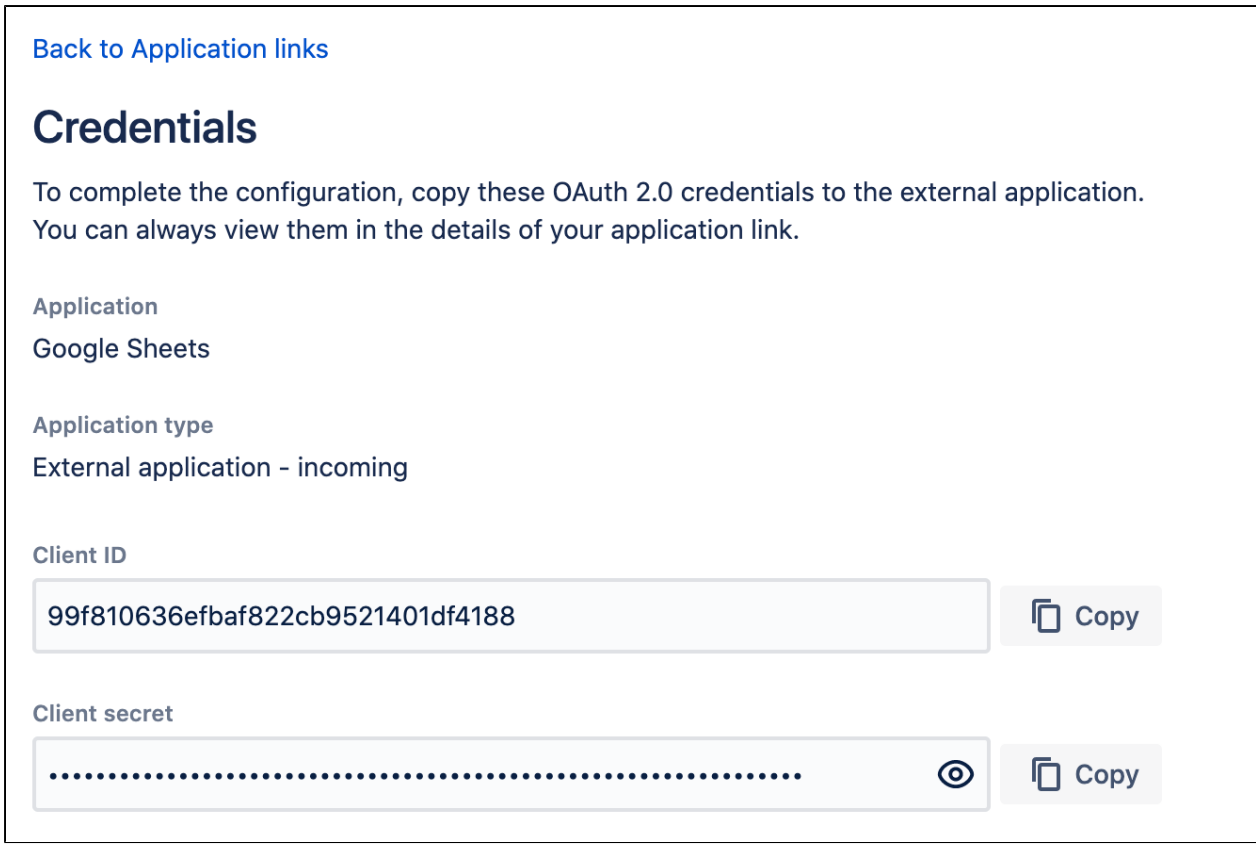
1. From the top navigation bar, select **Administration ( )** > **Overview**.
2. On the **Bamboo administration** page, select **Application links** from the left menu.
3. Select **Create link**.
4. Select **External application**, and then choose **Incoming** as the direction.
5. Provide the **Name** for the link and the **Redirect URL** (also known as Callback URL) from your external application.

   After authorizing the application, the user will be redirected to this URL with the authorization code.
6. Select permissions the application can have on your instance. You can choose the following permission scopes:

   - `READ`
   - `TRIGGER`
   - `USER`

   > Even if you grant higher permissions, the application won't be able to do more than the user authorizing it. For more info on what each of these scopes do, see OAuth 2.0 scopes for incoming links.

   After providing the redirect URL and selecting the scopes, Bamboo will generate the OAuth credentials that include these details.
7. Copy the generated credentials to your external application to complete the link.



The incoming application link is now set up. You can view its details on the **Application links** administration page, including the OAuth credentials in case you needed to access them later.

Viewing OAuth credentials for an existing link

If you lose your OAuth credentials, you can access them any time in the details of the application link you created:

1. From the top navigation bar, select **Administration ( )** > **Overview**.
2. On the **Bamboo administration** page, select **Application links** from the left menu.
3. Find the application link you're interested in, and then select **More actions** > **View credentials**.

OAuth 2.0 scopes for incoming links

When configuring an incoming application link, you need to select scopes; that is, the permissions that the application can have on behalf of a user in your Bamboo instance.

What the application can do with scopes

As an admin, you can select which scopes the application can request from the authorizing user, but the actual permissions will always be limited to what a particular user can do themselves. For example, even if you select the `TRIGGER` scope, the application won't be able to use the permissions associated with that scope if the authorizing doesn't have the Build permission.

Scopes

Here are the scopes you can select when configuring the link. The same scopes will be displayed to users when they authorize the integration. They can later be accessed in their user profile in **Authorized applications**, where they can also revoke the granted access.

Scope	Description
`READ`	The access token will only have permissions to read the same data that the associated user normally has access to in Bamboo.
`TRIGGER`	The access token will have permissions to start the same builds and deploy the same environments that the associated user normally can.
`USER`	The access token will have the same set permissions as the associated users (including Administrator permissions, if the associated user has them).

Bamboo OAuth 2.0 provider API

Bamboo Data Center and Server provides APIs to allow external services to access resources on a user's behalf with the OAuth 2.0 protocol.

If you already have an integration that you'd like to add to Bamboo, see Con figuring an incoming link for detailed steps. If not, this page will help you understand the details of our OAuth 2.0 implementation so you can create such an integration.

On this page:

Supported OAuth 2.0 flows

We support the following OAuth 2.0 flows:

- Authorization code with Proof Key for Code Exchange (PKCE)
- Authorization code

We don't support Implicit Grant and Resource Owner Password Credentials flows, as they will be deprecated in the next version of the OAuth specification.

For more information on how these flows work, see OAuth RFC. This should help you understand the flows and choose the right one for you.

Security recommendations

Here are some recommendations on how to improve security:

Preventing CSRF attacks

To protect redirect-based flows, the OAuth specification recommends the use of "One-time use CSRF tokens carried in the state parameter, which are securely bound to the user agent" using the `state` query parameter, with each request to the `/rest/oauth2/latest/authorize` endpoint. This can prevent CSR F attacks.

Using HTTPS in production

For production environments, use HTTPS for the `redirect_uri` parameter. This is important as OAuth 2.0 bases its security on the transport layer. For more info, see the OAuth 2.0 RFC and the OAuth 2.0 Threat Model RFC.

For the same reason, we also enforce HTTPS for the base URL of production environments. You can use insecure URIs and base URLs for staging or development environments by enabling the relevant system properties.

Authorization code with Proof Key for Code Exchange (PKCE)

This flow lets you securely perform the OAuth exchange of client credentials for access tokens on public clients. The following steps and parameters describe our implementation of this flow.

Parameters

Here are parameters you'll use in this flow:

Parameter	Description	Required
`redirect_uri`	URL that the user will be redirected to after authorizing the request.	Yes
`client_id`	Client ID received from Bamboo after registering your application.	Yes
`response_time`	Authorization code.	Yes
`scope`	Scopes that define application's permissions to the user account. For more info, see Scopes.	Yes
`code_challenge`	For `sha256`, generate this using the following pseudocode: `BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))` For `plain`, this can be the generated `code_verifier`.	Yes
`code_challenge_method`	Can be `plain` or `sha256` depending on how the `code_challenge` was generated.	Yes
`code_verifier`	High-entropy cryptographic random string using the unreserved characters: `[A-Z] / [a-z] / [0-9] / "-" / "." / "_" / "~"`. Must be between 43-127 characters. For more info, see section 4.1 of RFC7636.	Yes
`state`	An optional parameter, which value can't be predicted. It'll be used by the client to maintain state between the request and callback. It should also be used as a CSRF token. It can be generated in a similar manner to `code_verifier`.	No

Before you begin

- Register your application in Bamboo by creating an incoming. During registration, you can enable proper scopes to limit the range of resources that the application can access. After creating the link, you should receive the OAuth Client ID and Client secret — make sure to keep these secure. For more info, see Configuring an incoming link.
- Before starting the flow, generate `state` (optional), `code_verifier`, `code_challenge`, and `code_challenge_method`.

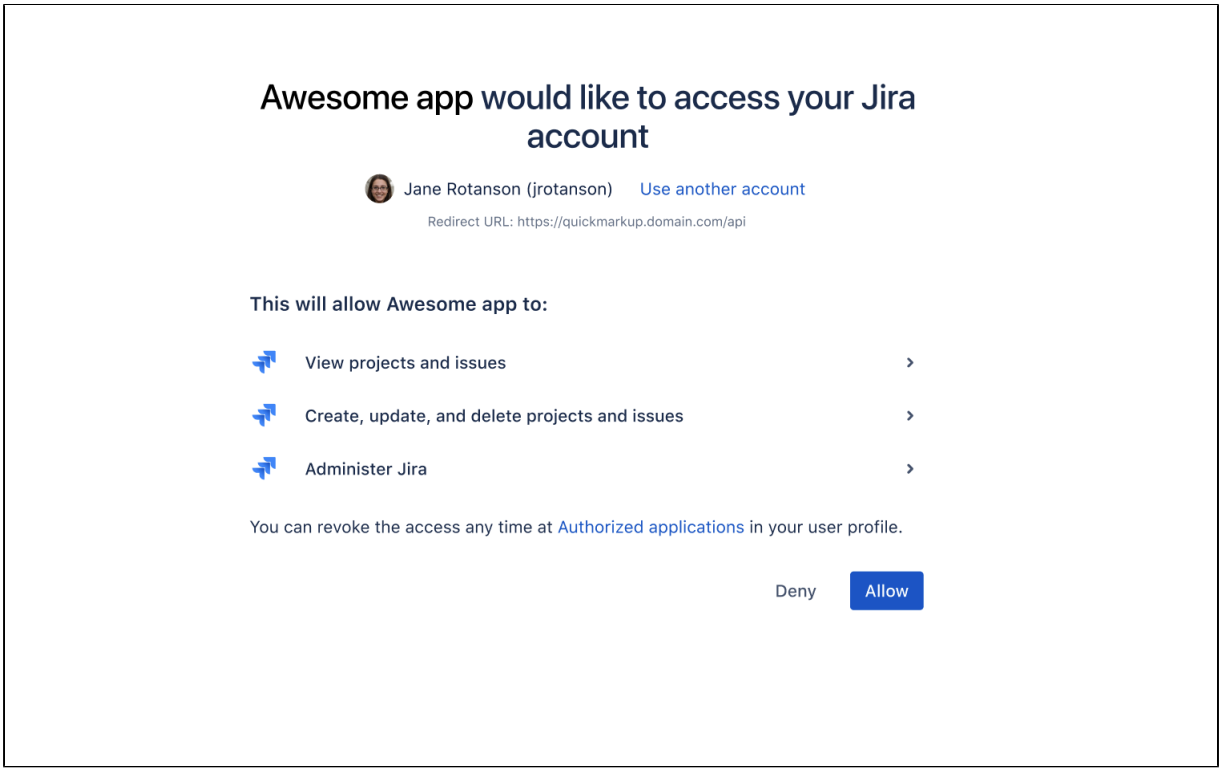
Steps

1. Request authorization code by redirecting the user to the `/rest/oauth2/latest/authorize` page with the following query parameters:

```
curl https://atlassian.example.com/rest/oauth2/latest/authorize?
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&state=STATE&scope=SCOPE&code_chal
lenge=CODE_CHALLENGE&code_challenge_method=S256
```

   This is the consent screen that asks the user to approve the application's request to access their account with the scopes specified in `scope`. The user is then redirected to the URL specified in `redirect_uri`. The redirect includes the authorization code. For example:

```
https://atlassian.example.com/plugins/servlet/oauth2/consent?
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE&code_chal
lenge_method=CODE_CHALLENGE_METHOD&code_challenge=CODE_CHALLENGE
```



2. With the authorization `code` returned from the previous request, you can request an `access_token`, with any HTTP client. The following example uses curl:

```
curl -X POST https://atlassian.example.com/rest/oauth2/latest/token?
client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&grant_type=authorization_code&redirect_u
ri=REDIRECT_URI&code_verifier=CODE_VERIFIER
```

Example response:

```
{
 "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjNmMTQ3NTUzYjg3OTQ2Y2FhMWJhYWJkZWQ0MzgwYTM4In0.
EDnpBl0hd1BQzIRP--xEvyW1F6gDuiFranQCvi98b2c",
 "token_type": "bearer",
 "expires_in": 7200,
 "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6ImMwZTMxYmZjYTI2NWI0YTkwMzBiOGM2OTJjNWIyMTYwIn0.
grHOsso3B3kaSxNd0QJfj1H3ayjRUuA75SiEt0usmiM",
 "created_at": 1607635748
}
```

3. To retrieve a new `access_token`, use the `refresh_token` parameter.

> Refresh tokens may be used even after the `access_token` itself expires.

```
curl -X POST https://atlassian.example.com/rest/oauth2/latest/token?
client_id=CLIENT_ID&client_secret=CLIENT_SECRET&refresh_token=REFRESH_TOKEN&grant_type=refresh_tok
en&redirect_uri=REDIRECT_URI
```

This request:

- invalidates the existing `access_token` and `refresh_token`
- sends new tokens in the response

Example response:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6ImJmZjg4MzU5YTVkNGUyZmQ3ZmYwOTEwOGIxNjg4MDA0In0.
BocpI91mpUzWskyjxHp57hnyl8ZcHehGJwmaBsGJEMg",
  "token_type": "bearer",
  "expires_in": 7200,
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6Ijg1NjQ1YjA1NGJiYmZkNjVmMDNkMzliYzM0YzQ4MzZjIn0.
4MSMIG46zjB9QCV-qCCglgojM5dL7_E2kcqmiV46YQ4",
  "created_at": 1628711391
}
```

You can now make requests to the API with the returned access token. For more info, see Accessing the Bamboo API with the access token.

Authorization code

This flow lets you securely perform the OAuth exchange of client credentials for access tokens on public clients.

Parameters

Here are the parameters you'll use in this flow:

Parameter	Description	Required
`redirect_uri`	URL that the user will be redirected to after authorizing the request.	Yes
`client_id`	Client ID received from Bamboo after registering your application.	Yes
`response_type`	Authorization code.	Yes
`scope`	Scopes that define application's permissions to the user account. For more info, see Scopes.	Yes
`state`	An optional parameter, which value can't be predicted. It'll be used by the client to maintain state between the request and callback. It should also be used as a CSRF token.	No

Before you begin

- Register your application in Bamboo by creating an incoming. During registration, you can enable proper scopes to limit the range of resources that the application can access. After creating the link, you should receive the OAuth Client ID and Client secret — make sure to keep these secure. For more info, see Configuring an incoming link.
- If you want to use the optional `state` parameter, generate it before starting the flow.

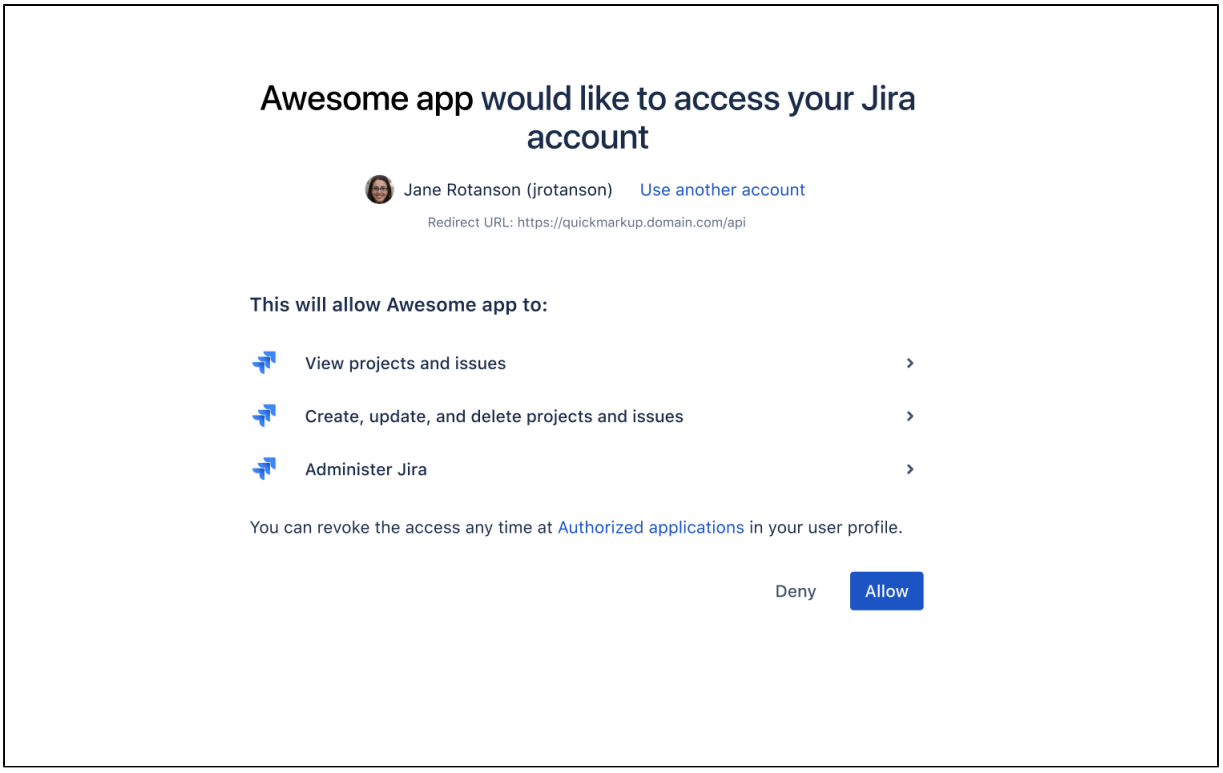
Steps

1. Request the authorization code by redirecting the user to the `/oauth/authorize` page with the following query parameters:

```
curl https://atlassian.example.com/rest/oauth2/latest/authorize?
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&state=STATE&scope=SCOPE
```

This is the consent screen that asks the user to approve the application's request to access their account with the scopes specified in scope. The user is then redirected to the URL specified in redirect_uri. The redirect includes the authorization code, like in the following example:

```
https://atlassian.example.com/plugins/servlet/oauth2/consent?
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE
```



2. With the authorization code (response_type) returned from the previous request, you can request an access_token, with any HTTP client. The following example uses Ruby's rest-client:

```
curl -X POST https://atlassian.example.com/rest/oauth2/latest/token?
client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&grant_type=authorization_code&redirect_u
ri=REDIRECT_URI
```

Example response:

```
{
 "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjNmMTQ3NTUzYjg3OTQ2Y2FhMWJhYWJkZWQ0MzgwYTM4In0.
EDnpBl0hd1BQzIRP--xEvyW1F6gDuiFranQCvi98b2c",
 "token_type": "bearer",
 "expires_in": 7200,
 "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6ImMwZTMxYmZjYTI2NWI0YTkwMzBiOGM2OTJjNWIyMTYwIn0.
grHOsso3B3kaSxNd0QJfj1H3ayjRUuA75SiEt0usmiM",
 "created_at": 1607635748
}
```

3. To retrieve a new `access_token`, use the `refresh_token` parameter.

> Refresh tokens may be used even after the `access_token` itself expires.

```
curl -X POST https://atlassian.example.com/rest/oauth2/latest/token?
client_id=CLIENT_ID&client_secret=CLIENT_SECRET&refresh_token=REFRESH_TOKEN&grant_type=refresh_tok
en&redirect_uri=REDIRECT_URI
```

This request:

- Invalidates the existing `access_token` and `refresh_token`.
- Sends new tokens in the response.

Example response:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6ImJmZjg4MzU5YTVkNGUyZmQ3ZmYwOTEwOGIxNjg4MDA0In0.
BocpI91mpUzWskyjxHp57hnyl8ZcHehGJwmaBsGJEMg",
  "token_type": "bearer",
  "expires_in": 7200,
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6Ijg1NjQ1YjA1NGJiYmZkNjVmMDNkMzliYzM0YzQ4MzZjIn0.
4MSMIG46zjB9QCV-qCCglgojM5dL7_E2kcqmiV46YQ4",
  "created_at": 1628711391
}
```

You can now make requests to the API with the returned access token. For more info, see Accessing the Bamboo API with the access token.

Accessing the Bamboo API with the access token

To access the Bamboo API with the previously returned access token, put the token in the Authorization header. For example:

```
curl --header "Authorization: Bearer OAUTH2-TOKEN" "https://atlassian.example.com:8085/bamboo/rest/api
/latest/plan/{projectKey}-{buildKey}"
```

Scopes

The scope parameter is required in both flows. It allows you to specify the permission scopes your application can request from the authorizing user. Note that regardless of which scopes you choose, the actual permissions will always be capped at what the user can actually do.

Here you can find the scope keys you can use in your requests, as values of the scope parameter:

Scope	Description
`READ`	The access token will only have permissions to read the same data that the associated user normally has access to in Bamboo.
`TRIGGER`	The access token will have permissions to start the same builds and deploy the same environments that the associated user normally can.
`USER`	The access token will have the same set permissions as the associated users (including Administrator permissions, if the associated user has them).

OAuth 2.0 provider system properties

When configuring Bamboo as an OAuth 2.0 provider (see Configuring an incoming link), you can use the following system properties. For more information, see Configuring your system properties.

atlassian.oauth2.provider.enable.access.tokens	
Default	`true`
Description	Disables the ability to authenticate using access tokens for that node.

atlassian.oauth2.provider.skip.base.url.https.requirement	
Default	`false`
Description	Disables the HTTPS requirement for the base URL. If this is disabled, the OAuth 2.0 provider will be enabled even if the product is using HTTP.

atlassian.oauth2.provider.skip.redirect.url.https.requirement	
Default	`false`
Description	Disables the HTTPS requirement for the Redirect URL. If this is disabled, the OAuth 2.0 provider will allow Redirect URLs using HTTP.

atlassian.oauth2.provider.max.lock.timeout.seconds	
Default	10
Description	Number of seconds a request will await lock access before timing out.

atlassian.oauth2.provider.max.client.delay.seconds	
Default	10
Description	Max lifetime of authorization codes (seconds). The limit is 600 seconds.

atlassian.oauth2.provider.prune.expired.authorizations.schedule	
Default	`* * * * * ?`
Description	Cron expression for a job that removes expired authorization codes. Default is 1 minute.

atlassian.oauth2.provider.access.token.expiration.seconds	
Default	`3600` (1 hour)
Description	Max lifetime of access tokens (seconds).

atlassian.oauth2.provider.prune.expired.tokens.schedule	
Default	`* * * * * ?`
Description	Cron expression for a job that removes expired access tokens. Default is 1 minute.

atlassian.oauth2.provider.access.token.expiration.seconds

Default	`7776000` (90 days)
Description	Max lifetime of refresh tokens (seconds).

atlassian.oauth2.provider.invalidate.session.enabled	
Default	`true`
Description	Invalidates a session after a successful authentication using an OAuth token.

atlassian.oauth2.provider.validate.client.secret	
Default	`true`
Description	Validates the client ID and client secret when revoking and creating tokens.

atlassian.oauth2.provider.use.quotes.in.sql	
Default	`false`
Description	Controls whether to add quotes to SQL statements. This is a sanity system property used for database requirements. PostgreSQL will always use quotes unless the `atlassian.oauth2.provider.do.not.use.quotes.in.sql` property (below) is enabled.

atlassian.oauth2.provider.do.not.use.quotes.in.sql	
Default	`false`
Description	Controls whether to add quotes to SQL statements. This is a sanity system property used for database requirements.

atlassian.oauth2.provider.token.via.basic.authentication	
Default	`true`
Description	Enables extracting tokens through the basic authentication password field for access token authentication.

Configuring an outgoing link

When you configure an outgoing link to an external application, Bamboo requests data from this application, which means that it acts as the OAuth client. This type of link is primarily used in Bamboo to create the OAuth 2.0 integration for popular mail servers. To learn more about the type of links and additional details, see Linking to another application.

> ⚠ While Application links allow you to configure an outgoing link, Bamboo does not yet make use of this functionality. We're planning to add support for this in a future release of Bamboo.

When to use this

We've created an outgoing OAuth 2.0 integration primarily because Google and Microsoft announced deprecating basic authentication. This means you wouldn't be able to use these providers (Gmail, Microsoft Exchange Online) to let users create issues and comments from emails if you were authenticating using basic auth. To fix this, you need to configure the OAuth 2.0 integration with these providers, and then update the configuration of your mail servers.

You don't need to take any actions if you're using IMAP or POP3, these will continue to work.

Before you begin

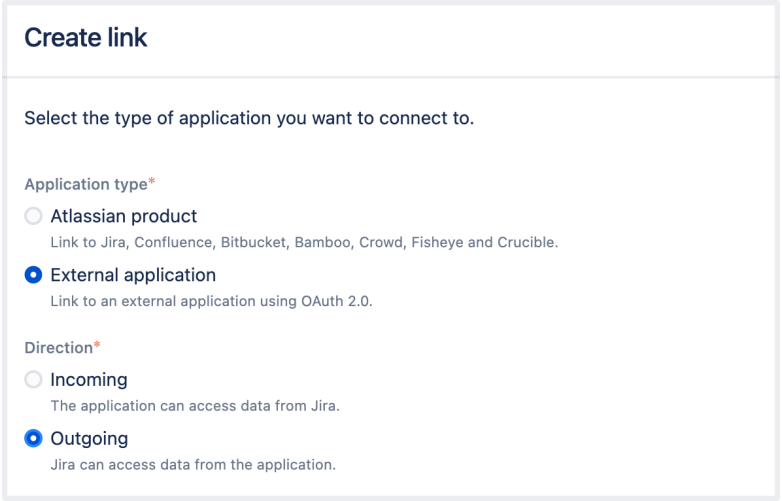
You need to ensure the following:

- Your server needs to run over HTTPS. If it doesn't, you will not be able to configure OAuth 2.0.
- Your base URL needs to be configured correctly. This is important as the redirect URL you'll need to provide is based on Bamboo's base URL.

Steps

To create and configure an outgoing link:

1. From the top navigation bar, select **Administration ( )** > **Overview**.
2. On the **Bamboo administration** page, select **Application links** from the left menu.
3. Select **Create link**.
4. In the **Create link** dialog, select **External application**, and then choose **Outgoing** as the direction.



5. Choose a service provider:
    - **Google**
    - **Microsoft**
    - **Custom** (for internal tools or other providers)

> Choosing Google or Microsoft lets you create an OAuth 2.0 integration for mail servers – in this case, some of the configuration fields will be pre-filled.

6. Copy the redirect URL and register it in your external application to obtain the client ID and client secret required to complete the configuration.

   If you're using Google or Microsoft as service providers, you'll be able to copy the redirect URL right away. For custom providers, you need to first provide the **Authorization endpoint** and **Token endpoint**. For more information on registering the URL with Google or Microsoft, check out the following guides:

   - OAuth 2.0 in Google
   - OAuth 2.0 in Microsoft

   > Different providers might have different requirements related to the redirect URL. For example, Google doesn't allow it to be a private IP address. Make sure you provide an external URL (for example, of a load balancer for Bamboo Data Center).

7. Fill-in the remaining configuration details:

Name	Description
Client ID	The client ID that's generated by the external application after registering Bamboo's redirect URL. This is the public identifier of the application.
Client secret	The client secret that's generated by the external application after registering Bamboo's redirect URL. This is the shared secret between Bamboo and the application, which ensures the authorization is secure.

Scopes	The required OAuth 2.0 scopes (permissions) that control what Bamboo can do in the external application. You need to specify different scopes for email servers.
	For Google, we recommend this scope: `https://mail.google.com` (for IMAP, POP3, and SMTP).
	
	For Microsoft, we recommend that you always use the `offline_access` scope and at least one additional scope, depending on what protocol you want to use. The scopes will vary depending on your Microsoft account type and the mail protocol type:
	• If you're using non-GCC (Government Community Cloud) accounts, we recommend the following scopes:
	`https://outlook.office.com/IMAP.AccessAsUser.All (for IMAP)`
	`https://outlook.office.com/POP.AccessAsUser.All (for POP3)`
	`https://outlook.office.com/SMTP.Send (for SMTP)`
	`offline_access`
	• For GCC accounts, use:
	`https://outlook.office365.com/IMAP.AccessAsUser.All (for IMAP)`
	`https://outlook.office365.com/POP.AccessAsUser.All (for POP3)`
	`https://outlook.office365.com/SMTP.Send (for SMTP)`
	`offline_access`
	
	For more information about scopes available in Google and Microsoft, see the detailed information at the Microsoft & Google sites.

Authorization endpoint	The HTTPS URL where authorization to use OAuth 2.0 is started.

Token endpoint	The HTTPS URL where refresh token requests are sent. As OAuth 2.0 tokens have an expiry, Bamboo will periodically update the token.
Redirect URL	The redirect URL that must be registered in the external application to obtain its client ID and client secret. This redirects the authentication flow back to Bamboo.

8. Save the link.

Troubleshooting

If you're facing some issues while configuring outgoing links for applications, check out our Application Links Troubleshooting Guide.

Integrating Bamboo with JIRA applications

Integrating Bamboo with Atlassian's Jira applications combines Bamboo's continuous integration capabilities with your issue tracker to give you a unified view of your software development project.

Configuring Bamboo and Jira applications to work together simply requires you to set up an application link (two-way) between a Jira application and Bamboo.

Note that application links have nothing to do with using a Jira application as a user directory for Bamboo; these 2 configurations can exist separately. See also Linking to another application.

On this page:

Benefits

See Viewing Bamboo activity in a Jira application for a full description of the benefits of integrating Jira applications with Bamboo.

Briefly, these are:

Integration feature	Description	Compatibility	
		Jira	Bamboo
Development info in issue search	See Jira 7.7 release notes	7.7 +	5.4 +
Development panel	See Bamboo development activity	6.2 +	5.4 +
Build result dialog	View detailed Bamboo build result information	6.2 +	5.4 +
Deployment dialog	View detailed Bamboo deployment information	6.2 +	5.4 +
Run Bamboo builds	Run a Bamboo build when releasing a Jira application version	4.4 +	3.0 +
Linked plan branches	Have Bamboo automatically link a plan branch with an issue	4.4 +	3.0 +
Jira issues view	View the issues linked to a build result	4.4 +	3.0 +
Jira Projects	View the Bamboo builds that relate to a Jira application project or version	4.4 +	3.0 +

If you are using an earlier version of Bamboo and/or a Jira application, you can also download an older version of the Jira Bamboo plugin from the Atlassian Plugin Exchange. However, we strongly advise you to upgrade Jira to version 6.2 or later and Bamboo to version 5.4 or later, to get the most out of Bamboo - Jira applications integration.

Requirements

Requirement	Description
Jira 6.2+	Earlier versions of Jira do not support the Development panel feature.
Bamboo 5.4+	
Jira/Bamboo Applink	See below for details about application links.
Permissions	Users will require the View development tools permission in Jira applications.
Issue key	The issue key must be included in the commit message, and must use the default issue key format.

Configuration

Integration of Bamboo and a Jira application requires an application link between them. The application link needs both 2-legged (2LO) and 3-legged OAuth (3LO) authentication:

- 2LO is required for information from Bamboo to be included in the summaries in the Development panel.
- 3LO checks that a user has authenticated with Bamboo before they get to see the information in any of the details dialogs. Users who can see summarized data in the Development panel may not have permission to see all the information that contributed to those summaries and which is visible in the details dialogs. That is, the details dialogs respect the access permissions that users have in the connected Bamboo server.

When you create a new link between a Jira application and Bamboo, both 2-legged (2LO) and 3-legged OAuth (3LO) are enabled by default.

- You will need to set up a two-way link. That is, select the **Create a link back to this server** option when adding the application link.

An existing application link between Jira and Bamboo may need to have 2LO authentication explicitly enabled.

An existing application link between a Jira application and Bamboo (that perhaps used Trusted Apps authentication) needs to have 2-legged authentication (2LO) enabled for both outgoing and incoming authentication, so that information from the application can be included in the Development panel summaries.

When updating an older application link to use OAuth, 3-legged authentication is applied by default, but you need to explicitly enable 2LO. Enable 2-legged authentication for the application link from within Jira as follows:

1. Go to the Jira admin area and select **Applications** > **Application Links**.
2. Select **Edit** for the app link with the other application.
3. For both **Outgoing Authentication** and **Incoming Authentication**:
    a. Select **OAuth**
    b. Check **Allow 2-legged OAuth**.
    c. Select **Update**.

The application link update process will involve logging you into the other application for a short time to configure that end of the link, before returning you to Jira application.

Note that:

- Application links must have Trusted Applications and Basic Access authentication disabled. The Development panel in Jira only supports OAuth authentication.
- You will need to configure 2-legged OAuth enabled for both incoming and outgoing authentication in both Jira application and Bamboo for your application link.
- See Configuring authentication for an application link for more information.
- If you are running Bamboo behind a proxy, you may need to configure the AJP connector.

Notes

Known issues

- Jira applications and Bamboo cannot run in the same Tomcat instance due to a known issue with the Bamboo plugin for Jira applications (see JRA-19662).
- When integrating Bamboo with a Jira application, you should not change the Jira application project key format from the default, as Bamboo only supports the default project key format.

If you need further help, please raise a support request in our support system, in the Bamboo project. You may also want to view articles in the Bamboo Knowledge Base and browse our forums.

Viewing linked Jira application issues

If your organization uses Atlassian's Jira application and your administrator has integrated Bamboo with a Jira application, you will be able to view the issues that have been linked to a build. This provides an easy way to jump to relevant issues in the Jira application to see details about what the code is intended to achieve.

Linked issues can be viewed on:

- the **Issues** tab of the Plan summary page, for all issues linked to the plan
- the Build result summary page, for just two of the issues linked to a build
- the **Issues** tab of the Build result summary page, for issues linked to a build.

Issue links can be created automatically by Bamboo when you specify an issue key in your build comments, labe , or commit messages, or they can be added manually.

On this page:

Related pages:

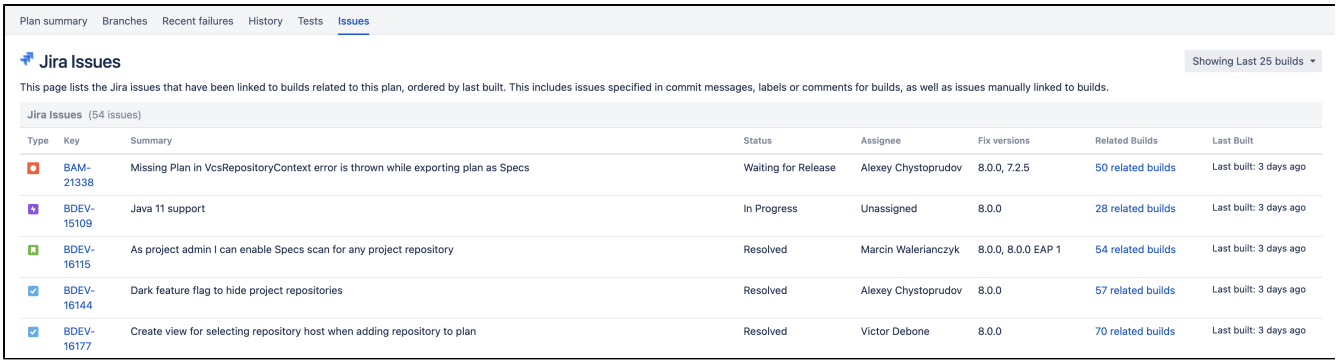
- Creating Jira issues from a build
- Linking Jira application issues to a build

Viewing the Jira issues linked to a plan's builds

To view the issues linked to all builds for a plan:

1. Navigate to the desired plan, as described on Configuring plans.
2. Select the **Issues** tab. A list of all of the issues linked to builds for the plan are displayed, sorted by build date. You can constrain the list using the build filter (e.g. Showing last 25 builds) next to the tabs.

- Select the issue key to view the issue in the Jira application.
- Select the **N related builds** link (where N is a number of builds) to view the builds linked to that issue on the **Builds** tab in the Jira application.



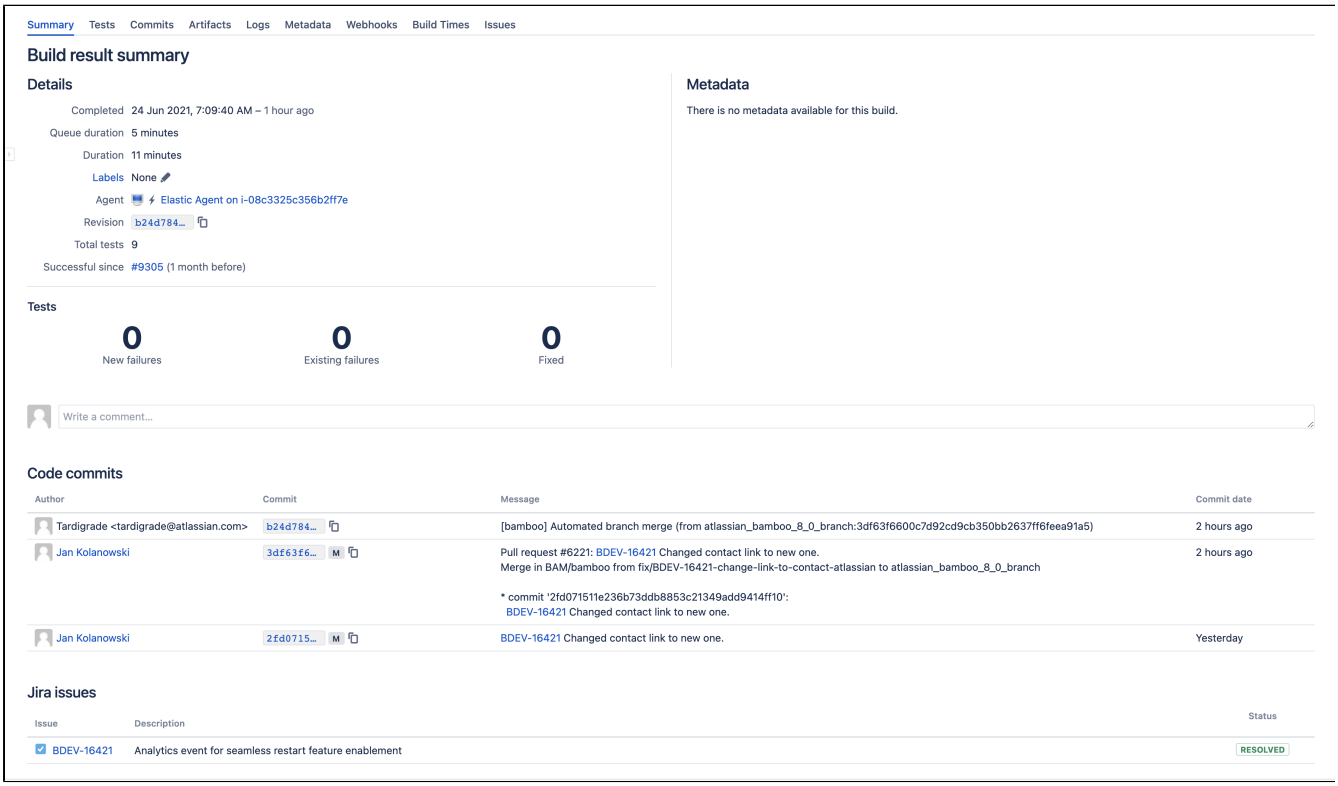
Viewing issues for a build result

To view the issues linked to a particular build result:

1. Navigate to the build results for the plan, as described in Viewing a build result.
2. Select the build number for the desired build result.

- **Build summary** tab — the Jira issues section displays up to two of the issues linked to the build.

- **Issues** tab — displays all of the Jira issues linked to the build. Select **Add linked issue** to link this build to an issue in a Jira application.



Linking Jira application issues to a build

If your organization uses Atlassian's Jira and your administrator has integrated Bamboo with Jira:

- Bamboo will automatically link Jira issues to builds.
- You can manually link an issue to a build.

Automatically linking issues to a build

Bamboo will automatically link an issue to a build if you specify a Jira issue key in a Bamboo build comment or label, or in a code commit message.

The issue key must be of the default Jira issue key format (that is, two or more uppercase letters (`[A-Z][A-Z]+`), followed by a hyphen and the issue number, for example BAM-123).

On this page:

Related pages:

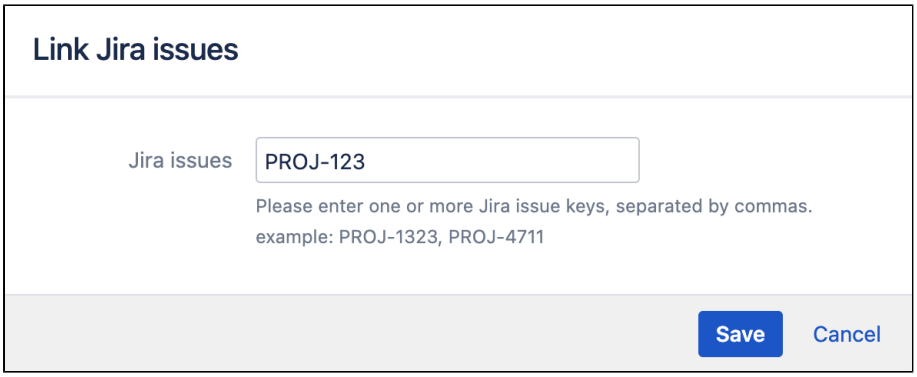
- Creating Jira application issues from a build
- Integrating Bamboo with JIRA applications

Manually linking issues to a build

If an issue has not been linked automatically to your build, you can manually create a link from that issue to your build.

To manually link a Jira Issue to a build result:

1. Go to the plan in Bamboo.
2. Select the build number for a build result.
3. Select the **Issues** tab. All of the Jira issues linked to your build will be listed.
4. Select **Add linked issues**.
5. Enter the Jira issue key of the issue you want to link to this build. Please note, the issue key must be of the default Jira issue key format (that is, two or more uppercase letters (`[A-Z][A-Z]+`), followed by a hyphen and the issue number, for example BAM-123).
6. Select **Save**.



Creating Jira application issues from a build

When Bamboo is integrated with Jira Software Server, you can create new issues right from your Bamboo build result. You can easily:

- *Capture* critical infrastructure failures that are keeping your build from passing.
- *Request* that a successful build be deployed to the next environment.
- *Create* a searchable knowledge base of failure causes and solutions.
- *Log* time spent on build failures and use Jira Software dashboard gadgets to discover trends over time.

When you create an issue from Bamboo, the issue in Jira Software links back to the build result it was created from.

A link to the new issue is displayed in the Jira issues section of the **Summary**, and on the **Issues** tab, in Bamboo.

To take advantage of Jira Software issue creation in Bamboo:

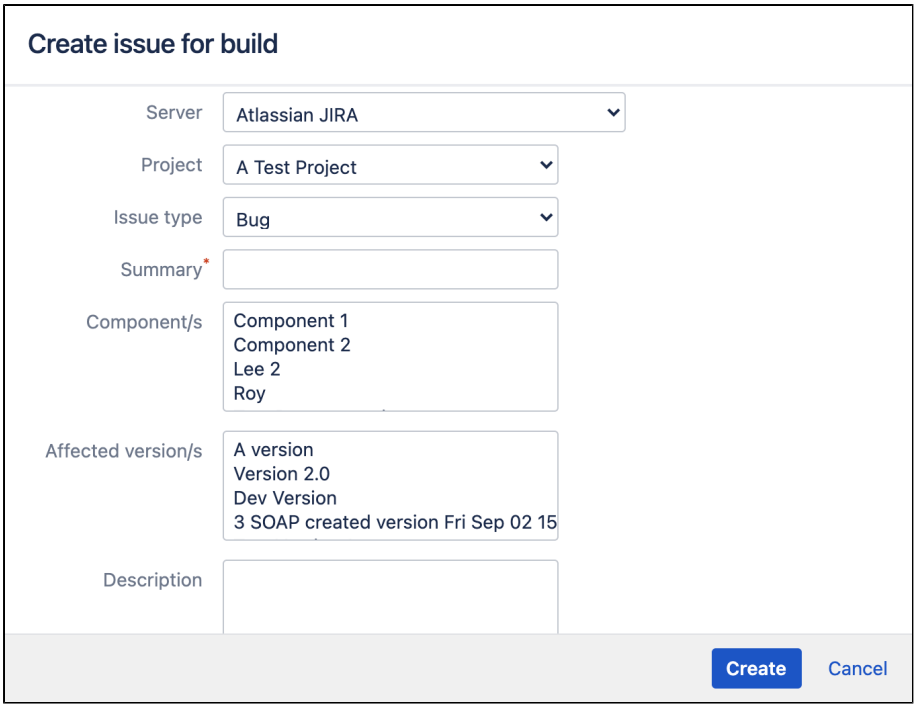
- You require Jira 5.0, or higher.
- There must be an application link already set up between Jira Software and Bamboo.
- Your Jira application administrator needs to have enabled fully reciprocal issue linking in Jira Software Server.

Related pages:

- Linking Jira application issues to a build

To create a new Jira Software issue from a Bamboo build:

1. Go to the desired build, and select **Actions** > **Create issue.**
2. Complete the form.
3. Select **Create**.



Viewing Bamboo activity in Jira applications

Overview

When Bamboo is integrated with Jira applications, Bamboo can pass important development information back to a Jira application. Currently Bamboo can pass Jira application information relating to:

- Build results
- Deployment statuses

With supported versions of Bamboo and a Jira application, this information is collated within the Development Panel. This panel summarizes the status of all work related to an issue, and can assist in identifying where an issue's build is failing, and where it has been deployed.

Example

- If you are working on issue BAM-12443, then you can see if it has been deployed to a development server yet
- A QA can also check to see if it's on their QA server and ready for testing
- A manager can see if a bug has made it to production.

Linking Jira applications and Bamboo has the benefit of improved information exchange during your development process.

See Integrating Bamboo with JIRA applications for information about permissions and configuration.

Related pages:

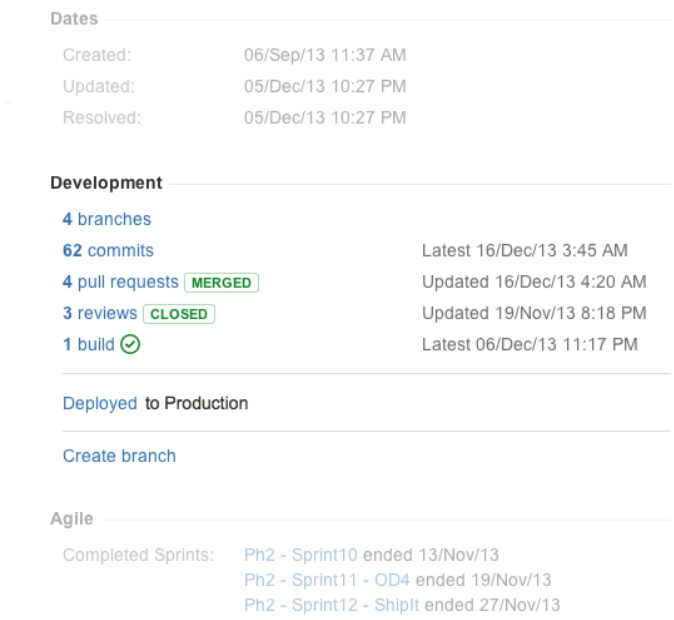
- Linking issues to a build

On this page:

The Development panel

The Development panel provides an at-a-glance development information resource, and is visible to anyone with the View development tools project permission. The panel replaces the Builds tab and Issue deployment panel, and summarizes an assortment of development data passed to Jira applications from Bamboo and other Atlassian products. Examples include:

- Feature branch creation from Jira applications
- Viewing repository branches in Bitbucket Cloud or Bitbucket Server
- Viewing commits and pull requests to Git repositories managed by Bitbucket Cloud or Bitbucket Server
- Viewing commits, branches and reviews in Fisheye/Crucible
- Viewing build result and deployment information in Bamboo



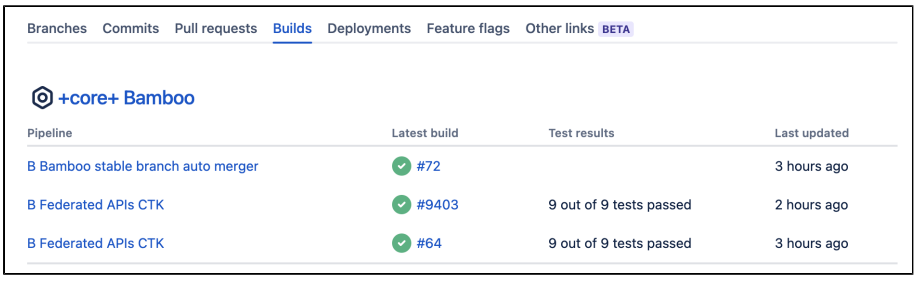
Viewing build result information

The Development panel shows the status of the latest Bamboo builds related to your linked issue. Using simple status icons builds are reported as:

- all the different builds (for example, unit tests, functional tests, deploy to staging) succeeded.
- at least one run failed for any build by any linked instance of Bamboo.

A build is automatically linked to an issue if one of the build's commits includes the issue key in its commit message. The issue key must be included in the commit to activate this feature.

Select the associated build link to see additional build details including the name of the plan branch and how many tests passed or failed:

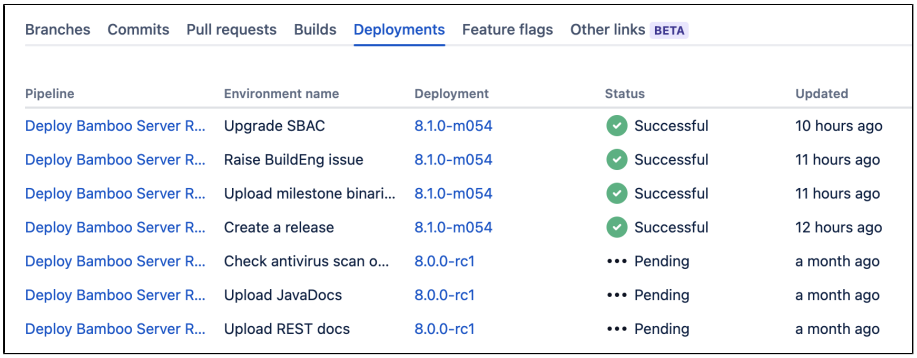


Viewing deployment information

A deployment to an environment, such as Production or Testing, is linked to an issue if a commit associated with the deploy contains the issue key in its commit message. The Development panel details the environments that associated Bamboo builds have been deployed to.

The issue key must be included in the commit to activate this feature.

Select a deployment name to see deployment details including the deployment status, release date, and select to view a particular deployment:



Note the Deployment panel is no longer displayed on an issue when the Development panel is available.

Integrating builds with your issues workflow

You can configure a workflow in Jira applications, so that the workflow is **actioned** when a build completes succ essfully. For example, you can configure a workflow to automatically progress an issue from *Building* to *Resolved* status. You could also configure the same workflow to progress an issue from *Building* to *Build broken* status if a build related to that issue fails. A build is related to an issue if the build involves a commit that had the issue key added to commit message.

On this page:

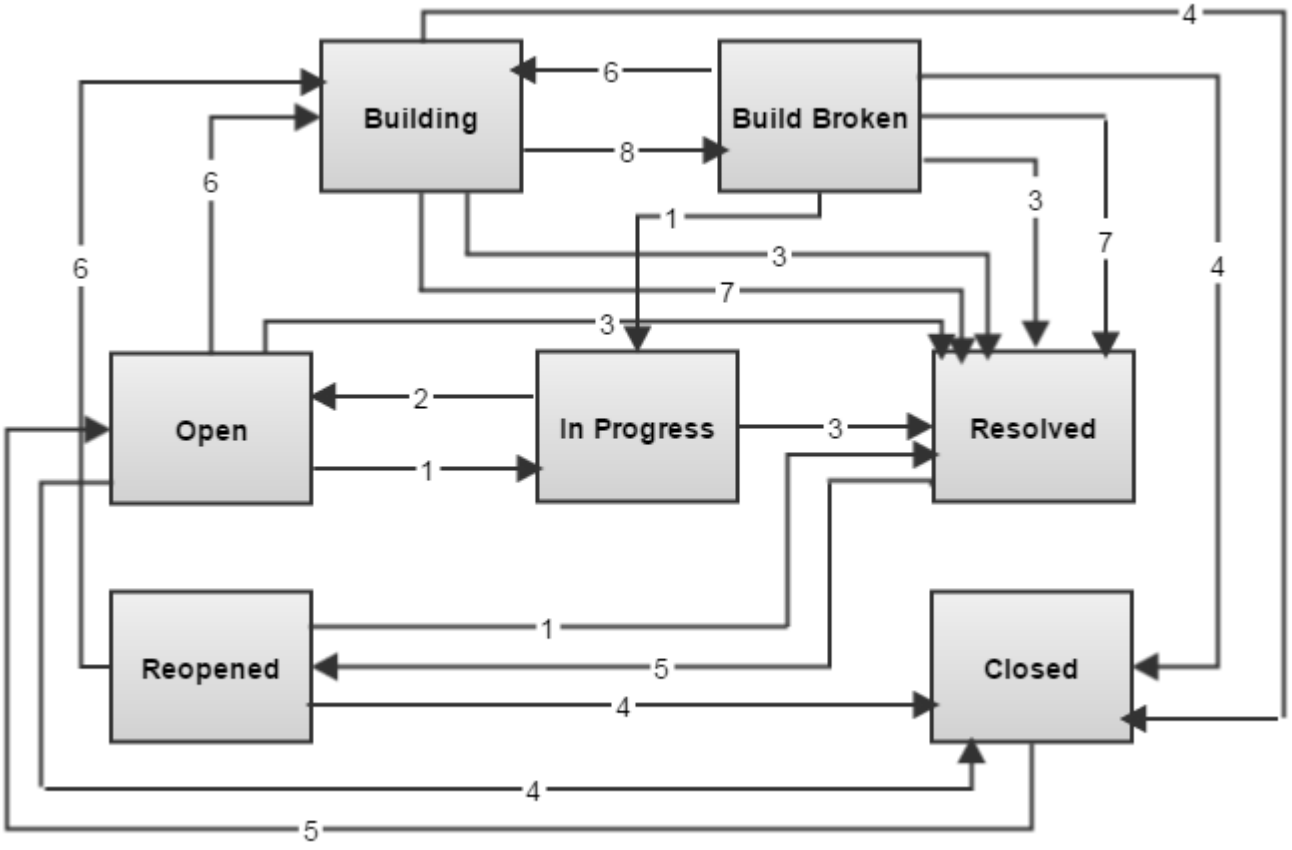
A **Builds workflow** exists in Jira applications, and it incorporates the common statuses and transitions (see the Understanding the Builds workflow section below).

- **If you are new to Jira applications and Bamboo**, we recommend that you use the Builds Workflow as modifying an existing workflow is not a trivial task.
- **If you have an existing workflow that you would like to modify** to include build statuses and transitions, we recommend that you take a copy of the Builds Workflow and modify it.
- **If you want to integrate Bamboo transitions into your existing workflow**, you can edit your workflow to add the transitions. We recommend that you avoid doing unless you have a good understanding of the workflows.

Understanding the 'Builds Workflow'

Diagram: The default Builds Workflow

#	Transition
1	*Start Progress*
2	*Stop Progress*
3	*Resolve Issue*
4	*Close Issue*
5	*Reopen Issue*
6	*Wait for Build*
7	*Build Passed*
8	*Build Failed*



The *Wait for Build*, *Build Passed* and *Build Failed* transitions are Bamboo-specific transitions:

- *Wait for Build* — This transition will be triggered when code is committed for this issue (and a build started) using the `#wait` or `#wait-for-build` commit command. Note, you must manually enter the commit command in your commit message to trigger the transition, as described in Using Smart Commits.
- *Build Passed* — This transition will be automatically triggered when a build for this issue passes.
- *Build Failed* — This transition will be automatically triggered when a build for this issue fails.

Using the Builds Workflow in your projects

The following instructions describe how to create a workflow scheme that uses the Builds Workflow, and then associate the workflow scheme with a project. If you want to add the Builds Workflow to an existing workflow scheme, ignore steps 4-6 below and assign the workflow to your existing workflow scheme instead.

Procedure

1. Creating a workflow scheme that uses the Builds Workflow

1. Log in as an admin for your site.
2. In the administration console of a Jira application, go to **Workflows** > **Workflow schemes**. The Workflow schemes page will display.
3. Select **Add workflow scheme**.
4. Enter a **Name** and **Description** for your workflow scheme and select **Add**. Your workflow scheme will be created and you will see the page for editing the workflow.
5. Select **Assign a workflow**.
6. In the **Issue type** dropdown, select the issue types that you want the Builds Workflow to apply to. In the **Workflow** dropdown list, select **Builds Workflow**. Select **Add.**

2. Associating the workflow scheme with your project

1. Log in as a user with the Jira application Administrators global permission.
2. Go to the Project summary page. Keyboard shortcut: **g + g +** start typing "projects".

3. Select **Workflows** on the left of the Project summary page (you can also select the **More** link in the Workflows section in the middle of the screen). The Workflows page is displayed, indicating the current workflow scheme used by the project.
4. Select **Switch scheme** to display the Associate Workflow Scheme to Project page.
5. Select the relevant workflow scheme from the **Scheme** list, and select **Associate** to begin the migration process. The Builds Workflow will be associated with your project via your workflow scheme.
6. Select **Acknowledge** to finish the process.
7. Select the project you wish to use the Builds Workflow with.

Issues (of the issue types specified in your workflow scheme) will now use the Builds Workflow. If you add the issue key of an issue to the commit message when committing, the issue will be automatically transitioned along the workflow when the build starts/succeeds/fails.

Modifying the Builds Workflow

You cannot modify the Builds Workflow itself because it is non-editable. However, you can copy it and edit the copy if the original Builds Workflow doesn't suit the needs of your project. You can then activate the new (copied) workflow by adding it to a workflow scheme and then associating that scheme with your projects.

Copying and editing the Builds Workflow

1. Log in as an admin for your site.
2. In the Jira administration console of a Jira application, go to **Schemes** > **Workflow schemes**. The Workflow schemes page will display.
3. Select **View all workflows**.
4. Locate the Builds Workflow and select **Copy** in the **Operations** column.
5. Enter a **Name** and **Description** for the new (copied) workflow scheme and select **Copy**. The new workflow will be created and displayed on the View Workflows page.
6. You can now edit and activate your new workflow as needed. See Configuring Workflow and Activating workflow in the Jira Server Administration documentation for more information on how to do this.

Integrating build transitions into your custom workflow

If modifying a copy of the Builds Workflow is not feasible for your projects, it is possible to manually modify your existing workflow to include the Bamboo transitions. It is recommended that you avoid doing so unless you have a good understanding of the workflows.

To integrate build transitions into your existing custom workflow, edit your workflow and configure appropriate issue statuses and issue transitions as described below.

Configuring your issue statuses

We recommend that you set up issue statuses for your workflow to indicate when a build related to an issue is building or the build is broken (e.g. *Building*, *Build Broken)*. There is no technical restriction preventing you from incorporating Bamboo-specific build transitions into a workflow without these intermediate states, however, in practice it will cause problems.

For example, a developer may work on an issue, and commit several times over the course of a few days for that issue.
Even if earlier commits cause the build to pass, the developer may not have finished working on the issue and will need to commit more code without successful builds resolving the issue. Hence, an intermediate state (e.g. *Building*) is required which a developer will only transition the issue into (i.e. using the `#build` commit command), if they want the issue to be resolved from that particular build.

Configuring your issue transitions

Automatic issue transitioning via builds is controlled by both commit commands and Bamboo-specific transition properties in Jira applications, as described below:

- **Commit command** — Commit commands are mapped to transition names. Hence, if you add the *Waiting for Build* transition to your workflow, your users will be able to automatically trigger the transition by using the `#wait` or `#wait-for-build` commit command in their commit messages.
- **Bamboo-specific transition properties**— The Bamboo-specific transition properties on the transitions that you want to be triggered when a Bamboo build passes or fails. The following properties are supported:

Property	Value	Description
`build.passed.transition`	*anything*	A transition with this property will be triggered when a build for this issue passes, and the transition is available to the issue in its current state.
`build.failed.transition`	*anything*	A transition with this property will be triggered when a build for this issue fails, and the transition is available to the issue in its current state.
`build.passed.resolution`	*any valid resolution, e.g. Fixed*	The issue resolution will be set as specified by this property, if the transition with this property is triggered by a build.

Please note, you cannot set up common transition properties in Jira applications. You will need to manually re-enter the transition property on each transition that you want it added to.

Integrating Bamboo with Confluence

Integrating Bamboo with Atlassian's Confluence combines Bamboo's continuous integration capabilities with your wiki to give you a unified view of your software development project.

When Bamboo is integrated with Confluence, you can add the following Bamboo gadgets to a Confluence wiki page:

- Bamboo Charts
- Bamboo Plan Summary Chart
- Bamboo Plan Status

Configuring Bamboo and Confluence to work together simply requires you to set up an application link (two-way) between Confluence and Bamboo.

On this page:

Related pages:

- Registering External Gadgets (Confluence documentation)
- Linking to Another Application

Before you begin

Version Requirements

Application	Version Requirement
Bamboo	Version 5.2 or later
Confluence	Version 3.5.9 or later

Set up an application link

Before you begin:

- Security Considerations — The instructions below recommend setting up authentication for the application link between Confluence and Bamboo. Please ensure that you read the Security implications for each authentication type (Applinks documentation). For example, if you use basic HTTP authentication for the Confluence to Bamboo link, you must specify a user that Confluence uses to log in to Bamboo. Hence, this user's Bamboo permissions will be used (not the Bamboo permissions of the user who is currently logged into Confluence).

Follow the Linking to Another Application instructions to configure the application link in Confluence.

- You will need to set up a two-way link, i.e. select the Create a link back to this server option when adding the application link.
- You will need to configure either OAuth or Trusted Apps authentication for your application link. See Linking to Another Application for instructions.

Congratulations! You have successfully integrated Bamboo and Confluence.

Try your new configuration

You may wish to read about how to use these two applications together in the following pages:

- Add Bamboo gadgets to Confluence, see Registering External Gadgets (Confluence documentation).

Notes

If you need further help, please raise a support request in our support system, in the Bamboo project. You may also want to view articles in the Bamboo Knowledge Base and browse our forums.

Integrating Bamboo with Fisheye

When Bamboo is integrated with Atlassian's Fisheye, you can:

- view the code changes that triggered a build
- explore a failed build in Fisheye and jump directly into the changeset that broke the build
- view the history of the changeset to see what the author was trying to fix
- analyze the change using the side-by-side diff view
- open the relevant files in your IDE.

A Bamboo administrator can make links to individual source-code files available by connecting the plan to the source repository, as described below.

You can specify repositories at the following levels in Bamboo:

- global – repositories are available to all plans in Bamboo.
- plan – repositories are available to all jobs in the Bamboo plan.
- job – repositories are available to all tasks in the Bamboo job.

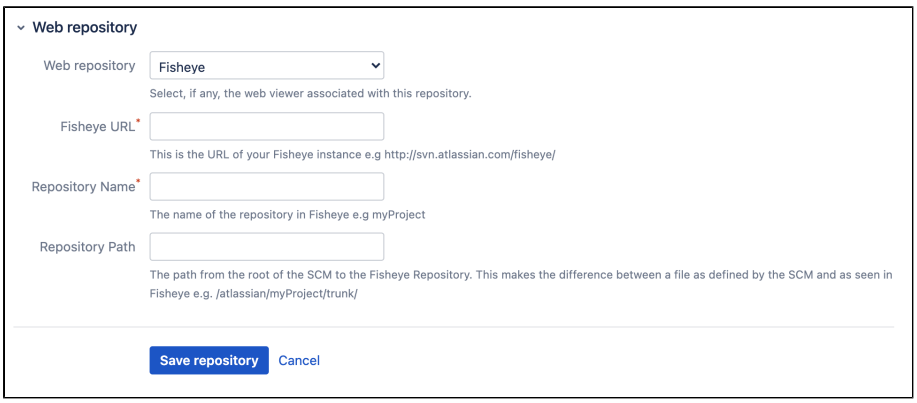
The recommended approach is to set up linked source repositories at the global level – see Linking to source code repositories.

Related pages:

- Integrating Bamboo with Atlassian applications
- Linking to Another Application

To integrate Bamboo with Fisheye:

1. Navigate to the repository configuration for a linked repository, plan or job. See Linking to source code repositories.
2. Select a repository name, and select **Web repository** > **Fisheye**.
3. Specify the **Fisheye URL**, **Repository name** and **Repository path**.



⚠️ *If links to Fisheye are broken in Bamboo builds, make sure that the Repository Path configured in Bamboo matches the Repository Location (under **SCM Details**) in Fisheye, for the specific repository.*

Integrating Bamboo with Bitbucket Data Center

When you integrate Bamboo with Atlassian's Bitbucket Data Center Git repository management solution, commit, branch, build, and deployment information is shared for users of both applications.

On this page:

Benefits of integration

When Bamboo and Bitbucket Data Center are integrated, you and your team get all the following advantages:

Bitbucket Data Center tells Bamboo when to build

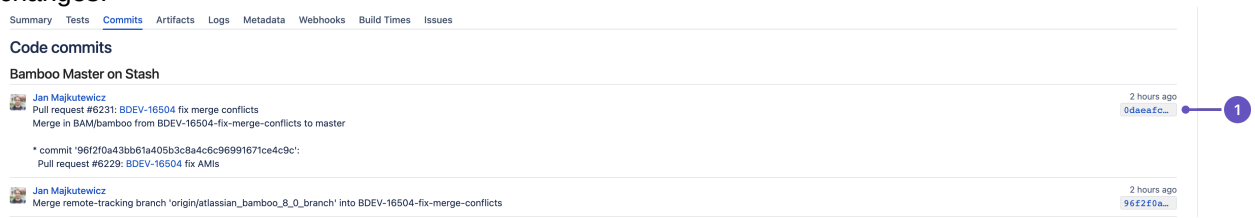
- When a developer pushes to a repository the build is automatically started.

Bitbucket Data Center tells Bamboo when to update plan branches to match changes in repository branches

- When a developer pushes a new branch to a repository a branch plan is automatically created.
- When a developer deletes a branch in a repository, the branch plan is automatically deleted or disabled.

Bitbucket Data Center commits are displayed in the relevant Bamboo builds

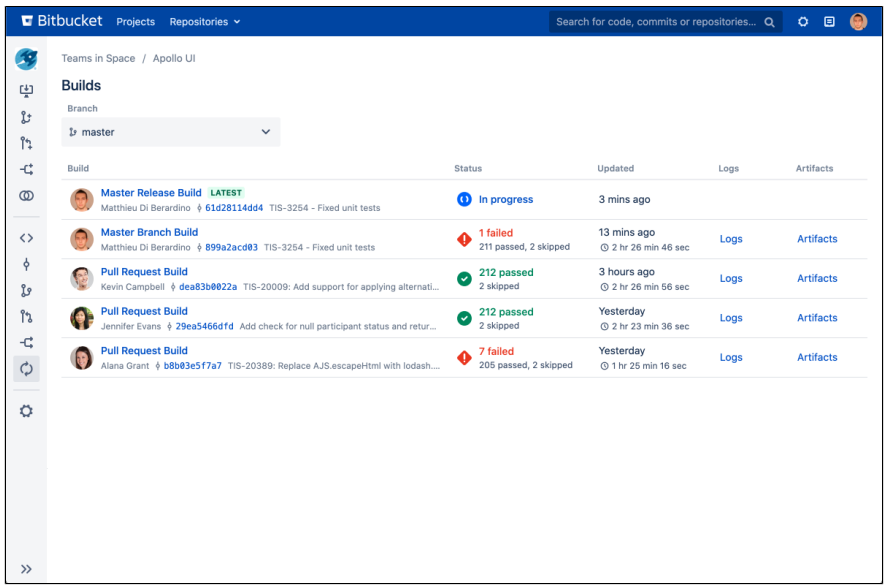
- In Bamboo, you can view all of the commits involved in the build, allowing you to accurately track changes.



1. **Commit changeset**: select a changeset to go to Bitbucket Data Center, where you can see the commit diff for all of the files that are part of the build.

Bamboo notifies Bitbucket Data Center automatically about build results

- Build notifications are automatically enabled when you link a build plan to a Bitbucket Data Center repository.
- Notifications are sent to all linked Bitbucket Data Center servers.
- You can see build results and other related information on the Builds, Pull request, Commits, and Branches pages so you can easily check the build status of a branch when deciding whether to merge change.



Bitbucket Data Center displays the overall status of the build results. The status is *passed* if all the different builds (for example, unit tests, functional tests, deploy to staging) have succeeded, and *failed* if at least one run failed for any of those.

For example, when viewing the Commits tab for a Bitbucket Data Center project, you will see icons that indicate the status of the latest build results. The red fail icon is displayed if there is at least one failed build run for the commit.

Note that the legacy Bitbucket Data Center notification type is deprecated – it is still available in Bamboo 5.6 but will be removed in Bamboo 5.7.

Bamboo provides support for Pull Request

Starting from version 6.0, Bamboo can create plan branches by pull requests. Create a pull request when ready to share your work with teammates and the CI system. Bamboo will detect new pull requests and create plan branch.

Note that Bamboo doesn't provide pull request support for forked repositories yet.

Configuration

There are just a few simple configuration steps to get the integrations described above with Bamboo (versions 5.6 and later) and
Bitbucket Data Center.

Bamboo will be automatically configured to respond to repository events published by Bitbucket Data Center, and to notify Bitbucket Data Center about build results – you don't have to configure repository polling for new commits anymore in Bamboo, or set up dedicated web hooks in your Bitbucket Data Center instance.

1. Create an Application link

You only need to do this once for each pair of Bitbucket Data Center and Bamboo instances.

See Linking to another application.

Once linked, all the Bitbucket Data Center repositories are available to your plans in Bamboo.

2. Choose the Bitbucket Data Center repository for the Bamboo plan

Create a build plan (if necessary) and specify the repository in the plan (or job) configuration.

To connect to a Bitbucket Data Center repository, select **Bitbucket Data Center/Stash** and provide the Bitbucket Data Center details.

You must enable SSH access on Bitbucket Data Center. Otherwise, the integration features won't work and you will have to provide an alternative HTTP repository type to connect to the Bitbucket Data Center repository.

BAM-15464 - Provide HTTP(S) authentication method option for Bitbucket Server type repository
GATHERING INTEREST

See Bitbucket Data Center for more information about using Bitbucket Data Center source repositories in Bamboo.

3. Build!

Managing your user profile

You can manage your user details, password, notifications preferences, and other preferences using your user profile.

To change your personal details:

1. From the top navigation bar select your avatar, then select **Profile**.
2. Select **Edit profile**.
3. Update your personal details as required.

Note that if your user profile is managed using a single sign-on application, like Atlassian's Crowd, you will only be able to edit your **Instant Messaging address** and **Repository aliases**.

Related pages:

- Changing your password
- Changing your notification preferences
- Associating your author name with your user profile

Changing your password

To change your Bamboo password:

1. From the top navigation bar select your avatar, and select **Profile**.
2. Select **Change password**.
3. Complete the form.

If your password is managed via a single sign-on application, like Atlassian's Crowd, this function will not be available.

Changing your notification preferences

Notifications in Bamboo are triggered by a range of events for a plan, including build completion, build outcomes and comments being posted against build results.You can choose whether to receive notifications by email, IM, both, or neither.

You can see which notifications are currently applicable to you, in your user profile: from the top navigation bar select your avatar, and select **Profile** > **Notifications**.

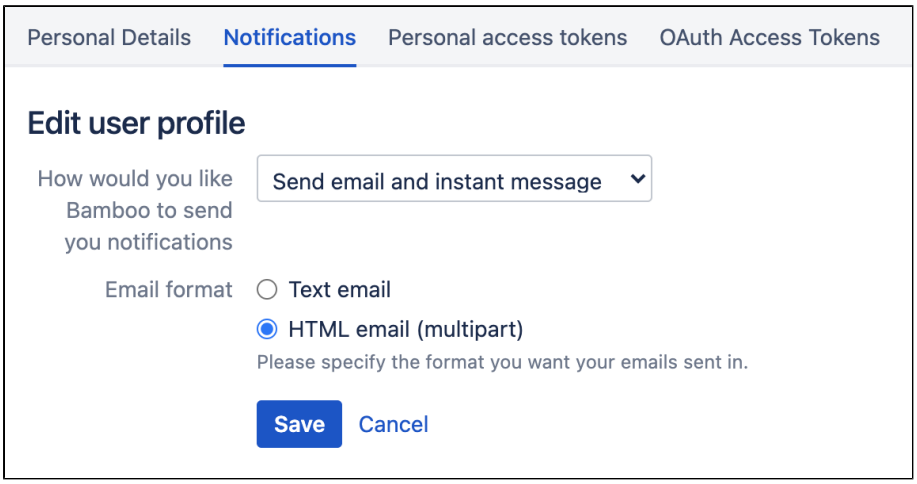
You must have the Edit permission for a plan to add or remove notifications for it.

Related pages:

- Configuring notifications for a plan and its jobs
- Managing your user profile

To change your notification preferences:

1. From the top navigation bar select your avatar, then select **Profile** > **Notifications** > **Edit notification preferences**.
2. Select an option from **How would you like Bamboo to send you notifications**. If you select one of the IM options, you also need to specify an **Instant messaging address** on the **Personal details** tab.
3. Select an **Email format** option.
4. Select **Save**.



Associating your author name with your user profile

An *author* is any person who checks in code to a repository that is associated with a Bamboo plan. An author need not be a Bamboo user.
Your *Author name* is your login name for the source-code repository. This is the identity that the SCM associates with tasks you perform on the repository. However, if this is not the login you use for Bamboo, then Bamboo may not be able to make the connection between your SCM login and your Bamboo login. See also Managing authors.

When your Bamboo user profile is associated with your author name, then Bamboo is able to:

- match your SCM activity with your Bamboo activity.
- show information about your recent builds on your My Bamboo page.
- show a **User details** tab in your Author information.

To associate your author name with your user profile:

1. From the top navigation bar select your avatar, then select **Profile** > **Edit profile**.
2. Select your author name from the **Repository aliases** list. If your name does not appear in the list, select **Add alias**. Note that your author name (alias) shouldn't be identical to your user name.
3. Select **Save**.

You can link more than one author name to a Bamboo user name.

Related pages:

- Managing your user profile
- Managing authors

Bamboo variables

Variables can be used to make values available when building plans in Bamboo.

- **Build-specific variables** are evaluated by Bamboo dynamically at build time. The source of a build-specific variable can either be a Bamboo property or one of the default plugins (assuming it is enabled).
- **Deployment variables** are available when deploying a project.
- **System variables** also apply across your entire Bamboo instance and inherit their values from system or environment variables of the same name.
- **Global variables** are defined across your entire Bamboo instance, and have the same (static) value for every plan that is built by Bamboo. See Defining global variables.
- **Project variables** are defined for specific projects. Project variables can override global variables with the same name. See Defining project variables.
- **Plan variables** are similar to global variables, but are defined for specific plans. Plan variables override global and project variables with the same name. You can also override a plan variable for a build if you trigger the build manually. See Defining plan variables.

Using variables

Variables can be used in all fields of a task or deployment, with the exception of password fields. Use the following format when referencing a variable:

```
${bamboo.variableName}
```

You can override a plan variable for a build, if you trigger the build manually. See Triggering a plan build manually.

You can reference a variable from another variables, e.g. consider having the following variables:

- `var1 = Hello`
- `var2 = world`

You can create another variable which references the two other one.

greet

```
${bamboo.var1} ${bamboo.var2}!
```

Bamboo will resolve the variable as *Hello World!*

You can:

- reference a global or context specific variable in a build plan or deployment project
- reference a variable which references another one, deep recursion is allowed

There are couple of limitations:

- referencing a variable which isn't defined is an error, whole build or deployment will fail if you reference such variable
- cycles are not allowed and are considered as build or deployment project error.

Defining custom variables

You can define your own custom variables, using a similar format to that above, however you cannot create a variable name that is already in use by Bamboo.

For information on how to define your own variables in Bamboo, see:

- Defining global variables
- Defining project variables
- Defining plan variables
- Defining deployment environment variables

On this page:

Related pages:

- Running a plan build manually
- Defining deployment environment variables
- Configuring plugins

Build-specific variables

The following build-specific variables are available by default:

- *System variables* apply across your entire Bamboo instance and inherit their values from system or environment variables of the same name.
- In the variable names from the table, <position> is an optional parameter that specifies the position of the repository in the plan's repository list. If omitted, the first repository in the list is used.
- Third-party repository plugins can expose their own variables.

Build-specific variable	Description
bamboo.buildKey	The job key for the current job, in the form PROJECT-PLAN-JOB, e.g. `BAM-MAIN-JOBX`
bamboo.planKey	The key of the current plan, in the form PROJECT-PLAN, e.g. `BAM-MAIN`
bamboo.shortPlanKey	The short key of the current plan (without project part), e.g. `MAIN`
bamboo.shortPlanBranchName	The name of the current plan branch where the job is being run
bamboo.shortJobKey	The short key of the current job (without project and plan parts), e.g. `JOBX`

bamboo.buildResultKey	The result key when this job executes, in the form PROJECT-PLAN-JOB-BUILD e.g. `BAM-BOO-JOB1-8`, where '8' is the build number. For deployment projects this variable will not have the JOB component e.g. `PROJ-TP-6`.
bamboo.buildResultsUrl *or* bamboo.resultsUrl	The URL of the result in Bamboo once the job has finished executing.
bamboo.buildNumber	The Bamboo build number, e.g. `123`
bamboo.buildPlanName	The Bamboo job name e.g. `Some Project name - Some plan name - Some job name`
bamboo.planName	The current plan's name e.g. `Some project name - Some plan name`
bamboo.shortPlanName	The current plan's name without project part, e.g. `Some plan name`
bamboo.shortJobName	The current job's name without project and plan parts, e.g. `Some job name`
bamboo.buildTimeStamp	The time when a job was started in ISO 8601 format (for example, `2010-01-01T01:00:00.000+01:00`)
bamboo.agentId	The ID of the agent that the deployment is executed on.
bamboo.agentWorkingDirectory	The path to the working directory on the agent, for example `<HOME>/xml-data/build-dir`. The agent working directory is not the same as the build working directory described below.
bamboo.build.working.directory	The working directory on which the build is being executed, for example `<HOME>/xml-data/build-dir/AV-AVT-JOB1.`
bamboo.ManualBuildTriggerReason.userName	The user who triggered the manual build.
bamboo.triggerReason.key	The trigger key that caused the build to launch; for example: `com.atlassian.bamboo.plugin.system.triggerReason:ManualBuildTriggerReason`
Generic repository variables	
bamboo.planRepository.<position>.branchName	The name of the branch in the repository (depends on availability from the VCS used) e.g. `default`
bamboo.planRepository.<position>.name	The name of of the repository, as shown in the repository for the plan e.g. Mercurial

bamboo.planRepository.<position>.revision	The revision use to build this release. Format depends on the VCS used.
bamboo.planRepository.<position>.previousRevision	The previous revision number (this might not exist, for example for the initial build).
bamboo.planRepository.<position>.type	The type of the repository, as defined by a repository plugin e.g. hg, svn, git
bamboo.repository.pr.key	Pull request key if plan branch was created from pull request
bamboo.repository.pr.sourceBranch	Pull request source branch name if plan branch was created from pull request
bamboo.repository.pr.targetBranch	Pull request destination branch name if plan branch was created from pull request
bamboo.planRepository.branchDisplayName	The name of the branch displayed in the branch details section.
Subversion	
bamboo.planRepository.<position>.username	User name, used for repository authentication.
bamboo.planRepository.<position>.repositoryUrl	The repository URL.
CVS	
bamboo.planRepository.<position>.last.update.time	The last updated timestamp.
bamboo.planRepository.<position>.last.update.time.label	The last updated timestamp to be used as a label for post build result labeling. The spaces in the cvs version string are replaced with '_'.
Perforce	
bamboo.planRepository.<position>.revision.number	The change set number.
bamboo.planRepository.<position>.username	User name, used for repository authentication.

bamboo.planRepository.<position>.port	Port used for repository communication.
bamboo.planRepository.<position>.client	Client used for repository communication.
Git	
bamboo.planRepository.<position>.branch	The branch
bamboo.planRepository.<position>.repositoryUrl	The repository URL
Mercurial	
bamboo.planRepository.<position>.repositoryUrl	The repository URL
bamboo.planRepository.<position>.branch	The branch
bamboo.planRepository.<position>.username	User name, used for repository authentication.

- *System variables* also apply across your entire Bamboo instance and inherit their values from system or environment variables of the same name.
- In the variable names from the table above, <position> is an optional parameter that specifies the position of the repository in the plan's repository list. If omitted, the first repository in the list is used.
- Third-party repository plugins can expose their own variables.

Build dependency variables

The following build dependency variables are also available:

Build-specific variable	Description
bamboo.dependency.parent.#	Allows a child build to query the build key of the triggering parent build, where # represents the position in the build tree - 0 at the top, 1 the following, and so on. The ${bamboo.dependency.parent.0} variable can be viewed in the child plan's metadata tab.
bamboo.dependency.parent.total	The total # of parent builds.

Deployment variables

Bamboo manages a number of standard reserved variables that are available when deploying a project.

Variables later in the following list override the previous ones in case of repeating names:

- global variables
- project variables of the plan linked to the deployment project
- plan variables of the plan linked to the deployment project
- release variables as defined below
- user variables defined at the environment level
- the autogenerated variables in the following table:

Variable	Description
`bamboo.agentId`	The id of the agent that the deployment is executed on.
`bamboo.agentWorkingDirectory`	The path to the working directory on the agent. This is not the same as the Bamboo working directory.
`bamboo.build.working.directory`	The path to the working directory for Bamboo. This is used by both the build plan and the deployment project.
`bamboo.deploy.environment`	The name of the environment that the release is to be deployed to.
`bamboo.deploy.project`	The name of the deployment project.
`bamboo.deploy.rollback`	True if the release being deployed is older than the release being replaced.

`bamboo.deploy.release`
 	
`bamboo.deploy.version`
	The name of the release that is being deployed. Either `.release` or `.version` can be used. Both return the name of the release being deployed.

`bamboo.deploy.release.previous`
 	
`bamboo.deploy.version.previous`
	The name of the release that is being replaced (if available). Either `.release` or `.version` can be used. Both return the name of the release being replaced.

`bamboo.resultsUrl`	The URL to the screen in Bamboo that displays build results.
`bamboo.triggerReason.key`	The trigger key that caused the deployment to launch. For example: `com.atlassian.bamboo.plugin.system.triggerReason:ManualBuildTriggerReason`

You can generate variables of your own, using a similar format, however you cannot create a variable that is already in use by Bamboo. See Defining deployment environment variables for more information.

Releases variables

Bamboo makes the following types of variables available for deployment releases:

- Snapshots of values of global variables.
- Snapshots of values of project variables.

- Snapshots of values of plan variables.
- Snapshots of values of repository variables.
- The autogenerated release variables in the following table:

Variable	Description
bamboo.buildNumber	The build result from which the release is created.
bamboo.buildResultKey	The key of the build result from which the release is created e.g. KUNG-FOO-35
bamboo.planKey	The key of the plan related to the release e.g. KUNG-FOO
bamboo.planName	The name of the plan related to the release e.g. Kung - Foo
bamboo.shortPlanKey	The short key of the plan related to the release (without project part), e.g. `MAIN`
bamboo.shortPlanName	The plan's name without project part, e.g. `Some plan name`

Note that several of the variables in the above table are actually those associated with the build plan. The *snaps hots* mentioned above do not contain *password* variables.

System variables

The usage format for all system variables is:

```
${system.<variable>}
```

For example, if you have a *system variable* `MYPATH=C:\MyPath;` you can use a *Bamboo system variable* `system.MYPATH` which will inherit the same value as the system variable.

In older Bamboo versions using *PATH* in the Environment Variables field (of a Script task) doesn't set the windows PATH variable, whereas using *Path* sets Path and PATH in cmd shell.

Using variables in bash

Bamboo variables are exported as bash shell variables. All full stops (periods) are converted to underscores. For example, the variable `bamboo.my.variable` is `$bamboo_my_variable` in bash. This is related to File Script and Inline Script tasks.

Jira applications variables

Note that these variables can be accessed from a Bamboo build only when that build was triggered by releasing a version in Jira Software Server .

Jira variable	Description
${bamboo.jira.baseUrl}	The URL of your Jira application server.
${bamboo.jira.projectKey}	The key of the triggering Jira application project.
${bamboo.jira.projectName}	The name of the triggering Jira application project.
${bamboo.jira.version}	The release version of the triggering Jira application project.
${bamboo.jira.username}	The username of the user who triggered the release build.

Examples

Maven examples

For example, you may want your Maven 2 version to be determined by Bamboo. In Maven 2 `pom.xml` you may have:

```
...
<groupId>com.atlassian.boo</groupId>
<artifactId>boo-test</artifactId>
<packaging>jar</packaging>
<version>1.1.${bambooBuildNumber}-SNAPSHOT</version>
...
```

You can then specify the following in the **Goal** field of your build plan:

```
clean package -DbambooBuildNumber=${bamboo.buildNumber}
```

When the command runs, Bamboo will replace the `buildNumber` with the actual number (e.g. 1102), which will be passed to the underlying Maven build to use. The command will then produce a jar that looks like this: `boo-test-1.1.1102-SNAPSHOT.jar`.

Ant examples

You can then specify the following in the **Target** field of your build plan:

```
-f build.xml -DbambooBuildNumber=${bamboo.buildNumber}
```

When the command runs, Bamboo will replace the `buildNumber` with the actual number (e.g. 1102), which will be passed to the underlying Ant build to use.

Specifying capabilities as variables

You can also specify a capability to be used in a similar way to a global variable.

The format of the capability should be as follows:

```
${bamboo.capability.<capability_key>}
```

For example,

- Custom

  ```
  ${bamboo.capability.<capability_key>}
  ```

- JDK

  ```
  ${bamboo.capability.system.jdk.<jdk_label>}
  ```

- Builder

  ```
  ${bamboo.capability.system.builder.<builder_type>.<builder_label>}
  e.g. ${bamboo.capability.system.builder.maven.Maven1}
  ```

- Perforce

  ```
  ${bamboo.capability.system.p4Executable}
  ```

If you select a capability, the specific capability key will be contained in the URL.

Please note, the space characters in the URL will be replaced with "+" characters. We recommend that you do not use capability labels with space characters, if you wish to use them as variables. A possible solution for space characters is to format them with "${}" symbols, however, this does not work in all cases.

Using capabilities

Global and Build-Specific Variables can be used in specific fields of your build plan, as specified above. For capabilities,

- **System capabilities** are available to all of these fields, (i.e. global and build-specific).
- **Agent capabilities** (i.e. agent-specific and shared/server capabilities) are available only to the build-specific fields. (i.e. not available to **Repository URL**, **CVS Root** or **Branch name**.)

For example,

If you wanted to specify a system variable, but have it set to different values on each agent, do the following:

- Set the following as a system environment variable field on the **Builder** tab:

```
${bamboo.capability.thatsystemvariable}
```

- Specify the system environment variable as a custom capability on each of your agents, and set to the capability to the different values, as desired.

Deprecated variables

The following variables are deprecated and are subject for removal in future Bamboo releases:

Generic repository variables	
bamboo.repository.revision.number	The revision number. If many repositories are linked to the plan, the variable stores the revision number of the *last* repository in the list.
bamboo.repository.branch.name	The repository branch name (for Bamboo version 4.2 or later). If many repositories are linked to the plan, the variable stores the branch name of the *first* repository in the list.
bamboo.repository.previous.revision.number	The previous revision number (might not exist, for example for the initial build). If many repositories are linked to the plan, the variable stores the previous revision number of the *first* repository in the list.
Subversion	
bamboo.custom.svn.revision.number	The revision number.
bamboo.custom.svn.lastchange.revision.number	The last changed revision number.
bamboo.custom.svn.username	User name used for repository authentication.
bamboo.repository.svn.repositoryUrl	The repository URL.

bamboo.planRepository.<position>.revision.number	The revision number.
bamboo.planRepository.<position>.lastchange.revision.number	The last-changed revision number.
CVS	
bamboo.custom.cvs.last.update.time	The last updated timestamp.
bamboo.custom.cvs.last.update.time.label	The last updated timestamp to be used as a label for post build result labeling. The spaces in the CVS version string are replaced with '_'.
Perforce	
bamboo.custom.p4.revision.number	The change set number.
bamboo.custom.p4.username	User name used for repository authentication.
bamboo.custom.p4.port	Port used for repository communication.
bamboo.custom.p4.client	Client used for repository communication.
Git	
bamboo.repository.git.branch	The branch.
bamboo.repository.git.repositoryUrl	The repository URL.
Mercurial	
bamboo.repository.hg.repositoryUrl	The repository URL.
bamboo.repository.hg.branch	The branch.
bamboo.repository.hg.username	User name, used for repository authentication.

Defining global variables

When configuring a plan, you may want to specify variables to be used in the build process. For details on how variables are used, see Bamboo variables.

Global variables are one type of variable that is available to you. Global variables are defined across your entire Bamboo instance, and have the same value for every plan that is built by Bamboo. If you want to define a variable for a specific plan rather than across all plans, define a plan variable as described in Defining plan variables.

Global variables can be accessed by using `${bamboo.globalVarName}`. Global variables can also be overridden at runtime when running a manual build. For more information, see Running a plan build manually.

Related pages:

- Bamboo variables
- Defining plan variables
- Running a plan build manually

To access the Global variables page:

1. From the top navigation bar select  > **Build resources** > **Global variables**.
2. Add, update, or delete the global variables, as desired:
    - **Add** a new variable once you have entered the key and value for it.
    - Updates to existing rows will be saved as you move between cells in the table.
    - Select the cross icon to delete a variable. Bamboo will ask you to confirm its deletion.

Note that if your new global variable contains the word *password*, then the value field will be automatically encrypted. If you change a variable to include the word *password*, then the value field will change from viewable text to an asterisk string.

Defining plan variables

When configuring a plan, you may want to specify variables to be used in the build process. For details on how variables are used, see Bamboo variables.

Plan variables are one type of variable that is available to you. A plan variable is defined for one specific plan, and has the same value every time that plan is built. If you want to define a variable across all plans rather than a single plan, define a global variable as described in Defining global variables.

Plan variables can be accessed by using ${`bamboo.varName`}. Plan variables can also be overridden at runtime when running a manual build. For more information, see Running a plan build manually.

Related pages:

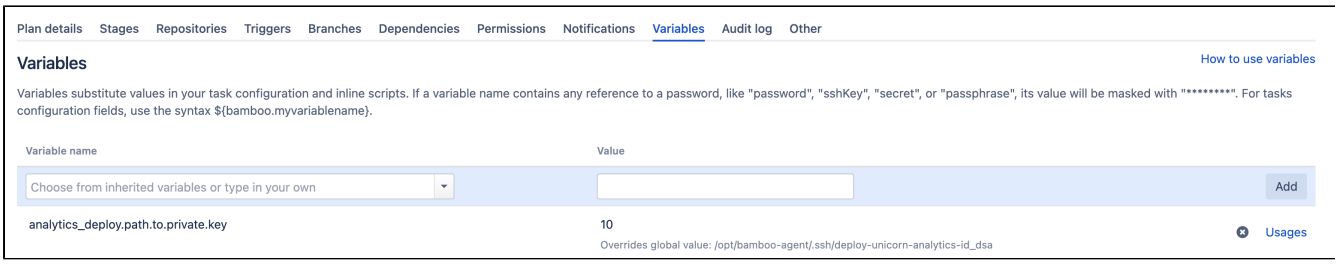
- Bamboo variables
- Defining global variables
- Running a plan build manually

Before you begin:

- Note that plan variables override global variables with the same name.

To define a plan variable:

1. From the top navigation bar select **Build** > **All build plans**, and select the plan you want to edit.
2. Select **Actions** > **Configure plan**.
3. Select the **Variables** tab.
4. Add, update, or delete plan variables, as desired:
    - Select **Add** to add a new variable once you have entered the key and value for it.
    - Updates to existing rows will be saved as you move between cells in the table.
    - Select the cross icon to delete a variable. Bamboo will ask you to confirm deletion.



Passing Bamboo variables to a build script

Bamboo global and build specific variables can be referred to in build scripts or maven pom.xml. Bamboo variables are not directly available in the builder execution context however. They can be passed as parameters to the builder.

Maven

For example, you may want your Maven 2 version to be determined by Bamboo. In Maven 2 `pom.xml` you may have:

```
...
<groupId>com.atlassian.boo</groupId>
<artifactId>boo-test</artifactId>
<packaging>jar</packaging>
<version>1.1.${bambooBuildNumber}-SNAPSHOT</version>
...
```

You can then specify the following in the **Goal** field of your build plan:

```
clean package -DbambooBuildNumber=${bamboo.buildNumber}
```

When the command runs, Bamboo will replace the `buildNumber` with the actual number (e.g. 1102), which will be passed to the underlying Maven build to use. The command will then produce a jar that looks like this: `boo-test-1.1.1102-SNAPSHOT.jar`.

Ant

You can pass Bamboo variables as ant parameters along with ant targets like:

```
clean test -Dbuild.key=${bamboo.buildKey}
```

In your ant build script just refer to this variable:

```
    ...
    <echo message="bamboo.buildKey = ${build.key}/>
    ...
```

Defining project variables

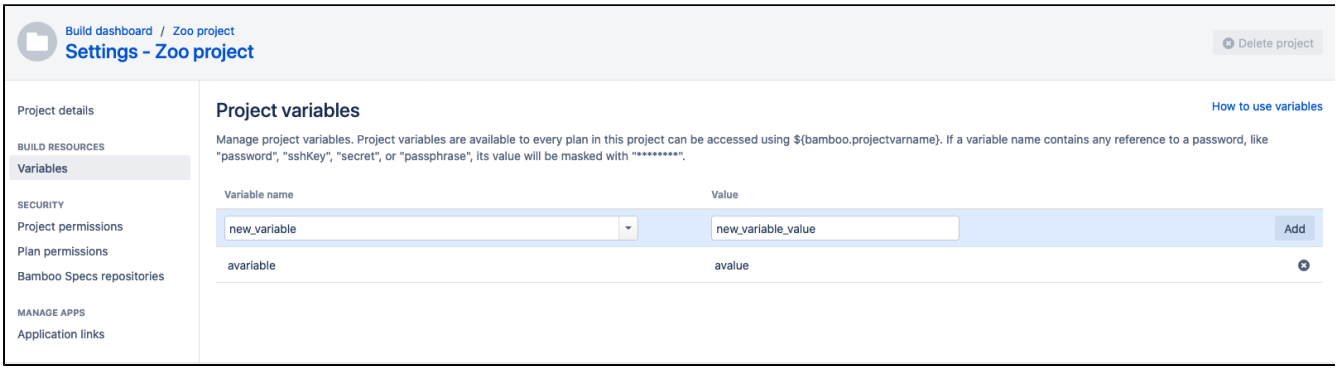
When configuring a plan, you may want to specify variables to be used in the build process. For details on how variables are used, see Bamboo variables.

Project variables are defined for a specific project, and have the same value for every plan that belongs to the project. If you want to define a variable for a specific plan, define a plan variable as described in Defining plan variables.

Project variables can be accessed by using `${bamboo.globalVarName}`. Project variables can also be overridden at runtime when running a manual build. For more information, see Running a plan build manually.

To access the plan variables page:

1. Select a project.
2. Select **Project settings** > **Variables**.
3. Add, update, or delete the plan variables, as desired:
    - Select **Add** to add a new variable once you have entered the key and value for it.
    - Updates to existing rows will be saved as you move between cells in the table.
    - Select the cross icon to delete a variable. Bamboo will ask you to confirm its deletion.



If your new plan variable contains the word *password* then the value field will be automatically encrypted. If you change a variable to include the word p*assword*, then the value field will change from viewable text to an asterisk string.

Bamboo permissions

Bamboo provides the following types of permissions to allow fully customizable control of access to the continuous delivery workflow:

- Global permissions
- Build plan permissions
- Project permissions
- Deployment permissions
  - Deployment project permissions
  - Deployment environment permissions

On this page:

Permissions key:

✅ - Permission is set by default

⭐ - Permission is available as an option

❌ - Permission not available, even as an option

⚠ Starting from Bamboo version 6.2, permissions become additive. Once you're assigned permissions on any level, you'll automatically have permissions on lower levels. You can't override or remove permissions on lower levels. For example, if you have **Create** permission of a global level, you can create plans on all levels. Another example, if you have **Build** permission assigned to you on a project level and none assigned on the plan level explicitly, you will still have build permissions for that plan anyhow.

⚠ Some permissions are available for Bamboo Data Center only. They're marked with DATA CENTER flag.

Global permissions

The Global permissions level controls the ability to view the system, create a new build plan and use administration tools. Global application permissions are accessed from the Global permissions page within the Bamboo administration pages.

User type	Access	Create	Create repository	Restricted admin	Admin
Anonymous	✅	❌	❌	❌	❌
Logged-in	✅	❌	⭐	⭐	⭐
Administrator	✅	✅	✅	⭐	✅

Key:

- **Access** - log in to Bamboo; this permission does not give you any additional permission.
- **Create** - create new plans, projects, and deployment projects in Bamboo.
- **Create repository** - create and manage linked repositories.
- **Restricted admin** - perform some administration operations and view all plans in Bamboo; this role excludes permissions that directly influence the host on which the Bamboo Server is located.

  ⓘ Restricted Admins can't perform agent dedication by default. You can change that by setting the **Allow users to dedicate agents to builds and deployments** option in the Security Settings.

- **Admin** - perform all operations and view all plans in Bamboo.

Build plan permissions

Build plan permissions allow a user to control access to the functions of the build plan. These include viewing, editing, building, cloning, and administering a build plan. Build plan level permissions are accessed from the build plan configuration page.

User	View	Edit	View configuration	Build	Clone	Admin
Anonymous	★	✗	✗	✗	✗	✗
Logged-in	★	★	★	★	★	★
Administrator	✓	✓	✓	✓	✓	✓

Key:

- **View** - view the plan and its builds; when creating a new plan, check the **Allow all users to view this plan** to allow anonymous and logged-in users view your plan.
- **View configuration** - view the configuration of the plan and its jobs. DATA CENTER
- **Edit** - view and edit the configuration of the plan and its jobs, not including permissions or stages.
- **Build** - trigger a manual build, or suspend and resume the plan.
- **Clone** - clone the plan.
- **Admin** - edit all aspects of the plan including permissions and stages.

Project permissions

⚠ Starting from Bamboo 6.9, to access any plans in a project you must have the View permission granted. Without the project View permission, you won't be able to see, run, or administer any plans.

Project permissions allow you to control access to project permissions and settings. See Configuring project permissions.

User	View	Create plan	Create repository	Admin
Anonymous	★	✗	✗	✗
Logged-in	★	★	★	★
Administrator	✓	✓	✓	✓

Key:

- **View -** access the project and plans or repositories for the project
- **Create plan -** create plans for the project
- **Create repository** - create repository for the project. DATA CENTER
- **Admin** -
  - manage permissions for the project
  - manage permissions for all plans in a project
  - change project settings

Deployment permissions

Bamboo's deployments features allow you to control permissions for both deployment projects and deployment environments.

Deployment projects

User	View	View configuration	Approve release	Edit
Anonymous	✓	✗	✗	✗
Logged-in	✓	★	★	★
Administrator	✓	✓	✓	✓

Key:

- **View** — view the project and its associated environments.
- **View configuration** — view the project configuration. DATA CENTER
- **Approve release** — allow users to approve or decline deployment releases.
- **Edit** — edit the project, its related plan, and environment configuration, and create releases.

Deployment environments

User	View	View configuration	Edit	Deploy
Anonymous	✓	✗	✗	✗
Logged-in	✓	★	★	★
Administrator	✓	✓	✓	✓

Key:

- **View** - view the environment. You must also have view permission on the deployment project.
- **View configuration** - view the environment configuration. DATA CENTER
- **Edit** - edit the environment configuration.
- **Deploy** - deploy releases to this environment and create releases for this project.

Permission dependencies

To ensure the consistency of Bamboo permissions, starting with Bamboo 6.3, we provide an update mechanism which will fix all inconsistencies for all permissions in your Bamboo environment. We have also modified all the pages where you can edit permissions in a way which won't allow granting inconsistent or clashing permissions in the future.

If you want to revoke a lower-level permission for a user, you must revoke the higher-level permissions first. Also, when granting a higher-level permission to a user, all relevant lower-level permissions will be granted automatically to that user.

⚠ Permission consistency might still be broken when using third-party plugins.

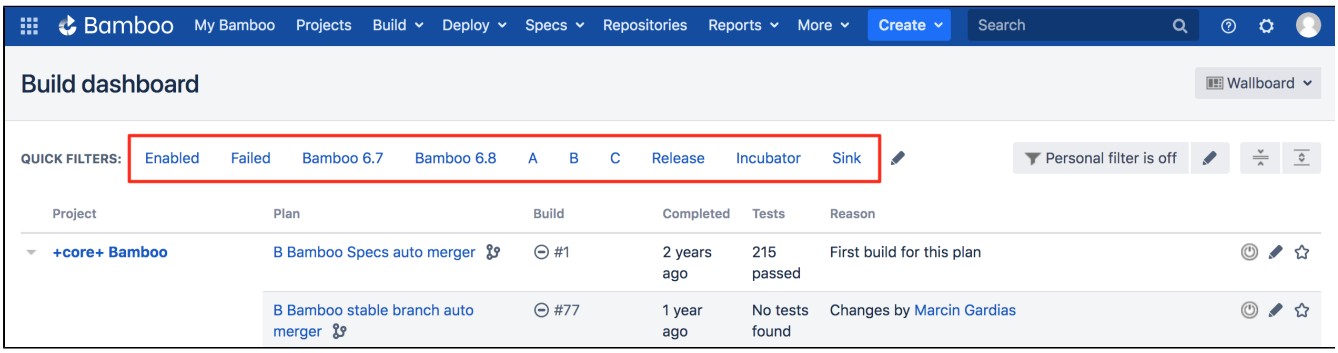
Quick filters for Bamboo

Use quick filters for handy search shortcuts in your Bamboo build dashboard. Create filters based on configurable rules and never miss a build plan again.

Quick filters can be created only by administrator and work only with the plans displayed in Bamboo dashboard, which means they don't include plan branches.

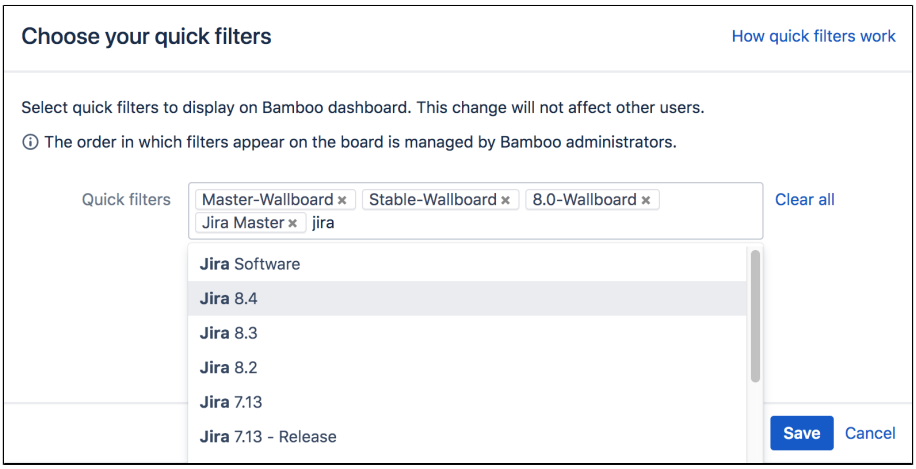
Using quick filters

All quick filters created by an administrator are available to all users of a Bamboo instance in the build dashboard. Select a filter name in the build dashboard to display plans that match the rules assigned to the filter. A plan or plans are displayed only if they match all the rules specified in a filter.



Starting from version 6.10, all users can select which quick filters they want to see on their dashboard. To do that, select the pencil ( ) icon and select your quick filters. This way you can avoid unnecessary clutter, and create a workspace that suits your needs best.

If you want to bring back a quick filter that you've previously removed from a dashboard, you can also do that by selecting the pencil ( ) icon and searching for your filter name.



Managing quick filters

Only administrators can add, edit, and delete quick filters.

If you're an administrator in Bamboo and want to add, edit, or delete quick filters for a dashboard:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Plans**, select **Quick Filters**.
3. Select **Create filter**.
4. Give your new quick filter a name.
   Your new filter, still with no rules, will appear at the bottom of the quick filters list.

5. In the actions column, next to your new filter, select **Configure**.
6. Start adding rules you filter.

   You can create filters with combinations of the following rules:

   By completion date
   Displays plans that completed within a specific time frame. For example, you can display plans that completed in the last three days.

   By label
   Displays plans with a specific label. You can define multiple labels. The plan displays when it has at least one label specified in the rule.

   To assign a label to a plan, select the name of a plan to display the plan summary, and go to **Actions** > **Modify plan label**.

   By name
   Displays plans with a specific name or plans that match a regular expression.

   By project
   Displays plans that are assigned to a specific project. You can select one or more projects.

   By result status
   Displays plans that have completed with a specific result. You can select from:

   - Successful
   - Failed

   By status
   Displays plans based with a specific status. You can select from:

   - Enabled
   - Disabled

Unpacking large .ZIP archives

We noticed that the TrueZIP archiver used by Bamboo might be incompatible with some binaries for archives exceeding 4 GB. In order to extract files from a large archive, use the following script:

TrueZIP extractor

```
#! /bin/sh

TRUEZIP_JAR=truezip-samples-7.7.9-jar-with-dependencies.jar

if [ ! $# -eq 1 ]; then
    echo "Usage:"
    echo "  $0 [bambooCloudExportFile]"
    exit 1
fi

BAMBOO_EXPORT_FILE=$1

if [ ! -f $BAMBOO_EXPORT_FILE ]; then
    echo Can not access $BAMBOO_EXPORT_FILE
    exit 1
fi

if [ ! -f $TRUEZIP_JAR ]; then
    echo Downloading TrueZip JAR ...
    curl -s https://repo1.maven.org/maven2/de/schlichtherle/truezip/truezip-samples/7.7.9/$TRUEZIP_JAR -o
$TRUEZIP_JAR
fi

if [ -d bamboo-home ]; then
    echo Directory bamboo-home exists, please remove it before proceeding
    exit 1
fi

if [ -d database ]; then
    echo Directory database exists, please remove it before proceeding
    exit 1
fi

echo Unzipping Bamboo HOME directory ...
java -jar $TRUEZIP_JAR cp $BAMBOO_EXPORT_FILE/bamboo-home bamboo-home

echo Unzipping Bamboo database ...
java -jar $TRUEZIP_JAR cp $BAMBOO_EXPORT_FILE/database database
```

Personal access tokens

Personal access tokens replace username and password authentication in REST calls. They are a secure way to use scripts and integrate external applications with Bamboo. If an external system is compromised, you simply revoke the token instead of changing passwords, and consequently changing it in all scripts and integrations.

For added security, when you're creating a token you can also set it to automatically expire. This is optional, but if your admin has made this a requirement you'll need to select an expiry date that's within the limits they've set. Once a token has been created, its expiry date can't be changed. You can see the expiry dates for all your tokens in the HTTP access tokens page list.

⚠ You can't authenticate with personal access tokens in Bamboo UI.

Using personal access tokens

To use a personal access token for authentication, you have to pass it as a bearer token in the Authorization header of a REST API call.

Here's an example of rest using a bearer token:

```
curl -H "Authorization: Bearer NDc4NDkyNDg3ODE3OstHYSeYC1UgXqRacSqvUbookcZk" http://localhost:8085/bamboo
/rest/api/latest/plan/PROJ-PLAN
```

Managing personal access tokens

To view and manage your personal access token in Bamboo:

⚠

- Admins **can't** create tokens for users.
- Admins **can** revoke tokens from **Administration** > **Security** > **Users** > {*user_name*} > Personal access tokens page.

Creating a token

1. From the top navigation bar select your avatar, and select **Profile**.
2. Select the **Personal access tokens** tab.
   Here you can view your existing tokens or create a new one.
3. Select the **Create token** button.
4. Give your token a name.
5. Assign permissions to your token.

   Permissions are set when creating a token and can't be modified later. By default, for security reasons, personal access tokens have read-only permissions:

   - **Read-only permissions** — the token will be only allowed to read data from Bamboo that you can normally view. It won't be allowed to read data, that the associated user can't read.
   - **Triggering permissions** — the token will be able to start builds and deploy environments that you normally can run. It won't be allowed to trigger builds or deployments that the associated user can't run.
   - **Same as user** — token will have the same set of permissions as you (i.e. edit or admin).

   It's recommended that you assign the lowest possible set of permissions to the token. This way even if the token gets compromised, it will be possible to perform only a limited set of actions with it.
6. Optionally, set an expiration date for your token.

> This step may be required if your system admin has made setting personal token expiration a requirement.
>
> Learn more about requiring personal access token expiration

7. Record your token in a safe manner. For security reasons, the token value is shown only once. If you don't record the token value or lose it you won't be able to recover it and will have to create a new token.
8. Select **Finish**.

Revoking a token

1. From the top navigation bar select your avatar, and select **Profile**.
2. Select the **Personal access tokens** tab.
3. Hover over your token name.
   The revoke button appears on the right.
4. Select **Revoke**.
5. Select **Confirm**.

Bamboo Best Practice

Bamboo is a fantastic tool for continuous integration and deployment. It offers a powerful tool for automating software development, however knowledge of some of the tips and tricks that our Bamboo masters use can help reduce friction within your own development cycles.

This user guide has information about how to get the best out of Bamboo, and includes a number of scenarios and best practice approaches. Please see Using Bamboo for more information on specific Bamboo installation, configuration, and usage.

Best practice topics

Using stages

Branching & DVCS

System Requirements

Sharing artifacts

Using Agents

Installing

Bamboo upgrade guide

Installing Bamboo on Linux

Installing Bamboo on Mac OS X

Installing Bamboo on Windows

Connect Bamboo to an external database

Bamboo remote agent installation guide

Supported platforms

Bamboo Best Practice - System Requirements

The recommendations in this guide are not universal and the final configuration of your Bamboo instance will depend on your specific use case.

System requirements & considerations

Note that Atlassian currently only supports Bamboo on x86 and 64-bit x86-derived hardware platforms.

Hardware considerations

CPU and memory

For Bamboo, the minimum CPU and memory requirements depend on the size and complexity of your plans. You need to consider:

- Will your builds have functional tests as part of the plans?
- Are your plans executed simultaneously? If so, how many plans will be running at any given time?
- What are the requirements for your running builds, for example, do they need large amounts of memory /disk/swap space?
- How many users will be using Bamboo at any given time? Like any web application, the system resource needed is proportional to the load experienced by the server.
- How many local agents do you plan on running?

User scenario	Usage profile	Bamboo server
Individual user/ **Small team**	• 10 - 20 plans • Little concurrent building • Light server use	4 core, 4 GB RAM
Medium team	• 10 - 20 plans • Medium concurrency • Light server use	8 core, 8 GB RAM, remote agent use
Multiple small teams/ **Large team**	• 20 - 100s plans • Plan branches • High concurrency • Medium server use	8 core, 16 GB RAM, more remote agents
Multiple large teams/ **Department/Division**	• 1000s of plans • Frequent plan branches • High concurrency • High server use	16 core, 16 GB RAM, all remote agents

Storage

The Bamboo installation size is approximately 140 MB, however, Bamboo's storage requirements depend upon its usage pattern during use. The usage pattern depends on factors such as:

- How many plans you will run
- How many tests each plan will execute
- How many artifacts you are going to have and their size

Atlassian recommends that you allocate about 20 GB on top of the Bamboo installation size, and evaluate your usage patterns. Where usage is likely to grow, consider adding additional storage.

Software requirements

Bamboo is a pure Java application and should run on any platform, provided all the JDK requirements are satisfied.

The Supported platforms page lists the server and client software, and their versions, supported by .

Browser

Disabling JavaScript in your browser, or using a script blocking tool like NoScript, will limit access to Bamboo's full functionality. JavaScript should be enabled.

Java

Bamboo requires the full **Java Developers Kit (JDK)** platform to be installed on your server's operating system.

Application server

Bamboo is a web application that requires an application server. Currently, Apache Tomcat is supported. Tomcat is a stable, lightweight, and fast performing application server, however, please note the following:

1. Deploying multiple Atlassian applications in a single Tomcat container is **not supported.** We do not test this configuration and upgrading any of the applications (even for point releases) is likely to break it. There are also a number of known issues with this configuration (see this FAQ for more information).
2. We also do not support deploying multiple Atlassian applications to a single Tomcat container for a number of practical reasons. Firstly, you must shut down Tomcat to upgrade any application, and secondly, if one application crashes, the other applications running in that Tomcat container will be inaccessible.
3. Finally, we recommend not deploying *any other applications* to the same Tomcat container that runs Bamboo, especially if these other applications have large memory requirements or require additional libraries in Tomcat's `lib` subdirectory.

Database

Bamboo requires a relational database to store its data. Bamboo supports the most popular relational database servers, so we suggest using the one that you are most comfortable with administering. Bamboo ships pre-configured with an integrated HSQL database for evaluation purposes only. Since HSQLDB is prone to database corruption, we recommend configuring an external database for production environments.

Hence, if you intend to use Bamboo in a production environment, we **strongly recommend** that you connect Bamboo to a supported enterprise database system.

Other considerations

Bamboo also requires a number of services for efficient operation. You need to consider:

- Build log size
- Database connection pool size.
- Number of local agents.
- Number of remote or elastic agents.

Build log size considerations

In Bamboo, ActiveMQ is configured as a persistent queue, and messages sent over ActiveMQ are written to disk in the `$BAMBOO_HOME/shared/jms_store` directory. Build logs are sent over this queue, and large volumes of logs can slow the queue as it attempts to persist the message to disk. If the queue slows sufficiently it can cause the agent message broker to stop responding to other agents. This may present as large numbers of agents suddenly disconnecting from the server without any errors appearing in the server logs, and failing to reconnect until the server is restarted.

Typically any log that is 100MB+ is a cause for concern, although large numbers of logs slightly smaller than this limit may cause similar issues.

Database connection pool size

The number of database connections available to Bamboo is the lower of two values: your DBMS connection limit and the configured Bamboo connection pool size. From Bamboo 4.2 and later, the Bamboo connection pool size has a default value of 100.
For a small to medium instances (~5 concurrent users, ~5 busy/building local agents, 20 remote agents, 50 plans), the default values are sufficient.

You should increase the connection limit if you notice UI freezes or general sluggish UI performance. Do not decrease the number of available connections below 25.
Note that having too many connections available to Bamboo carries no performance penalty as long as your DBMS can handle the load.

Example: How to estimate the number of db connections

The following formula gives a rough estimate of the number of database connections that will be required:

```
(Concurrent users)/5 + (Busy remote agents)/5 + (Local agents)*1.1 + (Amount of concurrent change detections)
```

For example, an instance with:

- 5 concurrent users
- 30 busy remote (or elastic) agents
- 30 busy local agents
- 60 plans with repository polling set to 60-second intervals (assume 3 seconds per change detection)

would require 1 + 6 + 33 + 3 = 43 connections.

Bamboo ships with a pre-configured connection limit, however this can be modified by editing the following value in your bamboo.cfg.xml file:

```
<property name="hibernate.c3p0.max_size">100</property>
```

Local agents considerations

If you run more than 5 concurrently building local agents, you'll probably need to adapt the connection limit because each busy local agent requires a live database connection.
Also, note that large amounts of busy (building) local agents can negatively influence the performance of a Bamboo server (and other services running on that host).

Remote and elastic agent considerations

Remote and elastic agents do not require special database connection settings.

Bamboo Best Practice - Using stages

Overview

The basic process for continuous delivery is Build > Test > Publish, which can be repeated multiple times before a release candidate is identified and shipped.

This page describes two approaches to using stages in Atlassian Bamboo. Many people will find that the first approach, *Continuous integration*, will meet their requirements, and we recommend that as a starting point. When you have that operating, you can build on it using more advanced methodologies.

On this page:

See also:

- Bamboo Best Practice - Sharing artifacts
- Bamboo Best Practice - Branching & DVCS

Fail fast – detect failures as early as possible

Fail fast is used here in the context of continuous integration. It's a development paradigm that emphasizes the early detection, notification and correction of build failures. Early detection allows early correction, so reducing impact on the project. Furthermore, if we detect problems early, we won't need to execute the rest of the build process, so saving time and resources.

Example Scenario

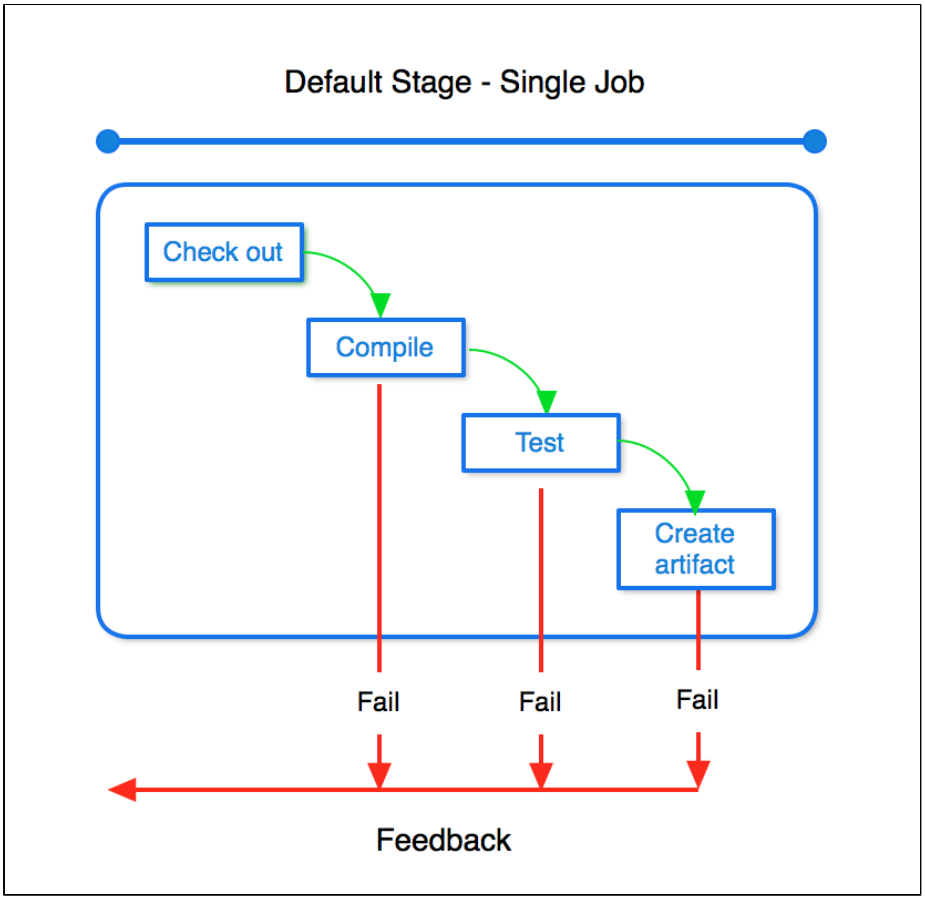
Let's consider the following simple scenario that uses a series of tasks within a single job. We only need a single stage for this. Typically, unit tests exist in close association with the source files, and are run at, or soon after, compile time.

Task 1 – Check out: We need to check out the relevant code from the repository. Best practice with Bamboo is to set up a linked repository that can be referenced by several plans and that can be updated in just one place. See Checking out code.

Task 2 – Compile: We can configure a builder task to compile the code. If syntax errors are detected, there is no point in performing the unit tests. See Configuring a builder task.

Task 3 – Run unit tests: Unit tests rapidly identify problems with how code runs. This quickly identifies semantic errors. See Configuring a test task.

Task 4 – Create artifact: Often, you will want Bamboo to keep build artifacts, such as reports and binaries, that can be used later. See Sharing artifacts.



Artifact promotion – ship the tested binary

The promotion of build artifacts, especially binaries for use in later phases of the pipeline, is a key concept in con tinuous integration. Not only can this save time and resources, but crucially, it ensures that a release candidate that could potentially ship to customers contains exactly the code that was tested throughout the pipeline.

In Bamboo, artifact sharing between stages is the mechanism used to promote artifacts.

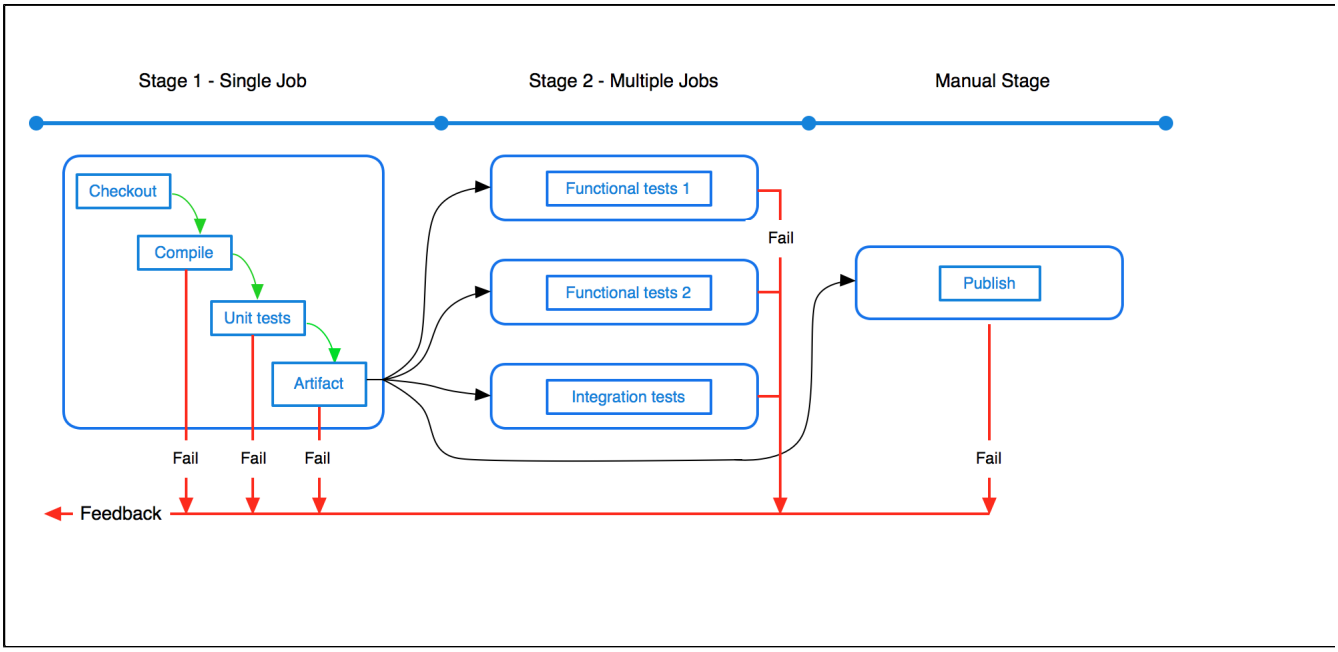
Example scenario

Let's consider the following scenario that adds further stages compared with the Fail Fast scenario above. These extra stages are used to add functional and integration testing, and to provide a manual stage to provide control over when publishing happens.

Stage 1 – Fail fast: This is just the continuous integration stage described in the earlier scenario. The generated artifacts are marked for sharing in later stages. See Sharing artifacts.

Stage 2 – Run functional and integration tests: We can split these tasks into multiple jobs so they will run in parallel, so reducing the time taken. Each job uses the same artifacts generated in Stage 1. If any job fails, then later stages will not be run.

Stage 3 – Publish: We introduce a manual stage here for this example, but this could an automated stage. A manual stage gives us a control point that pauses the pipeline, to allow us, for example, to make a business decision about whether the release candidate should be published, and when. We only run this stage when it has been approved.



Bamboo Best Practice - Branching and DVCS

General overview

No matter how scary it may seem, branching your code is unavoidable - and also a very powerful way to let developers work in isolation on different aspects of your project.

The simplest branching model is that of a master branch and a development branch. The master (or mainline) branch contains the production versions for release. Parallel to master runs the development branch, where developers work on features that will be merged back into master. When sufficient new features have been developed, they will be merged back into master and form the next production release.

The simple model can be extended with other branches to make development work more flexible. These include:

- Feature branches
- Release branches
- Hotfix branches

See also:

- Bamboo Best Practice - Sharing artifacts
- Bamboo Best Practice - Using stages

But because a developer isn't constantly merging changes from master into their development branch, there may be uncertainty about whether the code will work when it is eventually merged back into master. The last thing you want is to pollute your master with non-functioning code from the branch.

Bamboo offers a number of useful tools for tackling branches. This best practice guide explores some of the ways that Bamboo handles branching to improve your development practices.

You may also want to refresh your Git knowledge with the Atlassian Git tutorials page before you read any further.

Best practice approaches

Feature branching with Bamboo plan branches

Objectives and learning outcomes

Understand what feature branching is, and how it can be useful as a development process. After completing this section, you will understand:

1. How feature branching works
2. How feature branches improve quality by eliminating risky merges

What is feature branching?

Feature branching is a lightweight way for a developer to make changes to a software project without having to worry about sharing those changes if they are uncompleted.

The main reasons to use feature branching are to ensure accurate conflict mitigation and to reduce the possibility of pushing code into the master branch or to other people until you are ready to do so. Utilizing rapid, regular code merges assists in reducing code drift across the development process.

Bamboo uses a concept called plan branches to help teams easily test branches using continuous integration and to avoid merge problems.

Example scenario

Let's examine the following scenario for traditional feature branching:

1. A developer assigns an issue to themselves and creates a new branch (the feature branch) from master.
2. The developer works on the code, makes regular local commits to the feature branch, reaches a finishing point and pushes the commits to the repository.
3. When the issue is completed, the feature branch is merged back into master.

So, what's wrong with this? The developer hasn't run their builds on the feature branch and it is unknown whether the tests pass or not and any defective code from the feature branch will reach the rest of the team when it's merged to master.

Now let's see how it works using Jira and Bamboo plan branches:

1. A developer assigns the issue to themselves in Jira and creates a new branch from master. The name of the branch starts with the issue key so that it can be easily identified and tracked by both Bamboo and Jira.
2. Bamboo detects the new feature branch and creates a new plan branch. A plan branch is created automatically for any build that has plan branching enabled.
3. The developer works on the code, makes regular local commits and pushes the commits to the repository.
4. Bamboo identifies the changes and builds the corresponding plan branch.
5. Optionally, to ensure that the branch and master will work together when merged, Bamboo can then merge the contents of master (including any new changes the team has made) into the the feature branch and have the build run.
6. If the tests pass, Bamboo pushes the updated feature branch back to the repository.
7. When the issue is completed, the feature branch is merged back into master with the knowledge that their new feature will not break on master.

We can already see that the Bamboo plan branch helps us by running build plan tests against the newly merged code. Only if the tests are passed is the code pushed, which prevents incorporating defective code. If the build fails, the merge is thrown away and the developer is notified.

Extending feature branching

We can usefully extend the concept of feature branching to include an integration branch workflow. This concept mirrors the approach of feature branching in that it also advocates frequent merging. However, it provides an integration branch during development of a particular story. When the story is completed, it is merged into master, but offers two different approaches to working around the integration branch:

1. Some teams merge their code into the integration branch while the story development is in progress; when the story is complete, it is then merged directly into master and closed.
2. Other teams may work exclusively around the integration branch during their code development, but will wait until the very end when their stories are tested and validated before merging integration onto master.

Conclusion

Feature branching offers a flexible and accurate conflict mitigation tool for developers. By using frequent and regular code merges, code drift and defective code implementation across the project is minimized. Feature branching works particularly well when developers have permission to toggle auto merging on and off to suit their individual development cycle. And of course, Bamboo provides an ideal environment to give developers access to these permissions.

Approaches to branching

Objectives and learning outcomes

Identify and describe how Bamboo can use feature and plan branches. After completing this section, you will understand:

1. The two mechanisms for merging branched code back into the master branch
2. A high level concept view of the branching process

Overview

Feature (or topic) branches are used to develop new features for an upcoming or future release. A feature branch exists only as long as the feature is being developed, and will eventually be merged back into the development branch.

Plan branches represent a branch in the version control system for development of a specific feature. The plan branch inherits all of the configuration defined by the parent plan, but may be built against any other specified plan. Any new branch created can be automatically built and tested using the same build configuration as that of the parent plan. Alternatively, you can override the parent plan and individually configure the branch plan. When the branch succeeds, it is merged back into master.

There are two ways in which plan branches can be merged with the master branch.

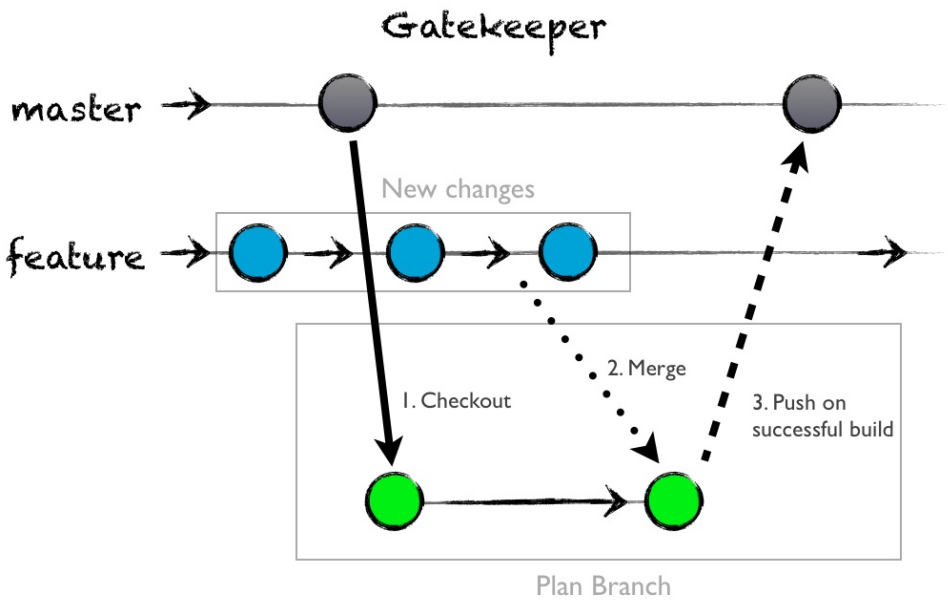
Example scenario

Let's consider the following branch scenarios:

Scenario 1: Gatekeeper

The gatekeeper method works in the following way:

1. Both master and feature branch are checked out from the repository
2. Changes are merged into master from the feature branch
3. The build plan is run against the merged code, and held in memory by Bamboo
4. If successful, the merged code is pushed to master



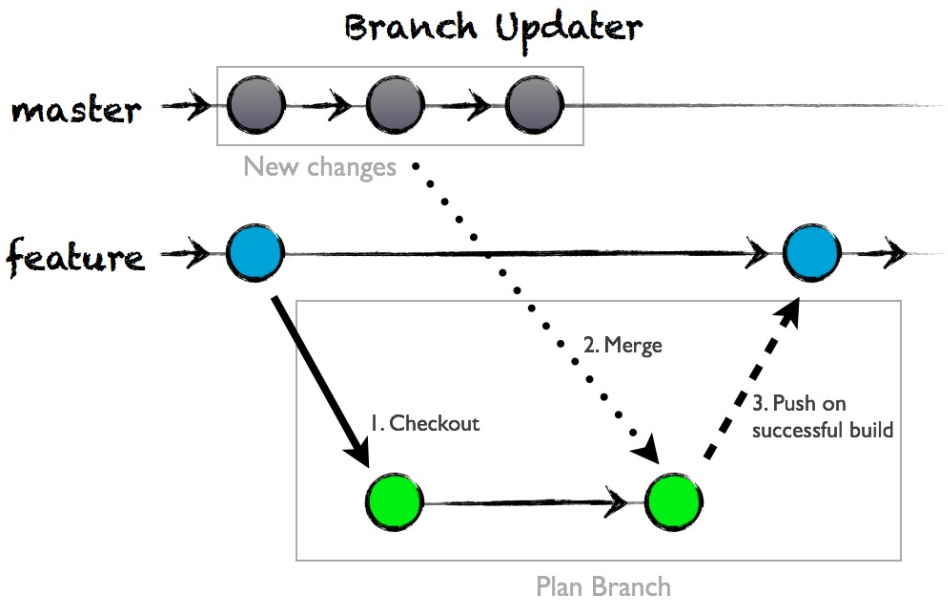
You should use the Gatekeeper strategy when you want to:

1. Automatically merge your feature branch back into the master branch after a successful build of the merged changes
2. Quickly identify when a build of combined changes fails, preventing the feature branch from being merged back into the master branch

Scenario 2: Build updater

The build updater is an alternative approach where changes flow in the opposite direction. It works in the following way:

1. Both master and a feature branch are checked out from the repository
2. Changes are merged into the plan branch from master
3. The build plan is run against the merged code and held in memory by Bamboo
4. If successful, the merged code is pushed to the feature branch



You should use the Build Updater strategy when you want to:

1. Automatically merge changes from the team's master branch into your feature branch, either after a successful build of the master branch, or at the start of builds against the feature branch.
2. Get notified when the changes on your feature branch are no longer compatible with the team's master branch.

Now we know how plan branching works, but how do we implement it using Bamboo? Bamboo actually makes it very easy for us. Let's have a look at another example:

Scenario 3: Plan branching in DVCS

This is a typical high level DVCS plan branching scenario:

Step 1: Create branch - Use your version control system's branching feature to create a new branch in your repository.

Step 2: Branch detection - Bamboo will auto detect the new branch for Git, Mercurial, and SVN. Perforce and CVS users will have to manually create the branch on Bamboo's behalf. This can be done from the Branches tab in your build plan's configuration screen.

Step 3: Plan cloning - Bamboo automatically clones all plans associated with the repository and connects the clones to the new branch.

Step 4: Configure plan variables - The configuration of plans pointing to the master branch will be inherited by the plan branches. Jobs, stages, and artifact sharing work exactly as defined in the original plan. Variables, notifications, and triggers may be customized for each plan branch. Other configuration options for plan branches include:

1. Merge strategies (see gatekeeper and build updater above)
2. Toggling auto cleanup on/off
3. Branch removal after a defined inactivity period

Step 5: Branch build - The feature branch is built in accordance with its triggers. The optional merge strategies are applied at build time.

Conclusion

Feature and plan branching offers a range of flexible methods for developers to branch and work on different code segments during the development process. The Gatekeeper and Branch Updater methods allow alternative approaches to branching your code, while plan branching in DVCS allows Bamboo to automatically detect new branches in Git, Mercurial, and SVN repositories.

Branching with Jira integration

Objectives and learning outcomes

Understand how Jira integration can be used to track development changes branching, and how it improves oversight of a development project. After completing this section, you will understand:

1. What Jira integration is
2. How it can be used to track changes within the code development

Overview

Jira integration in plan branches relies on including a Jira issue key as part of the branch name. Bamboo and Jira work together to ensure that Jira issues are attached to development branches, allowing developers and other interested parties to examine which issue has informed the code development within the branch.

Example scenario

Let's examine the following scenario for Jira integration:

1. A developer picks up a Jira issue and creates a feature branch for it
2. Bamboo creates a link between the issue and the branch, and all the branch's builds
3. The developer works on the issue, making regular pushes to the feature branch, which are built by the corresponding plan branch/es in Bamboo
4. The Jira issue shows the current build status of the feature branch
5. When work on the feature branch is complete, it can be merged to master manually through the version control system, or automatically, by enabling Bamboo's gatekeeper merge strategy

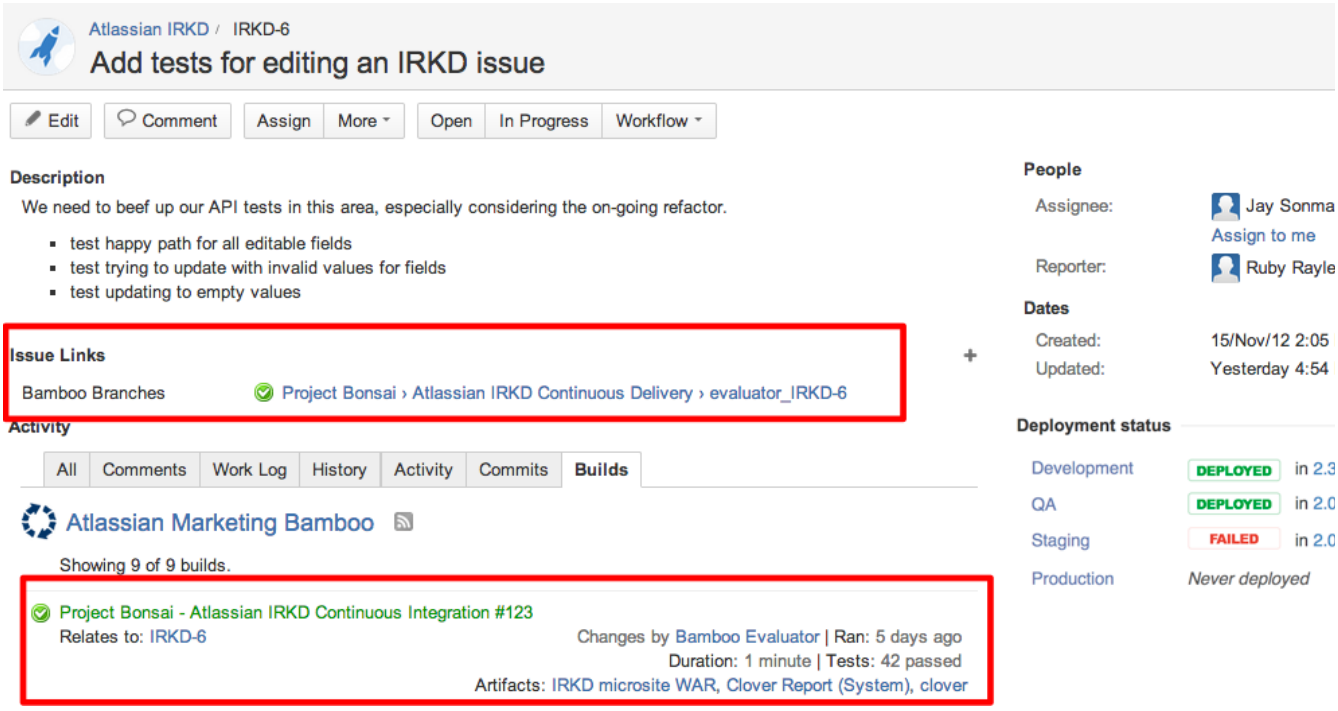
Why use Jira integration?

By including a related Jira issue as part of the branch name, Bamboo can link the issues to the related builds and to the branch itself. This makes oversight of individual stories much easier:

- Product owners can view the development of user stories from within the Jira issue.
- QA can select an artifact for testing from within Jira, and identify which issues have informed its development.
- Developers can examine builds and artifacts, and see which Jira issues have informed the development process.

The Jira Bamboo plugin

The Jira Bamboo plugin provides enhanced information sharing between Jira and Bamboo, allowing you to view the status of all builds and branches associated with an issue from within the issue itself. Apart from DVCS and branching, the plugin also surfaces deployment information for issues when Bamboo's deployment projects are used.



Learn more about the Jira Bamboo plugin here.

Conclusion

Jira integration with branching provides an effective mechanism for tracking changes in code development and identifying what issues have informed the process. Jira integration also provides an effective way for interested parties to track progress and locate relevant artifacts.

Bamboo Best Practice - Sharing artifacts

General overview

We've already had a look at techniques such as *fail fast* and *artifact promotion* as ways of improving your Bamboo processes in the using stages Best Practice guide, but here we're going to dig a little deeper and look at some ways that you can get artifact sharing to work for you.

See also:

- Sharing artifacts between jobs
- Sharing artifacts between build plans
- Sharing artifacts from a build plan to a deployment environment

Best practice approaches

Sharing build artifacts with downstream processes

See Artifact promotion for a description of this technique.

How do I configure artifact sharing between jobs?

In Bamboo, artifact sharing between jobs is configured using the Artifacts tab on the plan's configuration. Find the artifact you want to share, select **Edit**, and select the **Shared** checkbox.

Check out Sharing artifacts between jobs to learn how to configure your Bamboo server to take advantage of artifact sharing between jobs.

Sharing artifacts between plans

Objective

Identify and describe how artifact sharing between plans can be achieved.

Learning outcomes

After completing this section, you will understand how to share an artifact between plans.

Overview

We discussed above how we can achieve significant time benefits from capturing and sharing artifacts to downstream processes rather than checking out and compiling each time the artifact is required. Generating an artifact at the top of the development pipelie, and passing it to successive downstream processes also has the benefit of ensuring the integrity of the code is maintained throughout the pipeline, because we know it is the *exa ct* same code that we tested earlier on. We also discussed how we can manage passing artifacts within a build plan, but let's suppose that we want to pass artifacts between two plans. Easy: we use the download artifact task to make the artifact available from one plan to another.

Example scenario

Let's consider the following artifact sharing example:

Imagine that we have a build plan that creates and uses an artifact - Artifact A. Now let's suppose that we also have a child plan, and we would like to use Artifact A in this plan for some other purpose. Bamboo doesn't technically allow you to share artifacts between plans (but watch this space), so we can use a work around to get our artifact shared into the child plan. We copy it from the parent plan to a remote storage location, then use the artifact download task to obtain it for the new plan. **Note:** this approach differs significantly to the process for sharing artifacts between jobs.

Parent plan

Step 1: Checkout & compile - We need to check out the relevant code from the repository and compile it into an artifact. Our artifact is now defined and available for use by downstream jobs. Let's give it a name - Artifact A - and specify its location, so downstream jobs can find it, though of course only jobs in downstream stages can consume it.

Step 2: Testing - We can use some *fail fast* methodology and run some tests on our artifact before we go any further. We can conduct short and rapid unit tests and longer functional testing on our artifact. But we already know that artifact sharing can be used to increase testing speed in both cases.

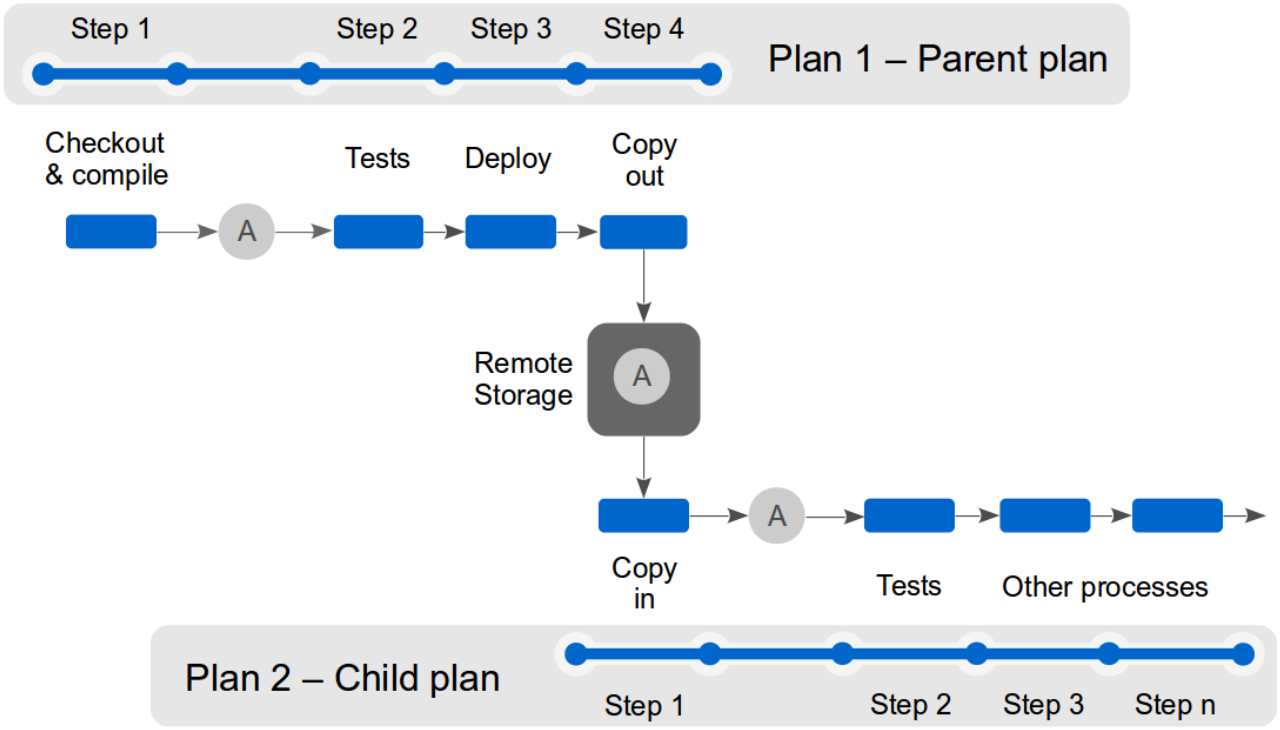
Step 3: Deployment - When testing is complete, the artifact can be deployed to a QA environment by a consuming job that runs a deployment script against it, but we still need to share it with the child plan.

Step 4: Copy artifact out - The final step of this plan is to use a task to copy the artifact out to a remote location such as Nexus or Artifactory, using the applicable Bamboo plugin. Alternatively, simply run a script task to copy the artifact to a remote file server location on your own network.

Child plan

Step 1: Copy artifact in - The first step of the child plan is to use a task to copy the artifact in from where the parent plan left it. Depending on the method you used to copy it out, you may require a task that utilizes a Bamboo plugin.

Step 2: Business as usual - Now that we have copied the artifact in, we can perform regular Bamboo operations as part of an ongoing build plan. These could be additional tests, or deployments into different environments.



Extending artifact sharing

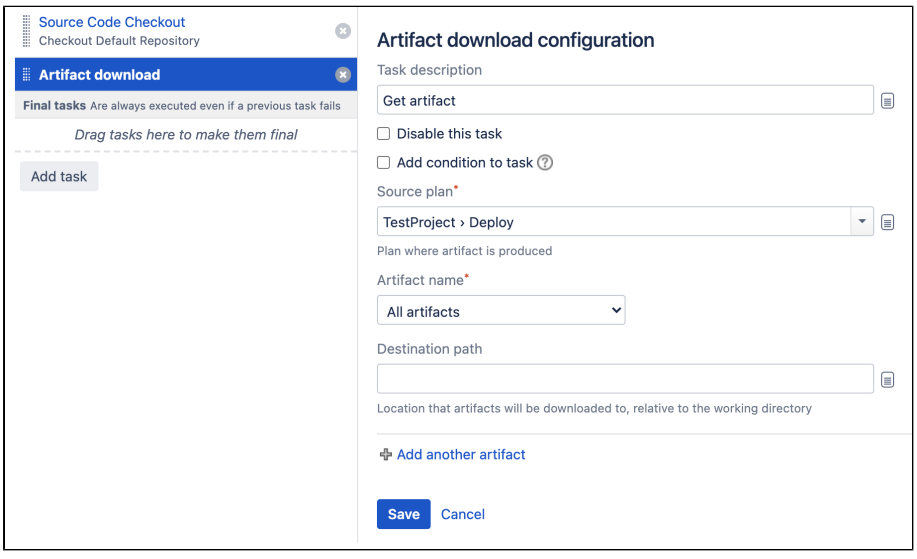
And of course, once we have our artifact neatly stored in remote storage, the artifact download task means that it can associated with any build plans that we may want to run.

Conclusion

Artifact sharing is a powerful technique for making single artifacts available. Artifact sharing across plans allows us to make artifacts available for different build plans from one checkout and compile. We know that we will always be using a consistent artifact which reduces the time overhead of multiple checkout and compile steps.

How do I configure artifact sharing between jobs?

Artifact sharing between jobs is configured using the Artifact download task:



Check out Sharing artifacts between build plans to learn how to configure your Bamboo server to take advantage of artifact sharing between plans.

Bamboo Best Practice - Using Agents

General overview

We've already had a look at how we can improve Bamboo's efficiency in the Using stages and Sharing artifacts best practice guides, so here we're going to have a look at how we can improve raw build speeds using Bamboo agents.

Let's consider this simple Bamboo scenario:

Imagine a set of plans that we have developed and are queued, awaiting a build agent to become available to execute the build. This is great, and exactly what Continuous Integration is all about, but we notice that certain plans seem to sit and wait consistently longer than others. This has the effect of slowing our progress, and may be felt later down our development streamline. But why is it happening? And what can we do about it?

See also:

- Bamboo Best Practice - Using stages
- Bamboo Best Practice - Sharing artifacts
- Bamboo Best Practice - Branching & DVCS

Let's examine exactly what's going on.

Each build agent offers a set of capabilities, and each plan will have capability requirements that the that the build agent must meet. These could include a range of executables, tasks and JDKs. Build agents are tailored to match specific plan requirements and as a result not all agents can build all plans. Often, only a small subset of agents will meet all of the requirements for a specific plan. Typically, plans that demonstrate consistently long wait times, are the ones that are waiting for a specific combination of capabilities to become available.

The Build Queued Duration report tells us the average time that a plan sits in the queue until build agents become available to execute it. By examining the report, we can identify which builds are too slow, and also if we are sporting wasted capacity on our systems. So how does this help us, and how can we even out our wait times? Adding required capabilities to a greater number of agents helps to improve parallel builds and even out our build loading. We can achieve this in a number of ways.

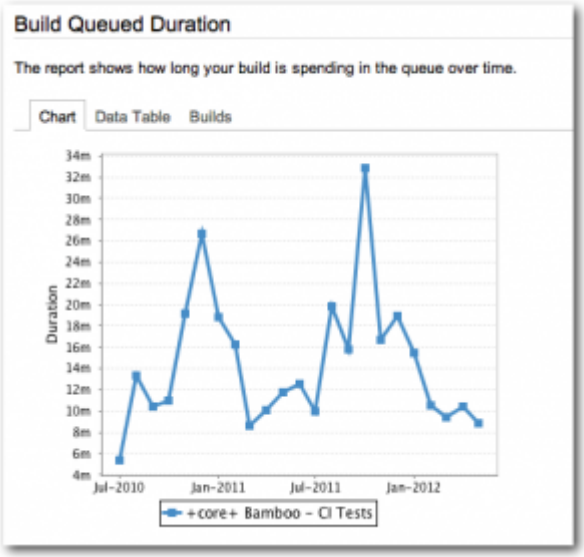
Best practice approaches

Using remote agents

Why use remote agents?

Adding popular capabilities to more agents is one way to tackle our wait time problem, however we can also take advantage of remote agents to boost our capabilities. By increasing the number of remote agents to the maximum allowed by our license tier, we can add significant amounts of available build capability which will in turn lead to reduced wait times. We could also consider using elastic agents on AWS.

The following graph shows how adding additional remote agents helped the Bamboo team to reduce build wait times for building Bamboo itself:



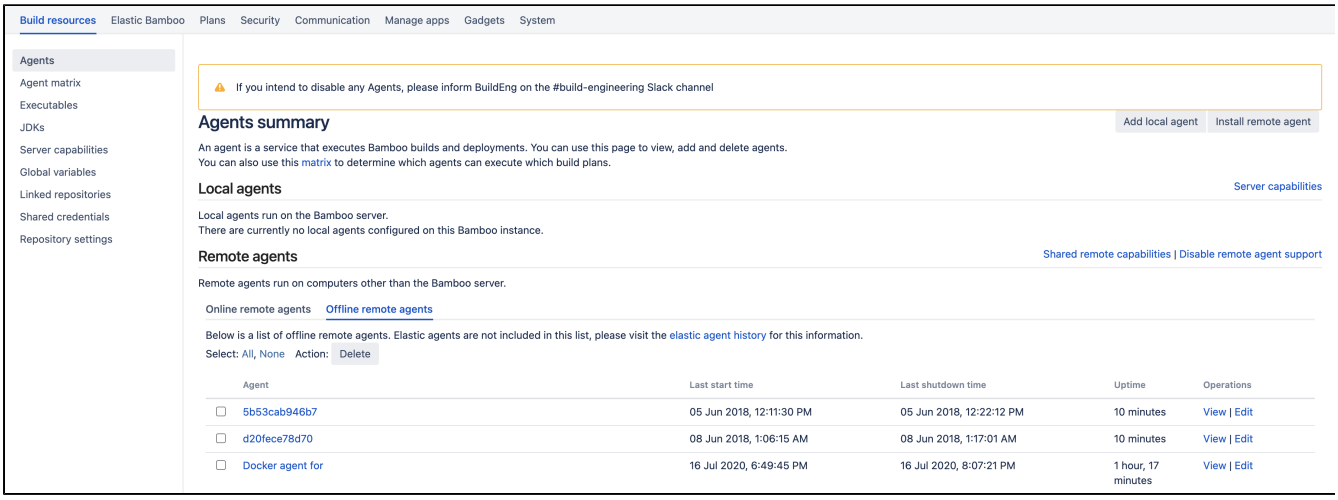
When build wait times approached 27 minutes in late 2011, adding additional remote agents with well defined capabilities reduced wait times to less than 10 minutes. The same is also true when wait times approached 33 minutes - additional remote agents ultimately reduced wait times back to less than 10 minutes.

Unknown capabilities

Sometimes remote agents have capabilities that are unknown, so Bamboo will not automatically utilize these when it's looking for agents for a build. Luckily, even if Bamboo doesn't know about these capabilities, we can quickly and easily detect them. To do so, go to  > **Build resources** > **Server capabilities** > **Detect server capabilities** to identify the capabilities available on a remote agent.

Adding remote agents

To add remote agents, go to  > **Build resources** > **Agents** > **Install remote agent**.



Learn more about adding additional remote agents in the remote agent installation guide.

Using local agents

Why use local agents?

If your license doesn't allow the addition of any more remote agents, then adding a small number of local agents can also help. A sound strategy is to add one or two local agents in the first instance, then evaluate the effect they have had on your build wait times.

Remember: too many local agents can start to impact Bamboo's performance because local agents run inside the same JVM as Bamboo itself. Unless you have 8 cores and 64GB RAM, ~3 local agents is about as many as you can accommodate comfortably.

Bamboo Server share permissions and accesses rights with with local agents. Keep in mind that by using local agents in your environment, you're giving other Bamboo users access to potentially sensitive information you might be storing on the server.

Adding local agents

To add local agents, go to  > **Build resources** > **Agents** > **Add local agent**.

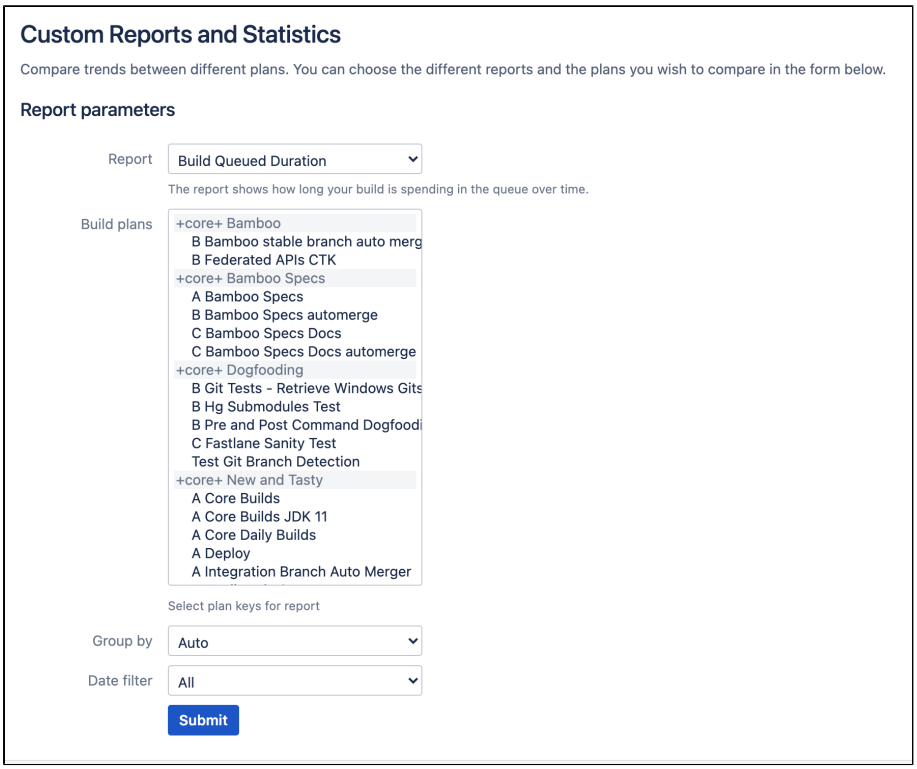
Learn more about adding additional local agents in the Creating a local agent guide.

Monitoring agents

Build Queued Duration

The Build Queued Duration report shows how long each build is spending in the build queue, and is an important tool for evaluating build wait times. The build queued duration report also allows you to compare build wait time between different plans.

You can access the report by selecting **Reports** > **Reports** > **Build Queued Duration**, and selecting the appropriate build plan for analysis.



Bamboo integrations

Add my product to this page

BrowserStack

BrowserStack is a cross-browser testing tool, used to extensively test public websites and protected servers, on real mobile and desktop browsers. The infrastructure consists of servers and mobile device cloud across the globe which can be used for interactive, Selenium and JavaScript testing.

Integrate Bamboo with BrowseStack

GitLab

GitLab is a web-based Git repository manager with wiki and issue tracking features, using an open source license, developed by GitLab Inc.

Integrate Bamboo with GitLab

JFrog

JFrog provides solutions to automate software package management from development to distribution. JFrog Artifactory is an artifact repository manager that fully supports software packages created by any language or technology. JFrog Bintray gives developers full control over how they store, publish, download, promote and distribute software with advanced features that automate the software distribution process. With JFrog, build managers can push their build info and artifacts directly to Artifactory and Bintray.

Integrate Bamboo with JFrog

Octopus Deploy

Octopus is a deployment and automation tool that works with your build server to enable reliable, secure, automated releases of ASP.NET applications and Windows Services into test, staging and production environments, whether they are in the cloud or on-premises.

Integrate Bamboo with Octopus Deploy

Sauce Labs

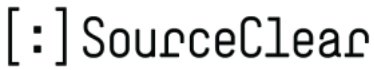
Sauce Labs provides a cloud based platform for the automated testing of web and mobile applications. Optimized for Continuous Integration (CI) and Continuous Delivery (CD), Sauce Labs eliminates the time and expense of maintaining an in house testing infrastructure, freeing development teams of any size to innovate and release better software, faster.

Integrated Bamboo with Sauce Labs

Sonatype

With more than 100,000 installations, companies around the globe use Sonatype's Nexus solutions to manage reusable components and improve the security, quality and speed of their software supply chains.

Integrate Bamboo with Sonatype



SourceClear

SourceClear provides automatic vulnerability detection for your open source dependencies that fits perfectly into your workflow. Detection is available for Java, Python, Ruby, Node, and JavaScript projects.

Integrated Bamboo with SourceClear

Visual Studio

Visual Studio is a complete set of development tools for building ASP. NET Web applications, XML Web Services, desktop applications, and mobile applications. Visual Basic, Visual C#, and Visual C++ all use the same integrated development environment (IDE), which enables tool sharing and eases the creation of mixed-language solutions. In addition, these languages use the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web Services.

Integrate Bamboo with Visual Studio

Zephyr

Zephyr's real-time test management products enable development and QA teams to ship high quality software on time in 100 countries.

Integrate Bamboo with Zephyr

Administering Bamboo

Bamboo is a continuous integration (CI) and deployment server. Bamboo assists software development teams by providing:

- automated building and testing of software source-code status.
- updates on successful/failed builds.
- reporting tools for statistical analysis.
- visibility into, and control over, release artifacts and environments.

This administration guide has information about managing the Bamboo server itself. Please see Using Bamboo for help with setting up CI builds and deployments.

Administering

System settings
Configuring the Bamboo server.

Agents and capabilities
Setting up services, including Elastic Bamboo, to perform builds.

Users and permissions
Managing users, groups and their permissions.

Apps
Extending Bamboo.

Data and backups
Managing databases, data and backups.

Security
Managing security for agents and Elastic Bamboo.

Installing

Installing Bamboo on Linux

Bamboo installation guide for Mac OS X

Installing Bamboo on Windows

Connect Bamboo to an external database

Bamboo remote agent installation guide

Supported platforms

See also

Getting started

Using Bamboo

Bamboo Release Notes

Bamboo security advisories

System settings

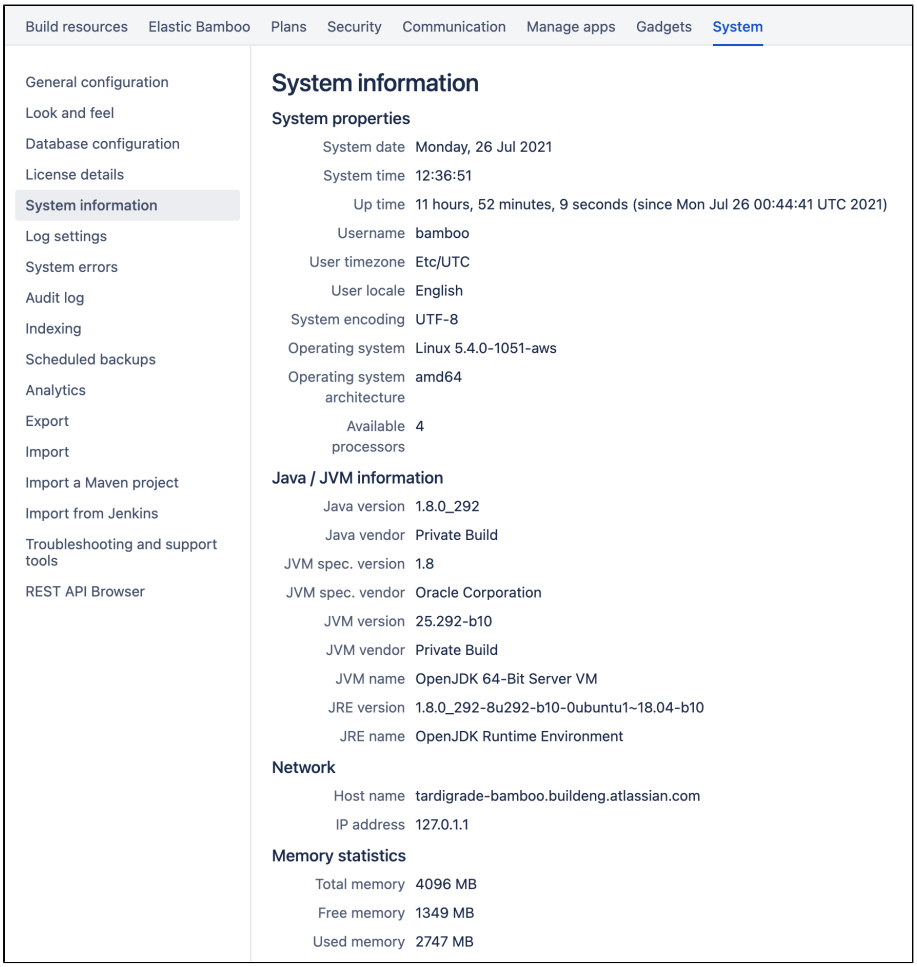
You can view system information for Bamboo from the administration console.

The system information contains useful data for you to send to Atlassian when requesting support.

See Locating important directories and files for more information.

Viewing your Bamboo system information

Go to  > **System** > **System information**.



Configuring system settings

For information on configuring system settings, see the following topics:

- Updating your Bamboo license details
- Specifying Bamboo's title
- Specifying Bamboo's URL
- Logging in Bamboo
- Enabling GZIP compression
- Enabling Bamboo's Remote API
- Starting Bamboo
- Configuring your system properties
- Configuring Gravatar support
- Tracking changes to your Bamboo server
- Customizing Bamboo headers
- Globally disabling the repository quiet period

Updating your Bamboo license details

⚠ **Before updating the license:**

- It is recommended to back up the `license.string` contents found on <bamboo-home> /bamboo.cfg.xml before attempting any license changes. This is particularly useful if the applied license needs to be rolled back.
- Some vendors would require different licenses once the main Bamboo one is modified. Suppose any custom apps or plugins are used in Bamboo. In that case, we recommend that **before updating any new licenses**, reach out to every third-party vendor and **validate if any of the plugins would require additional licenses** after the new Bamboo license is applied.

When you upgrade or renew your Bamboo license, you will receive a new license key. You will need to update your Bamboo server with the new license key.

See the Licensing FAQ if you have questions about licensing.

Related pages:

- System settings

To update your Bamboo license key:

1. Go to  > **System** > **License details**.
2. Paste your new license into **License key**.
3. Select **Save new license**.

When using Bamboo DC in an active-standby cluster:

Bamboo standby nodes will not update their license keys automatically. Modify the `license.string` contents found on `<bamboo-home>/bamboo.cfg.xml` manually and restart Bamboo on the standby node.

Specifying Bamboo's title

Bamboo's *name* is the displayed title of this installation of Bamboo. It will appear throughout Bamboo (e.g. on the Dashboard), and in the window title of your users' browsers.

Related pages:

- System settings

To specify Bamboo's title:

1. Go to  > **System** > **General configuration**.
2. Type the display title for your Bamboo server (e.g. MyCompany's Bamboo) into the **Name** field.
3. Select **Save**.

Specifying Bamboo's URL

This is the base URL of this installation of Bamboo. All links created (for links in Bamboo email notifications etc.) will be prefixed by this URL.

To specify Bamboo's URL:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **System**, select **General configuration**.
3. In the **Base URL** field, type the URL address of your Bamboo server (for example, "http://keg:8080/bamboo").
4. Select **Save**.

Related pages:

- System settings

Notes

- **Accessing Bamboo from Outside a Firewall** — When accessing Bamboo through a web browser, most Bamboo URL links (which provide navigation throughout the product) will use the base URL that was originally entered into your browser's URL field. For example, to access Bamboo through a web browser on the same machine running Bamboo itself, you may have entered the base URL:

  ```
  http://localhost:8085/...
  ```

  into your browser's URL field. Consequently, most Bamboo URL links will use the base URL:

  ```
  http://localhost:8085/...
  ```

  However, URL links to a Bamboo instance that are provided in Bamboo email notifications and by some Bamboo apps, will use the base URL set on this General configuration page. Hence, if you configure the **Base URL** field above to one that can only be accessed internally, behind a firewall, then you may have problems accessing this Bamboo instance externally.

Logging in Bamboo

Bamboo generates the following sets of logs:

Build logs
The build logs are generated each time a plan is executed. All information specific to the build is stored in these logs, which can be downloaded as an artifact (see Viewing a build's artifacts). You cannot change the logging configuration for the build logs.

The build logs are located in the `<Bamboo-Home>/shared/builds/` sub-directories.

On this page:

Related pages:

- System settings
- Viewing a build's artifacts
- Locating important directories and files

Bamboo server logs

Bamboo records all server activity in the `atlassian-bamboo.log`. The location of the `atlassian-bamboo.log` file can be viewed in Bamboo's System information under the Bamboo paths section.

In the case of a Tomcat webapp deployment, the logs are piped out to catalina.out file.

atlassian-bamboo logs for elastic agents

Elastic agent activity is logged inside the elastic instance where the elastic agent runs. To access the elastic agent logs (atlassian-bamboo.log and bamboo-elastic-agent.out) use ssh to log in to your elastic instance as described in Viewing an elastic instance and retrieve the logs.

atlassian-bamboo logs for remote agents

All agent activity is recorded in `atlassian-bamboo-agent.log` file stored on the agent machine. These are generated in the running directory of the agent. The running directory can be viewed in the remote agent's system properties under the Bamboo paths section.

See Locating important directories and files for information on where to find other important files in Bamboo.

Configuring the level of logging on the Bamboo server

Bamboo uses the Log4j – Apache Log4j 2 for logging during runtime. The logging levels can be changed by editing the `<Bamboo-Install>/atlassian-bamboo/WEB-INF/classes/log4j2.properties` file. There are five logging levels available: 'DEBUG', 'INFO', 'WARN', 'ERROR', and 'FATAL'. Each logging level provides more logging information that the level after it:
`DEBUG > INFO > WARN > ERROR > FATAL`
i.e. `DEBUG` provides the most verbose logging and `FATAL` provides the least verbose logging.

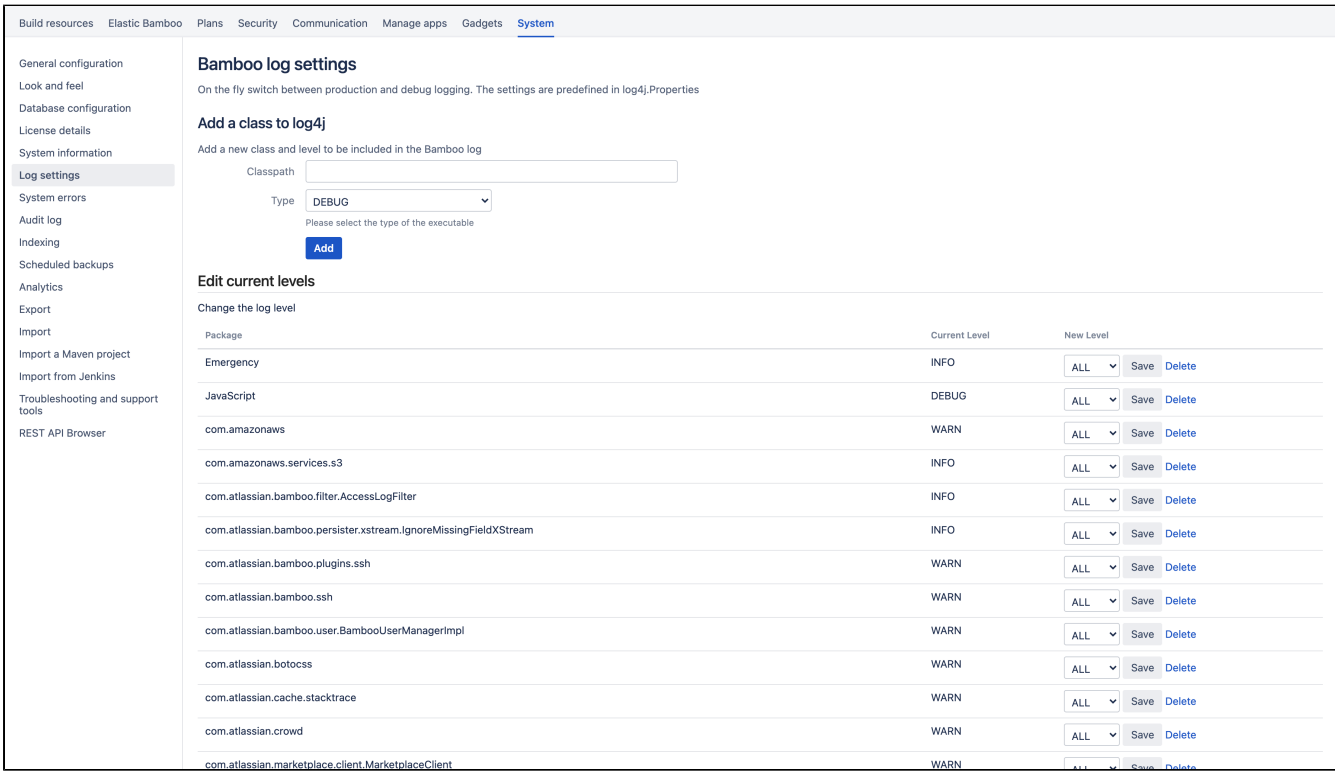
You can adjust the logging levels for the different Bamboo packages on the fly, using the runtime log4j2 configuration tool in the Bamboo administration console. The default log settings are still stored in the log4j2.properties file. When you view the log settings page for the first time you will see the default log settings as defined in log4j2.properties. All changes to the log settings via the runtime log4j2 configuration tool *will not be persisted* and are valid during Bamboo runtime only.

Before you begin:

- Note that you don't need to restart your Bamboo server for any logging changes to take effect.

Change the level of logging on your Bamboo server

1. Go to  > **System** > **Log settings**.
2. The Bamboo log settings page will display showing the Bamboo packages being logged.
    - To change the logging level of a package that is already being logged, locate the Bamboo package, select the desired logging level from the list next to it, and select **Save**.
    - To start monitoring a package in the Bamboo logs, enter the class name in the text box at the top of the page, select the desired logging level from the list next to it, and select **Add**.
    - To stop logging a package, locate the Bamboo package and select **Delete** next to it.



Configuring the level of logging on remote agents

Configuring the level of logging on remote agents depends on your Bamboo version. For more information, see How to increase debug logging to investigate Agent problems.

Configuring the location of the atlassian-bamboo logs

To change the directory that the `atlassian-bamboo` logs are generated to, you must set the environment variable for the target location of the logs, as seen below:

```
appender.filelog.fileName=/my/path/to/atlassian-bamboo.log
```

Note that the new log file location applies to both the server and remote agents. If using an absolute path this may result in aggregated logs.

Note: If you change the location of your log files, they will no longer be included when you generate a support zip. This means you'll need to attach your logs to any support requests manually.

Verbose mode

Starting from version 7.2, less data is being logged in Bamboo by default to limit the amount of noise that may get in your way so that you can process and analyze information stored in logs more efficiently. If you want Bamboo to start logging additional data, you must manually enable the verbose mode. With the verbose mode enabled, Bamboo logs information on environment variables and logs coming from VCS, which is omitted by default.

To enable verbose mode

You can enable the verbose mode in two cases:

- when you run a customized plan
- when you start a manual deployment

> Job reruns are always run in verbose mode.

Enabling GZIP compression

You can enable GZIP compression in order to reduce the size of Bamboo's web pages. This is useful if Bamboo is being run over slow networks. There is a slight performance penalty, and note that GZIP may not work for languages other than English.

Related pages:

- System settings

To enable GZIP compression:

1. Go to  > **System** > **General configuration**.
2. Select **Apply gzip compression to reduce the size of Bamboo's web pages?**.
3. Select **Save**.

Enabling Bamboo's Remote API

Note that the Bamboo Remote API *has been deprecated* in favor of the new Bamboo REST API.

You can access Bamboo's data from an external program by using Bamboo's REST-style remote API.

Starting Bamboo

Configuring Bamboo system properties

The default settings on a number of Bamboo functions can be configured by setting the appropriate system properties.

Bamboo on UNIX-based operating systems (such as Solaris, Linux or Mac OS X) can be started by using the setenv.sh script.

Bamboo on Windows-based operating systems can be started by running the `setenv.bat` file from the command line (which is the same as running the Start in console option from the Windows Start menu) or as a Windows Service.

On this page:

Related pages:

- System settings
- Configuring your system properties

Please see Configuring your system properties for more information on configuring your Bamboo system properties.

Configuring your system properties

This page describes how to set Java properties and options on startup for Bamboo.

On this page:

Linux

To configure system properties in Linux installations:

1. From `<bamboo-install>/bin`, open **setenv.sh**.
2. Find the section **JVM_SUPPORT_RECOMMENDED_ARGS=**.
3. Refer to the list of parameters below.

> Add all parameters in a space-separated list, inside the quotations.

Windows (starting from .bat file)

To configure system properties in Windows installations when starting from the .bat file:

1. From `<bamboo-install>/bin`, open **setenv.bat**.
2. Find the section **set JVM_SUPPORT_RECOMMENDED_ARGS=**
3. Refer to the list of parameters below.

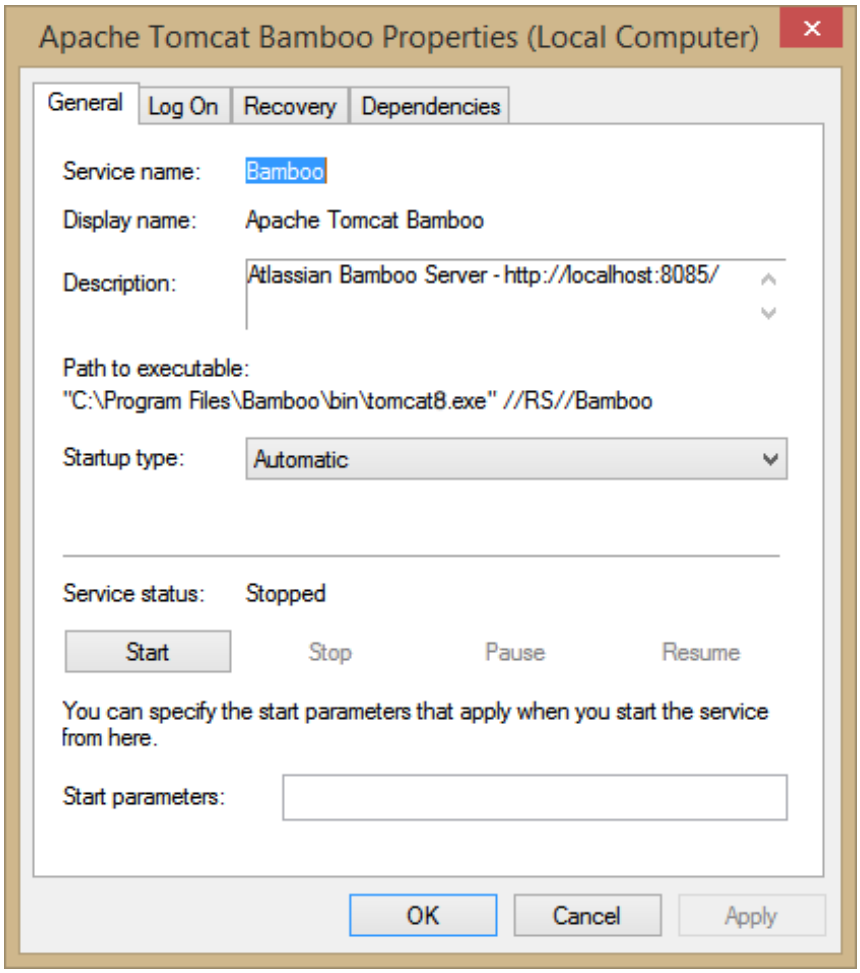
> Add all parameters in a space-separated list, inside the quotations.

Windows service

There are two ways to configure system properties when starting Bamboo as a service — either in the command prompt or in the Windows registry.

Setting properties for Windows services from the command line

1. Identify the name of the service that Bamboo is installed as in Windows (**Control panel** > **Administrative tools** > **Services**):



2. Open a command prompt by selecting **Start** > **Run**.
3. In the **Run** window, type `cmd` and press the **Enter** key.
4. In the command prompt, navigate to the `bin` subdirectory of your Bamboo installation directory.
5. Run the following command:

```
bamboow.exe //ES//%SERVICENAME%
```

> Following the example above, the full command would be:
>
> ```
> bamboow.exe //ES//Bamboo
> ```

6. Select the **Java** tab to see the list of current start-up options.
7. Append any new option on its own new line by adding to the end of the existing Java Options. Refer to the list of parameters below.

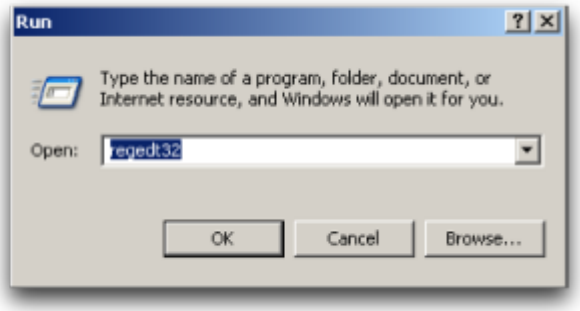
> If you want to change the heap size configured for the JVM, use the **Initial memory pool** and **Maximum memory pool** fields instead of adding the -Xms and -Xmx parameters to the list of Java options.

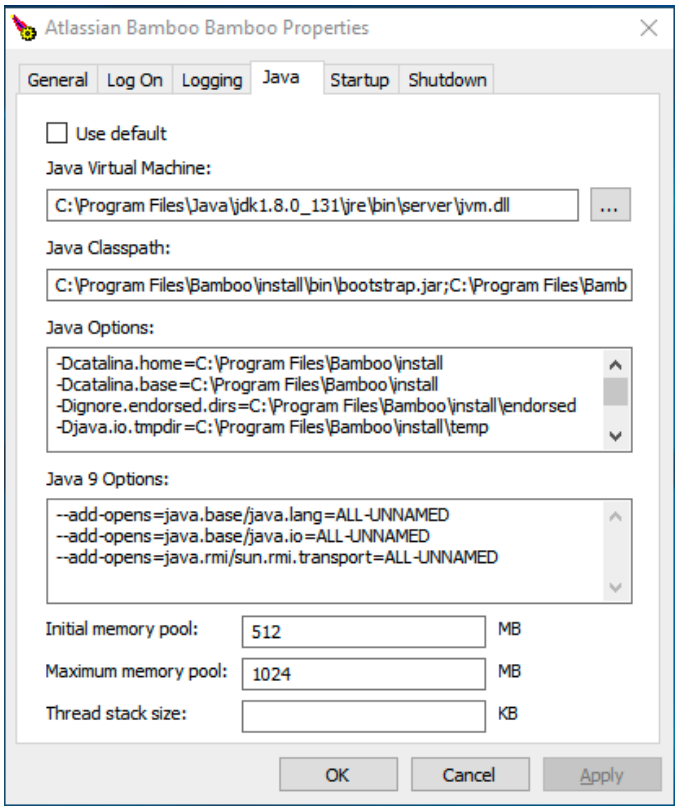
Setting properties for Windows services using the Windows registry

In some versions of Windows, there is no option to add Java variables to the service. In these cases, you must add the properties by viewing the option list in the registry.

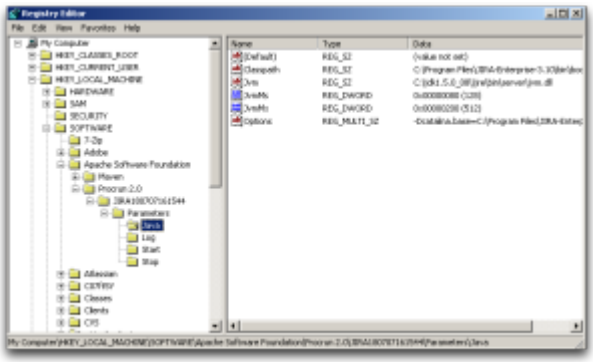
To set properties for Windows services using the Windows registry

1. Go to **Start** > **Run**, and run "regedit32.exe".

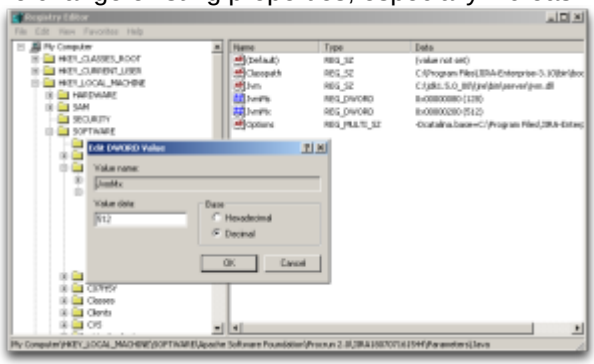




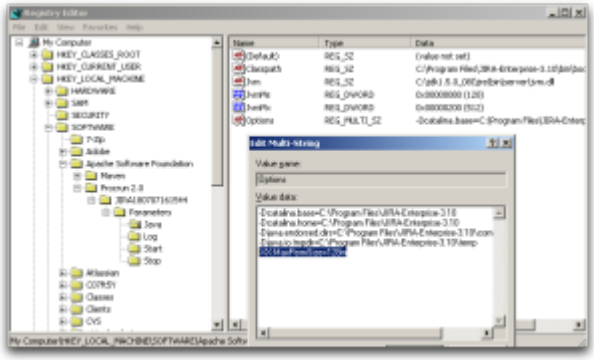
2. Find the Services entry:
32-bit: `HKEY_LOCAL_MACHINE >> SOFTWARE >> Apache Software Foundation >> Procrun 2.0 >> Bamboo`
64-bit: `HKEY_LOCAL_MACHINE >> SOFTWARE >> Wow6432Node >> Apache Software Foundation >> Procrun 2.0 >> Bamboo`



3. To change existing properties, especially increasing Xmx memory, double-click the appropriate value.



4. To change additional properties, double-click options.



5. Refer to the list of parameters below. Enter each on a separate line.

Changing the Bamboo start port

1. Stop Bamboo.
2. Edit `<Bamboo install directory>/conf/server.xml`
3. Update the following so that `Connector port` is set to the port value you require:

```
<Service name="Catalina">
       <Connector port="8085"
                  maxThreads="150"
                  minSpareThreads="25"
                  connectionTimeout="20000"
                  enableLookups="false"
                  maxHttpHeaderSize="8192"
                  protocol="HTTP/1.1"
                  useBodyEncodingForURI="true"
                  redirectPort="8443"
                  acceptCount="100"
                  disableUploadTimeout="true"/>
```

4. Restart Bamboo.

List of startup parameters

Memory Property	Notes	Related Pages
-Xmx -Xms XX:MaxPermSize	These properties are pre-existing. See related pages for instructions.	Tuning the Java heap
-XX:+PrintGCTimeStamps -verbose:gc -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError	Set these for Garbage Collection tuning.	Tuning Java VM garbage collection

Configuring Gravatar support

Bamboo is configured to support Gravatars by default. This means that Bamboo will attempt to use user's emails to retrieve profile pictures from the Gravatar service. The profile pictures will be displayed against user activity, e.g. comments, in Bamboo.

Related pages:

- System settings

Enabling Gravatar support:

⚠ You must have set up an external Gravatar server if you want to specify your own server

1. Go to  > **General configuration**.
2. Select the **Enable gravatar support** checkbox.
3. Enter the URL of your Gravatar server in the URL field, or leave as default if you wish to use the default Gravatar service.
4. Select **Save**.

Disabling Gravatar support:

1. Go to  > **General configuration**.
2. Uncheck the **Enable gravatar support** checkbox.
3. Select **Save**.

Tracking changes to your Bamboo server

Tracking configuration changes

You can track changes to the configuration of your Bamboo server, as well as track changes to any plans it may be running.

To track changes, you must enable Audit logging. To enable Audit logging:

1. Go to  > **System** > **Audit log**.
2. Select **Enable audit logging**.

The Audit log will record details of changes made to the configuration of the Bamboo server. It will record:

- the time and date
- the user
- the changed field
- the old value and
- the new value

The Audit log does not record change of permissions.

Audit logging will also record details of changes made to any plans, including:

- Plan branch creation
- Plan deletion

Related pages:

- System settings

Deleting Audit Logs

You may wish to delete audit logs, particularly when the plans or configuration changes have expired.

To delete your configuration change history, select **Delete all global audit logs**.

To delete all audit logs, including any plan audit logs, select **Delete all audit logs**.

Customizing Bamboo headers

Starting from version 7.0, you can customize your Bamboo instance's look and feel. You can change the color of Bamboo header, the **Create** button, set a custom logo in Bamboo header, and set a custom favicon. This can come in handy when navigating among multiple Bamboo instances or if you want to introduce your own branding in Bamboo.

Before you begin

To change the logo and favicon in Bamboo, you must have access to the Bamboo home folder.

To customize Bamboo header

1. Go to  > **System** > **Look and feel**.
2. To customize Bamboo logo or favicon:
    a. Place your image and icon files in the following directory:

    ```
    <Bamboo_home>/attachments/logos/
    ```

    > The image file name must be `bamboo-logo.png`.
    > The icon file name must be `bamboo-favicon.ico`.

    b. Refresh you web browser cache.
3. To customize Bamboo colors:
    a. Select the Bamboo header color from the color picker, or insert a hexadecimal color value.
4. Select **Save**.

Globally disabling the repository quiet period

The quiet period is a configurable delay between detecting a commit having been pushed to a repository and starting a build. This delay allows multiple commits to be aggregated into one build instead of triggering a build after each commit. You can set a quiet period for each linked repository individually.

While the quiet period is in effect for a build, there isn't any indication of it on the **Build dashboard** and the build itself is not yet in the build queue. However, the build status will appear as "Queued". If this behavior is causing confusion, you can disable the quiet period globally for all repositories.

To disable the quiet period globally across all linked repositories:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. In the left panel, under **System**, select **Global configuration**.
3. Under **Global system configuration**, select the **Disable quiet period** checkbox.

Agents and capabilities

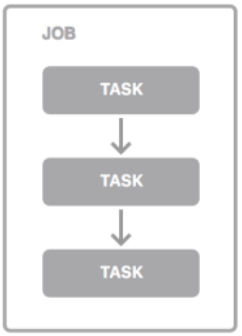
An agent can run a job if its capabilities match the requirements of a job. Each job inherits the requirements from individual tasks that it contains.

On this page:

Related pages:

- Configuring agents
- Configuring capabilities
- Remote agents
- Ephemeral agents

Capablities



You can define the following capabilities for an agent:

- an executable (e.g. Maven)
- a JDK
- a Version Control System client application (e.g. Git)
- a custom capability. This is a key-value property which defines a particular characteristic of an agent (e.g. 'operating. system=WindowsXP' or 'fast.builds=true').

Capabilities typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of those.

Capabilities can be defined specifically for an agent, or they can be shared between all remote agents. Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

See also:

- Configuring capabilities
- Viewing a capability's agents and jobs
- About capabilities and requirements

Viewing the agents and plans related to a capability

To view the agents and plans related to a capability, see Viewing a capability's agents and jobs.

Configuring agents

A Bamboo *agent* is a service that can run job builds. There are the following types of Bamboo agents:

- *local agents* run as part of the Bamboo server.
- *remote agents* run on computers, other than the Bamboo server, that run the remote agent tool.
- elastic agents run in the Amazon Elastic Compute Cloud (EC2).

Local agents run in the Bamboo server's process, i.e. in the same JVM as the server. Each remote agent runs in its own process, i.e. has its own JVM.

Each agent has a defined set of capabilities and can only run builds for jobs whose requirements match the agent's capabilities.

If you are looking for information on **elastic agents**, please refer to the documentation on Working with Elastic Bamboo.

On this page:

Creating a new agent

To create a new agent, see:

- Creating a local agent, or
- Creating a remote agent.

Configuring an agent's capabilities

To configure an existing agent's capabilities, see:

- Configuring capabilities
- Configuring remote agent capabilities

Disabling or deleting an agent

To disable or delete an agent, see Disabling or deleting an agent.

Notes

- A *capability* is a feature of an agent. A capability can be defined on an agent for:
  - an executable (e.g. Maven)
  - a JDK
  - a Version Control System client application (e.g. Git)
  - a custom capability. This is a key-value property which defines a particular characteristic of an agent (e.g. 'operating.system=WindowsXP' or 'fast.builds=true').

  Capabilities typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of those.

  Capabilities can be defined specifically for an agent, or they can be shared between either all local agents or all remote agents. Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

Viewing a Bamboo agent's details

A Bamboo *agent* is a service that can run job builds. There are the following types of Bamboo agents:

- *local agents* run as part of the Bamboo server.
- *remote agents* run on computers, other than the Bamboo server, that run the remote agent tool.
- elastic agents run in the Amazon Elastic Compute Cloud (EC2).

Local agents run in the Bamboo server's process, i.e. in the same JVM as the server. Each remote agent runs in its own process, i.e. has its own JVM.

Each agent has a defined set of capabilities and can only run builds for jobs whose requirements match the agent's capabilities.

On this page:

Viewing an agent's details

To view an agent's details:

1. Go to  > **Build resources** > **Agents.**
2. Select the name of the desired agent. The agent's page will be displayed.
3. Select one of the following tabs to see corresponding details for the agent:

   Capabilities
   Displays a list of all agent-specific and shared capabilities. The capabilities in each of those sections are grouped into the following subsections:

   - Custom — custom capabilities
   - Executable — executable capabilities
   - JDK — JDK capabilities
   - Perforce**,** Mercurial**,** Git — VCS capability

     You'll only see a subsection if a capability of that type is defined in Bamboo. To define a new capability, see Configuring capabilities.

   Executable jobs
   Displays a list of jobs, arranged by plan, that the agent can build.

   System properties
   Displays information about the agent.

   Audit logs
   Displays a record of changes that have been made to the agent.

Viewing the agents that can build jobs

To view which agents can build which jobs:

1. Go to  > **Build resources** > **Agent matrix**.

Editing an agent's name or description

To edit an agent's name or description:

1. Navigate to the desired agent, as described above.
2. Select **Edit details**.
3. Update the details for the agent.
4. Select **Save**.

Creating a local agent

⚠ Local agents are no longer supported in Bamboo Data Center. Because of that we decided to standardize the naming and from now on we will refer to remote agents as simply **agents**.

A Bamboo *agent* is a service that can run job builds. There are the following types of Bamboo agents:

- *local agents* run as part of the Bamboo server.
- *remote agents* run on computers, other than the Bamboo server, that run the remote agent tool.
- elastic agents run in the Amazon Elastic Compute Cloud (EC2).

Local agents run in the Bamboo server's process, i.e. in the same JVM as the server. Each remote agent runs in its own process, i.e. has its own JVM.

Each agent has a defined set of capabilities and can only run builds for jobs whose requirements match the agent's capabilities.

Related pages:

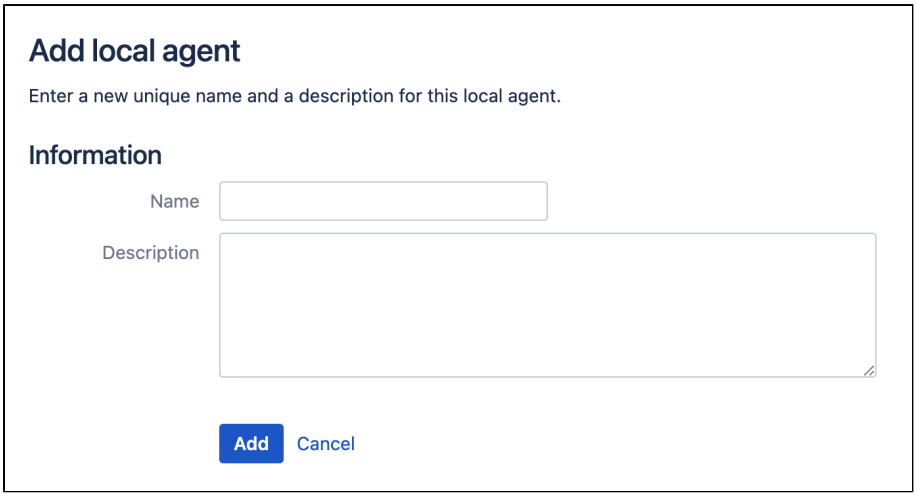
- Configuring agents
- Bamboo remote agent installation guide

To create a new local agent:

1. Go to ⚙ > **Build resources** > **Agents**.
2. Select **Add local agent**.
3. Enter details for the agent. The name is displayed on the dashboard. The description is only visible to administrators.
4. Select **Add**.

Note that your new local agent:

- will be enabled by default.
- will inherit all *local server capabilities* that are defined in your Bamboo system.
- will be able to run builds for all jobs whose requirements are met by the agent's capabilities (see Configuring a job's requirements).



Disabling or deleting an agent

Bamboo allows you to disable or delete an agent, to prevent that agent from running any further builds.

- **Disabling an agent** lets you keep the agent in Bamboo, but stops it from running builds.
  If you need to prevent Bamboo from building any plans at all (e.g. while you re-index Bamboo), you can disable all agents. By doing so, all builds will wait in the queue until you re-enable the agents.
- **Deleting an agent** removes it from Bamboo altogether. If you need to use the agent again in future, you will need to recreate it (see Creating a local agent and Creating a remote agent for more information).

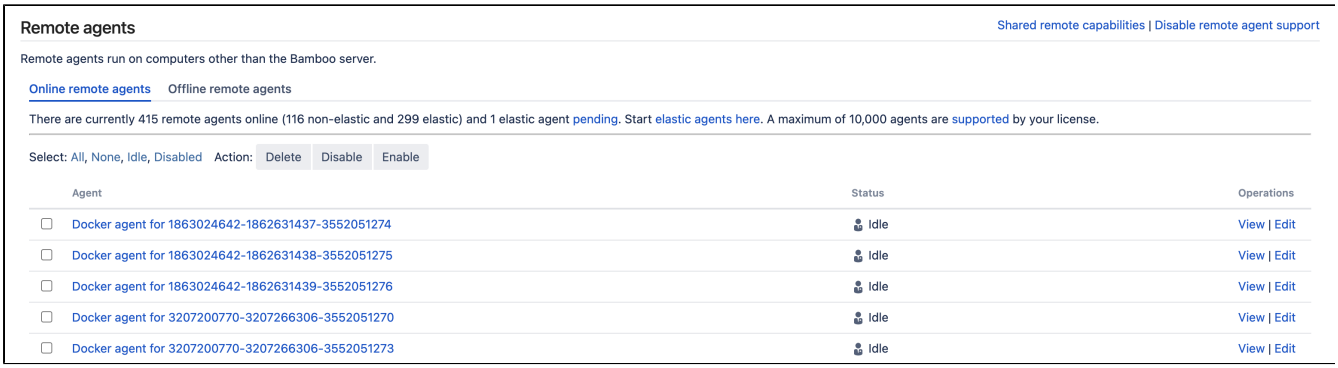
Note that you can also delete/disable individual plans and/or their jobs. This prevents the plans and/or their jobs from being submitted to the build queue. See Disabling or deleting a plan and Disabling or deleting a job.

Related pages:

- Disabling or deleting a plan
- Disabling or deleting a job
- Creating a remote agent
-
    - Creating a local agent

To disable (or delete) an agent:

1. Go to ⚙ > **Build resources** > **Agents**. A list all agents that currently exist in your Bamboo system will be displayed. The Status column indicates which agents are currently enabled or disabled. Scroll down if you require remote agents.
2. Select the checkbox for the agent (or agents) you wish to disable or delete.
3. Select the **Disable** (or **Delete)** button above the table.



Dedicating an agent

Bamboo allows you to dedicate an agent, to run, for example, only specific build projects, deployment projects or associated activities. You must dedicate an agent to a type of activity and a specific entity of that type.

Note that when you dedicate an agent, then no other activity will be able to use it, unless it is dedicated to that activity as well. Similarly, no other agent will be able to build this plan other than the agent(s) that are dedicated to build it.

Related pages:

- Disabling or deleting an agent

To dedicate an agent:

1. Go to  > **Build resources** > **Agents**. This displays all local and remote agents currently available to your Bamboo system. The Status column indicates which agents are currently enabled or disabled.
   Select **Image configurations** for EC2 elastic agents.
2. Select the agent you wish to dedicate, and select the **Dedicate agent** tab.
3. Using the menu, select the type of activity you wish to dedicate the agent to. Available choices are:
   - Build project
   - Build plan
   - Build job
   - Deployment project
   - Deployment environment
4. Select an entity to assign the dedicated agent to. This will be a specific project, plan, job, or environment. Use the menu or type-ahead field to locate a suitable entity.
5. If required, select **Add** for an additional entity dedication.
6. Select **Save** to dedicate your agent.

Monitoring agent status

You can monitor your agents' status to check that all agents are functioning as expected.

Online versus Offline agents:

- An Online agent is an agent which is currently available for use by Bamboo. Local agents are always online, although remote agents may be either online or offline.
- An Offline agent is a remote agent which has been registered with the Bamboo server, was online but is now unavailable for builds because:
  - The Bamboo remote agent process (running on the remote hardware) was stopped.
  - The Bamboo server (for whatever reason) cannot communicate with the remote hardware that is running the Bamboo remote agent process.

Bamboo administrators can manually disable an online agent to prevent it from being used in build generation. The agent will still be online and it can be enabled at a later point in time. It is not possible to disable offline agents.

Related pages:

- Bamboo remote agent installation guide

To monitor the status of your agents:

1. Go to  > **Build resources** > **Agents**. This will display the Agents screen, showing lists of all local agents and all remote agents that currently exist in your Bamboo system.
   Agents can have one on the following statuses:

   Idle
   Available to execute builds.

   Building
   Currently executing a build.

   Canceling
   Currently canceling a Job build.

   Disabled
   Not available to execute builds (see Disabling or deleting an agent).

   Disabled - Building
   Currently executing a build but disabled so cannot execute further builds.

   Disabled - Canceling
   Currently canceling a build, and disabled so cannot execute further builds.

Note that to see the jobs that are currently being built, look at the **Current Activity** tab on the dashboard.

Configuring capabilities

A *capability* is a feature of an agent. A capability can be defined on an agent for:

- an executable (e.g. Maven)
- a JDK
- a Version Control System client application (e.g. Git)
- a custom capability. This is a key-value property which defines a particular characteristic of an agent (e.g. 'operating.system=WindowsXP' or 'fast.builds=true').

Capabilities typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of those.

Capabilities can be defined specifically for an agent, or they can be shared between either all local agents or all remote agents. Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

See also Configuring agents.

On this page:

Defining a new capability

To define a new capability, see:

- Defining a new executable capability
- Defining a new JDK capability
- Defining a new custom capability
- Defining a new version control capability
- Defining a new Docker capability

Editing and deleting a capability

To edit an existing capability, see Modifying and deleting capabilities.

Renaming a capability

To rename an existing capability, see Renaming a capability.

Notes

- A *requirement* is specified in a job or a task. A requirement specifies a capability that an agent must have for it to build that job or task. A job inherits all of the requirements specified in its tasks.

  Together, capabilities and requirements control which agents can execute builds for particular jobs. Each job can only be built by agents whose capabilities match the job's requirements.

  See Configuring a job's requirements for more information.

About capabilities and requirements

A *capability* is a feature of an agent. A capability can be defined on an agent for:

- an executable (e.g. Maven)
- a JDK
- a Version Control System client application (e.g. Git)
- a custom capability. This is a key-value property which defines a particular characteristic of an agent (e.g. 'operating.system=WindowsXP' or 'fast.builds=true').

Capabilities typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of those.

Capabilities can be defined specifically for an agent, or they can be shared between either all local agents or all remote agents. Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

See Configuring capabilities for more information.

On this page:

Related pages:

- Configuring capabilities
- Configuring agents
- Remote agents
- Working with Elastic Bamboo

How do capabilities work with requirements?

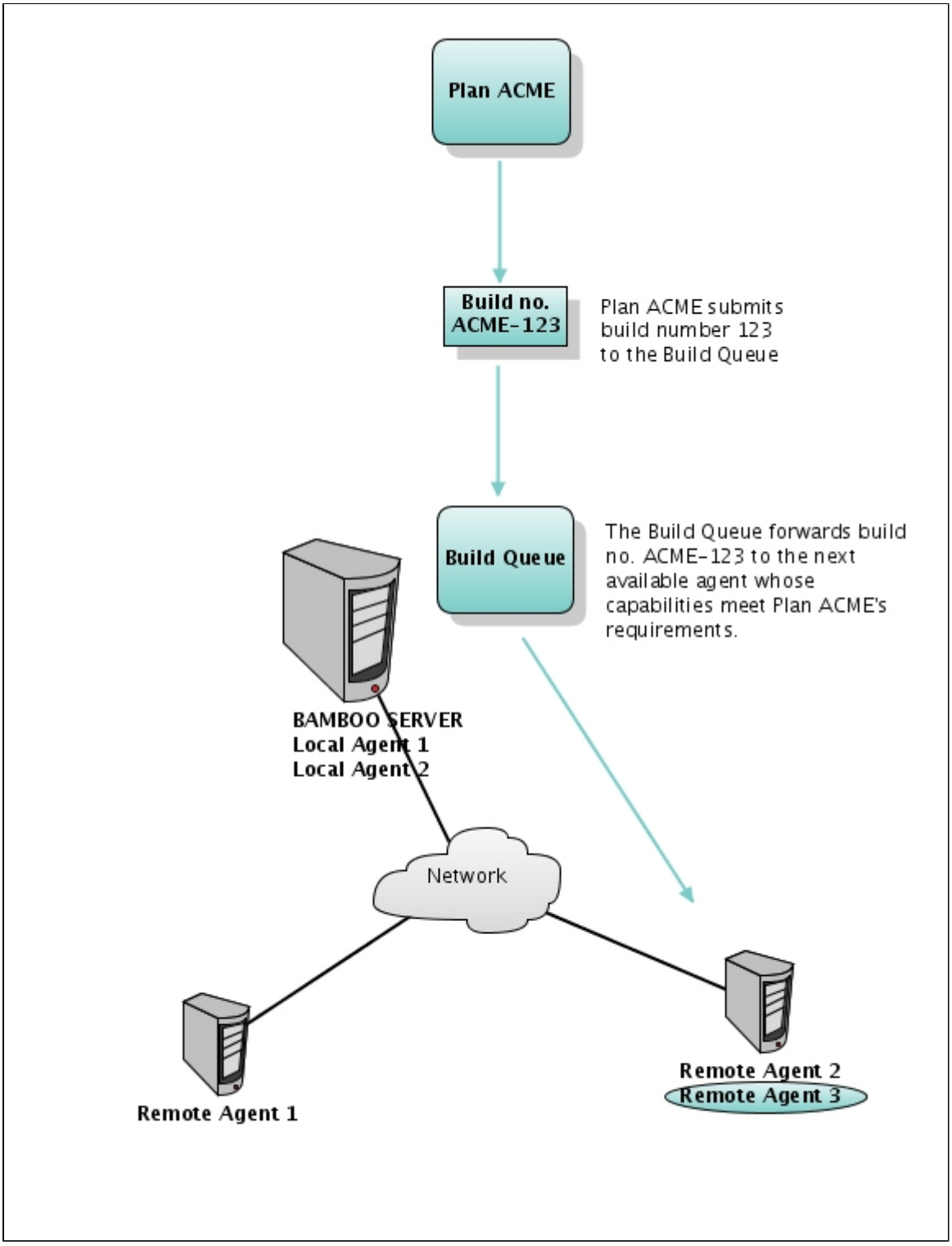
A *requirement* is specified in a job or a task. A requirement specifies a capability that an agent must have for it to build that job or task. A job inherits all of the requirements specified in its tasks.

Together, capabilities and requirements control which agents can execute builds for particular jobs. Each job can only be built by agents whose capabilities match the job's requirements.

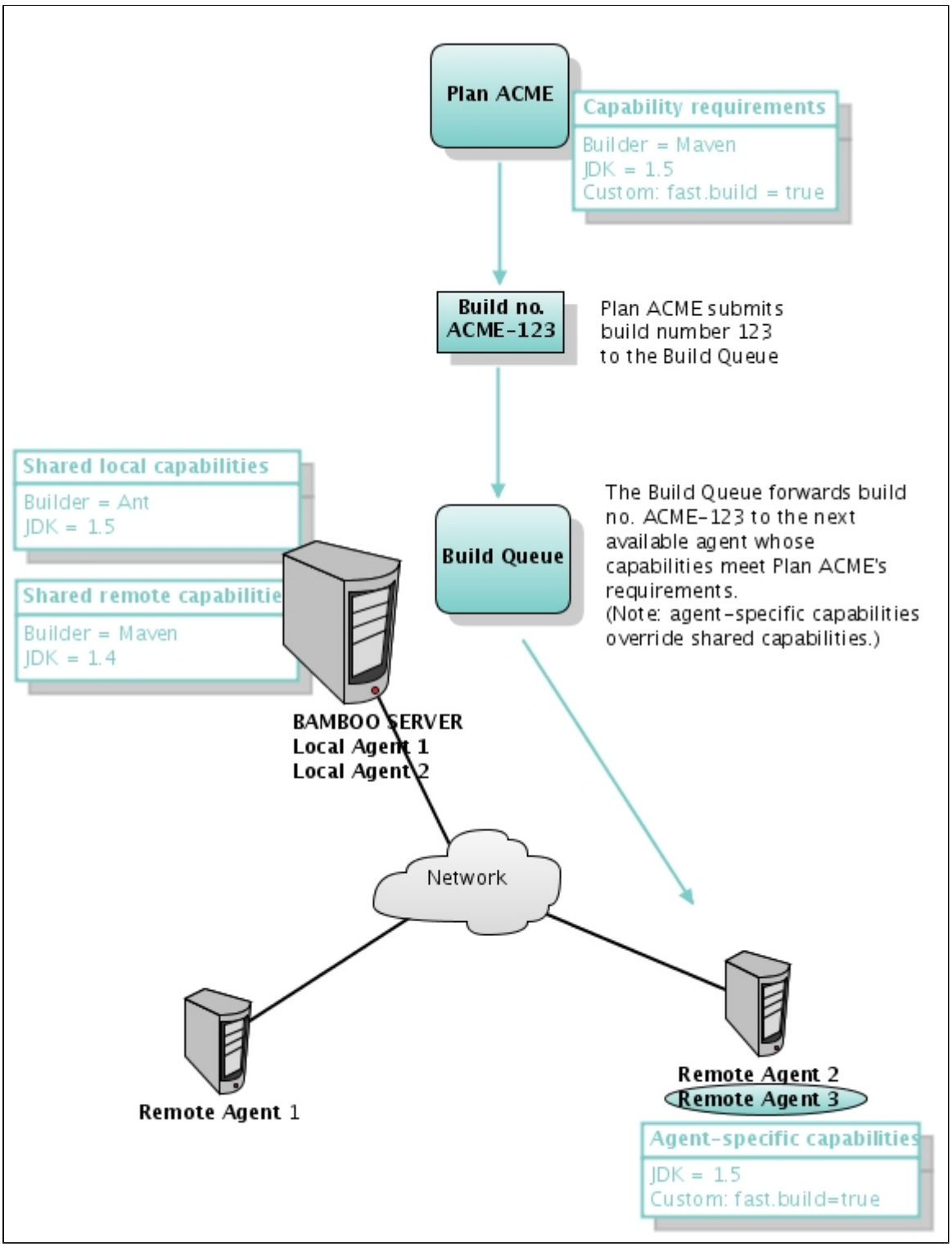
See Configuring a job's requirements for more information.

How are builds distributed to agents?

An agent will consume a single job at a time and will block any other Bamboo jobs from being processed until that job build is complete. If you would like to build multiple jobs simultaneously on the Bamboo server, then simply set up multiple local agents. If the agents are remote, then you will need to install that number of agent instances on the machine. Separate installations are required because each remote agent will need its own home and log directories.



How do capabilities affect the distribution of builds to agents?



Modifying and deleting capabilities

Depending on the capability type, you can edit parameters such as **Path**, **Java Home** and **Value** for the capability.

Note that:

- Because each agent can only run builds for jobs whose requirements are met by the agent's capabilities (see Configuring a job's requirements), modifying or deleting a capability may mean that some plans can no longer be built.
- Renaming a capability involves changing its key. See Renaming a capability .

On this page:

Related pages:

- Configuring capabilities
- Renaming a capability

Modifying an agent-specific capability

To delete an agent-specific capability:

1. Navigate to the desired agent.
2. Select either **Edit** or **Delete** for the capability you wish to modify.

Modifying a local server capability

To delete a local server capability:

1. Go to  > **Build resources** > **Server capabilities**.
2. Select either **Edit** or **Delete** for the capability you wish to modify.

Modifying a shared remote capability

To delete a shared remote capability:

1. Go to  > **Build resources** > **Agents** > **Shared remote capabilities**.
2. Select either **Edit** or **Delete** for the capability you wish to modify.

Renaming a capability

To rename a capability you have to change its key value.

Renaming an agent-specific capability

To rename a capability:

1. Go to  > **Build resources** > **Agents**.
2. Select **View** for the agent that has the capability you wish to rename. A list of agent-specific capabilities and shared capabilities for that agent is displayed.
3. Select **View** for the capability you wish to rename.
4. Select **Rename capability**.
5. Enter a value for **New key** and select **Rename capability**.

On this page:

Related pages:

- Configuring capabilities

Renaming a local server capability

To rename a local server capability:

1. Go to  > **Build resources** > **Server capabilities**.
2. Select **View** for the capability you wish to rename.
3. Select **Rename capability**.
4. Enter a value for **New key** and select **Rename capability**.

Renaming a shared remote capability

To rename a shared remote capability:

1. Go to  > **Build resources** > **Agents** > **Shared remote capabilities**.
2. Select **View** for the capability you wish to rename.
3. Select **Rename capability**.
4. Enter a value for **New key** and select **Rename capability**.

Viewing a capability's agents and jobs

You can view a capability to see the following information about it:

- which agents have/inherit the capability. Select one of the listed agents to show further information about that agent:
  - **Executable jobs** tab — all the jobs whose requirements match the capabilities of this agent
  - **Capabilities** tab — the capabilities of the agent itself
  - **System properties** tab — system information about this agent
  - **Recent activity** link — recent builds for the agent
- which jobs have the capability specified as a requirement.
- which elastic images have this capability and the Bamboo plans that rely on this capability. See also Viewing an elastic image.

On this page:

Related pages:

- Configuring capabilities
- Renaming a capability
- Modifying and deleting capabilities

Viewing an agent-specific capability

To view an agent-specific capability:

1. Navigate to the desired agent.
2. Select the **Capabilities** tab.
3. Select **View** for the capability you wish to view.

Viewing a local server capability

To view a local server capability:

1. Go to  > **Build resources** > **Server capabilities**.
2. Select **View** for the capability you wish to view.

Viewing a shared remote capability

Before you begin:

- Shared remote capabilities are **not shared** with elastic agents.

To view a shared remote capability:

1. Go to  > **Build resources** > **Agents** > **Shared remote capabilities**.
2. Select **View** for the capability you wish to view.

Defining a new executable capability

An executable is an external program that Bamboo uses during the build process. Bamboo supports the following executables:

- Ant
- Maven
- Grails
- NAnt
- devenv.com
- msbuild.exe
- PHPUnit
- Custom command (e.g. 'make')
- Script

On this page:

Executables must be defined as capabilities (that is, registered) in Bamboo before they can be used by tasks in a Bamboo job. At least one capability was automatically defined when you installed Bamboo, but you can define additional capabilities for other executables.

You can define an executable capability that is:

- for a specific local or remote agent
- shared by all local agents
- shared by all remote agents

Once you have defined a new executable capability in your Bamboo system, its label (e.g. Ant) will appear in the **Executable** list when you use the executable in a task (see Configuring tasks). The executable you select will be used every time the task is run during a build. That is, the task can only be run by agents which have a capability that matches the executable specified in the task's **Executable** list.

Note that agent-specific capabilities override any shared capability of the same name.

Defining an agent-specific executable capability

An *agent-specific capability* applies to one agent only. Note that the value of an agent-specific capability will override the value of a *shared capability* of the same name (if one exists).

To define a new agent-specific executable capability:

1. Navigate to the desired agent.
2. In the Agent-specific capabilities section of the **Capabilities** tab, select **Add capability**.
3. Select **Capability type** > **Executable**.
4. Select the appropriate executable from the **Type** list.
5. In the **Executable label**, type a name/label for the executable. Bamboo uses this name in the **Executables** list whenever a task's executable is configured.
6. In the **Path** field, type the path to the installed executable. This will vary depending on the **Type** you selected in the previous step.
   For Ant and Maven, Bamboo requires the path to be the location of the executable installation folder.
7. Select **Add**. This will verify whether the executable and path you have specified are valid.

Defining a local server executable capability

Local server capabilities are inherited by all local agents. This means that local agents can all make use of the executables installed on the Bamboo server machine.

Before you begin:

- If you want to run multiple Maven agents on your local server, you will need to configure repository isolation for your Maven executables. See Configuring repository isolation for Maven executables for details.

To define a new local server executable capability:

1. Go to  > **Build resources** > **Server capabilities**.
2. Select **Capability type** > **Executable** in the Add capability section at the end of the page.
3. Select the appropriate type of executable from the **Type** list.
4. In the **Executable label** field, type a name/label for the executable, which Bamboo presents in the **Executable** list whenever a Task's executable is configured.
5. In the **Path** field, type the appropriate path. This will depend on the **Type** you selected in the previous step.
   Note that for Ant and Maven, Bamboo requires the path to be the location of the executable installation folder.
6. Select **Add**.

Defining a shared remote executable capability

Shared remote capabilities are inherited by all remote agents. However, Bamboo remote agents inherit only the paths of the shared executable capabilities, not the actual executable files. This means that every time you define a capability for an agent, you must make sure that the executable (for example, Ant or Maven) has actually been installed in that location on the remote server on which the remote agent will run.

Note that the value of a shared capability will be overridden by the value of an *agent-specific capability* of the same name (if one exists).

Shared remote executable capabilities are **not shared** with elastic agents.

To define a shared remote executable capability:

1. Go to  > **Build resources** > **Agents** > **Shared remote** capabilities.
2. Select **Capability type** > **Executable** in the Add capability section.
3. Select the appropriate type of executable from the **Type** list.
4. In the **Executable label** field, type a name/label to help you identify this executable.
5. In the **Path** field, type the appropriate path. This will depend on the **Type** you selected in the previous step.
   Note that for Ant and Maven, Bamboo requires the path to be the location of the executable installation folder.
6. Select **Add**.

Notes

- **Pre-defined executables** — The executable that was automatically defined when you installed Bamboo depends on the system environment variables (e.g. 'ANT_HOME=/opt/java/ant') that were present on the machine that Bamboo was installed on.
  - On the Bamboo server, environment variables that were present during installation were saved as *local server capabilities* in Bamboo.
  - On remote agents, environment variables that were present during installation were saved as *agent-specific capabilities* in Bamboo.
- **Using other executables** – If you need to use an executable that is not natively supported by Bamboo, a number of third-party plugin modules are available. You can also create your own executable plugin (see the Bamboo Plugin Guide for details).

- **msbuild.exe** — You will need to install the .NET framework SDK and reference the default path for msbuild.exe, (e.g. `C:\Windows\Microsoft.NET\Framework*64*\v2.0.50727`), to use this executable.
- **PHPUnit** — You will need to install PHPUnit and reference the path to your PHP command-line interpreter, (e.g. `/usr/bin/phpunit` on Ubuntu), to use this executable.

Viewing your executable capabilities

You can view all of the executable capabilities that have been defined in Bamboo on the Executables page. These include local server capabilities, local agent-specific capabilities and remote agent-specific capabilities.

An *executable* is an external program that Bamboo uses during the build process. Generally, executables compile source code to generate compiled executable files (referred to as artifacts in Bamboo). Ant, Maven, MS Build or PHPUnit are just some examples of executables that can be used as part of your build process.

New executables can be defined as capabilities in Bamboo. Once an executable has been defined in Bamboo, it can be configured as part of a task.

On this page:

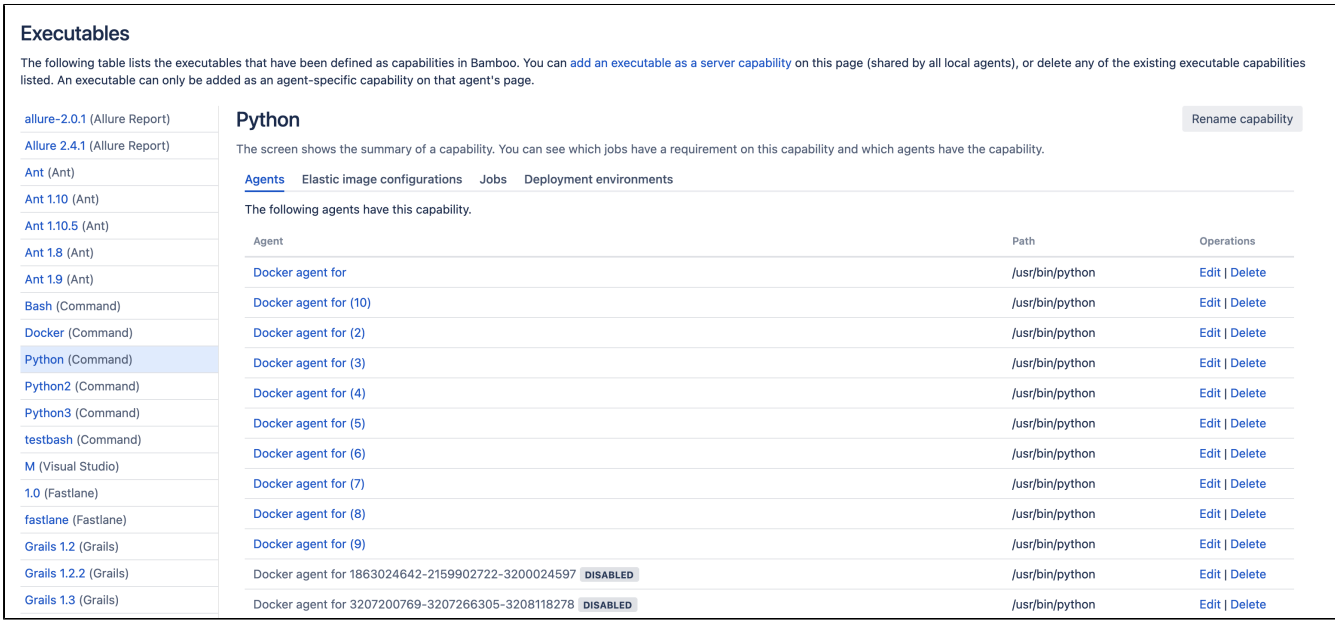
Related pages:

- Configuring capabilities

Viewing and configuring executable capabilities

To view and configure the executable capabilities defined in Bamboo:

1. Go to  > **Build resources** > **Executables**.
2. Select a specific executable's tab to see the agents and jobs related to this executable capability.
    - View more details about an agent with this executable capability — select the linked name of the agent in the Agent column. This will show you the complete list of capabilities and jobs associated with that agent.
    - Edit the executable path of an agent with this capability — select **Edit** in the Operations column for the agent you wish to configure. See Defining a new executable capability.
    - Remove this executable capability from an agent — select **Delete** in the Operations column for the agent that currently possesses this executable capability.
      ⚠ Be aware that you can only remove an executable capability from *all* local agents, not from individual local agents. See the note below for more information.
    - View details about (and configure) an elastic image with this executable capability — select the linked name of the elastic image in the Elastic Image Configuration column.
    - Configure a job that relies on or requires this executable capability — select the linked name of the job in the Plan column.
    - If you are currently viewing a Maven (2.x or later) executable capability, you can configure repository isolation for it by selecting **Edit capability configuration**. Please refer to Configuring repository isolation for Maven executables for more information.
    - To add a new executable as a local server capability, select **Add executable to server capabilities** to navigate to the Server capabilities page.



Notes

- **Bamboo's automatic detection of executables** — When you install the Bamboo server application or the Bamboo Remote Agent application on another machine, either of these applications will automatically look for existing executables installed on the same machine (based on a combination of the machine's environment variables and other conditions). An executable capability will be created for each executable that either of these Bamboo applications find.

  The environment variables and conditions that Bamboo uses to automatically detect and create executable capabilities are listed below. With the exception of the Command executable, the paths for each automatically detected executable are based on the path string values found within these environment variables.
  - Ant — the `ANT_HOME` environment variable
  - Maven — the `MAVEN_HOME` environment variable (Maven 1), `M2_HOME` or `MAVEN2_HOME` environment variable (Maven 2.x)
  - Grails — `GRAILS_HOME` environment variable
  - Command — the existence of the `/bin/bash` file
  - PHPUnit — the existence of the `phpunit` file anywhere within the machine's `PATH` environment variable value
- **Local agents and executable capabilities** — Since Bamboo automatically looks for executables installed on the same machine and creates an executable capability for each executable installation it finds, all existing and subsequent local agents that you create will possess these executable capabilities. Hence, when you access the Executables page and view these executable capabilities, all local agents will be grouped together in the All local agents category and you will only be able to remove these executable capabilities from *all* local agents, not from individual local agents.

Configuring repository isolation for Maven executables

Bamboo allows you to isolate Maven (2.x or later only) executables on an agent-specific basis. If you configure repository isolation for a particular Maven executable capability, each agent that uses this executable will have its own private Maven 2.x artifacts directory, thereby allowing you to avoid these jar and dependency file corruptions. Each isolated repository directory has the path:

```
$BAMBOO_HOME/.m2/AGENT-${bamboo.agentId}/repository
```

Related pages:

- Defining a new executable capability

You may want to configure repository isolation for Maven executables, if you run multiple Maven executables on one server machine which *run under the same user account on that server*, but belong to *different Bamboo agents*. In this case, the agents will use the same default Maven artifacts directory: `$HOME/.m2/repository` (or `%USERPROFILE%\.m2\repository` for Windows-based servers). This is the directory to which Maven dependency jars are downloaded and where project artifacts are installed during the "install" phase of a Maven build.

Hence, problems can arise if Bamboo uses these multiple Maven executables simultaneously. For example, if multiple agents on a single computer, each with a different Maven executable capability, start to run Maven builds simultaneously from the queue, the different Maven executables may attempt to download the same dependency to the same artifacts directory location, resulting in corruption of the downloaded jar and dependency files.

Before you begin:

- This feature is not available for Maven 1.x executables.
- When configuring any Maven executables in Bamboo in which you want to force local repository isolation, ensure that the executable label you use is one that identifies it as such — for example, Maven 2.x with local repository isolation.

To configure a new local server Maven capability with repository isolation:

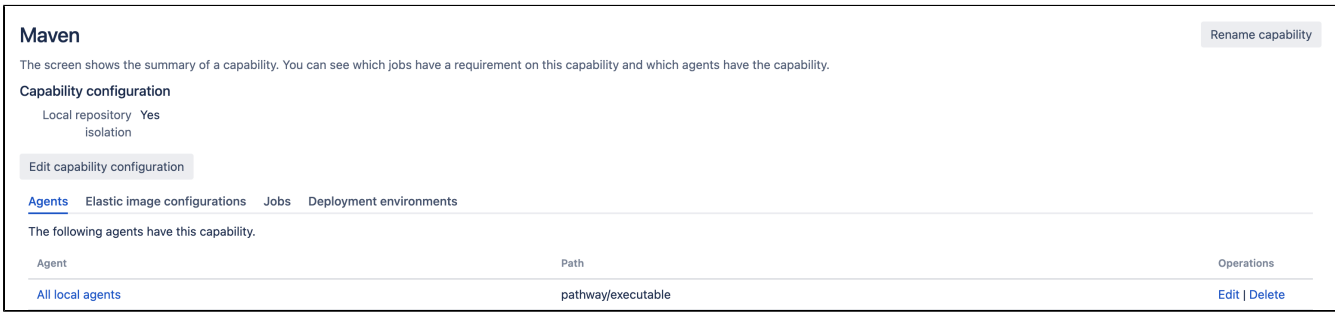
1. From the top navigation bar select  > **Build resources** > **Server capabilities**.
2. In the Add capability section, select your executable and enter its details as described:

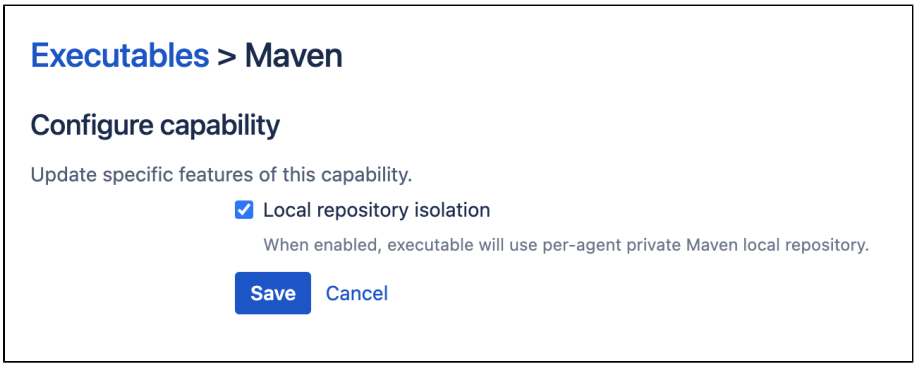
   Capability type
   Select **Executable**.

   Type
   Select one of the Maven options (2.x or later).

   Executable label
   Enter Maven with local repository isolation.
   You can use any label you wish. However, it will help you and your Bamboo users if you enter an appropriate executable label that identifies this Maven 2.x executable as one that uses local repository isolation.

   Path
   Enter the path for your Maven executable.
3. Select **Add**.
4. Select the label for the executable you have just added. The executable capability summary screen will be displayed (see 'Maven 2.x Executable' screenshot below).
5. Select **Edit capability configuration**. The Configure capability screen will be displayed (see 'Maven 2.x Repository Isolation' screenshot below).
6. Select the **Local repository isolation** checkbox.
7. Select **Save**.





Defining a new JDK capability

A JDK must be installed, and defined in Bamboo as a capability, before Bamboo can make use of it when building jobs.

At least one JDK was automatically defined when you installed Bamboo. You can define additional JDK capabilities that are:

- for a specific local or remote agent
- shared by all local agents
- shared by all remote agents.

On this page:

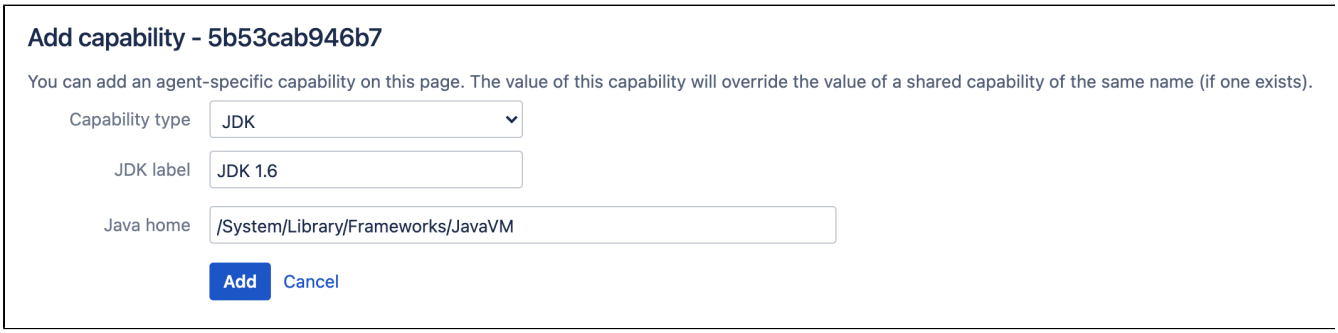
Once you have defined a new JDK capability in your Bamboo system, its label (e.g. 1.5) will appear in the **Build JDK** list when you configure a job's builder (see Configuring tasks). The JDK you select will be used for every one of that job's builds. That is, the job can only be built by agents which have a JDK capability whose label is specified in the job's **Build JDK** field.

Note that if an agent has its own specific JDK capability, that value will override the value of a shared JDK capability of the same name (if one exists).

Defining a JDK capability on an agent

To define a new agent-specific JDK capability:

1. From the top navigation bar select > **Build resources** > **Agents**.
2. Select the name of the required agent.
3. Go to the **Capabilities** tab, and then **Add capability**.
4. Select **Capability type** > **JDK**.
5. In the **JDK label** field, type a name/label for the JDK. Bamboo displays this in the **Build JDK** list whenever a job's builder is configured.
6. In the **Java home** field, type the location of the JDK Home Directory.
7. Select **Add**.



Defining a local server JDK capability

Local server capabilities are inherited by all local agents.

To define a new local server JDK capability:

1. From the top navigation bar select  > **Build resources** > **Server capabilities**.
2. From the **Add capability** section select **Capability type** > **JDK**.
3. In the **JDK Label** field, type a name/label for the JDK. Bamboo displays this in the **Build JDK** list whenever a job's builder is configured.
4. In the **Java Home** field, type the location of the JDK Home Directory.
5. Select **Add**.

Defining a shared remote JDK capability

Shared remote JDK capabilities are **not shared** with elastic agents.

To define a new shared remote JDK capability:

1. From the top navigation bar select  > **Build resources** > **Agents** > **Shared remote capabilities**.
2. Select **Capability type** > **JDK**.
3. In the **JDK Label** field, type a name/label for the JDK. Bamboo Bamboo displays this in the **Build JDK** list whenever a job's builder is configured.
4. In the **Java Home** field, type the location of the JDK Home Directory.
5. Select **Add**.

Notes

- **Configuring generic JDK capabilities** — If you want to indicate that an agent is capable of running builds for a set of related JDKs (e.g. all point versions of JDK 1.5), you set up generic JDK capabilities to encompass these JDKs.
  For example, you can set up the following JDK capabilities for your Bamboo agent(s):
  - **JDK** (where 'JDK Label' = '`JDK`' and 'Java Home' = '`/usr/java/jdk1.5.0_07`') — this JDK capability indicates that an agent(s) is capable of running builds with **any** JDK requirement.
  - **JDK 1.5** (where 'JDK Label' = '`JDK 1.5`' and 'Java Home' = '`/usr/java/jdk1.5.0_07`') — this JDK capability indicates that an agent(s) is capable of running builds with a JDK 1.5 requirement or any point version of JDK 1.5, e.g. 1.5.0_07, 1.5.0_08, etc.
  - **JDK 1.5.0_07** (where 'JDK Label' = '`JDK 1.5.0_07`' and 'Java Home' = '`/usr/java/jdk1.5.0_07`') — this JDK capability indicates that an agent(s) is only capable of running builds with a JDK 1.5.0_07 requirement.

- If you wish to find redundant JDK capabilities, you can view the list of JDK capabilities set up in Bamboo and delete any unwanted JDK capabilities.

- **Automatically defined capabilities** – This depends on the system environment variables (e.g. '`JAVA_HOME=/opt/java/java_sdk1.5`') that were present on the machine on which Bamboo was installed:
  - On the Bamboo server, environment variables that were present during installation were saved as shared local capabilities in Bamboo.
  - On remote agents, environment variables that were present during installation were saved as agent-specific capabilities in Bamboo.

Viewing your JDK capabilities

You can view all the JDK capabilities that have been defined in your Bamboo system on the **JDKs** page. These include local server capabilities, local agent-specific capabilities and remote agent-specific capabilities.

Note the following:

- **Bamboo's automatic detection of JDKs** — When you install either Bamboo or the Bamboo Remote Agent, it will automatically look for an existing JDK installed on the same machine (based on the machine's `JAVA_HOME` environment variable) and create a 'JDK capability' for that JDK installation, with its path being the value of `JAVA_HOME`.
- **Local agents and JDK capabilities** — Since Bamboo automatically looks for an existing JDK installed on the same machine and creates a 'JDK capability' for it, all existing and subsequent local agents that you create will possess this JDK capability. Hence, when you access the 'JDKs' page and view this JDK capability, all local agents will be grouped together in the 'All local agents' category and you will only be able to remove this JDK capability from *all* local agents, not from individual local agents.

Related pages:

- Defining a new JDK capability

To view and configure the JDK capabilities defined in Bamboo:

1. From the top navigation bar select  > **Build resources** > **JDKs**.
2. Select the tab for a specific JDK to see the agents and jobs related to this JDK capability.
    - View the capabilities and jobs associated with an agent with this JDK capability — select the linked name of the agent in the **Agent** column. See Viewing a capability's agents and jobs.
    - Edit JAVA_HOME for an agent — select **Edit** in the **Operations** column for the agent you wish to configure. See Defining a new JDK capability.
    - Remove this JDK capability from an agent — select **Delete** in the **Operations** column for the agent that currently possesses this JDK capability.
      ⚠ Be aware that you can only remove a JDK capability from *all* local agents, not from individual local agents. See the note above for more information.
    - View details about (and configure) an elastic image with this JDK capability — select the name of the elastic image in the **Elastic image configuration** column. See Viewing an elastic image.
    - Configure a job that relies on this JDK capability — select the name of the job in the **Plan** column.
    - To add a new JDK as a local server capability, select **add a JDK as a server capability** at the top of the page. This opens the Server capabilities page at the Add capability section, with the JDK selected as the **Capability type**.

Defining a new version control capability

Version control capabilities let Bamboo know where the client application for a version control system is located, so that Bamboo can perform a checkout while building. Bamboo requires that a capability for at least one of the following version control repositories be set so that Bamboo can check out source code from that repository type:

- Bitbucket Cloud
- Git
  If no capability is provided, Bamboo will use its built-in Git implementation. Note that the built-in Git implementation does not support symbolic links, submodules, automatic branch detection, or automatic merging.
- Perforce

Note that there is no need to create a SVN capability as SVN support is built into every Bamboo agent.

Example version control executable paths

For the version control systems that require capabilities to be set on agents, the following table offers example paths for both Linux and Windows systems.

Note that these paths may differ on your actual system's configuration.

Capability type	Example paths
Git	• /usr/bin/git • C:\Program Files\Git\git.exe
Mercurial	• /usr/local/bin/hg • C:\Program Files\Mercurial\hg.exe

To define a new version control capability

1. Navigate to the desired agent.
2. Select either a local or remote agent.
3. Select the version control type you require from **Capability type**.
4. Provide the full path to client executable on the agent machine.

If you install a new agent on a machine that has Git already installed, the agent will find the Git client automatically.

Defining a new custom capability

Custom capabilities can be used to control which jobs will be built by a particular agent, since agent capabilities are required to match job requirements. For example, if the builds for a particular job should only run in a Windows environment, you could create a custom capability '`operating.system=WindowsXP`' for the appropriate agent(s), and specify it as a requirement for this job.(See Configuring a job's requirements.)

You can define a custom capability that is:

- for a specific local or remote agent
- to be shared by all local agents
- to be shared by all remote agents.

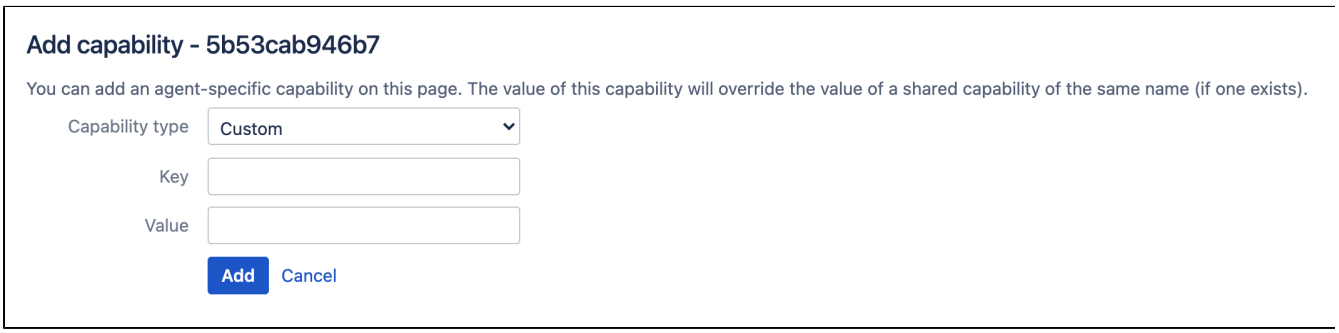
Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

On this page:

Defining an agent-specific custom capability

To define a new agent-specific custom capability:

1. Navigate to the desired agent.
2. From the **Agent-specific capabilities** section select **Add capability**.
3. Select **Capability type** > **Custom**.
4. Specify values for **Key** and **Value**.
5. Select **Add**.



Defining a local server custom capability

Local server capabilities are inherited by all local agents.

To define a new local server custom capability:

1. From the top navigation bar select  > **Build resources** > **Server capabilities**.
2. Specify values for **Key** and **Value**.
3. Select **Add**.

Defining a shared remote custom capability

Shared remote custom capabilities are **not shared** with elastic agents.

To define a new shared remote custom capability:

1. From the top navigation bar select  > **Build resources** > **Agents**.
2. In the **Remote agents** section select **Shared remote capabilities**.
3. In the **Add capability** section select **Capability type** > **Custom**.
4. Specify values for **Key** and **Value**.
5. Select **Add**.

Defining a new Docker capability

A capability typically defines the path to a module or executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of that. That is why you need to define a Docker capability in Bamboo before you can use a Docker task or the Docker Runner feature in Bamboo builds and deployments.

Before you begin

- Make sure you have Docker installed. We recommend using the latest version of Docker. If you have restrictions on the version of Docker that you can run, review the Supported platforms page for your Bamboo version.
- For Bamboo 5.8, and later versions, Stock images already provide Docker, but you might still need to add the capability manually if you have upgraded from Bamboo 5.7 or an earlier version. See Existing stock images require manual update if new capabilities are needed for more details.

Define a Docker capability on an elastic image

1. From the top navigation bar select  > **Elastic Bamboo** > **Image configurations**.
2. In the **Operations** section select **Capabilities** for the relevant elastic image.
3. Use the **Add capability** panel at the end of the page to add the new Docker capability to the image:
    - From **Capability type** select **Docker**.
    - For **Path**, enter the path to the Docker executable, for example `/usr/bin/docker`.
4. Select **Add**.

Define a Docker capability on an agent

1. From the top navigation bar select  > **Build resources** > **Agents**.
2. Select the name of the required agent.
3. Select the **Capabilities** tab, and then **Add capability**.
4. In the **Add capability** panel:
    - From **Capability type** select **Docker**.
    - For **Path**, enter the path to the Docker executable, for example `/usr/bin/docker`.
5. Select **Add**.

Define a Docker capability on the Bamboo server

1. From the top navigation bar select  > **Build resources** > **Agents** >**Server capabilities**.
2. Use the **Add capability** panel at the end of the page to add the new Docker capability to the server:
    - From **Capability type** select **Docker**.
    - For **Path**, enter the path to the Docker executable, for example `/usr/bin/docker`.
3. Select **Add**.

For more information about Bamboo and Docker integration, see Getting started with Docker and Bamboo

Remote agents

For information about installing and using remote agents, see the following pages:

- Bamboo remote agent installation guide
- Configuring remote agent capabilities using bamboo-capabilities.properties
- Disabling and enabling remote agents support

Disabling and enabling remote agents support

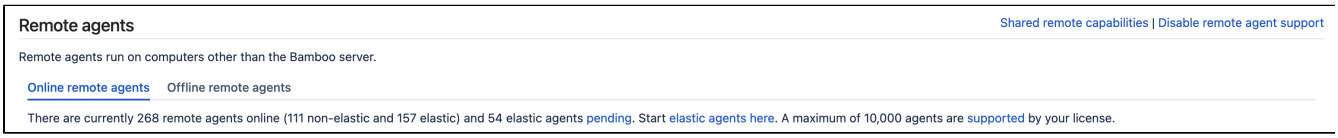
Remote agent support

Disabling remote agent support in Bamboo will disable all remote agents and prevent any users from creating new remote agents. This function will not delete any remote agents that you have already created. To delete a remote agent, see Disabling or deleting an agent.

Note that remote agent support must be enabled to use Elastic Bamboo. Disabling remote agent support will disable Elastic Bamboo.

To enable or disable remote agent support:

1. From the top navigation bar select  > **Build resources** > **Agents**.
2. Select either **Enable remote agent support** or **Disable remote agent support**.



Related pages:

- Configuring agents
- Agents and capabilities
- Configuring a job's requirements

Additional remote agent options

This page describes additional options for running a Bamboo remote agent. Additional options can be found in *How to extend the Remote agent installation command using JVM parameters*.

By default, the remote agent will store its data in a USER_HOME/`bamboo-agent-home`. If you wish to specify a different directory, add the following command line parameter before the JAR file name:

```
-Dbamboo.home=RemoteAgentHome
```

Where *RemoteAgentHome* is the path to the Bamboo agent home directory you created in step 1.1.

Your command line should appear similar to the example provided below. Note that the command will work only in the cmd (Command Prompt for Windows) and Terminal (for MacOS and Linux).

```
java -Dbamboo.home=RemoteAgentHome -jar atlassian-bamboo-agent-installer-X.X-SNAPSHOT.jar http://bamboo-host-server:8085/agentServer/
```

The name of the jar file (e.g. atlassian-bamboo-agent-installer-2.2-SNAPSHOT.jar) will vary depending on the version of Bamboo you are running.
There may be situations where you want to prevent Bamboo from automatically detecting and adding capabilities (such as JDKs) to the remote agent, or where you don't want to run the remote agent with default capabilities.

⚠ Elastic agents don't support the `DISABLE_AGENT_AUTO_CAPABILITY_DETECTION` system property.

The `DISABLE_AGENT_AUTO_CAPABILITY_DETECTION` system property is handled only from the wrapper.conf file, not from the command line.

To update the property for a remote agent, add the following line to the <bamboo-agent-home>/conf/wrapper.conf file:

```
wrapper.java.additional.3=-DDISABLE_AGENT_AUTO_CAPABILITY_DETECTION=true
```

Then restart the agent with regular command `java -jar agent.jar URL_TO_SERVER SECURITY_TOKEN`
You can specify a custom log4j2 file in the wrapper.conf file. The file is located in *<bamboo-agent-home>/conf/wrapper.conf*.

Find the `#wrapper.java.additional.3=-Dlog4j2.configurationFile=` line, uncomment it and provide path to log4j2.properties file at your disk. Then restart the agent with regular command `java -jar agent.jar URL_TO_SERVER SECURITY_TOKEN`.

Changing the logging on the remote agent

By default, the remote agent will use the same logging level as the Bamboo server. However, you can control the level of logging of your remote agent independently of your Bamboo server by setting up a separate logging configuration file.

See Logging in Bamboo for further details.
Use the text-based keytool utility or GUI-based Portecle to add the self-signed certificate to the trusted certificates in your keystore.

Add "socket.verifyHostName=false" settings to Client Broker URL at Bamboo general configuration.

If you are connecting to your Bamboo server via https (SSL is enabled) using a hostname or IP address that does not match the Common Name of your Bamboo server's certificate (or any of its Subject Alternative Names) you need to disable hostname verification:

Add the following parameter to the remote agent's command line:

```
-Dbamboo.agent.ignoreServerCertName=true
```

Your command line will look something like this:

```
java -Dbamboo.agent.ignoreServerCertName=true -jar atlassian-bamboo-agent-installer-X.X-SNAPSHOT.jar
https://bamboo-host-server:8085/agentServer/
```

⚠ Note that this **reduces the security of your configuration**, as the identity of your Bamboo server will not be authenticated by the remote agent.

The remote agent supervisor is included in the remote agent JAR bundled with Bamboo. The appropriate remote agent supervisor for the operating system of your remote machine, will be automatically installed when you run the default remote agent start-up command line.

> *The remote agent supervisor cannot be installed on a small number of operating systems (i.e. the remote agent will start without the remote agent supervisor). If the remote agent supervisor fails to install, please check the operated systems list on the remote agent supervisor page. If your operating system is on the list and the remote agent supervisor still fails to install, please raise a support request in the Bamboo project.*

If you need to run the remote agent without running the remote agent supervisor, you can execute the classic version of the remote agent JAR.
The classic agent jar is available from Bamboo's agent installation page for download. Follow the steps below to run the classic version of the remote agent:

1. Browse to:

   ```
   http://<host>:8085/admin/agent/addRemoteAgent.action
   ```

2. Select the the **direct agent JAR is available at bamboo-agent-2.2.2.jar** link and save classic agent jar.
3. Start the agent with or without token key:

   ```
   java -jar bamboo-agent-2.2.2.jar http://<host>:8085/agentServer/ <tokenKey>
   ```

   > `<tokenKey>` is required if Security Token Verification is enabled.

The name of the jar file (e.g. bamboo-agent-2.2.2.jar) will vary depending on the version of Bamboo you are running

The remote agent supervisor is executed by default when you run the default remote agent start-up command line. The remote agent supervisor is implemented via a Java Service wrapper. The wrapper allows you to execute a number of general start-up commands when the remote agent is run. These commands are appended to the end of the default remote agent start-up command line:

```
java -jar atlassian-bamboo-agent-installer-2.2-SNAPSHOT.jar http://bamboo-host-server:8085/agentServer
<wrapper_command>
```

where `<wrapper_command>` is one of the keywords described below:

- console — runs the remote agent in the foreground, i.e. display all of the commands on the screen. The agent home directory will be populated only if it is empty. This parameter is used by default.
- start — runs the remote agent in the background, i.e. no commands are displayed on screen. If you have installed the remote agent as a Windows service, this command will work with the service.
- stop — stops a remote agent that is running. If you have installed the remote agent as a Windows service, this command will work with the service.
- status — (non-Windows OS only) returns the status of the remote agent, e.g. "Remote agent is not running."
- install — installs the files for the remote agent, but does not start it. This will overwrite any changes that have been made to the wrapper.conf file. The agent home directory will be populated, regardless of whether it is empty or not, i.e. existing files will be overwritten. You may wish to use this option, if you want to customize the remote agent files before starting it.

The name of the jar file (e.g. atlassian-bamboo-agent-installer-2.2-SNAPSHOT.jar) will vary depending on the version of Bamboo you are running.

The remote agent supervisor is executed by default when you run the default remote agent start-up command line. The remote agent supervisor is implemented via a Java Service wrapper. The wrapper allows you to install or uninstall the remote agent as a service in Windows (i.e. start the Bamboo remote agent automatically when the machine boots). This is done by appending the appropriate wrapper commands to the end of the default remote agent start-up command line:

```
java -jar atlassian-bamboo-agent-installer-2.2-SNAPSHOT.jar http://bamboo-host-server:8085/agentServer
<wrapper_command>
```

where `<wrapper_command>` is one of the keywords described below:

- **`installntservice`** — *(Windows only)* installs the remote agent as a Windows service.
- **`uninstallntservice`** — *(Windows only)* uninstalls the remote agent as a Windows service.

The name of the jar file (e.g. atlassian-bamboo-agent-installer-2.2-SNAPSHOT.jar) will vary depending on the version of Bamboo you are running.

If you have installed the NT service, you will be able to use the `start` and `stop` start-up console commands wit h the service.

- The remote agents connect to the Bamboo server on the normal http/https port and 54663. You need to ensure that the network firewall isn't blocking these ports.If you're having issues connecting the remote agent with the Bamboo server, please this Troubleshooting Guide
- On Windows, when the Bamboo remote agent is installed as a service under the Local System user account. The temporary folder created by the agent isn't accessible by any other application. To allow enable to that folder, configure the agent to use a commonly accessible temporary folder by defining a new Java property in the service wrapper:

  1. Go to: <BAMBOO_AGENT_HOME>/conf/wrapper.conf
  2. Add the following line using the next property number as X:
     `wrapper.java.additional.X=-Djava.io.tmpdir=C:\path\to\temp`
  3. Restart the agent.

Ephemeral agents

An ephemeral agent is a short-lived remote agent that starts on-demand inside a Kubernetes cluster to carry out a single build or deployment before being shut down. Each ephemeral agent has exactly the tools it needs to perform its job thanks to the concept of templates, which define the agent's capabilities and the Kubernetes pod layout.

Whether or not you're already running your own Kubernetes cluster, ephemeral agents are a great way to integrate Bamboo with it, offering such benefits as:

- the ability to run agents on any cloud or on-premise infrastructure (both virtualized and bare-metal)
- savings on resource costs (agents don't use computing resources when there's nothing to do)
- flexible and on-demand scalability (run as many agents as necessary at any given moment)
- robust security with pod and container isolation (each agent starts up in a fresh environment)
- automatic agent lifecycle management (Bamboo automatically starts and stops agents when needed)
- automatic matching of ephemeral agents with the builds and deployments they're capable of running

Check out the following pages to learn how to enable support for ephemeral agents in Bamboo, configure templates, and manage running pods:

- Enabling ephemeral agent support
- Enabling and disabling automatic pod removal
- Ephemeral agent template management
- Pod and ephemeral agent management

Enabling ephemeral agent support

Before you start creating ephemeral agent templates, you should enable support for ephemeral agents in Bamboo so that Bamboo can connect to and authenticate with your Kubernetes cluster.

Before you begin

Install the Kubernetes command-line tool, kubectl, on your Bamboo server. Kubectl is not installed by default.

To enable ephemeral agent support in Bamboo:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Configuration**.
3. Under **General**, select the **Enabled** checkbox.
4. In the **Path to configuration file** field, enter the full path to your Kubernetes configuration file relative to the Bamboo [shared] home directory.
5. In the **Resource label** field, enter the label that you want to attach to the resources Bamboo creates and manages in your Kubernetes cluster.

   > ⚠ Bamboo can only manage Kubernetes resources identified by the resource label you set here. If you change the resource label later, Bamboo will no longer be able to recognize resources identified by the previous label.

6. Select **Test connection** to make sure that the communication between Bamboo and your Kubernetes cluster is working correctly.
7. Optional: If you want Bamboo to automatically remove pods after ephemeral agents complete their work, enable automatic pod removal.
8. If the test connection was successful, select **Save**.

You can now create your first ephemeral agent template and configure its capabilities.

I'm getting a "Connection failed" error

The test connection may fail for a number of reasons. Bamboo will display the exact error message coming from Kubernetes that you can use to troubleshoot the connection with the cluster.

Here's what you can do in the most common situations:

- **An error occurred while loading the configuration file** — this may happen either due to an incorrectly configured path to the Kubernetes configuration file (there's a typo in the path or the file does not exist at the specified location) or a syntax error in the file. Verify that the path you specified is correct and that the file is well-formed.
- **Bamboo can't connect to the specified IP address or hostname** — this can mean that the Kubernetes cluster is unreachable, the specified authentication details are incorrect, or there is an error in the provided IP address or hostname. Verify that the connection and authentication details specified in your Kubernetes configuration file are correct.

If everything looks alright but the problem persists, contact Atlassian Support.

Related pages

- Enabling and disabling automatic pod removal
- Ephemeral agent template management
- Ephemeral agents
- Pod and ephemeral agent management

Enabling and disabling automatic pod removal

Bamboo can automatically delete a pod after an agent completes its work. Removing completed pods ensures that the health status of the cluster is reflected accurately and reduces the risk of unauthorized access to sensitive information stored in a pod's log files or environment variables.

You can enable or disable this behavior at any time. You can also configure the delay between two consecutive pod removal operations.

Before you begin

If you want to enable automatic pod removal, enable support for ephemeral agents first.

To enable or disable automatic pod removal:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Configuration**.
3. Under **General**, select the **Enabled** checkbox.
4. Under **Pod removal**, do the following:
    a. If you want to enable automatic pod removal, select the **Enabled** checkbox.
    b. If you want to disable automatic pod removal, clear the **Enabled** checkbox.
5. In the **Delay** field, enter the time in seconds Bamboo should wait between two consecutive pod removal operations.

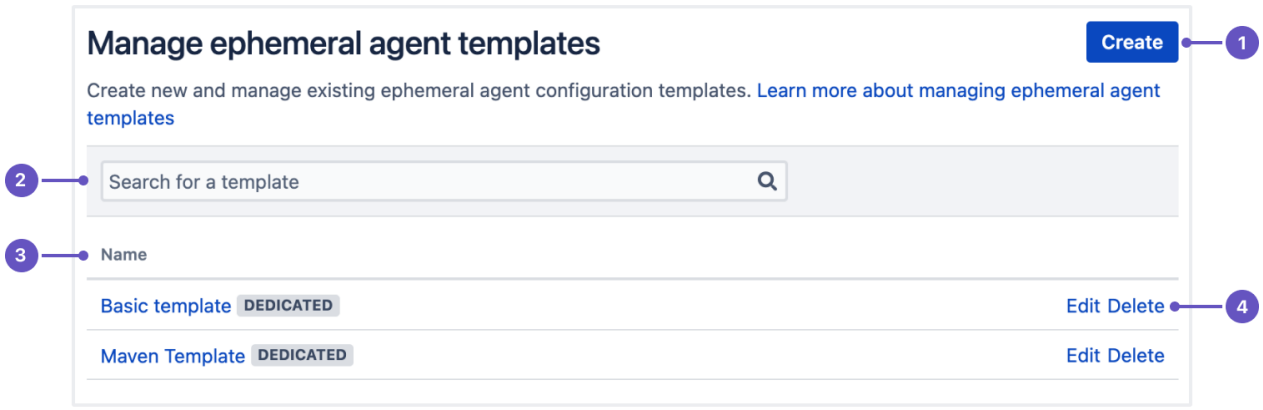
Related pages

- Enabling ephemeral agent support
- Ephemeral agent template management
- Ephemeral agents
- Pod and ephemeral agent management

Ephemeral agent template management

The **Manage ephemeral agent templates** page lets you create new and manage existing ephemeral agent templates. You can access the ephemeral agent management options by selecting **Templates** from the menu on the **Bamboo administration** page.

Here's what the page looks like:



The page is divided into the following elements:

1. **Create** button — create new ephemeral agent templates
2. Search bar — search for an existing template
3. List of existing ephemeral templates
4. Edit or delete each template directly in the list

Select one of the following topics to learn more about what you can do:

- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Dedicating ephemeral agent templates
- Viewing ephemeral agent template details
- Deleting ephemeral agent templates

About ephemeral agent templates

To start up an ephemeral agent in its own dedicated Kubernetes pod, Bamboo needs to know how to construct the pod. Templates let Bamboo know what it needs to build Kubernetes pods and run ephemeral agents in them. A template has a name and a YAML configuration with all the necessary details about the building blocks of the pod and the containers it's supposed to run.

> To start creating ephemeral agent templates, enable ephemeral agent support in Bamboo first.

On this page:

Basic template requirements

Every template has the following set of basic requirements:

- An `apiVersion` field with a value of `v1`
- A `kind` field with a value of `Pod`
- A `metadata` field to uniquely identify the pod with the following data:
  - A `name` field with a value equal to `'{{NAME}}'`
  - A `labels` field defining the following key and value pair that will be used as the resource label to identify resources it creates and manages in your cluster:

    ```
    '{{RESOURCE_LABEL}}': <VALUE>
    ```

    Replace <VALUE> with the resource label name you want Bamboo to use to identify resources in your Kubernetes cluster.
- A `spec` field defining the properties of exactly one ephemeral agent container under containers:
  - An `image` field declaring the Docker image to use. This can be a custom image or the base Docker image:

    ```
    atlassian/bamboo-agent-base:<YOUR_BAMBOO_VERSION>
    ```

    Replace <YOUR_BAMBOO_VERSION> with the Bamboo version number that you're currently running. For example: `atlassian/bamboo-agent-base:9.3.0`
  - A `name` field with a value equal to `'{{BAMBOO_AGENT_CONTAINER_NAME}}'`
  - An env field containing the `BAMBOO_EPHEMERAL_AGENT_DATA` environment variable with a string value equal to `'{{BAMBOO_EPHEMERAL_AGENT_DATA_VAL}}'`
- A `restartPolicy` field with a value of Never

Here's an example YAML configuration that you can use as the starting point for creating your own ephemeral agent template:

All of the variables wrapped in double curly braces are placeholders that Bamboo will automatically replace with appropriate values. Make sure to insert them explicitly as shown in the following example.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: '{{NAME}}'
  labels:
      '{{RESOURCE_LABEL}}': <VALUE>
spec:
  containers:
    - image: atlassian/bamboo-agent-base:<YOUR_BAMBOO_VERSION>
      name: '{{BAMBOO_AGENT_CONTAINER_NAME}}'
      env:
        - name: BAMBOO_EPHEMERAL_AGENT_DATA
          value: '{{BAMBOO_EPHEMERAL_AGENT_DATA_VAL}}'
  restartPolicy: Never
```

- Replace <VALUE> with the resource label name you want Bamboo to use to identify resources in your Kubernetes cluster.
- Replace <YOUR_BAMBOO_VERSION> with the Bamboo version number that you're currently running. For example: `atlassian/bamboo-agent-base:9.3.0`

Creating custom images

The base Docker image used in the example YAML configuration above contains only a minimal setup to run a basic ephemeral agent, which may not have sufficient capabilities to run your builds. If you need additional capabilities beyond what the basic image provides, you can extend the image as needed.

To extend the basic image:

1. Create a new Docker file that defines all the capabilities that you want your ephemeral agent to have. In the following example, we'll install Maven:

   ```
   FROM atlassian/bamboo-agent-base:<YOUR_BAMBOO_VERSION>
   RUN apt-get update && \
       apt-get install maven -y
   ```

   Replace <YOUR_BAMBOO_VERSION> with the Bamboo version number that you're currently running.

2. Build the new image:

   ```
   docker build --tag <MY_IMAGE_TAG>
   ```

   For example:

   ```
   docker build --tag my-agent-with-maven-image
   ```

3. Create a new template or modify an existing one to use the new image.
4. Add all the capabilities you need to the template.
   In our Maven example above, the configuration may look like this:

Field	Value
Capability type	Executable
Type	Maven 3.x
Path	/usr/share/maven

You can now use your customized template to run builds and deployments.

Adding multiple containers to a template

Apart from the Bamboo ephemeral agent container, your template can define other containers that the ephemeral agent can communicate with to exchange data or run other processes.

To add multiple containers, define each one as an entry in the `containers` list.

In the following example, we'll add extra containers for the PostgreSQL database and nginx server:

```
---
apiVersion: v1
kind: Pod
metadata:
  name: '{{NAME}}'
  labels:
      '{{RESOURCE_LABEL}}': <VALUE>
spec:
  containers:
    - image: atlassian/bamboo-agent-base:<YOUR_BAMBOO_VERSION>
      name: '{{BAMBOO_AGENT_CONTAINER_NAME}}'
      env:
        - name: BAMBOO_EPHEMERAL_AGENT_DATA
          value: '{{BAMBOO_EPHEMERAL_AGENT_DATA_VAL}}'
    - image: postgres:latest
      name: postgres-db
      env:
        - name: POSTGRES_PASSWORD
          value: password
    - image: nginx:latest
      name: nginx-sever
  restartPolicy: Never
```

- Replace <VALUE> with the resource label name you want Bamboo to use to identify resources in your Kubernetes cluster.
- Replace <YOUR_BAMBOO_VERSION> with the Bamboo version number that you're currently running. For example: `atlassian/bamboo-agent-base:9.3.0`

Delaying the start of the ephemeral agent container

If you're going to be running pods with multiple containers in them, it may happen that you'll want to delay the start of the ephemeral agent container until the other containers are successfully up and running.

An ephemeral agent will wait for all the remaining containers to report their readiness for up to 20 minutes before starting up.

To delay the start of the ephemeral agent container:

1. In the `.spec.volumes`, define an `emptyDir` volume, where your additional containers will report their readiness.
2. In the ephemeral agent container configuration section, under `volumeMounts`, declare a read-only mount point for the previously defined `emptyDir` volume by specifying a `name` and `mountPath`.
3. In each extra container, under `.spec.containers[*].volumeMounts`, declare a writeable mount point for the previously added `emptyDir` volume by specifying the `name`, `mountPath`, and `readonly: false` properties.
4. To each extra container, under `.spec.containers[*]`, add a `postStart` lifecycle hook that will execute a command to create a unique file in the mounted volume's file system. For example:

```
lifecycle:
  postStart:
    exec:
      command: ['/bin/sh', '-c', 'touch /registration/nginx']
```

5. In the ephemeral agent container configuration section, declare the following environment variables as key-value pair entries into the `env` list:
   - `KUBE_NUM_EXTRA_CONTAINERS` equal to the number of containers the agent should wait for
   - `EXTRA_CONTAINERS_REGISTRATION_DIRECTORY` equal to the registration directory `mountPath`

For example:

```
---
apiVersion: v1
kind: Pod
metadata:
  name: '{{NAME}}'
  labels:
    '{{RESOURCE_LABEL}}': <VALUE>
spec:
  volumes:
    -   name: registration-volume
        emptyDir: {}
  containers:
    -   image: atlassian/bamboo-agent-base:<YOUR_BAMBOO_VERSION>
        name: '{{BAMBOO_AGENT_CONTAINER_NAME}}'
        volumeMounts:
          -   name: registration-volume
              mountPath: /registration
        env:
          -   name: BAMBOO_EPHEMERAL_AGENT_DATA
              value: '{{BAMBOO_EPHEMERAL_AGENT_DATA_VAL}}'
          -   name: KUBE_NUM_EXTRA_CONTAINERS
              value: '1'
          -   name: EXTRA_CONTAINERS_REGISTRATION_DIRECTORY
              value: /registration
    -   image: nginx:latest
        name: nginx-sever
        volumeMounts:
          -   name: registration-volume
              mountPath: /registration
              readOnly: false
        lifecycle:
           postStart:
              exec:
                 command: ['/bin/sh', '-c', 'touch /registration/nginx']
  restartPolicy: Never
```

- Replace <VALUE> with the resource label name you want Bamboo to use to identify resources in your Kubernetes cluster.
- Replace <YOUR_BAMBOO_VERSION> with the Bamboo version number that you're currently running. For example: `atlassian/bamboo-agent-base:9.3.0`

Reducing server resource usage on ephemeral agent startup

When an ephemeral agent starts up, it fetches the files it needs to perform a build or deployment from the Bamboo server. Fetching these files on every ephemeral agent startup may put your Bamboo server under additional load and increase the ephemeral agent's overall startup time.

You can offload some of the agent's files to a persistent volume in your Kubernetes cluster to speed up the launch of ephemeral agents and make sure that your Bamboo server's resources are put to good use.

Related pages

- Configuring an ephemeral agent template’s capabilities
- Creating ephemeral agent templates
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Ephemeral agent template management
- Viewing ephemeral agent template details

Creating ephemeral agent templates

Templates let Bamboo know what it needs to build Kubernetes pods and run ephemeral agents in them. A template has a name and a YAML configuration with all the necessary information about the building blocks of the pod and the containers it's supposed to run.

Learn more about what goes into an ephemeral agent template's YAML configuration

Before you begin

Enable and configure support for ephemeral agents in Bamboo.

To create an ephemeral agent template:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. In the upper-right corner of the **Manage ephemeral agent templates** page, select **Create**.
4. On the **Create ephemeral agent template** page, complete the template configuration:
    - **Name** — A unique name to identify this template.
    - **Enabled** — Allows Bamboo to use this template for builds and deployments.
    - **YAML configuration** — The YAML configuration for this template.
5. Select **Save**.

Result

If you've selected the **Enabled** checkbox, Bamboo can now start using this template to create ephemeral agents. You can also enable the template at a later time.

Related pages

- About ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Ephemeral agent template management
- Viewing ephemeral agent template details

Configuring an ephemeral agent template’s capabilities

Capabilities define what an ephemeral agent can do and what it supports. When you define capabilities, Bamboo will use them to find jobs and deployment environments with matching requirements. This way, you can always see which agents can run which builds and deployments.

- Bamboo does not configure any capabilities by default. You have to manually configure any capabilities that you need an ephemeral agent to have.
- To make sure that any changes you've made to a template's capabilities take effect, you need to restart any builds in the queue that require these changes. This will ensure that your builds are up-to-date and running smoothly.

Before you begin

Create an ephemeral agent template.

To configure an ephemeral agent’s capabilities:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, select an ephemeral agent template.
4. On the **Ephemeral agent template details** page, select **Add a capability**.
5. In the **Add a capability** dialog, select a capability type and configure it:

Capability type	Configuration
Custom	• **Key** — a unique key that identifies the capability • **Value** — the value of the associated with the key
Executable	• **Executable type** — the type of the executable that you want to make available to your build plans. An executable capability requires a label and the full file system path to the executable. The available options include: ◦ **Ant** ◦ **Command** ◦ **Fastlane** ◦ **Grails** ◦ **Maven 1.x** ◦ **Maven 2.x** ◦ **Maven 3.x** ◦ **MSBuild** ◦ **NAnt** ◦ **Node.js** ◦ **NUnit 2** ◦ **NUnit 3** ◦ **PHPUnit** ◦ **PHPUnit 3.3.x** ◦ **Visual Studio** ◦ **VSTest Runner** ◦ **Xcode**
JDK	• **Label** — a unique label that identifies the JDK • **Java home path** — the full file system path to the corresponding Java home directory

Perforce	**Perforce executable** — the full file system path to the Perforce P4 client application executable
Xcode SDK	• **SDK name** — the name of the Xcode SDK (foor example, “iOS 16.4” or “macOS 13.3”) • **SDK label** — the canonical name of the Xcode SDK. To determine which SDKs are recognized by your version of Xcode, run `xcode build -showsdks` in the terminal.

Git	• **Executable type** — the available options are:
	  ◦ **Git**
	  ◦ **SSH**
	• **Path** — the full file system path to the executable. Depending on the selected **Executable type** option, this should be the path to either the Git executable or the SSH executable. For example:
	  ◦ `C:\Program Files (x86)\Git\git.exe` or `/usr/local/git/bin/git`
	  ◦ `/usr/bin/ssh`

Docker	**Path** — the full file system path to the Docker executable (for example, `/usr/bin/docker`)
VS Test Adapter Discoverer	• **Name** — the name of the VS Test Adapter Discoverer. The name should follow the <VS_VERSION>.<FRIENDLY_NAME> pattern (for example, VS 2013.Generic Test Discoverer) • **Supported file types** — the file types that the Discoverer will search for runnable tests (for example, `.generictest`) The **Supported file types** value is for informational purposes only. Copy it from the output of the `vstest.console.exe /UseVsixExtensions:true /ListDiscoverers` command.

6. Select **Add**.

Bamboo will now use the capability you defined to find matching jobs and deployment environments. If you want to add more capabilities, repeat these steps for each additional capability.

Related pages

- About ephemeral agent templates
- Creating ephemeral agent templates
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Viewing ephemeral agent template details
- Ephemeral agent template management

Editing an ephemeral agent template

You can change the name and YAML configuration of an existing ephemeral agent template.

Learn more about what goes into an ephemeral agent template's YAML configuration

> ⚠ To apply the changes you made to a template's YAML configuration to queued builds, you must restart all the builds in the queue that require the changes.

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, select **Edit** next to the template that you want to edit.

   > Alternatively, select the template from the list, and then select **Edit** on the **Ephemeral agent template details** page.

4. Modify the configuration of the selected template:
   - **Name** — A unique name to identify this template.
   - **Enabled** — Allows Bamboo to use this template for builds and deployments.
   - **YAML configuration** — The YAML configuration for this template. This is a required field.
5. Select **Save**.

Related pages

- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template’s capabilities
- Enabling and disabling ephemeral agent templates
- Viewing ephemeral agent template details
- Ephemeral agent template management

Enabling and disabling ephemeral agent templates

You can enable and disable ephemeral agent templates at any time. For example, you may want to disable an ephemeral agent template to make sure it's not being used while you're finalizing its configuration.

To enable or disable an existing ephemeral agent template:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, select an ephemeral agent template.
4. On the **Ephemeral agent template details** page, do the following:
    - If the template is disabled, select **Enable** to turn it back on.
    - If the template is enabled, select **Disable** to turn it off.

Related pages

- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Viewing ephemeral agent template details
- Ephemeral agent template management

Dedicating ephemeral agent templates

By default and as long as its capabilities permit it, an ephemeral agent template can be used to perform all available builds and deployments. By dedicating a template, you can make it available only to a specific build or deployment project, plan, job, environment, or any combination of those.

To dedicate an ephemeral agent template:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, select an ephemeral agent template.
4. From the **Type** menu, select one of the following options:
    - **Build project**
    - **Build plan**
    - **Build job**
    - **Deployment project**
    - **Deployment environment**
5. From the **Name** menu, select the name of a project, plan, job, or environment.

    You can use the search field in the menu to filter available objects.

6. Select **Add**.

The template is now dedicated to the selected build or deployment project, plan, job, or environment. To dedicate it to another one, repeat these steps.

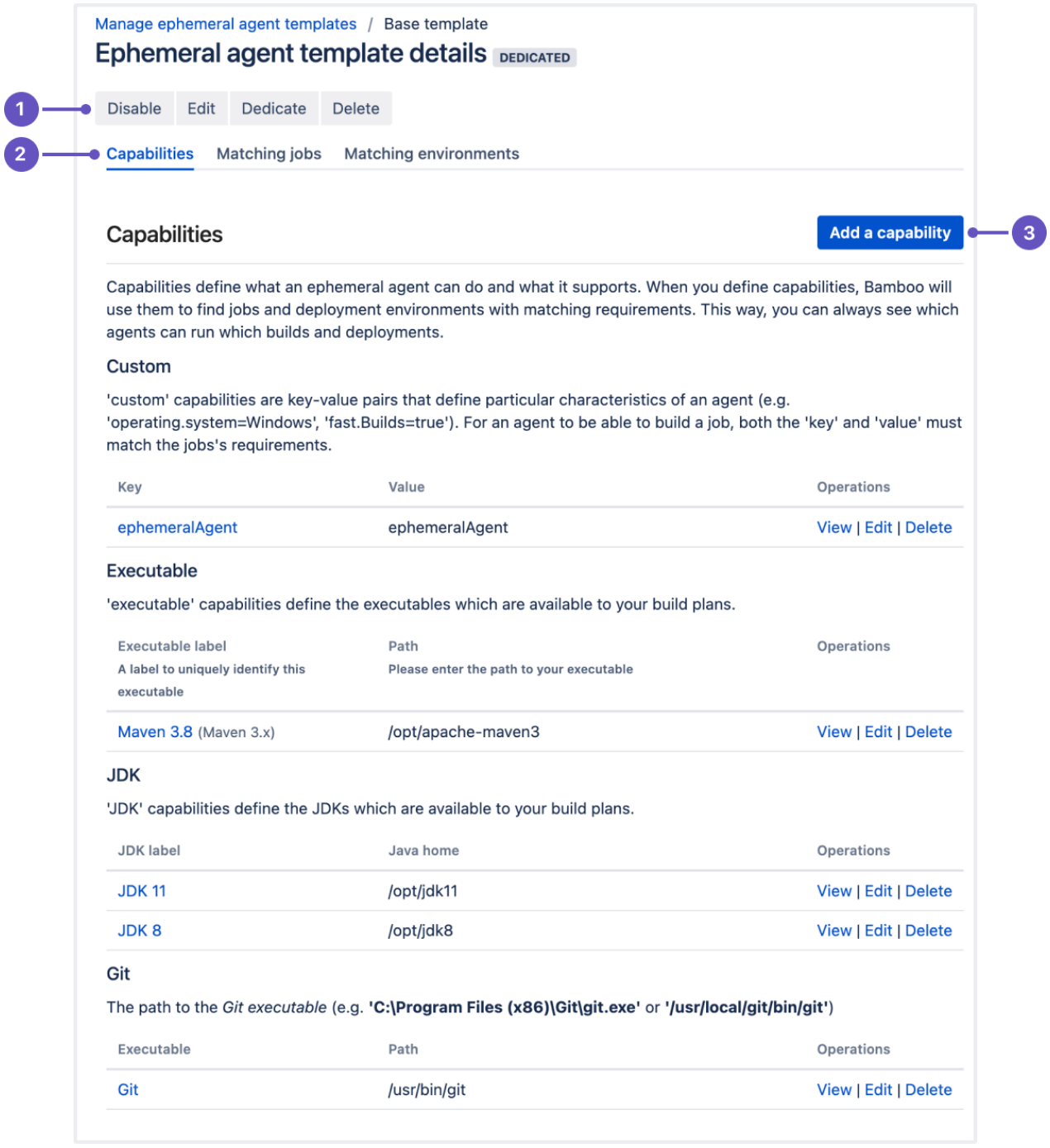
Related pages

- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Viewing ephemeral agent template details
- Ephemeral agent template management

Viewing ephemeral agent template details

After you've created an ephemeral agent template, you can go to the **Ephemeral agent template details** page to check the template's capabilities, matching jobs, and matching deployment environments.

The following image shows an example of the **Ephemeral agent template details** page:



The page is divided into the following components:

1. Toolbar — use the buttons in the toolbar to enable or disable a template, edit its name and YAML configuration, dedicate it to a specific project, plan, job, or environment, or delete it.
2. Tab strip — switch between the available tabs to see the template's capabilities as well as jobs and environments with requirements matching those capabilities.

3. **Add a capability** button — configure the ephemeral agent's capabilities.

To view the details of an ephemeral agent's configuration:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, select an ephemeral agent template.
4. On the **Ephemeral agent template details** page, select one of the available tabs:
    - **Capabilities** — displays a list of the template's configured capabilities.
    - **Matching jobs** — displays a list of jobs with requirements that match the template's capabilities.
    - **Matching environments** — displays a list of deployment environments with requirements that match the template's capabilities.

Related pages

- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Viewing ephemeral agent template details
- Ephemeral agent template management

Deleting ephemeral agent templates

You can delete a template directly from the **Manage ephemeral agent templates** page or when viewing the details of an existing ephemeral agent template.

To delete a template:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Templates**.
3. On the **Manage ephemeral agent templates** page, do one of the following things:
    - Select **Delete** next to an ephemeral agent template displayed in the list.
    - Select the ephemeral agent template, and then, on the **Ephemeral agent template details** page, select **Delete**.
4. In the confirmation dialog, select **Confirm**.

If you want to delete more ephemeral agent templates, repeat these steps for each template.

Related pages

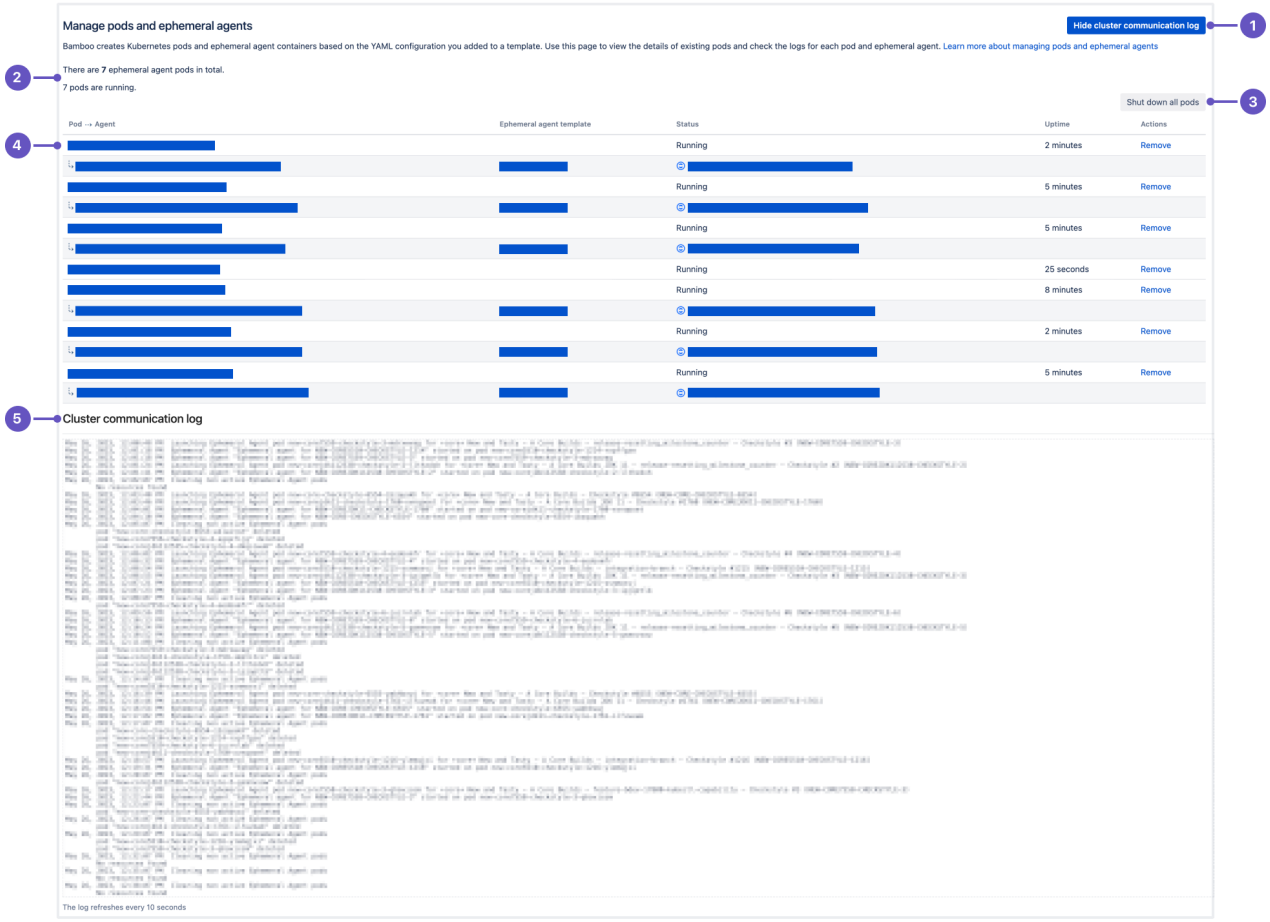
- About ephemeral agent templates
- Creating ephemeral agent templates
- Configuring an ephemeral agent template's capabilities
- Editing an ephemeral agent template
- Enabling and disabling ephemeral agent templates
- Viewing ephemeral agent template details
- Ephemeral agent template management

Pod and ephemeral agent management

The **Manage pods and ephemeral agents** page lists all existing pods and ephemeral agents. You can use this page to view the total number of pods and ephemeral agents as well as more detailed information about each pod and ephemeral agent, including their statuses and logs.

The page also displays a log that displays messages about the current status of the communication between Bamboo and your Kubernetes cluster.

Here's what this page looks like:



The page is divided into the following elements:

1. **Hide cluster communication log** button
2. Indicators displaying the total number of pods and the number of running pods
3. **Shut down all pods** button — shut down and remove all of the listed pods at once
4. List of all pods and ephemeral agents created by Bamboo — check the status of pods and ephemeral agents or remove each pod individually
5. **Cluster communication log** — a generalized log of what's happening between Bamboo and your Kubernetes cluster

Select one of the following pages to learn more about managing active pods and ephemeral agents:

- Viewing pod and ephemeral agent details
- Checking the log for a pod
- Checking ephemeral agent build and deployment logs
- Manually removing pods

Viewing pod and ephemeral agent details

The **Manage pods and ephemeral agents** page lists all existing pods and ephemeral agents. You can use this page to view more detailed information about each pod and ephemeral agent, including their statuses and logs.

To check the status of existing pods and ephemeral agents:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Pods**.
   The **Manage pods and ephemeral agents** page appears, where you can see a list of all pods and ephemeral agents created by Bamboo. There is also a generalized **Cluster communication log** that displays messages about what's happening between Bamboo and your Kubernetes cluster.
3. To view detailed information about a pod or an ephemeral agent, select it from the list.

When you select a pod from the list, you'll be taken to a page displaying information about the pod, including its status, name, start time, configuration details, existing containers, and more.

Selecting an ephemeral agent from the list will take you to the **Agent summary** page for that agent.

Related pages

- Checking ephemeral agent build and deployment logs
- Manually removing pods
- Checking the log for a pod
- Pod and ephemeral agent management

Checking the log for a pod

You can view the logs for each individual Kubernetes pod to check the status of your applications and troubleshoot problems. By default, you have to refresh the log manually, but there is also the option of enabling live logs, which makes the content refresh automatically every 10 seconds.

To view the logs for a pod:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Pods**.
3. On the **Manage pods and ephemeral agents** page, select a pod from the list.
4. On the **Pod details** page, under **Pod logs**, select **Show**.

   You can control the log's display size by selecting the number of lines to show from the **Log size** menu.

5. To refresh the log, select **Refresh**.
6. Optionally, if you want to see a live log (refreshed automatically every 10 seconds), under **Fetch log**, select **Automatically**.

Related pages

- Checking ephemeral agent build and deployment logs
- Manually removing pods
- Viewing pod and ephemeral agent details
- Pod and ephemeral agent management

Checking ephemeral agent build and deployment logs

If you want to find out what's happening with the build or deployment that an ephemeral agent is currently carrying out, you can go to the log displayed on the build dashboard either straight from the **Manage pods and ephemeral agents** page or from the **Agent summary** page.

To view the logs for an ephemeral agent:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Pods**.
3. In the list of existing pods and agents on the **Manage pods and ephemeral agents** page, complete one of the following actions:
    - Select the link in the **Status** column next to the ephemeral agent whose logs you want to view.
    - Select the name of the ephemeral agent whose logs you want to view, and then, on the agent summary page, select the link in the **Current status** field.

Related pages

- Manually removing pods
- Viewing pod and ephemeral agent details
- Checking the log for a pod
- Pod and ephemeral agent management

Manually removing pods

If you haven't enabled automatic pod removal in Bamboo, you can still remove a pod manually from the **Manage pods and ephemeral agents** page. If you want to, you can also remove all existing pods at once.

To remove a single pod from the cluster:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Pods**.
3. In the list of existing pods on the **Manage pods and ephemeral agents** page, select **Remove** next to the pod that you want to remove.
4. In the confirmation dialog, select **Confirm**.

To remove all existing pods at once:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. From the menu on the **Bamboo administration** page, under **Ephemeral agents**, select **Pods**.
3. In the upper-right corner of the **Manage pods and ephemeral agents** page, select **Shut down all pods**.

> The **Shut down all pods** button appears only if there is at least one pod in the list.

Related pages

- Checking ephemeral agent build and deployment logs
- Viewing pod and ephemeral agent details
- Checking the log for a pod
- Pod and ephemeral agent management

Working with Elastic Bamboo

Elastic Bamboo allows you to use computing resources from the Amazon Elastic Compute Cloud (EC2) to run builds. Elastic Bamboo uses a remote agent AMI (Amazon Machine Image) to create instances of remote agents in the Amazon EC2.

The following pages and sub-pages describe how to work with Elastic Bamboo:

- About Elastic Bamboo — Elastic Bamboo concepts.
- Getting started with Elastic Bamboo — setting up Elastic Bamboo for the first time. It contains instructions on enabling Elastic Bamboo for your Bamboo installation and running your first build.
- Configuring Elastic Bamboo — changing settings for Elastic Bamboo. This includes instructions on how to use Amazon's Elastic Block Storage to persist build information for your builds on Elastic Bamboo.
- Managing your elastic images
- Managing your elastic instances
- Managing your elastic agents
- Elastic Bamboo FAQ — running your builds using Elastic Bamboo.
- Elastic Bamboo Security — setting up secure communication between Bamboo and the EC2.

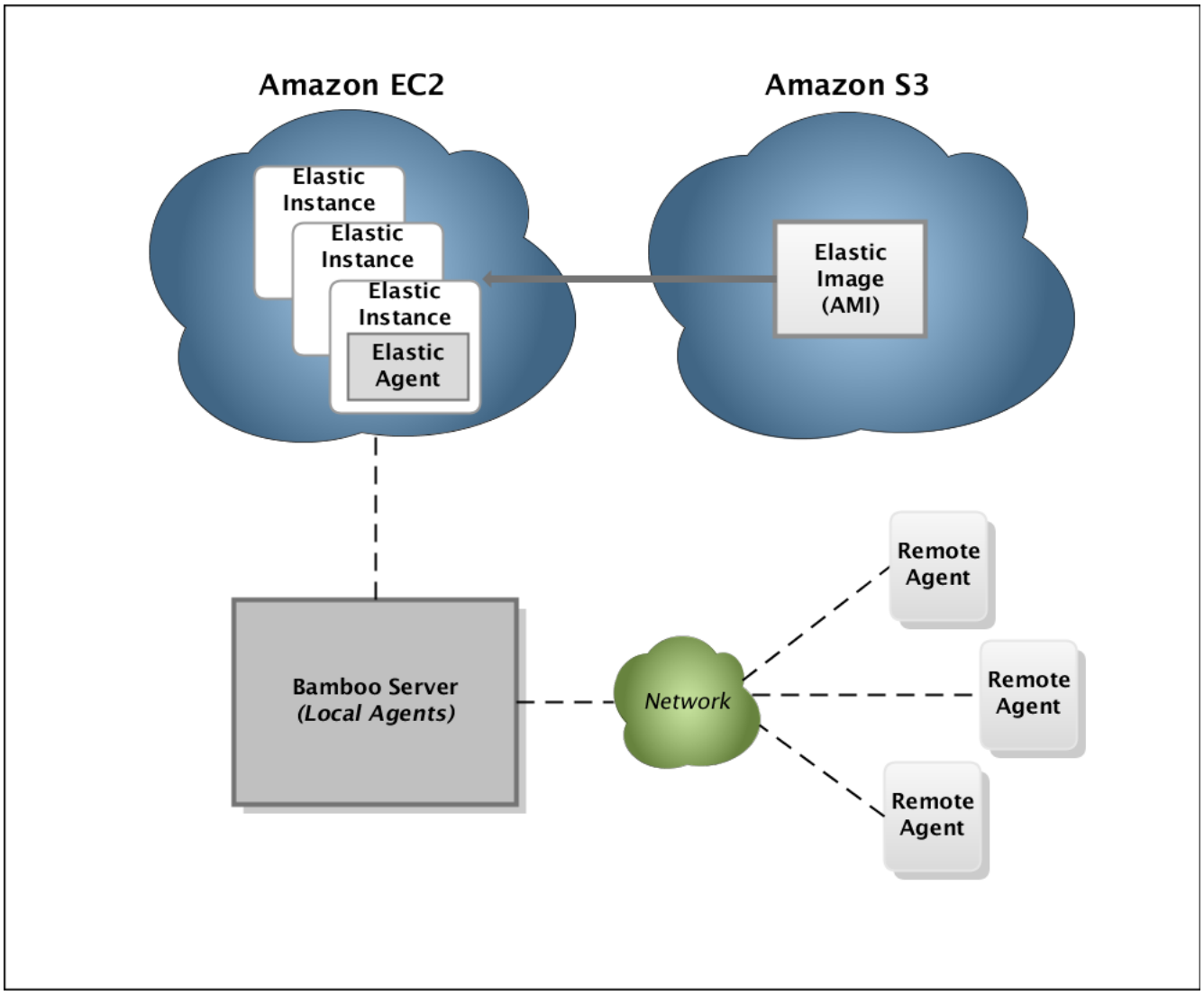
About Elastic Bamboo

On this page:

Conceptual Overview

Elastic Bamboo allows you to use computing resources from the Amazon Elastic Compute Cloud (EC2) to run builds. Elastic Bamboo uses a remote agent AMI (Amazon Machine Image) to create instances of remote agents in the Amazon EC2.

Elastic Bamboo Conceptual Overview



A Bamboo build job can be run on an elastic agent, provided that the capabilities of the elastic agent meet the requirements of the job. Bamboo will assign the relevant job to an available elastic agent from the build queue automatically. The elastic agent must already be running for a job to be assigned to it.

An elastic agent is started by creating a new instance of an elastic image. Creating this new elastic instance automatically runs an elastic agent process in the instance. The agent inherits the capabilities of the image it was created from. Only one agent process can be run in an instance, although multiple instances can be created from the same image.

Once a job has completed running on an elastic agent, its results are made available (like those of any other job executed on a non-elastic agent). The elastic agent and instance will continue to run until they are shut down. Shutting down an elastic instance will terminate the agent, not take it offline. However, Bamboo will store historical information about the terminated elastic agent, such as the job which it has run.

An Amazon Web Services (AWS) account is required to use Elastic Bamboo. Elastic Bamboo Costs are charged by Amazon, separate to Bamboo license costs, as Elastic Bamboo is powered by Amazon resources.

Did you know you can configure Bamboo to start and shut down elastic instances **automatically**, based on build queue demands? Please refer to Configuring Elastic Bamboo for more information.

Key Terms

Elastic Image
An *elastic image* is an Amazon Machine Image (AMI) that is stored in one of Amazon data centers for use with the Elastic Bamboo feature. An elastic image is used to create elastic instances, which in turn create elastic agents. Conceptually, an elastic image is equivalent to an operating system running on a computer's boot hard drive and elastic instances would be the software that runs on this operation system.

Each elastic image registered with the Amazon Web Services (AWS) has its own unique identifier, known as an **AMI ID**.

You can associate multiple elastic images with a Bamboo server. One default shared image is maintained by Atlassian in AWS, and is available to all Elastic Bamboo users.

You can also create your own custom elastic images.

Elastic Instance

An *elastic instance* is a running instance of an elastic image. One elastic instance is created whenever an elastic image is started. Hence, starting one elastic image multiple times, results in the creation of multiple elastic instances. Each time an elastic instance is created, one elastic agent is created on that instance.

Conceptually, an elastic instance can be thought of as a computer. The elastic agent's processes are run on this computer and the elastic image is the boot hard drive. Unlike computers, however, elastic instances are temporary and stateless. When an elastic instance is shut down:

- Any changes that an elastic instance makes to the boot hard drive (e.g. agent log file) will not persist
- Any customizations to the instance itself will also be lost.

The Amazon Elastic Block Store can provide persistent storage for your elastic instances.

Elastic Agent
An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

Setting Up Elastic Bamboo

If you would like to set up Elastic Bamboo for your Bamboo installation, please read Getting started with Elastic Bamboo. This document guides you through the initial configuration of Elastic Bamboo and running your first Job build.

Elastic Bamboo Costs

This page provides high level guidelines to Elastic Bamboo costs. As usage patterns vary from user to user, these guidelines are only intended to provide a picture of how Elastic Bamboo operates, not to make definitive pricing statements.

> The Bamboo pricing page on the Atlassian website details the costs for Elastic Bamboo. This page is intended to complement that information.

Amazon EC2 Pricing Information

You can use Elastic Bamboo to run remote agents on elastic instances in the Amazon Elastic Compute Cloud (EC2). If you choose to do this, you will be charged by Amazon for your EC2 compute usage. These charges will be billed to the AWS account that you provide.
Please note, if you do not have an AWS account, you must register for one on the AWS registration page before you can enable Elastic Bamboo.

Full details on Amazon EC2 pricing is available on the Amazon EC2 pricing page. Please also note the following important information, which is relevant to EC2 usage by Elastic Bamboo:

- **You are responsible for all EC2 usage costs incurred on your AWS account.**
- Elastic Bamboo creates "High-CPU Medium" Instances by default, however you can configure the EC2 instance type. Read Managing your elastic image configurations for instructions on how to change your default instance type. Please note the different costs for different instance types.
- You are responsible for creating and shutting down resources required to run agents in EC2.
- You can track your EC2 usage in near real-time on the AWS Account page.
- Your Elastic Bamboo compute usage will not be distinguishable from your non-Bamboo EC2 compute usage in your AWS billing.

General Notes about EC2 Usage and Costs

The following information is based on our usage of Elastic Bamboo at Atlassian. These points are intended to be guidelines to EC2 usage and costs only.

- The bulk of EC2 costs from using Elastic Bamboo is for the uptime of EC2 instances. We strongly recommend that you shut down your instances when not in use.
- The costs for storing and moving data in and out of the EC2 will vary. However these costs are minimal (e.g. storing image) compared to instance uptime costs. Using the Amazon Elastic Block Store (EBS) with Elastic Bamboo can significantly reduce the data transfer (and associated costs) in and out of the EC2. Read more about configuring elastic instances to use EBS.
- The costs for using the Amazon Elastic Block Store (EBS) is minimal, relative to instance uptime costs.

Elastic Bamboo Security

Elastic Bamboo is a feature in Bamboo that allows Bamboo to dynamically source computing resources from the Amazon Elastic Compute Cloud (EC2).

All traffic sent between the agents located in EC2 and the Bamboo server is tunneled through an SSL-encrypted tunnel. The tunnel will be initiated from the Bamboo Server to the EC2 instance, which means that you don't need to allow any inbound connections to your server. You will need to permit outbound traffic from the server on the tunnel port, however - the default port number is 26224. On the EC2 instance, only the tunnel port needs to be open for inbound traffic.

SSL tunneling is not implemented for VCS (Version Control System) to EC2 traffic though. You will need to make your VCS available for access from EC2 to use Elastic Bamboo. See the section on setting up your VCS for Elastic Bamboo, which contains guidelines on securing your VCS.

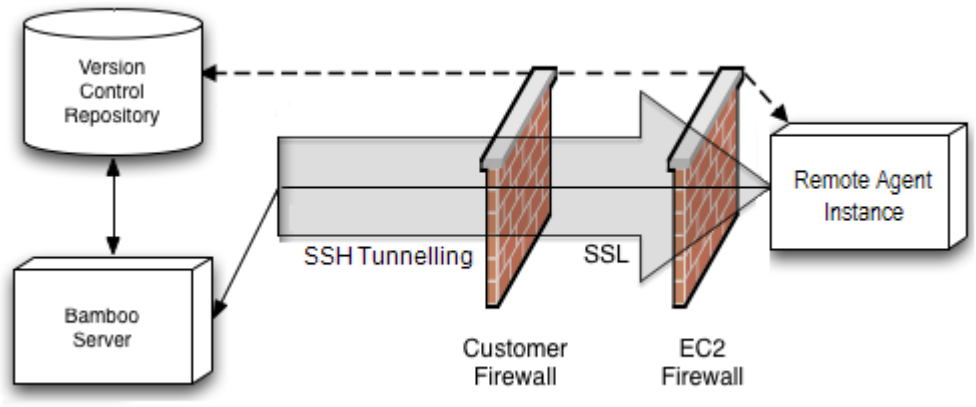
Be warned that just as with a regular host accessible from the Internet, if one of your remote agent instances is compromised, your Bamboo installation may be exposed to number of security vulnerabilities. These include confidential data (e.g. source code, VCS credentials) being stolen, malicious code being injected into elastic agents, unauthorized access to build queues and false information being submitted to Bamboo servers. Given that all Bamboo-related traffic is sent through a single encrypted connection, the risk of that happening is not high and can be further mitigated by setting up a VPC (Amazon Virtual Private Cloud). In a VPC, your elastic instances typically have no public IPs which means they are inaccessible from the internet other than through a regular, industry-standard VPN connection.

The sections below explain the default access rules for remote agent instances and how to change these rules, if desired.

On this page:

Related pages:

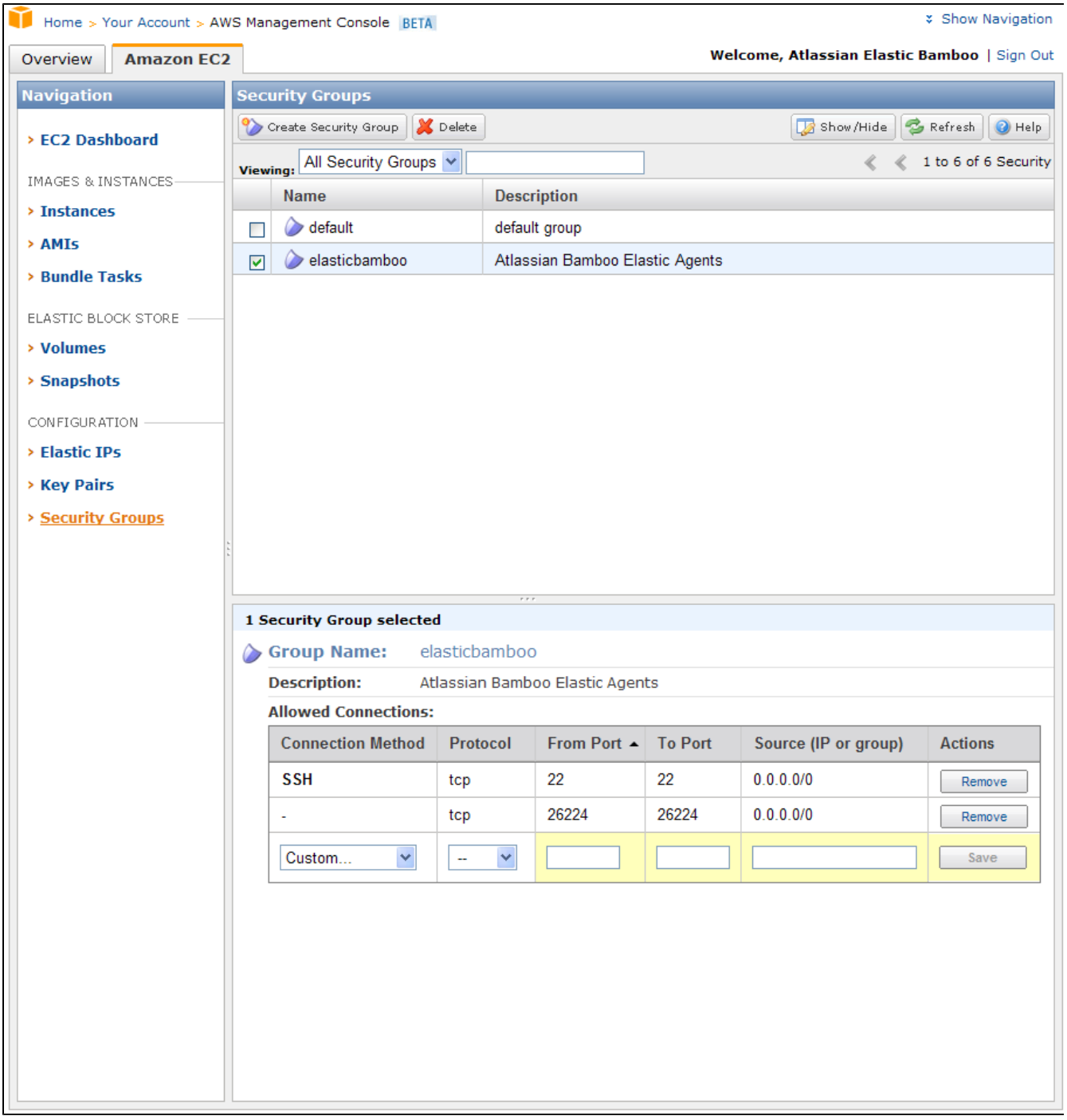
- Configuring Elastic Bamboo



Default EC2 Access Rules

When you first use Elastic Bamboo, i.e. start an elastic instance, an *elasticbamboo* security group will be set up for you on your AWS account. This security group is essentially a set of IP addresses that are permitted access to the EC2. By default, the security group will contain two rules — one to allow connections for Elastic Bamboo itself, and another to allow connections via SSH.

The EC2 security groups can be accessed via the AWS management console (see **Security groups** in the left-hand menu under **Configuration**).



Changing the Default EC2 Access Rules

If you wish to permit additional connections to your EC2 instance, you can do this by adding entries to the **Allowed connections** section for the *elasticbamboo* security group. See the previous section on Default EC2 Access Rules for instructions on how to access your EC2 security groups.

Using VPCs with Elastic Bamboo

VPC functionality is available with Bamboo 4.3. Amazon Virtual Private Cloud (Amazon VPC) lets you provision a private, isolated section of the Amazon Web Services (AWS) Cloud where you can launch AWS resources in a virtual network. By default, the instances running in that network will have no public IPs and will not be accessible to the computers outside of your VPC. You can also create a Hardware Virtual Private Network (VPN) connection between your company datacenter and your VPC and leverage the AWS cloud as an extension of your company datacenter. You can read more about VPCs on Amazon Web Services VPC page.

Using a VPC means that your agents (and other instances launched in the VPC) will not be available on the Internet. There are several basic scenarios that can be realized using a VPC:

- Secure access to your company datacenter - agents can securely access resources from your internal network through a VPN connection. In this way, you can safely use your Version Control System or other internal resources such us databases from your Elastic Agents - without making them publicly accessible.
- Hiding some EC2 instances from the Internet - agents can communicate with your other hosts on the VPC using the internal network. This lets you e.g. set up an agent with a Windows-based DBMS and another one that runs tests against that DBMS from a different platform. Computers from outside of the VPC will not be able to access the DBMS because it will have no external IP. You don't need to use VPN for that use case, it's enough to assign an Elastic IP to the agent.
- Full-cloud deployment - you can host your Bamboo server in an Amazon's VPC and hide all your agents in a VPC. This will also let you access your other resources located in a VPC. The Bamboo Server can be accessed using VPN or an Elastic IP.

Setting up your Version Control System (VCS) for Elastic Bamboo

We recommend that you take the following steps to ensure that your Version Control System is set up securely for Elastic Bamboo:

1. Make your Version Control System accessible to the public internet
2. Use VCS authentication and access control
3. Use encrypted connections to VCS

1. Make your Version Control System accessible to the public internet

> ⚠ You only need to do this if you aren't using a VPC for agent connectivity. See using Bamboo with VPCs for more information.

As SSL tunneling is not implemented for VCS to EC2 connections, you will need to make your VCS accessible to the public internet to use Elastic Bamboo. If your VCS is behind a firewall this will involve configuring an access point in your firewall. Consult the documentation for your firewall software for details on how to do this.

2. Use VCS authentication and access control

We highly recommend that you secure access to your VCS by enabling the authentication and access control features on your VCS. Consult the documentation for your VCS for details.

3. Use encrypted connections to VCS

We also highly recommend that you use encrypted connections for your VCS (e.g. SSL). Consult the documentation for your VCS for details.

Getting started with Elastic Bamboo

Elastic Bamboo allows you to use computing resources from the Amazon Elastic Compute Cloud (EC2) to run builds. Elastic Bamboo uses a remote agent AMI (Amazon Machine Image) to create instances of remote agents in the Amazon EC2.

On this page:

1. Read important documents

If you are using Elastic Bamboo for the first time, we highly recommend that you start by reading the following important documents:

- **About Elastic Bamboo** — This high-level overview explains the key concepts behind the Elastic Bamboo feature.
- **Elastic Bamboo Security** — We *strongly recommend* that you read this document to understand the security implications of enabling Elastic Bamboo. This includes important information on securing your version control system (VCS) for use with Elastic Bamboo.
- **Elastic Bamboo Costs** — Elastic Bamboo sources resources from the Amazon Elastic Compute Cloud (EC2) which are charged separately to your Bamboo license fee. We recommend that you read this document to understand how you will be charged for using Elastic Bamboo.

2. Enable and configure Elastic Bamboo

Once you have understood the concepts, security implications and costs of Elastic Bamboo, you can enable and configure Elastic Bamboo for your Bamboo installation. You will also need to make your version control system (VCS) available to Amazon for Elastic Bamboo to work correctly.

2.1. Enabling Elastic Bamboo

To enable Elastic Bamboo:

1. Enable remote agent support in Bamboo — if you have disabled remote agent support, you must enable it before you can enable Elastic Bamboo. The Disabling and enabling remote agents support documentation also contains instructions on how to enable remote agent support.
2. From the Bamboo top navigation bar select  > **Elastic Bamboo** > **Configuration**.
3. Select **Enable**.

2.2. Configuring Elastic Bamboo

Before you can use Elastic Bamboo, you must configure it as detailed in the Configuring Elastic Bamboo document. This is a simple three-step process:

1. Provide your Amazon Web Services account details.
2. Configure your Elastic Bamboo global settings.
3. Configure your elastic instance settings.

- **Read the Configuring Elastic Bamboo document**.

2.3. Providing access to your VCS

You need to make your version control system available to Amazon to run job builds using Elastic Bamboo. This has security implications, particularly if your VCS is behind a firewall.

- **Read the Elastic Bamboo Security document for further instructions,** if you have not read it already.

3. Start an Elastic Instance

Now that you have enabled and configured Elastic Bamboo for your Bamboo installation, you can try building a plan with Elastic Bamboo. You can manually start an elastic instance using the Bamboo administration console. Starting an elastic instance will automatically start an elastic agent process on it.

- **Read about starting an elastic instance**.

4. Run a plan build

To run a plan build on your elastic agent, you must set up a plan with its Default Job (plus any other optional jobs) all of whose requirements can meet your elastic agent's capabilities. Elastic agents inherit the capabilities of the image they are started from. We recommend that you use the Bamboo default image to start with.

- **Read about the capabilities of the default image**.

For the purposes of this guide, you should set up your plan so that its jobs' requirements can *only* be met by the elastic agent's capabilities. This will ensure that the jobs' builds run on your elastic agent. If you cannot set up your jobs' requirements to meet your elastic agent's capabilities, you can customize your elastic agent's capabilities to add a unique custom capability, e.g. '`elastic=true`').

- **Read about configuring the capabilities of elastic agents**.

Job builds on elastic agents are run just like job builds on any other agent. You will see the progress of your build on your dashboard and can view the build result when it has completed.

Tip: *You can significantly reduce the costs and time taken to run a job build by configuring Elastic Bamboo to use Amazon's Elastic Block Store (EBS).*

5. Shut down your Elastic instance

When your job builds successfully, shut down your elastic instance. As described in Elastic Bamboo Costs, the bulk of your Elastic Bamboo costs are from instance uptime. We strongly recommend that you shut down your elastic instances when not in use.

- **Read about shutting down an elastic instance**.

Please note, that when you shut down an elastic instance, the agent process it is running is terminated. This means that elastic agents are not present on the 'Agents' page in Bamboo unless they are online. If you wish to view information about a terminated elastic agent, you can find the agent in the elastic agent usage history.

- **Read about viewing your elastic agent usage history**.

Congratulations! You have successfully set up and run a job build with Elastic Bamboo.

Further information

You may be interested in reading the following related topics below to help you manage and improve Elastic Bamboo's handling of job builds:

- **Managing your elastic images**, **Managing your elastic instances**, **Managing your elastic agents** — information hubs for managing Elastic Bamboo images, instances and agents.

- **Elastic Bamboo FAQ** — general questions about running builds using Elastic Bamboo.
- **Configuring elastic instances to use the EBS** — information on configuring Elastic Bamboo to use the Amazon Elastic Block Store (EBS) to improve job build times.

Configuring Elastic Bamboo

Elastic Bamboo allows you to use computing resources from the Amazon Elastic Compute Cloud (EC2) to run builds. Elastic Bamboo uses a remote agent AMI (Amazon Machine Image) to create instances of remote agents in the Amazon EC2.

Builds run on these elastic agents in a similar way to how they run on local and remote agents.

⚠ *If you have disabled remote agent support, you **must enable it** before you can enable Elastic Bamboo. Refer to Disabling and enabling remote agents support for instructions on how to enable remote agent support.*

To configure your Amazon Web Services (AWS) account details or settings for Elastic Bamboo:

1. From the top navigation bar select  > **Elastic Bamboo** > **Configuration**.
2. Select **Edit configuration**.
3. Configure settings as described in the sections below.
4. Select **Save** when finished.

On this page:

Related pages:

- AWS account for Bamboo
- Configuring elastic instances to use the EBS
- Disabling Elastic Bamboo

Configuring AWS account settings

Before you use Elastic Bamboo for the first time in your Bamboo instance, enter your Amazon Web Services (AWS) account details into the Bamboo application. You can authenticate with AWS using an AWS Access Key or an EC2 instance profile.

ⓘ If you change your AWS account details, Bamboo will stop all currently running elastic agents.

Before you begin

- If you don't have an AWS account, sign up for one on the AWS registration page.
- Read Elastic Bamboo Costs for more details about how Amazon will charge you for EC2 compute resource usage. Amazon EC2 is sold separately from your Bamboo license and all fees are billed to your AWS account.

Using an AWS access key ID and secret access key

If you're unsure what your AWS access key ID and secret access key are, see Understanding and getting your AWS credentials — AWS General Reference.

To enter or update your AWS access key ID and secret access key:

1. From the top navigation bar, select **Administration** ( ) > **Elastic Bamboo**.
2. From the left menu, select **Configuration**.
3. Select **Edit configuration**.
4. Under **Amazon Web Services configuration**, select **Access key**.
5. Enter your AWS access key ID, secret access key, and select the region.

Using an EC2 instance profile

If you're running Bamboo on an EC2 instance in AWS, you can utilize an instance profile to configure Elastic Bamboo. An instance profile is a container for an IAM role attached to an EC2 instance that provides short-lived, periodically rotated credentials. Bamboo can automatically detect and use such credentials to manage EC2 instances for Elastic Agents.

To configure Elastic Bamboo using an EC2 instance profile:

1. Go to **Administration** ( ) > **Elastic Bamboo**.
2. From the left menu, select **Configuration**.
3. Select **Edit configuration**.
4. Under **Amazon Web Services configuration**, select **Instance profile**.

> The **Instance profile** option is available only if you're running Bamboo on an EC2 instance.

Global settings

Elastic Bamboo provides you with a number of global configuration options to help you optimize EC2 usage for your Bamboo job builds. These settings control how the Bamboo server operates and how it manages its elastic instances and agents.

Maximum number of elastic instances
The number of elastic instances that can be running at any one time. You may wish to decrease this value if you are concerned about EC2 compute costs, and you have a large number of concurrent job builds that cannot be supported by your non-elastic agents.

Automatically terminate elastic instance when elastic agent process ends

Controls whether your elastic instances will automatically shut down after the elastic agent processes running on them terminate.

Shutdown delay

Controls how long an elastic instance will wait before shutting down, after its elastic agent process terminates.

EC2 spot instances

Elastic Bamboo provides support for Amazon EC2 Spot Instances. Amazon spot instances allow you to buy unused EC2 capacity. You can configure Elastic Bamboo to buy you a spot instance of a particular type, and fall back to a regular instance after a set amount of time if no instances are available.

Enable support for spot instances
Select this checkbox to enable support for spot instances.

Fallback to a regular instance after
The time (in minutes) after which Elastic Bamboo will fall back to using a regular instance, if a spot instance hasn't become available.

AWS settings

These settings allow you to specify your AWS configuration settings in Bamboo so that Bamboo can operate elastic instances through your AWS account. This section includes settings that are used to configure elastic instances to work with the Amazon Elastic Block Store (EBS).

Using EBS with your elastic instances can significantly reduce the amount of data transfer required to run a job build, compared with starting a clean elastic instance. To find out more about this feature and how to set it up in Elastic Bamboo, read Configuring elastic instances to use the EBS.

Upload AWS account identifiers to new elastic instances
Select to upload the AWS Account Private Key File and Account Certificate File to all new elastic instances started. This is mandatory if you wish to use EBS to store job build information in a snapshot. However, you can also select this option if you are not using EBS (e.g. if you wish upload the AWS account identifiers in order to use Amazon's AWS command line tools).

> Note that your AWS access key ID and secret access key will be be uploaded to your Elastic Instances only if you configured Elastic Bamboo with an AWS access key. If you're using an instance profile, the AWS access key ID and secret access key won't be uploaded.

Key files location
Choose how private key and certificate will be provided.

Account Private Key File
You must specify the location of this file to use the Amazon EBS with Elastic Bamboo. This file is generated by Amazon.

Account Certificate File
You must specify the location of this file to use the Amazon EBS with Elastic Bamboo. This file is generated by Amazon.

If you haven't downloaded an AWS private key file or certificate file to your Bamboo server yet, see IAM best practices on the Amazon page.

Automatic elastic instance management

The Automatic Elastic Instance Management feature allows Bamboo to start and shut down elastic instances automatically (based on build queue demands), so that you do not have to perform these action manually. This feature reduces Bamboo administration overhead and can help minimize your overall elastic instance usage costs.

If a job's requirements cannot be met by any available online agents, this feature will start any elastic instance whose elastic agent has the capabilities to execute the job, so that the job's build can be generated. Regardless of how an elastic instance was started, all elastic instances will be shut down based on the settings specified below.

Elastic instance management
Select from the following elastic instance management presets. Each of these presets define values for the five criteria described in the Custom user-defined options (below). (Bear in mind that both the Aggressive and Passive presets have trade-offs.)

- **Default** — Balances build queue clearance rates with elastic instance usage costs.
- **Aggressive** — Favors higher build queue clearance rates but with higher elastic instance usage costs.
- **Passive** — Favors lower instance usage costs but with lower build queue clearance rates.
- **Custom** — Select your own settings, as described below.
- **Disabled** — Disables Bamboo's automatic elastic instance management feature.

Idle agent shutdown delay
Specify the number of minutes that an elastic agent must be idle before Bamboo shuts down the elastic instance running that agent.
Elastic instances running in the Amazon EC2 compute cloud are charged in hourly blocks from the time they are started. To maximize usage of elastic instances in a cost-effective manner, Bamboo only performs these checks just prior to the expiry of each hourly block.

Allowed non-Bamboo instances
The maximum number of elastic instances allowed on your AWS account that are not controlled by this Bamboo instance. When this limit is exceeded, Bamboo will not start any new instances.

Maximum number of instances to start at once
The maximum number of elastic instances that Bamboo can start in one go. Bamboo only starts this maximum number of elastic instances on a per minute basis.

Number of builds in queue threshold
The total number of builds in a queue. When this and all other thresholds have been reached, new elastic instances will be started.

Number of elastic builds in queue threshold
The number of builds in the queue that can be executed on elastic instances. When this and all other thresholds have been reached, new elastic instances will be started.

Average queue time threshold
The average number of minutes that job builds have been waiting in a queue. When this and all other thresholds have been reached, new elastic instances will be started.

Generating your AWS Private Key File and Certificate File

There are several security mechanisms associated with Amazon Web Services (AWS) and EC2:

- The AWS Access Key ID and Secret Access Key that are used by the Bamboo server to authenticate with AWS.
- A login key pair that you can use to log in to EC2 instances that have been started by Bamboo. The key pair is automatically generated, either the first time you use Elastic Bamboo, or if you delete the key pair. The key pair is listed as "elasticbamboo" in your AWS console. Bamboo does not use this key pair.
- The AWS private key file and certificate file that are generated by Amazon and used together to allow Elastic Bamboo to securely access some of the AWS services, such as EBS for elastic instances and the Amazon command line tools. These are described below.

On this page:

Related pages:

- Configuring Elastic Bamboo

AWS private key file and certificate file

⚠ These settings were used for older Bamboo setup. Since then, we've introduced a simplified mechanisms for working with instances as described above.

The Amazon Web Services (AWS) private key file and certificate file are generated by Amazon and work together to allow Elastic Bamboo to securely access your AWS account. These are required to enable certain features, such as EBS for elastic instances and the Amazon command line tools.

- The **certificate file** contains the public key associated with your AWS account. This file is kept by Amazon, not on your Bamboo server.
- The **private key file** contains the private key that is used to authenticate requests to AWS. This file must be stored on your Bamboo server, if you are using EBS for elastic instances or the Amazon command line tools.
- The public key and private key from these files together form an **X.509 certificate**.

Generating the files

The certificate file will be kept by Amazon (to inject into your elastic instances) and the private key file will be downloaded to your Bamboo server in your Bamboo Home directory. If you are setting up Elastic Bamboo on multiple Bamboo servers using the same AWS account, you can simply copy the private key file across from the original Bamboo server. You should not need to regenerate the private key file and certificate file unless your private key file is lost or corrupted.

If you do need to regenerate the private key file and certificate file, please follow the instructions in the Amazon X.509 Certificates documentation. The Amazon documentation also contains instructions on using your own certificate, if you wish.

Downloading the files

We recommend that you store the files in the Home directory of your Bamboo server.

Notes

- If you wish to use this security mechanism with **multiple Bamboo installations using the same AWS account** (e.g. you have configured your elastic instances on each installation to use EBS), you will need to copy the AWS private key file and certificate file to each Bamboo server.
- You can only download the AWS private key file **at the time it is generated**. If the private key file has already been generated for your AWS account, you will not be able to download it from AWS again (for security purposes). You will have to copy it from wherever it was previously downloaded to. Otherwise you will have to generate a new private key file and certificate file to go with it.
  ⚠️ *If you regenerate a new private key file and certificate file, any Bamboo servers using the old private key file and certificate file will no longer be able to access the Amazon EC2, as only one X.509 certificate can be associated with your AWS account.*
- You can download the AWS certificate file as many times as you want. This file does not need to be regenerated.

Configuring elastic instances to use the EBS

The Amazon Elastic Block Store (EBS) provides 'EBS volumes' which can attach to EC2 instances. EBS volumes (and the 'EBS snapshots' created from these volumes) provide persistent storage for your elastic instances.

If you have relatively static resources required for building your Bamboo jobs (such as, source code checkouts and Maven repository artifacts), you can add these to an EBS volume. From this volume, you can create an EBS snapshot, which effectively records the 'state' of an EBS volume at a given point in time.

After setting up an EBS snapshot, you can then associate it with an elastic image configuration. When this elastic image is started:

- its elastic instance will be started, along with the EBS volume (derived from the EBS snapshot associated with the elastic image) and
- this EBS volume will be attached to this elastic instance
  any build resources (added to the EBS volume prior to creating its snapshot) will be available to this elastic instance.

Why should I use the EBS with Elastic Bamboo?
Because an elastic instance is stateless, so also is the elastic agent that runs on it. Hence, every time an elastic instance is restarted from the same image:

- Any resources that its elastic agent must retrieve externally (for example, Maven repository artifacts), must be downloaded in their entirety.
- Full checkouts must be performed by elastic agents when new Jobs are built.

Therefore, you can use the EBS to store these external resources in an EBS volume and snapshot so that they do not have to be downloaded or source code checked out each time you start up an elastic instance from an image. If your jobs rely heavily on downloading such resources and/or you are not performing clean builds each time, the EBS may significantly **improve your build times**.

Additionally, the EBS provides an easy mechanism for **customizing elastic agents**, rather than you having to create a custom elastic image from scratch (with your own elastic agent capabilities). For example, you could upload files and scripts to your EBS volume that would install resources such as PostgreSQL databases for your elastic agents, which will be available when the agent's elastic instance is started.

On this page:

Related page:

- Configuring Elastic Bamboo
- Populating your EBS volume

Creating your first EBS snapshot

To create your first EBS snapshot:

1. Download Amazon Web Services (AWS) account identifiers to your Bamboo server — You will need to store the AWS private key file and certificate file on your Bamboo server to use Elastic Bamboo with EBS. If you haven't downloaded an AWS private key file or certificate file to your Bamboo server yet, please see Generating your AWS Private Key File and Certificate File for instructions.
2. Update your Bamboo configuration settings with the location of the AWS account identifier files you have downloaded. This will ensure that these files are uploaded to any new elastic instances started. See the Elastic Instance Settings section on the Configuring Elastic Bamboo for instructions (you will need to

update the **Upload AWS account identifiers to new elastic instances (mandatory if EBS Snapshot ID specified)** checkbox and **Account Private Key File** and **Account Certificate File** fields described on this page).

3. Start a single elastic instance via Bamboo. See Starting an elastic instance for instructions.
4. Access your elastic instance via SSH (see Accessing an elastic instance for instructions).
5. Log in as and administrator, such as **root** in Linux and, in Linux, make sure to load the root user's environment as below:

```
sudo su -
```

> In this case, the '-' or the '-l' or the '-login' parameters is required, otherwise some of the scripts may fail.

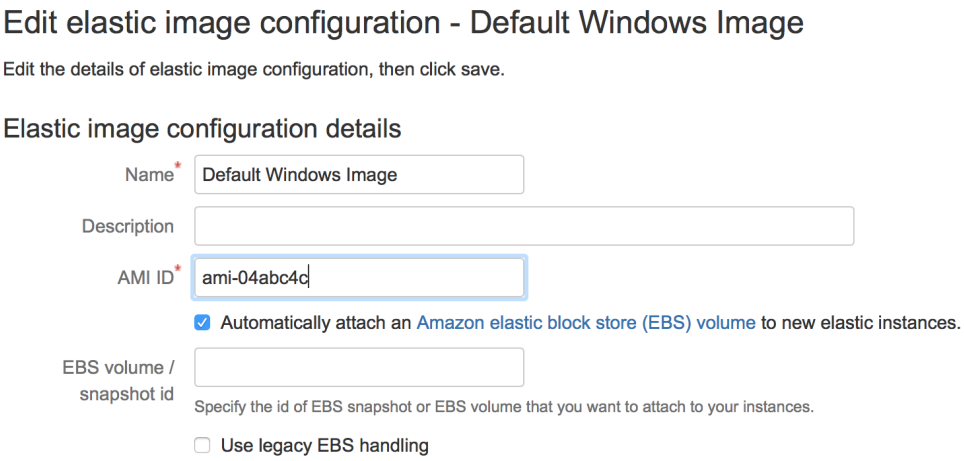
6. Follow the steps below to create an EBS volume and attach it to the elastic instance (*steps a & b*), upload content to the EBS volume (*step c & d*), and generate the snapshot (*step e & f*):
   All the scripts mentioned below are available in /opt/bamboo-elastic-agent/bin on Bamboo stock images. You can also download them from here (choose the latest version).
   a. Run `createInitialVolume.sh` *<volume size>* — This script creates a EBS volume (where <volume size> is the size of the volume), attaches the volume and mounts it on the elastic instance. For example, `createInitialVolume.sh 100` will create a 100GB EBS volume and attach and mount it on the elastic instance.
   b. Run `rewarmEbsSnapshot.sh` — This script sets up the standard structure for Elastic Bamboo on the EBS volume. The directories and files for this standard volume structure are detailed in the Important EBS Directories and Files section below.
   c. *(optional)* Populate your EBS volume — Your EBS volume can now be populated with any files and scripts that you wish to make available to the elastic instances that use the EBS volume. For example, you may want to upload maven repository data, source code, scripts and files to install databases on your elastic agents, etc. You must upload your files to the **`/mnt/bamboo-ebs`** folder or its subfolders, if you want them to be included in the snapshot. We recommend that you read Populating your EBS volume for guidelines on how to populate your EBS volume effectively.
      The EBS volume is attached to the elastic instance, so accessing your elastic instance via SSH will give you full access to the EBS volume (see Important EBS Directories and Files below).
   d. Ensure all uploaded content has the owner **bamboo:bamboo** — You can set the owner of a file by executing the following command: `chown -R bamboo:bamboo` *<filename>*
   e. Execute the `killall java` command — This command kills all processes on the instance, such as agent processes, so that the volume can be unmounted to be snapshotted.
   f. Run `generateSnapshot.sh` — This script unmounts and detaches the volume, before creating a snapshot based on the volume. The time taken to create the snapshot will vary depending on the amount of content that you have uploaded to the EBS volume. The **Snapshot ID** for the snapshot will be available in the logs for the elastic instance. See Accessing an Elastic Instance for instructions on how to access the logs for your elastic instance.
      The device can not unmount if any terminals are currently in the mounted volume.
7. Shut down your elastic instance. See Shutting down an elastic instance for instructions.

Configuring an elastic image to use an EBS snapshot

Once you have set up an EBS snapshot, the final step is to add the snapshot details to an elastic image configuration, so that any instances started from that image will have EBS volumes attached to them. You can associate different snapshots with different elastic image configurations.

To configure Elastic Bamboo to use an EBS snapshot:

1. Determine the Snapshot ID of the EBS snapshot you have just created. The Snapshot ID should be recorded in the logs of the elastic instance you created it on. You can also view your EBS snapshots in the AWS Console by selecting the **Snapshots** menu item.
2. From the Bamboo top navigation bar select  > **Elastic Bamboo** > **Image configurations**.
3. In the **Operations** column select **Edit** for the elastic image configuration that you would like to add your EBS snapshot to.



4. Select **Automatically attach an Amazon Elastic Block Store (EBS) volume to new elastic instances**.
5. Enter the Snapshot ID of your EBS snapshot in the **EBS Snapshot ID** field.
6. Select **Use legacy EBS handling** to resolve EBS issues for images older than two years.
7. Select **Save**. A new EBS volume will be created from the specified snapshot and attached to any new elastic instances started from that image.

Updating your EBS snapshot

If you are currently using EBS with Elastic Bamboo and want to update your snapshot, follow the instructions below. These are similar to the instructions for creating a new EBS snapshot.

To update your EBS snapshot:

1. Start a single elastic instance via Bamboo. See Starting an elastic instance for instructions.
2. *(optional)* Run a build on the elastic agent of the instance to populate the attached EBS volume. We recommend that you read Populating your EBS volume for guidelines on how to populate your EBS volume effectively.
3. Access your elastic instance via SSH (see Accessing an elastic instance for instructions) and do the following:
   All the scripts described below are bundled with Bamboo.
   a. Log in as and administrator, such as **root** in Linux and, in Linux, make sure to load the root user's environment as below:

      ```
      sudo su -
      ```

      In this case, the '-' or the '-l' or the '-login' parameters is required, otherwise some of the scripts may fail.

   b. *(optional)* Upload any additional content to the attached EBS volume via Secure Copy (SCP). You must upload your files to the **`/mnt/bamboo-ebs`** folder or its subfolders, if you want them to be included in the snapshot.
   c. Execute `killall java` — This command kills all agent processes, so that nothing is using the mounted volume.
   d. Execute `jps -vl` — This command displays a list of all java processes running on your instance. There should be no java processes running.
   e. Run `generateSnapshot.sh` — This script unmounts and detaches the volume, before creating a snapshot based on the volume.
      The device can not unmount if any terminals are currently in the mounted volume.
   f. Check the elastic instance logs for the Snapshot ID of the snapshot you just created. See Accessing an Elastic Instance for instructions on how to access the logs for your elastic instance.
   g. Update the new Snapshot ID in your Elastic Bamboo configuration, as described in Configuring an Elastic Image to use an EBS snapshot above.

Important EBS directories and files

By convention, Bamboo will attach an EBS device at `/dev/sdh`. This will be mounted at `/mnt/bamboo-ebs`. The contents of the standard structure are:

- `bin/customiseInstance.sh` - This script is run on startup of an elastic instance. We recommend that you do not customize this script, as it is overwritten when `rewarmEbsSnapshot.sh` is run.
- `bin/customise-extras.sh` - This script is run on startup of an elastic instance as the root (as opposed to being run as the Bamboo user). This script is safe to customize, as it will never be overwritten. You can customize this script to automate processes such as setting up your database, move files to custom locations on the instance, etc.
- `profile-extras.sh` - This script gets appended to the profile that is run under the Bamboo user (as opposed to being run as the root). It is useful for setting up environment variables.
- `bamboo-agent/bamboo-agent.cfg.xml` - This configuration file modifies the build working directory to point to build working directory on the EBS volume.
- `bamboo-agent/build-dir` - This is the build working directory.
- `maven/build.properties` - This properties file is copied to `/home/bamboo` on startup of an elastic instance. It points the Maven 1 default repository to `/mnt/bamboo-ebs/maven/.maven`
- `maven/.m2/settings.xml` - This configuration files is copied to `/home/bamboo/.m2` on startup of an elastic instance. It points the Maven 2 default repository to `/mnt/bamboo-ebs/maven/.m2/repository`.
- `tmp-extras` - The contents of this directory is copied to `/tmp` on startup of an elastic instance.

---

1. Start a single elastic instance via Bamboo. See Starting an elastic instance for instructions.
2. Access your elastic instance via SSH (see Accessing an elastic instance for instructions).
3.

```
sudo su -
```

    a. In this case, the '-' or the '-l' or the '-login' parameters is required, otherwise some of the scripts may fail.
4. Follow the steps below to create an EBS volume and attach it to the elastic instance (*steps a & b*), upload content to the EBS volume (*step c & d*), and generate the snapshot (*step e & f*):
   All the scripts mentioned below are available in /opt/bamboo-elastic-agent/bin on Bamboo stock images. You can also download them from here (choose the latest version).
    a. Run `createInitialVolume.sh` <*volume size*> — This script creates a EBS volume (where <volume size> is the size of the volume), attaches the volume and mounts it on the elastic instance. For example, `createInitialVolume.sh 100` will create a 100GB EBS volume and attach and mount it on the elastic instance.

Populating your EBS volume

This page is intended to complement the instructions for Configuring elastic instances to use the EBS. It lists different methods of for populating your EBS volume, depending on the data you wish to have available in your snapshot.

On this page:

Related pages:

- Configuring elastic instances to use the EBS

Uploading Maven 2 repository data

You can upload **Maven 2 repository data** to your EBS volume, so that it does not have to be downloaded every time an elastic agent (running on an instance which uses the EBS volume) is started.

To populate your EBS snapshot with your Maven repository data, we recommend that you upload it via SCP (see **step 5c** of the 'Creating your first EBS snapshot' section in Configuring elastic instances to use the EBS). In most cases, you will have a modified `settings.xml` file if you are using Maven 2. This means that you will need to upload this file and Maven repository data to your EBS volume, rather than populating your volume by running a build.

Uploading Ant repository data

You can upload **Ant repository data** to your EBS volume, so that it does not have to be downloaded every time an elastic agent (running on an instance which uses the EBS volume) is started.

To populate your EBS snapshot with your Ant repository data, we recommend that you run a build on an elastic agent with a blank EBS volume attached to the elastic instance (see **step 2** of the 'Updating your EBS snapshot' section in Configuring elastic instances to use the EBS). This is a faster and more reliable method of populating your volume, if you are using Ant.

Setting up PostgreSQL for elastic agents

You can upload scripts to your EBS volume so that the elastic agent started on any elastic instances which use this EBS volume, will have **PostgreSQL** automatically installed.
These elastic instances must be started from an elastic image which is associated with an EBS snapshot derived from this EBS volume.

To set up the automatic installation of PostgreSQL on your EBS volume for elastic agents, you will need to create the following script:

setupPostgreSQL.sh

```
#!/bin/sh
yum install -y postgresql-server
service postgresql initdb
cat > /var/lib/pgsql/data/pg_hba.conf << EOF
local all all trust
host all all 127.0.0.1/32 trust
EOF
/etc/init.d/postgresql start
```

This script uses the package management tools provided by Fedora to install and configure PostgreSQL on the agent when its started.

1. Uses **yum** to install the PostgreSQL server packages. Details on the yum tool can be found in the Fedora Software Management Guide.
2. Initializes the PostgreSQL server environment by creating the database directories and default config files.
3. Creates a new pg_hba.conf file which trusts all local connections and all connections coming from localhost.
4. Starts PostgreSQL.

You then need to update the `customise-extras.sh` file on your EBS volume (see Important EBS Directories and Files) to invoke this script.

Finally, you need to add a custom capability (e.g. `postgres=true`) to the elastic agents with PostgreSQL installed. You can do this by updating the elastic image configuration that the agents inherit their capabilities from. Read Configuring elastic agent capabilities for detailed instructions.

Setting up Selenium on elastic agents

You can upload scripts to your EBS volume so that the elastic agent started on any elastic instances which use the EBS volume, will be able to run **Selenium** tests.
These elastic instances must be started from an elastic image which is associated with an EBS snapshot derived from this EBS volume.

To set up elastic agents to support Selenium test, you will need to create the following script:

setupSelenium.sh

```
#!/bin/sh

centosMajorVersion=5
centosVersion=${centosMajorVersion}

cat >/etc/yum.repos.d/centos-$centosVersion.repo <<EOF
[centos-base]
name=CentOS - Base
mirrorlist=http://mirrorlist.centos.org/?release=${centosVersion}&arch=\$basearch&repo=os
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-${centosMajorVersion}
enabled=0

[centos-update]
name=CentOS - Updates
mirrorlist=http://mirrorlist.centos.org/?release=${centosVersion}&arch=\$basearch&repo=updates
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-${centosMajorVersion}
enabled=0

EOF

yum -y --enablerepo=centos-base install firefox

yum -y install xorg-x11-server-Xvfb xterm xorg-x11-server-utils xorg-x11-fonts-ISO8859-1-75dpi xorg-x11-
fonts-Type1

/usr/bin/killall Xvfb

#Start virtual screen
Xvfb :100 -ac -screen 0 1024x768x24 &
```

This script uses the package management tools provided by Fedora to install Mozilla's Firefox and enough of X to get a VNC (Virtual Network Computing) server running.

1. Uses **yum** to install the following packages. Details on the yum tool can be found in the Fedora Software Management Guide.
    - `vnc-server` — the vnc server used by the selenium test server.
    - `xorg-x11-server-Xvfb xterm xorg-x11-server-utils twm xorg-x11-fonts-` — these packages cover the xorg dependencies to get Firefox to run.
2. The script then copies some prepared VNC authentication files into the bamboo home directory and sets their permissions accordingly. These files are:
    - `vncpasswd` — this is the password file used by the VNC server (copied to `/home/bamboo/.vnc/passwd`)
    - `vncxstartup` — this is the script executed by the VNC server when a connection is made (copied to `/home/bamboo/.vnc/xstartup`)
3. The last step of this script is to manually install Firefox into `/opt/firefox` *(we manually install Firefox because the package that would be installed by the Fedora 8 package management appears to be outdated).*
    - The tar is extracted to the appropriate directory
    - The `.bashrc` file is customized to include the Firefox directory when searching for libraries. This is so Firefox will be able to find its libraries.

You then need to update the `customise-extras.sh` file on your EBS volume (see Important EBS Directories and Files) to invoke this script.

Finally, you need to add a custom capability (e.g. `selenium=true`) to the elastic agents with PostgreSQL installed. You can do this by updating the elastic image configuration that the agents inherit their capabilities from. Read Configuring elastic agent capabilities for detailed instructions.

Managing Elastic Bamboo

The following pages and the related sub-pages contain information on managing your elastic image, instances and agents.

- Managing your elastic images — please see this page and the related sub-pages for detailed information about Elastic Bamboo images in Bamboo. This includes instructions on how to view and customize the capabilities of your Elastic Bamboo images.
- Managing your elastic instances — please see this page and the related sub-pages for detailed information about Elastic Bamboo instances in Bamboo. This includes instructions on how to view, start, stop and access an elastic instance.
- Managing your elastic agents — please see this page and the related sub-pages for detailed information about Elastic Bamboo remote agent instances in Bamboo. This includes instructions on how to view and disable an elastic instance.

Managing your elastic images

An *elastic image* is an Amazon Machine Image (AMI) that is stored in one of Amazon data centers for use with the Elastic Bamboo feature. An elastic image is used to create elastic instances, which in turn create elastic agents. Conceptually, an elastic image is equivalent to an operating system running on a computer's boot hard drive and elastic instances would be the software that runs on this operation system.

Each elastic image registered with the Amazon Web Services (AWS) has its own unique identifier, known as an **AMI ID**.

You can associate multiple elastic images with a Bamboo server. One default shared image is maintained by Atlassian in AWS, and is available to all Elastic Bamboo users.

You can also create your own custom elastic images.

If you haven't provided your AWS details in Bamboo, you must set them before you can work with elastic instances. For more information, see Configuring Elastic Bamboo.

- To view an elastic image, including the image properties, capabilities and the jobs that an image can build, see Viewing an elastic image.
- To associate an elastic image with your Bamboo installation, see Managing your elastic image configurations.
- To customize the capabilities of an elastic image, see Configuring elastic agent capabilities.
- To create your own custom elastic image, see Creating a custom elastic image.

Viewing an elastic image

An elastic image is similar to an agent, so the Image page closely resembles the Agent page. A number of functions available for agents are also available for images.

- **Viewing an elastic image's capabilities** — your image has capabilities, similar to how agents have capabilities. Read more about viewing an agent's capabilities.
- **Viewing the jobs that an image can build** — you can also view the jobs that an image is capable of building (using the elastic agent created from the image), in a similar way to how you view the jobs that an agent is capable of building. Read more about viewing the jobs that an agent can build and determining which agents can build which jobs.

Related pages:

- Managing your elastic images

To view an image:

1. From the top navigation bar select  > **Elastic Bamboo** > **Image configurations**.
2. Select the name, or **View**, for the image that you want to view.

   Name
   The name of the image.

   AMI ID
   The Amazon Machine Image identifier that uniquely identifies the image.

   EBS Snapshot ID
   The ID of the EBS Snapshot that you have associated with this image. See Configuring elastic instances to use the EBS and Managing your elastic image configurations for more information on how to use EBS with Elastic Bamboo.

   Instance type
   The instance type of new instances started from this image. Read more about Amazon instance types.

   Availability zone preference
   New instances started from this image will be run in the Amazon availability zone named here.

   Active instances
   The number of currently active instances that were started from this image.

Managing your elastic image configurations

An *elastic image* is an Amazon Machine Image (AMI) that is stored in one of Amazon data centers for use with the Elastic Bamboo feature. An elastic image is used to create elastic instances, which in turn create elastic agents. Conceptually, an elastic image is equivalent to an operating system running on a computer's boot hard drive and elastic instances would be the software that runs on this operation system.

Each elastic image registered with the Amazon Web Services (AWS) has its own unique identifier, known as an **AMI ID**.

You can associate multiple elastic images with a Bamboo server. One default shared image is maintained by Atlassian in AWS, and is available to all Elastic Bamboo users.

You can also create your own custom elastic images.

On this page:

Related pages:

- Managing your elastic images

Associating custom elastic images with Bamboo

Associating a custom elastic image with your Bamboo installation allows you to start elastic instances with capabilities that are different from those inherited from the default image. For example, you may wish to associate a Ubuntu operating system-based elastic image with your Bamboo installation, so that you can run Ubuntu-related tests on the instances started from that image.

Once you have associated a custom elastic image with Bamboo, the settings for your elastic image are stored as an elastic image configuration.

To associate a custom image with Bamboo:

1. From the top navigation bar select  > **Elastic Bamboo** > **Image configurations**.

2. In the panel under Create elastic image configuration enter the details of your custom elastic image:

   Name
   The name of your custom elastic image. If you created your own custom image, you should have named it in step 6 of the Creating a custom elastic image instructions. You can also view the image name via the AWS console.

   Description
   A description for your image, which is used in Bamboo only.

   AMI ID
   The AMI ID obtained as an output from step 6 of the Creating a custom elastic image instructions. You can also view the AMI IDs of elastic images via the AWS console.

   Automatically attach an Amazon Elastic Block Store (EBS) volume to new elastic instances
   Select this option if you want the elastic instances started from this image to use the EBS. Read more about Configuring elastic instances to use the EBS.
   EBS Snapshot ID — Specify the EBS Snapshot ID of the EBS volume that you wish to attach to new instances.

   Instance type
   The instance type for new instances started from this image. Amazon offers a number of instance types that provide different computing capacity. Read more about Amazon EC2 instance types.

   Virtual Private Cloud Subnet
   The Subnet of the Virtual Private Cloud where your Elastic Bamboo agent will start up. Select multiple subnets from the list to enable Bamboo to automatically switch between Availability Zones when starting agents. This reduces the chance of a build failing because of a lack of available resources in a particular zone. For more about VPC, see the Amazon VPC FAQ.

   Availability zone
   The availability zone used to start your new instances from this image in (e.g. if you wish to use Elastic Bamboo with reserved instances). We recommend that you select "Default (chosen by EC2)" to allow Amazon to select the best zone for your instance. Read more about Amazon EC2 availability zones.

   Product
   The EC2 product name.

Creating elastic images with custom agent capabilities

You can customize the agent capabilities of an elastic image that is already associated with Bamboo. The initial process is similar to that of associating a custom elastic image with Bamboo (above). Here, however, you use the AMI ID of an image already associated with Bamboo — most commonly the default image.

To create an elastic image with customized agent capabilities:

1. From the top navigation bar select  > **Elastic Bamboo** > **Image configurations**.
2. Select the name, or **View**, for the image that you want to view.
3. Enter the details of your custom elastic image. See the section above for details.
4. You now have a new elastic image configuration based on an existing elastic image. Follow the procedure on Configuring elastic agent capabilities to customize this elastic image's agent capabilities.

Creating a custom elastic image

Atlassian doesn't provide support for customized images. Bamboo provides flexibility to use customized machine images, but it's impossible for us to support all individual configurations.

Use Bamboo stock images as the base for all image customizations to ensure a minimal level of consistency of your Elastic Bamboo setup.

An *elastic image* is an Amazon Machine Image (AMI) that is stored in one of Amazon data centers for use with the Elastic Bamboo feature. An elastic image is used to create elastic instances, which in turn create elastic agents. Conceptually, an elastic image is equivalent to an operating system running on a computer's boot hard drive and elastic instances would be the software that runs on this operation system.

Each elastic image registered with the Amazon Web Services (AWS) has its own unique identifier, known as an **AMI ID**.

You can associate multiple elastic images with a Bamboo server. One default shared image is maintained by Atlassian in AWS, and is available to all Elastic Bamboo users.

At a high level, the process for creating a custom elastic image consists of taking one of the existing Amazon Machine Images (AMIs) available on Amazon EC2, starting an instance of the AMI, customizing the instance and then creating an image from the customized instance. This image can then be used as an elastic image in your Bamboo installation.

Instead of creating a custom image (Linux/UNIX only) consider:

- Using the "Instance setup script" feature to run commands (executed as the root user) before the agent is started. This field is available for every image from the Administration > Image Configurations page. Select the **Edit** link under Operations for the image you want to use.
- Customizing an existing Bamboo image by using Amazon's Elastic Block Store (EBS), as described in Configuring elastic instances to use the EBS.

The options above are much simpler than creating a new custom image. If you are having problems, please don't hesitate to contact us for further help.

Before you begin:

- This is *not a trivial procedure* and chances are you don't need it.
- Note that Atlassian *does not support custom elastic images*. Consider customizing the elastic agents started from your stock images instead.
- A number of the EC2 commands in the steps below can be completed using the AWS console rather than command line tools (e.g. registering an image). You should use the method you feel most comfortable with.

On this page:

1. Requirements

First ensure that you have set up the following:

- **Amazon Web Services (AWS) account with EC2** — if you are already using Elastic Bamboo, you should already have an AWS account with EC2 set up. If not, please read Getting started with Elastic Bamboo.
- **Amazon EC2 API Tools** — you must install the EC2 API tools on your local machine, otherwise you will not be able to start and access your AMI instance. Note: you need Java Runtime Environment to run these tools. You can install the EC2 API tools by executing the following commands:

```
wget http://s3.amazonaws.com/ec2-downloads/ec2-api-tools.zip
unzip ec2-api-tools.zip
```

- **Environment Variables**— you must set up the following environment variables on your local machine before creating a custom elastic image:
  - `EC2_HOME` — set this to the path to the installed EC2 API Tools
  - `EC2_CERT` — set this to the path to the certificate assigned to EC2 account
  - `EC2_PRIVATE_KEY` — set this to the path to the private key assigned to your AWS account
- **Registered Key Pair**— you need a registered EC2 key pair, which consists of a private key file and certificate file, to use the EC2 API tools with your AMI instance. If you have previously generated and registered an EC2 key pair (e.g. to use the EC2 API tools), you can re-use it. If you need to generate a new key pair, you can use the following command to do so:

```
ec2-add-keypair <key_pair_name>
```

The content of the private key will be displayed in the command-line output on your console. Save this content in a file , starting with the line:

"--BEGIN RSA PRIVATE KEY--"

and ending with the line:

"--END RSA PRIVATE KEY--"

This private key file will be used to access your AMI instance. Set up the appropriate permissions on the private key file by executing the following command:

```
chmod 600 <private_key_file>
```

2. Selecting an existing AMI

We strongly recommend that you select an existing Linux/UNIX AMI to customize, rather than starting with a blank AMI. When choosing an AMI, decide whether you want to launch **32-bit** or **64-bit** instances from your custom elastic image and select an existing AMI matching your choice.

We recommend the following existing Linux/UNIX AMIs for customization (in order of preference):

Source	Description	AMI list
Atlassian	One of the Stock images provided by Atlassian. It is an Amazon image, for either Linux or Windows, updated and prepared for Bamboo, i.e. you will not have to install any Bamboo prerequisites.	Available on your Bamboo instance under Administration/Image Configurations
Amazon	CentOS-based image provided by Amazon. It does not have any Bamboo prerequisites installed. Typically, you will be better off using the Atlassian AMI.	Amazon's site
Canonical (Ubuntu)	An official Ubuntu image provided by Canonical (the company behind the Ubuntu Linux project). It does not have any Bamboo prerequisites installed.	Canonical's site

Atlassian's AMIs (and hence, their IDs) may change with each release of Bamboo, including both major and minor releases. To quickly access Atlassian's AMI IDs for your currently-running version of Bamboo, open that Bamboo site in a web browser and access its 'Image Configurations' page (see Managing your Elastic Image Configurations for more information). The AMI IDs of Atlassian's AMIs are labeled with "(stock image)".

If you want to find out the AMI IDs for a version of Bamboo you don't have running or you're starting an image from scratch and you need the image baseline:

1. Open https://packages.atlassian.com/maven/repository/public/com/atlassian/bamboo/atlassian-bamboo/
2. On the resulting directory page, select the link that represents the version of Bamboo you are currently running. For example, if you are running Bamboo 5.9.7, select on the 5.9.7 link. Another directory page opens, listing a .pom and some additional checksum files.
   ⚠️ Do not select a version number link that contains 'mX', 'rcX' or 'betaX' (where 'X' is a number), since these relate to publicly available developmental releases of Bamboo.
3. Open the atlassian-bamboo-x.x.x.pom file (where `x.x.x`) is your version of Bamboo). The image version/baseline is stored in the `elastic-image.version` tag. For example, this value is "4.2" for Bamboo version 5.9.4.
4. Open https://packages.atlassian.com/maven/repository/public/com/atlassian/bamboo/atlassian-bamboo-elastic-image/
5. Select the image version/baseline you found in the `elastic-image.version` tag.

   On the resulting directory page, the file with `ami` extension contains all stock image AMI IDs.

3. Starting an instance

After you have selected an existing AMI to customize, the next step is to start an instance of the AMI.

3.1 Starting an instance of Atlassian's default AMI

If you chose to customize Atlassian's default AMI, you will have to start the instance from the admin section of Bamboo. See Starting an elastic instance.

Note that Atlassian's default AMI cannot be started using the command line ec2 tools. This is because, on start up, the Bamboo agent on Atlassian's AMI checks to see if it was started from a Bamboo server, and immediately shuts itself down if it was not.

Once started, see Accessing an elastic instance for details on how to access the running instance.

3.2 Starting an instance from the command line

Use the `ec2-run-instances` command to start your instance, as follows:

```
ec2-run-instances <image_name> -k <key_pair_name>
```

where `<image_name>` is the name of the AMI selected in the previous step and `<key_pair_name>` is the name of the registered key pair generated in '1. Requirements'. The public certificate of this key will be injected into your instance.

For example, if you wanted to start an instance of image `ami-e55bbd8c` using key pair `my-keypair`, you would run the following command:

```
ec2-run-instances ami-e55bbd8c -k my-keypair
```

This command would produce the following command-line output:

```
INSTANCE        i-25b86743    ami-e55bbd8c     running    my-keypair
```

`i-25b86743` is the name of the new instance in the above example. You should note down the name of your new instance, as you will need that to access your instance in the next step.

Don't forget to shut down unused instances

Please note that once you start an instance, you will be billed by Amazon for instance uptime. If you decide to abandon the setup of a custom elastic image after this step, please ensure that you shut down your instance via the AWS console.

3.3 Starting an instance from Bamboo

You can also start a fresh, uncustomized image from Bamboo and begin customization. The drawback of this approach is that you have only 40 minutes before Bamboo shuts down your instance. The advantage is that you can customize the agent in a single step (as opposed to having to customize/create image/start from Bamboo/save image again).

4. Accessing your instance

If you started the instance from Bamboo, see Accessing an elastic instance for details on how to access the running instance.

Once your instance is running, you will need to obtain the address of the instance so you can access it. To do this, use the following command:

```
ec2-describe-instances <instance_name>
```

For example, if you wanted to find the address of instance `i-25b86743`, you would enter:

```
ec2-describe-instances i-25b86743
```

This command would produce the following command-line output similar to this:

```
RESERVATION        r-790f7210        121852097033        default
INSTANCE        i-25b86743        ami-e55bbd8c         ec2-174-129-94-241.compute-1.amazonaws.com
domU-12-31-39-04-38-87.compute-1.internal        running        elasticbamboo        0               m1.
small
2009-06-24T12:36:20+0000        us-east-1c        aki-a71cf9ce        ari-a51cf9cc
monitoring-disabled
```

The address of the instance in the above example is `ec2-174-129-94-241.compute-1.amazonaws.com`

You can then use this address to access the instance via SSH. See Accessing an elastic instance for instructions. If you are using the example command text from that document, you will need to adjust it as follows:

- replace `/opt/bamboo/home/xml-data/configuration/elasticbamboo.pk` in the example command text with the private key file you generated in '1. Requirements'.
- replace `ec2-68-111-185-197.compute-1.amazonaws.com` in the example command text with the address of your instance.

5. Customizing your instance

Now that you have a running instance, customization steps heavily depend on the operating system you're using. We've prepared separate pages with Linux-specific instructions and Windows-specific instructions.

6. Creating an image of your customized instance

The process of creating a new image varies depending whether you based your image on an instance-store or EBS-root image. You can check your image type via AWS console or using ec2-describe-images.

Creating an image from EBS-root instances

See here for instructions: Amazon Tutorial

Creating an image from instance-store (S3) instances

The final step is to create an image from your customized instance. To do this, you will require the following information:

- Amazon Account Number
- Access Key ID
- Secret Access Key
- Amazon S3 bucket name that will be used to store image (if you don't have access to Amazon S3, you can sign up on this page.)

1. **Transfer Amazon private key file and certificate to your instance**
   Transfer the key files to your instance by running these commands on your **local machine**:

   ```
   scp -i <private_key_file> $EC2_PRIVATE_KEY root@<instance_address>:/mnt
   scp -i <private_key_file> $EC2_CERT root@<instance_address>:/mnt
   ```

   where <private_key_file> is the private key file from your local machine created in step 'Registered Key Pair' of 1. Requirements and the <instance_address> is the address of your instance from '4. Accessing your Instance'.
2. **Set up EC2_HOME and JAVA_HOME environment variables**
   Set up these environment variables by running the following commands on your instance:

   ```
   export EC2_HOME=<location of your EC2 tools installation>
   export EC2_PRIVATE_KEY=/mnt/<ec2_private_key_file>
   export EC2_CERT=/mnt/<ec2_certificate_file>
   export JAVA_HOME=<path to JRE used to start the agent>
   ```

3. You can create an image of your customized instance by using the `ec2-bundle-vol` command, as follows:

   ```
   ec2-bundle-vol -c $EC2_CERT -k $EC2_PRIVATE_KEY -u <amazon_account_number> -p
   <elastic_image_name> --batch --debug
   ```

   where <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1')
4. Once the image is created, you need to upload it to Amazon S3 by running the command below:

   ```
   ec2-upload-bundle -b <s3_bucket_name> -m /tmp/<elastic_image_name>.manifest.xml -a
   <access_key_id> -s <secret_access_key> --retry --debug
   ```

where <s3_bucket_name>, <access_key_id> and <secret_access_key> are the Amazon S3 bucket name, Access Key ID and Secret Access Key described previously, and <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1').

You will then need to register your image with Amazon EC2 by using the `ec2-register` command:

```
ec2-register <s3_bucket_name>/<elastic_image_name>.manifest.xml
```

where <s3_bucket_name> is the Amazon S3 bucket name described previously and <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1'). The output of this command will show the AMI ID of your custom image.

7. Next steps

Now that you have created a custom elastic image, there are **two more steps** that you will need to complete before you can use it.

First, you will need to **associate your custom elastic image with your Bamboo installation** by creating an Elastic Image Configuration. Please note the AMI ID of your new custom image and read Managing your Elastic Image Configurations for further instructions.

Secondly, you will need to **configure the capabilities of the elastic agents** that will run on instances started from your image. This is done by adding the appropriate builder, JDK, Perforce and custom capabilities to your elastic image configuration, so that it reflects what your custom elastic image actually can do. For example, if you have created a custom elastic image with JDK 1.6 and Maven 2 installed, you will need to add capabilities for JDK 1.6 and Maven 2 to the elastic image configuration. Read Configuring Elastic Agent Capabilities for further instructions.

8. Need more help?

If you need **more help**, there are a number of resources that you can take advantage of:

- AWS Support Center — if you are having problems with any of your Amazon services, not specifically related to Bamboo, you can obtain basic support from the AWS Support Center. Note, you will need to sign up for Premium Support to get access to web/phone support.
- AWS Resource Center — the AWS Resource Center has links to online documentation, code samples and tools for AWS services.
- Bamboo Developer Forums — please feel free to discuss any useful tips or issues regarding this process in the Bamboo Developer Forums.

Creating a custom elastic image - Linux

Atlassian doesn't provide support for customized images. Bamboo provides flexibility to use customized machine images, but it's impossible for us to support all individual configurations.

Use Bamboo stock images as the base for all image customizations to ensure a minimal level of consistency of your Elastic Bamboo setup.

5. Customizing your instance

Customizing your instance is the most complicated part of creating a custom elastic image. You need to install the packages that are prerequisites for Bamboo onto your instance (if you didn't choose the Elastic Bamboo Stock images as your base AMI), add your customizations, deploy Bamboo onto your instance and set up an EC2 environment on your instance.

5.1 Installing Bamboo prerequisite packages

If you selected Atlassian's AMI as your base AMI in '2. Selecting an Existing AMI', you can skip this step and go to '5.2 Adding Customizations' as this image has been pre-configured for Bamboo. If you have selected a different AMI, you will need to install the following packages onto your instance using the commands shown below:

Amazon EC2 API tools

```
wget http://s3.amazonaws.com/ec2-downloads/ec2-api-tools.zip
unzip ec2-api-tools.zip
mv ec2-api-tools-* /opt/ec2-api-tools
```

Note: if your distribution already contains ec2-api-tools package, you can install it instead.

Java JRE

You need to install a JRE (or JDK) on your instance to be able to run the agent. The preferred way of doing this is to install a package that came with your distribution. For a list of supported JREs, see supported platforms.

5.2 Adding user customizations to your instance

Adding your own customizations is quite a simple process, once you have made it this far.

To add user customizations to your instance:

1. Log into your elastic instance (as previously described in '4. Accessing your Instance').
2. Once you have logged into your elastic instance, you can treat it as a standalone machine and install anything you want. For example, if you want to install Tomcat on an Ubuntu instance you would run '`sudo apt-get install tomcat6`', configure it, ensure that your startup scripts are in place, etc, just as you would when installing Tomcat on a standalone machine.
   ⚠*Please note however, you cannot customize the operating system of a running instance. If you want to create an instance with a customized operating system (e.g. Ubuntu), you will need to select an AMI with that operating system installed (as previously described in '2. Selecting an Existing AMI').*
3. Everything that you install will be saved in snapshot image created at the end of these instructions (see '6. Creating an Image of your Customized Instance'). Any instances started from this image will have all of your user customizations automatically installed.

5.3 Deploying Bamboo onto your instance

Once you have installed the Bamboo pre-requisites on you instance and added your customizations, you can deploy Bamboo elastic bootstrap files onto your instance.

5.3.1 Creating Bamboo user

First, you need to create a '`bamboo`' user on your instance by running the following command:

```
useradd -m bamboo
```

5.3.2 Downloading agent installer to the instance

Then, install Bamboo Agent binaries as described below. In this case we're using image version 2.2, you should use the latest version available at https://maven.atlassian.com/content/repositories/atlassian-public/com/atlassian/bamboo/atlassian-bamboo-elastic-image/.

```
imageVer=2.2
wget https://maven.atlassian.com/content/repositories/atlassian-public/com/atlassian/bamboo/atlassian-bamboo-elastic-image/${imageVer}/atlassian-bamboo-elastic-image-${imageVer}.zip
sudo mkdir -p /opt/bamboo-elastic-agent
sudo unzip -o atlassian-bamboo-elastic-image-${imageVer}.zip -d /opt/bamboo-elastic-agent
sudo chown -R bamboo /opt/bamboo-elastic-agent
sudo chmod -R u+r+w /opt/bamboo-elastic-agent
```

5.4 Instance configuration

At this stage, you should have a customized instance with Bamboo deployed onto it. The last step in creating a customized instance is to set up an EC2 environment on your instance. Carry out the following steps to set this up:

1. Run the following command on your instance to set permissions on the bamboo user directory:

   ```
   chown -R bamboo:bamboo /home/bamboo/
   ```

2. **Configure path variables**
   Create a file **profile.sh** in your instance's `/mnt` directory. This file contains the default Elastic Bamboo path configuration settings, as seen below:

   ```
   export JAVA_HOME=<path to JRE used to start the agent>
   export EC2_HOME=<location of your EC2 tools installation>
   export EC2_PRIVATE_KEY=/root/pk.pem
   export EC2_CERT=/root/cert.pem
   export PATH=/opt/bamboo-elastic-agent/bin:$EC2_HOME/bin:$JAVA_HOME/bin:$M2_HOME/bin:$MAVEN_HOME/bin:$ANT_HOME/bin:$PATH
   ```

   If all of the tools on this page were installed in recommended locations, no changes are required. Otherwise, you can update the file as required.
   Once **profile.sh** is correctly customized for your instance, you need to copy it to the `/etc/profile.d` directory by running the following command on your instance in the `/mnt` directory:

   ```
   mv profile.sh /etc/profile.d/bamboo.sh
   ```

3. Configure the Bamboo agent to start up automatically with the instance by sourcing Bamboo's `/opt/bamboo-elastic-agent/etc/rc.local` file in SysVinit or systemd.

Alternatively, you can use the AWS cloud-init startup scripts. To do that, create a script like the one below and add it to your cloud-init configuration in `/var/lib/cloud/scripts/per-instance/bamboo-agent-start`:

```
#!/bin/sh

# Bamboo agent startup
. /opt/bamboo-elastic-agent/etc/rc.local

exit 0
```

For more information, go to Run commands on your Linux instance at launch in the Amazon Elastic Compute Cloud documentation.

4. Ensure the bamboo user can write to and execute from `/tmp`. Validate if `noexec` option is NOT set on `/tmp` mountpoint.
   If your security policy enforces you to use `noexec` as a mount option in `/tmp`, you can alternatively specify a different `java.io.tmpdir` during your agent startup:

```
$ cat /opt/bamboo-elastic-agent/bin/runStartupScripts.sh
#!/bin/sh

echo Running startup scripts...

bambooAgentBin=$(cd -P -- $(dirname $0) && pwd)

#
# custom -Djava.io.tmp to bypass /tmp noexec option
#
java -Djava.io.tmpdir=/home/bamboo -cp $bambooAgentBin/*installer*.jar com.atlassian.bamboo.agent.
elastic.startup.RunStartupScripts 2>&1 | tee -a /tmp/BambooStartupLog.log
```

5. **Final settings and cleanup**
   Finally, create a Bamboo welcome screen and clean up keys on your instance by running the following command:

```
cp /opt/bamboo-elastic-agent/etc/motd /etc/motd
echo bamboo-<x.x.x>  >> /etc/motd
rm -f /root/firstlogin /etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_dsa_key.pub
/etc/ssh/ssh_host_key /etc/ssh/ssh_host_key.pub /etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_rsa_key.pub /root/.ssh/authorized_keys
touch /root/firstrun
```

   where <x.x.x> is the Bamboo version you are running (e.g. 4.1.2).

6. **Now, follow the instructions from section "Creating an image of your Customized Instance" to create an AMI.**
7. **Start the image from Bamboo. The agent should come up and download all necessary data to the EC2 instance.**
8. Run /opt/bamboo-elastic-agent/bin/prepareInstanceForSaving.sh.
9. **Now, follow the instructions from section "Creating an image of your Customized Instance" to create an AMI.** That's it, the newly created AMI contains everything you need to start Bamboo Agents.

Note: if you started your instance from Bamboo right at the start, instead of steps 5 & 6, you can just run:

```
su -c /opt/bamboo-elastic-agent/bin/bamboo-elastic-agent - bamboo
```

Creating a custom elastic image - Windows

Atlassian doesn't provide support for customized images. Bamboo provides flexibility to use customized machine images, but it's impossible for us to support all individual configurations.

Use Bamboo stock images as the base for all image customizations to ensure a minimal level of consistency of your Elastic Bamboo setup.

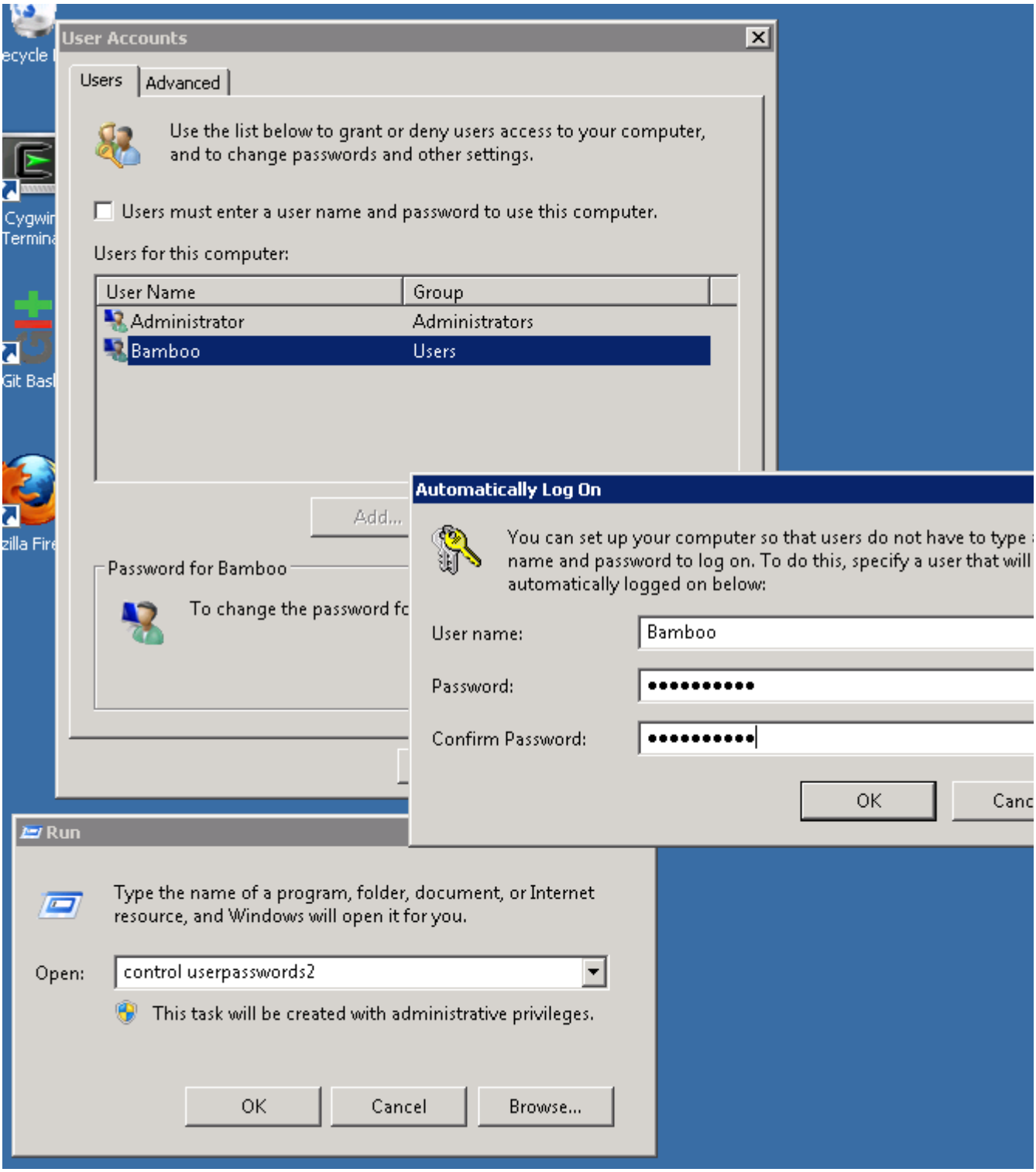
To perform the tasks listed below, log in to your instance with an Administrator account using Remote Desktop Client.

Setting up the user account

Create the user account that will be used by the Bamboo agent. The account name is up to you, I will use Bamboo in the examples below. All builds running on your machine will use this account. It can be a regular user (i.e. it does not need to be a Power User or Administrator, unless your builds require it). Set up a password for that user. The default user on a Windows image has a user name of Bamboo with a password of Atlassian1.

Important: by default, a newly created user should be denied remote login rights (which is as we want it to be). To be on the safe side, please make sure that you indeed **cannot** log in using that user's credentials (unless you change the credentials to non-default ones).

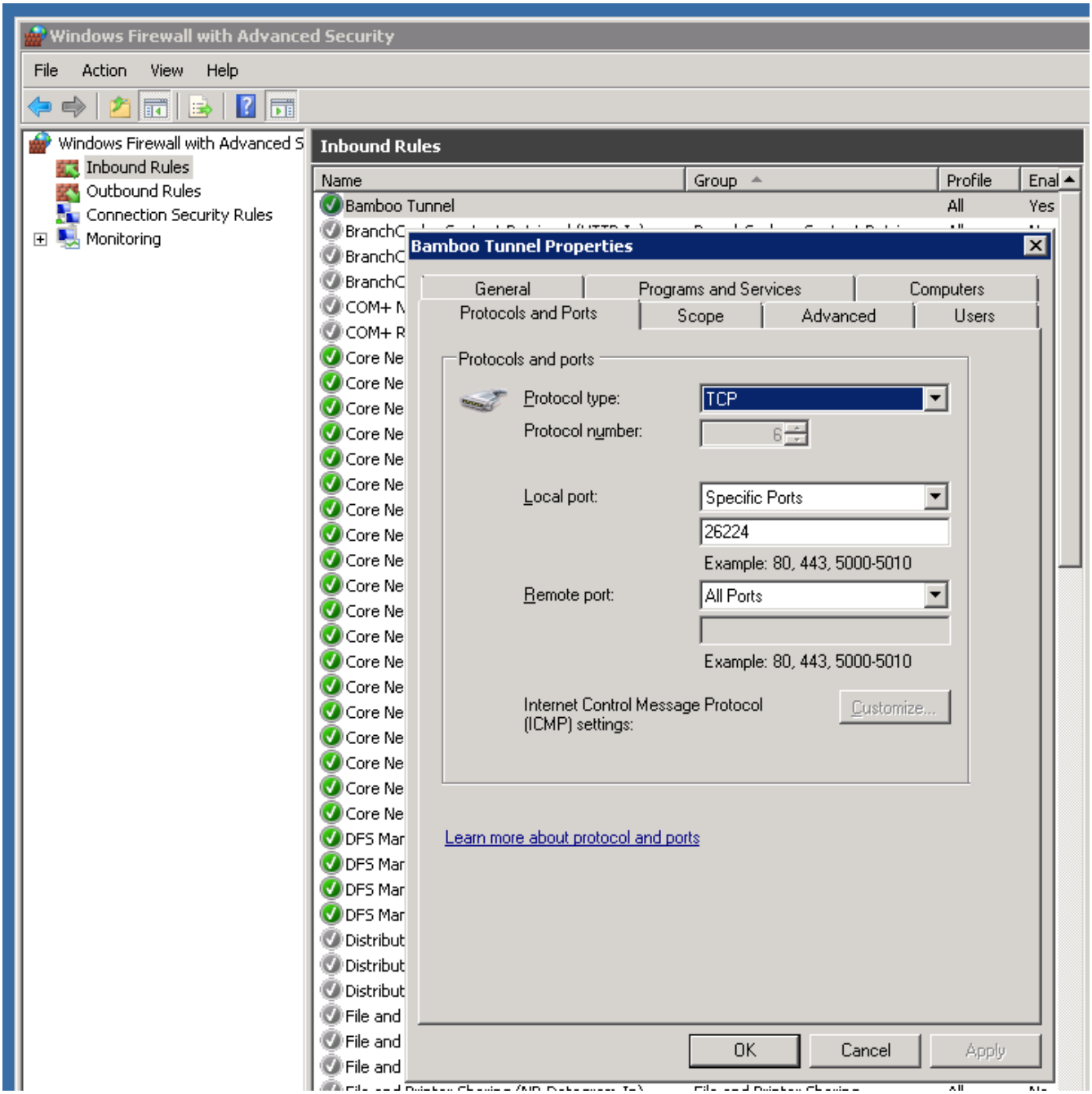
You'll need to set up autologin for your Bamboo account (don’t worry, this will not let remote users in). To do this, run `control userpasswords2` and uncheck **User must enter a user name and password to enter this computer**:



Setting up the firewall

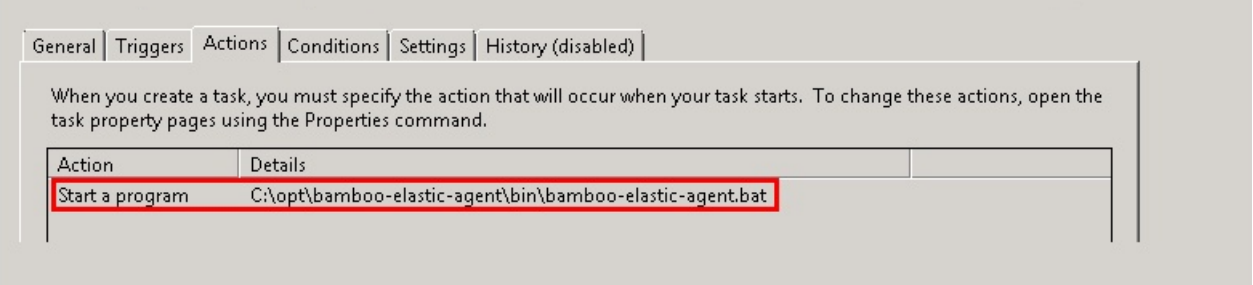
Reconfigure the Windows firewall to accept TCP connections on port 26224. No other inbound connections are necessary for Bamboo.

⚠ You don't need to worry about changing the EC2 security group setting for this port. Bamboo will set it up automatically:

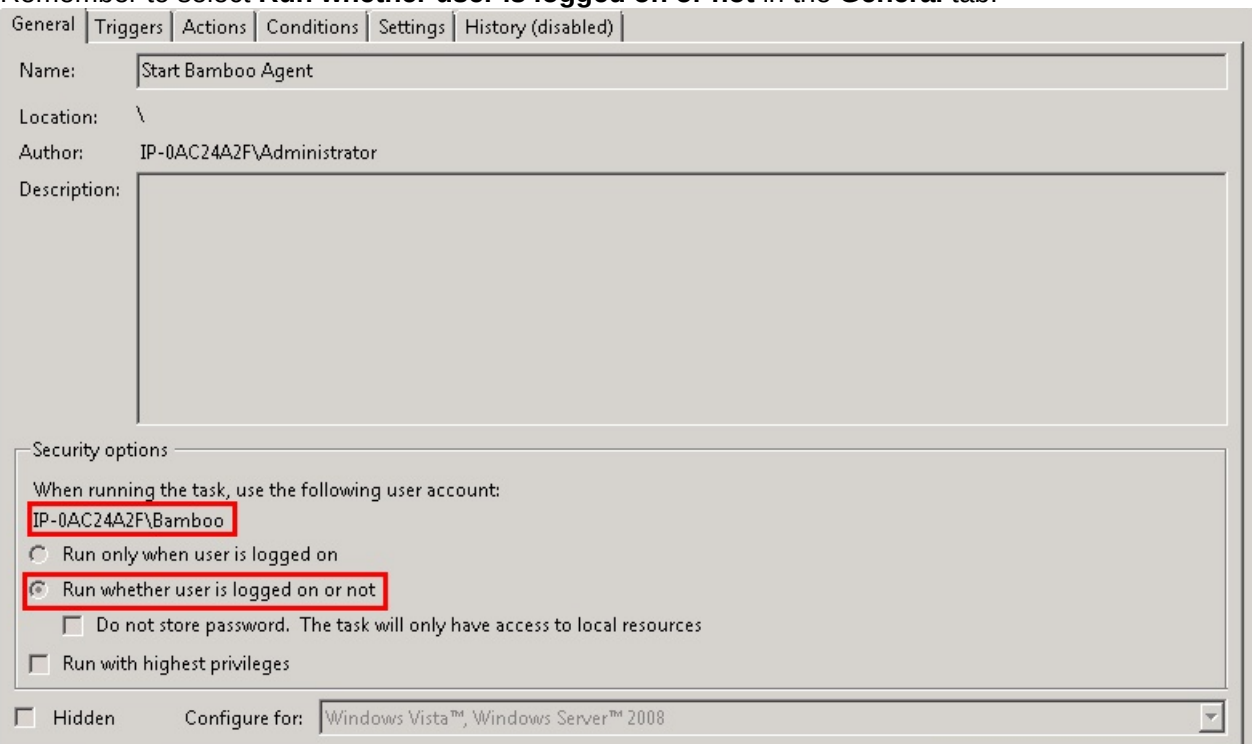


Installing the required software

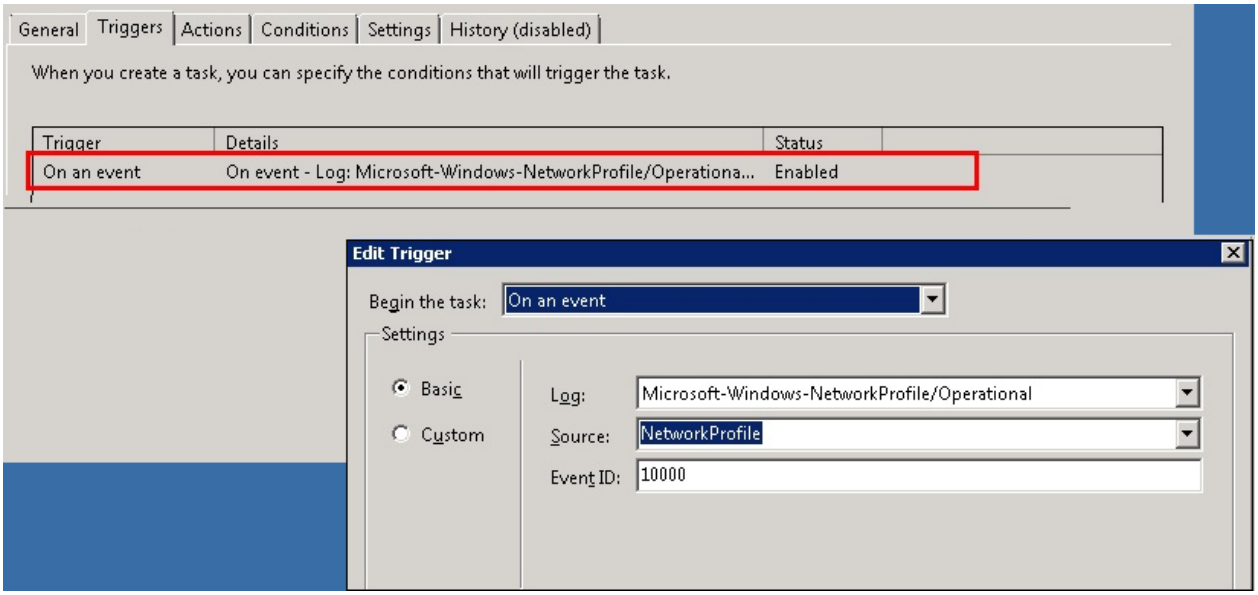
1. Install a supported Oracle Java version. See Supported platforms
2. Download the latest version of agent installer zip from this this location (at the time this guide was written, the latest version was this). Unpack it to a desired location, we suggest using `C:\opt\bamboo-elastic-agent` to match stock images distributed with Bamboo.
3. A batch file should launch with your Windows instance startup. In order to do this, use the Windows Task Scheduler (**Start** > **Administrative Tools** > **Task Scheduler**), and set up a new Action task of “Start a program" with the <PATH TO YOUR BATCH FILE> as the Details:



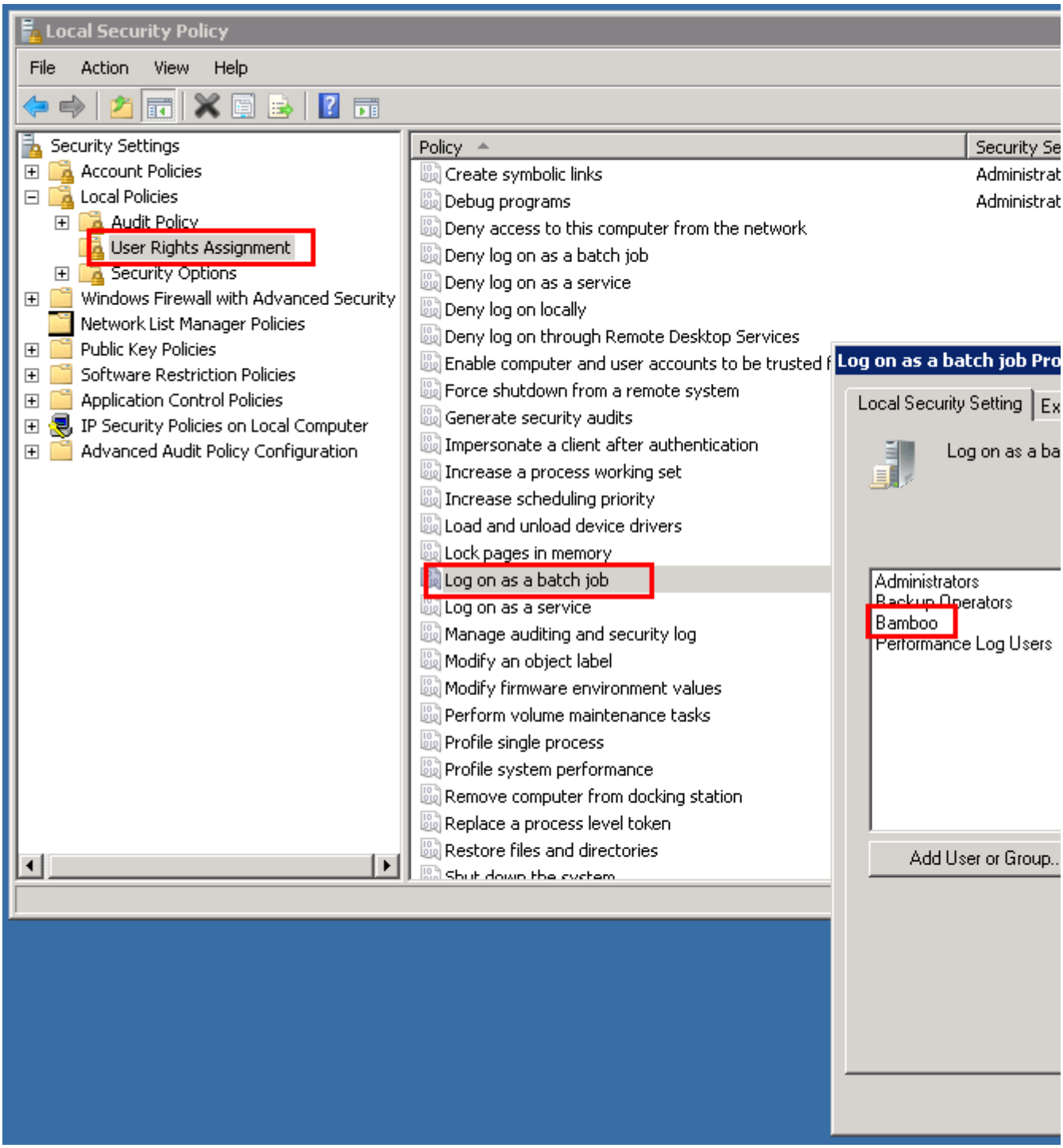
Remember to select **Run whether user is logged on or not** in the **General** tab:



And appropriately define the Trigger task so that the agent starts up only after the network connection is up and running:



The task manager will warn you that the account needs to be able to log in as a batch job. Make sure the **Log on as batch job** policy is set for the user. This policy is accessible by opening the **Control Panel** > **Administrative Tools** > **Local Security Policy**. In the Local Security Policy window, select **Local Policies** > **User Rights Assignment** > **Log on as a batch job**:



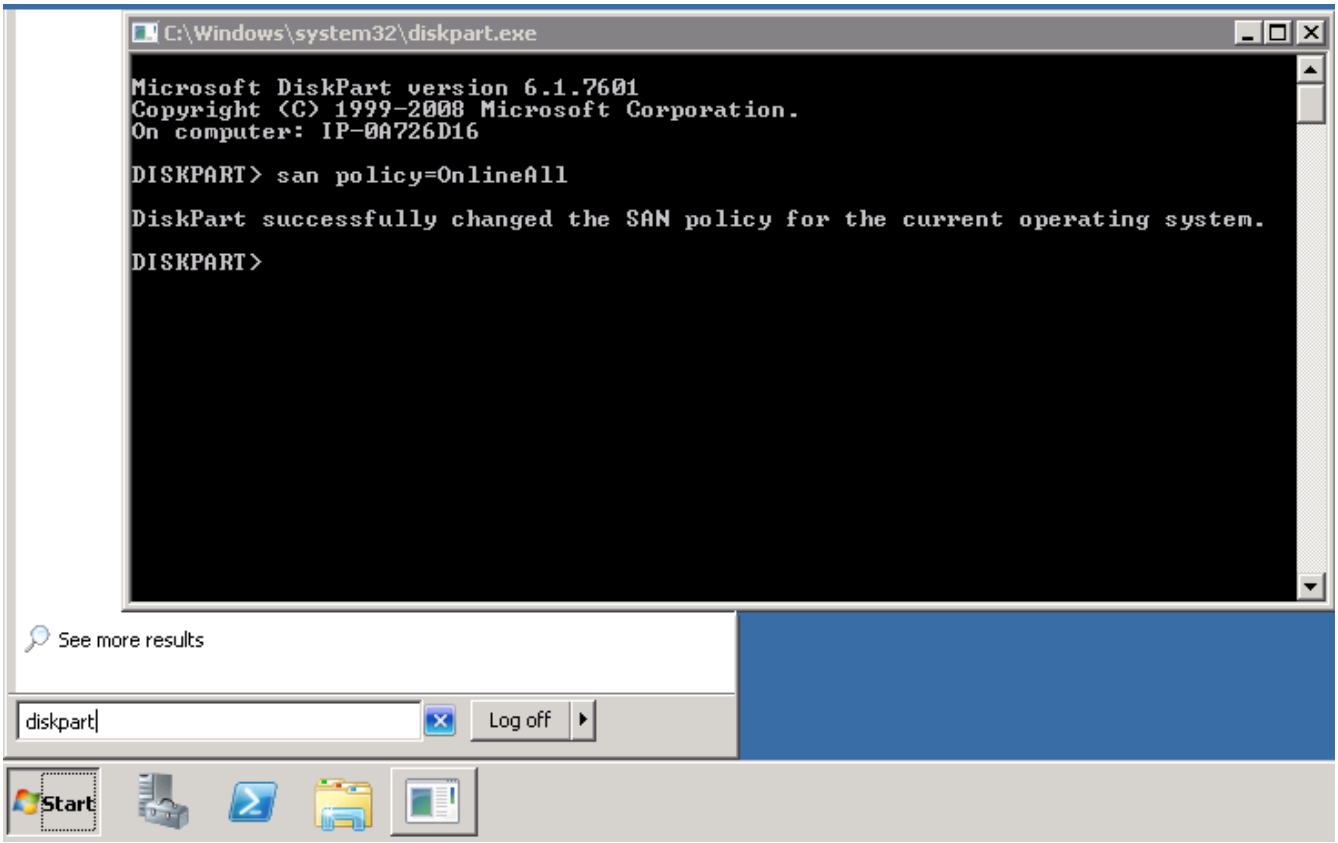
Enabling EBS usage on the instance

Starting with Bamboo 5, you'll be able to use custom Elastic Block Storage with your Windows instances. To do that, you need to change the SAN policy on your instance, otherwise the disks will be attached in Offline state with status text *the disk is offline because of policy set by an administrator*.

You can change the SAN policy using the Diskpart utility. Run `Diskpart`, type

```
san policy=OnlineAll
```

and press Enter. You can then quit `Diskpart;` the new policy will now be active.



Testing

The easiest way to check if everything is set up correctly is to run the task you've defined using Windows Task Scheduler (**Start** > **Administrative Tools** > **Task Scheduler**). Right-click on the task and select **Run**. Always test the script using the Task Scheduler; if you run the script manually, you'll use Administrator account, which is not what we want.

Look for the `%USERPROFILE%/bamboo-elastic-agent.out` file. If it exists and contains an error message stating that agent was **not** run within an EC2 instance started by Bamboo Server, you've successfully completed the customization.

Run `c:\opt\bamboo-elastic-agent\bin\prepareInstanceForSaving.bat`

Bundle your instance. Make a note of the AMI id of the new image.

Start your image from Bamboo

> ⚠ If you fail to complete the following steps within ~40 minutes, Bamboo will shut down your instance, so remember to save your work if you're running out of time (i.e. create an interim image).

In Bamboo, define an image configuration for the image you've just created, and start it from Bamboo. If everything went well, the agent will start together with the instance. It will perform the following steps:

- Update/*create /opt/bamboo-elastic-agent* directory structure by creating additional directories. If they appeared, Java is working correctly on that machine and the connection to S3 is working.
- Start the agent, which will create the Bamboo Agent Home directory and populate it with data pulled from the Bamboo server.

If everything went well, you should see the agent appear in the Bamboo instance list. Congratulations, you have a working Bamboo agent.

Because the agent has just synchronized itself with the Bamboo server (because it has downloaded all the jars exactly matching what you have on your server), as an extra step, you may want to save that state to speed up future instance startup and reduce bandwidth usage.

To do that, run

```
c:\opt\bamboo-elastic-agent\bin\prepareInstanceForSaving.bat
```

save the image, define a new image confguration, kill the instance, and try running it from Bamboo.

Upgrading the agent for your custom elastic image

The instructions below are valid if you were using Bamboo 3.4 or newer. If you're upgrading from an earlier version, you should first reinstall the agent installer (see Creating a custom elastic image).

If you customized your instance according to Creating a custom elastic image, your instance will keep itself updated across Bamboo. The synchronization process takes a while and the time required increases as your image gets older. If you notice slow startup, you may want to refresh your image. You can either customize the instance from scratch, as when you created your customized image, or update just the agent data, which is much faster.

Related pages:

- Managing your elastic image configurations
- Creating a custom elastic image

To refresh your agent data:

1. Start your instance from Bamboo.
2. Log into your instance.
3. Run /opt/bamboo-elastic-agent/bin/prepareInstanceForSaving.sh.
4. Create an Image of your Customized Instance.

   The final step is to create an image from your customized instance. To do this, you will require the following information:

   - Amazon Account Number
   - Access Key ID
   - Secret Access Key
   - Amazon S3 bucket name that will be used to store image (if you don't have access to Amazon S3, you can sign up on this page.)

   You can create an image of your customized instance by using the `ec2-bundle-vol` command, as follows:

   ```
   /usr/local/bin/ec2-bundle-vol -c $EC2_CERT -k $EC2_PRIVATE_KEY -u <amazon_account_number> -p
   <elastic_image_name> --batch --debug
   ```

   where <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1')

   Once the image is created, you need to upload it to Amazon S3 by running the command below:

   ```
   /usr/local/bin/ec2-upload-bundle -b <s3_bucket_name> -m /tmp/<elastic_image_name>.manifest.xml -a
   <access_key_id> -s <secret_access_key> --retry --debug
   ```

   where <s3_bucket_name>, <access_key_id> and <secret_access_key> are the Amazon S3 bucket name, Access Key ID and Secret Access Key described previously, and <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1')

   You will then need to register your image with Amazon EC2 by using the `ec2-register` command:

   ```
   $EC2_HOME/bin/ec2-register <s3_bucket_name>/<elastic_image_name>.manifest.xml
   ```

   where <s3_bucket_name> is the Amazon S3 bucket name described previously and <elastic_image_name> is the name that you want to assign to your custom image (e.g. 'CustomImage1')

   The output of this command will show the AMI ID of your custom image.

5. Associate the new Custom Image with Bamboo.
   Finally, you will need to associate your custom elastic image with your Bamboo installation by creating an Elastic Image Configuration. Please note the AMI ID of your new custom image and read Managing your elastic image configurations for further instructions.

Updating elastic images for Bamboo upgrades

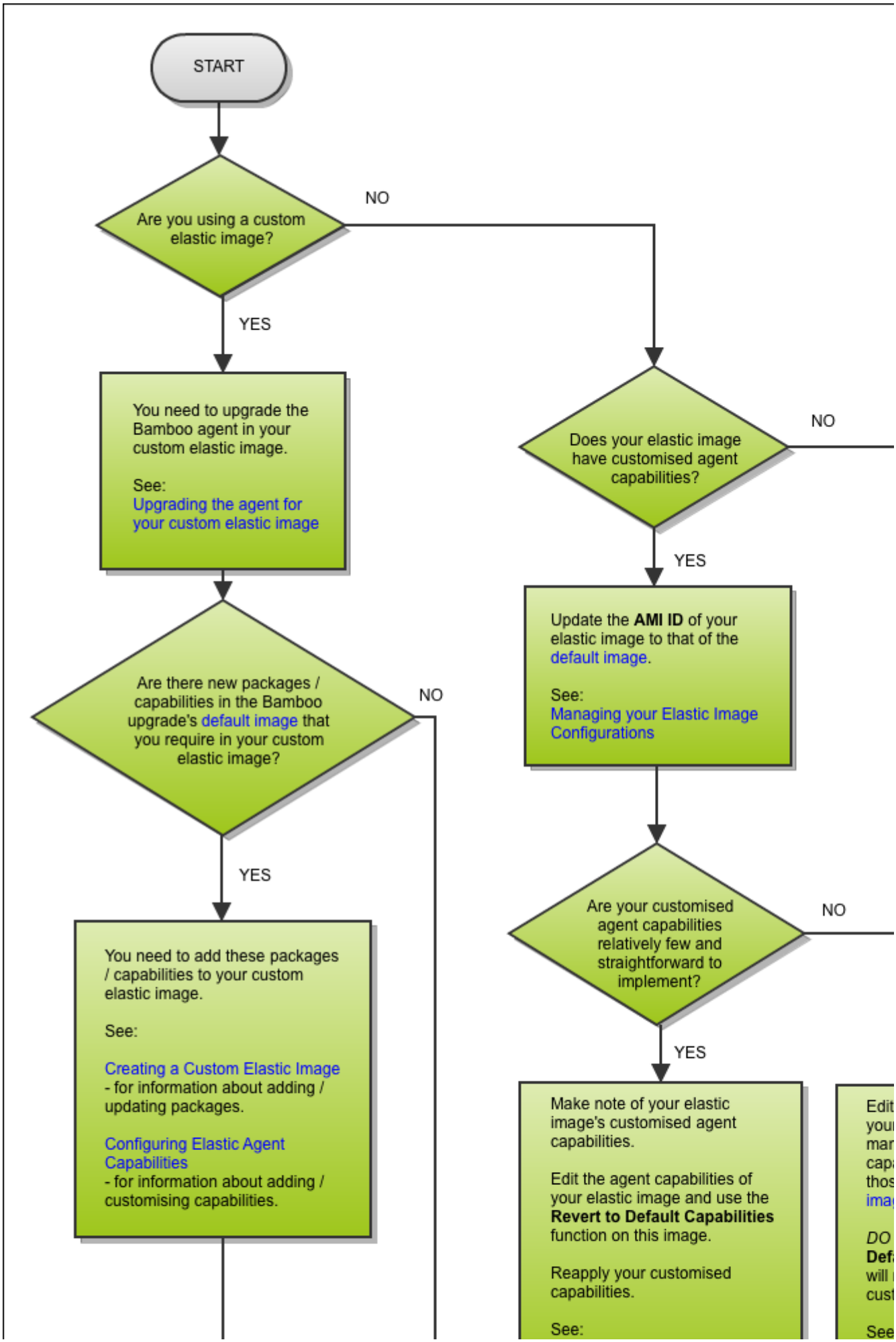
Various updates to default packages and capabilities are made to the default image with each major release of Bamboo.

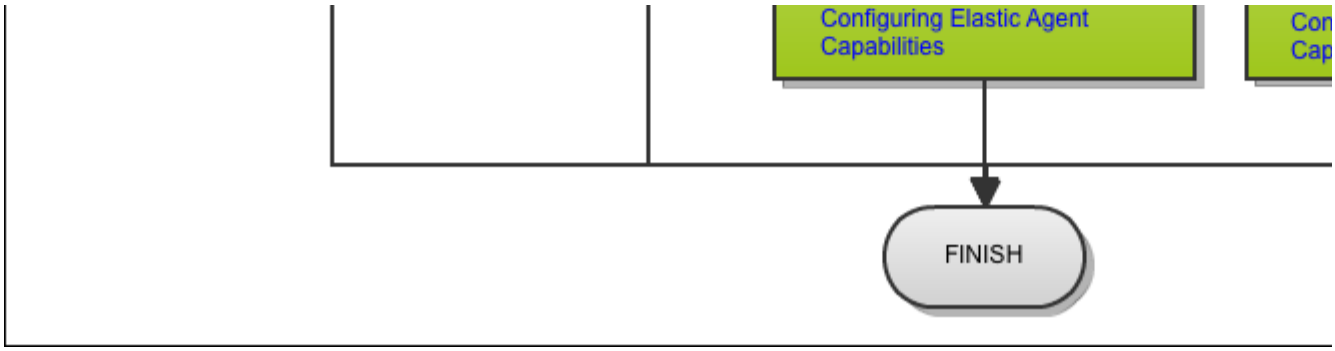
Therefore, if you are using either a:

- custom elastic image, or
- an elastic image with customized agent capabilities

then to ensure this elastic image acquires these package/capability updates, use the flow chart below to update your elastic image.

⚠️ Use this flowchart ***only after Bamboo has been upgraded***. For each elastic image you wish to update, follow this flow chart from the start.





Elastic Images with Customized Capabilities:

This flow chart assumes that all elastic images with customized agent capabilities are based off the default image.Please check the default image page to identify the packages and related capabilities available in the default image for .

Viewing the list of Bamboo stock images

Bamboo provides a collection of Amazon Machine Images (AMIs) that are ready for use or further customization. Each AMI is identified by its unique ID. For more information about the parameters of the stock images, see Stock images.

⚠ Atlassian AMIs and their IDs may change with each minor or major Bamboo release.

Viewing a list of AMI IDs available for a release

To generate a list of AMI IDs for a Bamboo version:

1. Save the following script as a .sh file:

```
if [ $# -eq 0 ]; then
  echo "Usage: `basename $0` [7.2.4] (Your Bamboo version)"
  exit 0
fi

BAMBOO_VERSION=$1
echo For Bamboo version: $BAMBOO_VERSION
ELASTIC_VERSION="$(curl -L -v --silent https://packages.atlassian.com/content/groups/public/com
/atlassian/bamboo/atlassian-bamboo/$BAMBOO_VERSION/atlassian-bamboo-$BAMBOO_VERSION.pom 2>&1 |
grep \<elastic-image.version\> | sed -e 's/<[^>]*>//g' -e 's/^[[:space:]]*//' -e 's/[[:space:]]*$
//')"

echo "Elastic bamboo version is $ELASTIC_VERSION"

curl -L -v --silent https://packages.atlassian.com/content/groups/public/com/atlassian/bamboo
/atlassian-bamboo-elastic-image/$ELASTIC_VERSION/atlassian-bamboo-elastic-image-$ELASTIC_VERSION.
ami 2>&1 | grep image.
echo "REMEMBER: Use the image from the appropriate region!"
```

2. In the terminal, go to the directory where you saved the file and run it with the following command:

```
./<name-of-your-file>.sh <your-bamboo-version>
```

Example

```
./amis.sh 7.2.4
```

✓ If you get the Permission denied error, you can modify the permissions of the .sh file with the following:

```
chmod +x <name-of-the-file>.sh
```

Results

Select the example to see the full script output.

For Bamboo 7.2.4, the results can be the following:

```
For Bamboo version: 7.2.4

Elastic bamboo version is 6.23

> GET /content/groups/public/com/atlassian/bamboo/atlassian-bamboo-elastic-image/6.23/atlassian-bamboo-
elastic-image-6.23.ami HTTP/2
< x-artifactory-filename: atlassian-bamboo-elastic-image-6.23.ami
< content-disposition: attachment; filename="atlassian-bamboo-elastic-image-6.23.ami"; filename*=UTF-
8''atlassian-bamboo-elastic-image-6.23.ami
image.SOUTH_AMERICA_1.EBS.x86_64.linux.HVM.Ubuntu=ami-03c3d6d69adbc5946
image.EU_CENTRAL_1.EBS.x86_64.windows.HVM.Windows=ami-00a1727df3812c2d6
image.EU_WEST_3.EBS.x86_64.linux.HVM.Ubuntu=ami-0d65fbb37cfe81d00
image.SOUTH_AMERICA_1.EBS.x86_64.linux.PV.Ubuntu=ami-0471e1dc9b77376a9
image.US_WEST_1.EBS.x86_64.linux.PV.Ubuntu=ami-005e7ea0adfbae676
image.SOUTH_AMERICA_1.EBS.x86_64.windows.HVM.Windows=ami-01bf6de9c569da505
image.US_EAST_1.EBS.x86_64.linux.PV.Ubuntu=ami-0791f1d390f64157f
image.US_EAST_1.S3.x86_64.linux.PV.Amazon\ Linux=ami-0c89a893d848a8593
image.EU_WEST_3.EBS.x86_64.windows.HVM.Windows=ami-053a58739cc428775
image.ASIA_PACIFIC_SE_2.EBS.x86_64.linux.HVM.Ubuntu=ami-074ba1801d88c78f0
image.ASIA_PACIFIC_NE_1.EBS.x86_64.linux.PV.Ubuntu=ami-05d051f31169095da
image.AP_NE_2.EBS.x86_64.windows.HVM.Windows=ami-06d2a504ca4b877fa
image.US_EAST_2.EBS.x86_64.windows.HVM.Windows=ami-06e93f4ef8d13d47f
image.US_EAST_1.EBS.x86_64.linux.HVM.Ubuntu=ami-02d922f88e5fbbb00
image.US_WEST_2.EBS.x86_64.windows.HVM.Windows=ami-0db289e52d607daa3
image.AP_S_1.EBS.x86_64.windows.HVM.Windows=ami-09cc63730fd18cdbe
image.AP_S_1.EBS.x86_64.linux.HVM.Ubuntu=ami-014757183c34279b9
image.EU_WEST_2.EBS.x86_64.linux.HVM.Ubuntu=ami-0f8a4ff5beddc0420
image.EU_WEST_2.EBS.x86_64.windows.HVM.Windows=ami-06d2ec543b878d597
image.US_WEST_2.EBS.x86_64.linux.PV.Ubuntu=ami-053c2a69af96c9f47
image.US_EAST_1.EBS.x86_64.windows.HVM.Windows=ami-0922a872659cf255b
image.US_WEST_1.EBS.x86_64.windows.HVM.Windows=ami-04f9fdc823ead49dc
image.ASIA_PACIFIC_SE_1.EBS.x86_64.linux.HVM.Ubuntu=ami-07d01de5aebebe238
image.CA_CENTRAL_1.EBS.x86_64.windows.HVM.Windows=ami-0a55890a368f2fb68
image.US_WEST_2.EBS.x86_64.linux.HVM.Ubuntu=ami-06cb734294afa43b8
image.EU_WEST_1.EBS.x86_64.windows.HVM.Windows=ami-07f0b01b03bf29e1b
image.ASIA_PACIFIC_SE_1.EBS.x86_64.linux.PV.Ubuntu=ami-056c397105ecfb8eb
image.ASIA_PACIFIC_NE_1.EBS.x86_64.linux.HVM.Ubuntu=ami-03f2a6b757cd9b939
image.ASIA_PACIFIC_SE_2.EBS.x86_64.windows.HVM.Windows=ami-02d6b280b0c4e260a
image.AP_NE_2.EBS.x86_64.linux.HVM.Ubuntu=ami-007d28bdc6b2b8dcf
image.EU_WEST_1.EBS.x86_64.linux.HVM.Ubuntu=ami-093b0aa989ca066bc
image.US_EAST_1.EBS.x86_64.linux.PV.Amazon\ Linux=ami-023f0ce24fb18bf8f
image.EU_CENTRAL_1.EBS.x86_64.linux.HVM.Ubuntu=ami-055e7272fc013fc02
image.ASIA_PACIFIC_NE_1.EBS.x86_64.windows.HVM.Windows=ami-02cf4e13b931415c8
image.CA_CENTRAL_1.EBS.x86_64.linux.HVM.Ubuntu=ami-008dfdcaef7743d95
image.US_WEST_1.EBS.x86_64.linux.HVM.Ubuntu=ami-0aa05539ff48549ad
image.ASIA_PACIFIC_SE_1.EBS.x86_64.windows.HVM.Windows=ami-01564eb2283f136b9
image.US_EAST_2.EBS.x86_64.linux.HVM.Ubuntu=ami-00ede1dff70710545
image.EU_CENTRAL_1.EBS.x86_64.linux.PV.Ubuntu=ami-053a73d3547e266e8
image.ASIA_PACIFIC_SE_2.EBS.x86_64.linux.PV.Ubuntu=ami-0d6f81a7502c1366b
image.EU_WEST_1.EBS.x86_64.linux.PV.Ubuntu=ami-02563a74df85e9d7b
REMEMBER: Use the image from the appropriate region!
```

Troubleshooting

If the script above doesn't work, you can also find the list of Bamboo stock images in the following way:

1. In the following URL:

   ```
   https://packages.atlassian.com/content/groups/public/com/atlassian/bamboo/atlassian-bamboo
   /$BAMBOO_VERSION/atlassian-bamboo-$BAMBOO_VERSION.pom
   ```

   change *$BAMBOO_VERSION* to the Bamboo version number for which you want to list the AMIs.

   Example

   For Bamboo 7.2.4:

```
https://packages.atlassian.com/content/groups/public/com/atlassian/bamboo/atlassian-bamboo/7.2.4
/atlassian-bamboo-7.2.4.pom
```

2. Open the URL in a browser.
3. In the atlassian-bamboo-$BAMBOO_VERSION.pom file (where $BAMBOO_VERSION is your version of Bamboo), find the elastic image version for the release. The image version (baseline) is stored as an `elastic-image.version` property.
   Example
   For the version 7.2.4 (atlassian-bamboo-7.2.4.pom), the elastic image version was 6.23:

```
<elastic-image.version>6.23</elastic-image.version>
```

4. In the following URL:

```
https://packages.atlassian.com/content/groups/public/com/atlassian/bamboo/atlassian-bamboo-
elastic-image/$ELASTIC_VERSION/atlassian-bamboo-elastic-image-$ELASTIC_VERSION.ami
```

   change *$ELASTIC_VERSION* to the Bamboo elastic version number from Step 3.

   Example

   For Bamboo elastic version 6.23, which is the baseline for Bamboo 7.2.4:

```
https://packages.atlassian.com/content/groups/public/com/atlassian/bamboo/atlassian-bamboo-
elastic-image/6.23/atlassian-bamboo-elastic-image-6.23.ami
```

5. Open the URL in a browser.
6. The .ami file that opens contains the list of all stock AMI IDs available for the selected version of Bamboo.

   ⚠ Make sure you select the image from the correct region. For example:

   image.US_EAST_1.EBS.x86_64.linux.HVM.Ubuntu=ami-02d922f88e5fbbb00

Related topics

- Creating a custom elastic image
- Stock images

Managing your elastic instances

An *elastic instance* is a running instance of an elastic image. One elastic instance is created whenever an elastic image is started. Hence, starting one elastic image multiple times, results in the creation of multiple elastic instances. Each time an elastic instance is created, one elastic agent is created on that instance.

The following list directs you to details on managing elastic instances manually in Bamboo. However, you can configure Bamboo to automatically manage your elastic instances. Please refer to Automatic Elastic Instance Management for more information.

- **To view a running elastic instance,** see Viewing an elastic instance.
- **To access your elastic instance via a client**, see Accessing an elastic instance.
- **To start one or more elastic instances,** see Starting an elastic instance.
- **To shut down one or more elastic instances,** see Shutting down an elastic instance.
- **To configure your Elastic Bamboo settings for elastic instances,** see the Elastic Instance Settings section in the Configuring Elastic Bamboo document.

Viewing an elastic instance

An *elastic instance* is a running instance of an elastic image. One elastic instance is created whenever an elastic image is started. Hence, starting one elastic image multiple times, results in the creation of multiple elastic instances. Each time an elastic instance is created, one elastic agent is created on that instance.

Conceptually, an elastic instance can be thought of as a computer. The elastic agent's processes are run on this computer and the elastic image is the boot hard drive. Unlike computers, however, elastic instances are temporary and stateless. When an elastic instance is shut down:

- Any changes that an elastic instance makes to the boot hard drive (e.g. agent log file) will not persist
- Any customizations to the instance itself will also be lost.

The Amazon Elastic Block Store can provide persistent storage for your elastic instances.

You can also view information about your elastic instances on the AWS Management Console. Please note, we strongly recommend that you use the console for *viewing* instance information only. You may experience errors if you attempt to manage your instances outside of Bamboo.

Related pages:

- Managing your elastic instances

To view an elastic instance:

1. From the top navigation bar select  > **Elastic Bamboo** > **Instances**.
2. Select the name of the instance that you want to view, e.g. '`i-05ff716c`'.

Current status
The status of the elastic instance. Values include *pending* (instance starting up), *running* and *shutting down*.

Public DNS
The public DNS address of the elastic instance. The IP address of the elastic instance is displayed here.

Start time
The start time of the instance, based on the Amazon EC2 timezone (US Eastern Time for Elastic Bamboo). Start time is the time when you sent the request to start an instance, not the time when the instance progresses to 'Running' status. Up time of the instance (including the time taken for the instance to start up) is shown in brackets after the start time.

Elastic agent
The elastic agent process currently running on your elastic instance. Currently, Elastic Bamboo only supports one elastic agent per elastic image. Click the link to view the elastic agent. If the agent is running a job, the job's key will be shown in brackets after the elastic agent name.

Current availability zone
The availability zone that your elastic instance is running in. Read more about Amazon EC2 availability zones.
Your availability zone preference is shown in brackets after the current availability zone. For instructions on how to set the availability zone for your instances, please see Managing your elastic image configurations.

Attached volumes
The IDs of the attached EBS volumes, if you have configured your elastic instances to use EBS.

Configuration
The name of the elastic image configuration that was used to create this elastic instance. Click the name to configure the elastic image.

AMI ID
The ID of the elastic image (i.e. Amazon Machine Image) that the elastic instance was created from (as part of the elastic image configuration).

EBS Snapshot ID
The ID of the EBS snapshot that was used to create the EBS volumes attached to your instance, if you have configured your elastic instances to use EBS.

Bamboo polls the EBS volumes for an elastic instance every 60 seconds by default. If you want to change this interval, you need to modify the following system property: `bamboo.agent.elastic.ebsVolumeSupervisionIntervalInSeconds`

Instance type
The instance type of your instance.

SSH access
Please see Accessing an elastic instance for information on using this function.

Accessing logs
Please see Accessing an elastic instance for information on using this function.

Accessing an elastic instance

It's possible to connect directly to a running elastic instance to access logs or upload files. Access is available through secure shell (SSH) and file transfer is enabled through secure copy (SCP).

> You can only access already running elastic instances. To do that, you may need to configure the automatic termination of elastic instances.

On this page:

Related pages:

- Managing your elastic instances

Using SSH

To access your elastic instance using SSH:

1. Navigate to an elastic instance as described in Viewing an elastic instance.
2. Run the command listed in the SSH Access section. For example:

```
ssh -i elasticbamboo.pk <username>@ec2-68-111-185-197.compute-1.amazonaws.com
```

Where `<username>` is a correct username. Common values are "ubuntu" (for the Stock Ubuntu Elastic Image and images derived from it) and "ec2-user" (for old Stock Elastic Images and images based on Amazon Linux).

> You can also download the private by using the link in the SSH Access section of Viewing an elastic instance to access your elastic instance via SSH. You can download the SSH private key file by clicking the link provided on-screen.

> If you are experiencing permission issues when attempting to SSH into your elastic instance, you may need to log in as root or modify the permissions on your Elastic Bamboo private key file. See this FAQ for more details.

Using SCP

You can use SCP to download logs or upload files to your elastic instance.

To access your elastic instance using SCP:

1. Navigate to the an elastic instance as described in Viewing an elastic instance.
2. Run the command listed in the Accessing logs section. For example:

```
scp -i elasticbamboo.pk <username>@ec2-68-111-185-197.compute-1.amazonaws.com:/home/bamboo/bamboo-elastic-agent.out ./
```

Where <username> is a correct username. Common values are "ubuntu" (for the Stock Ubuntu Elastic Image and images derived from it) and "ec2-user" (for old Stock Elastic Images and images based on Amazon Linux).

Starting an elastic instance

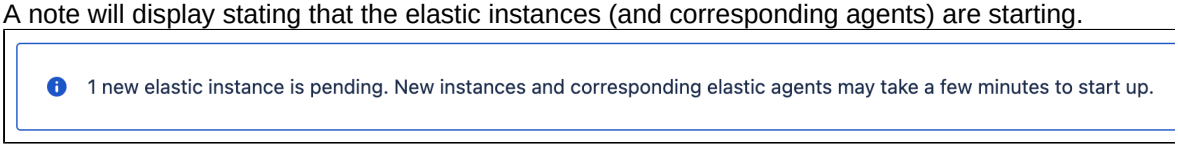
An elastic agent process runs in an elastic instance and will automatically start when an instance is started. If you want to run a Job build on an elastic agent, you can start an elastic instance for the agent to run in. The elastic agent will inherit the capabilities of the image that the instance is started from.

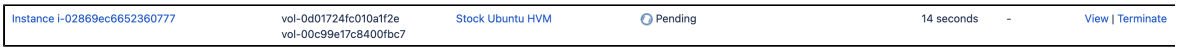
Limitations on the number of elastic instances — An elastic agent is counted as a remote agent for licensing purposes. Hence, if starting an elastic instance (and hence an elastic agent) causes you to exceed the total number of remote agents allowed under your license, you will not be able to start the instance.

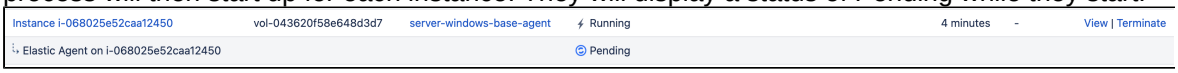
Related pages:

- Managing your elastic instances

To start an elastic instance:

1. From the top navigation bar select  > **Elastic Bamboo** > **Instances**.
2. Select **Start new elastic instances**.
    - Use **Number of instances** to specify the number of new instances you would like to start.
    - Use **Elastic image configuration name** to select the elastic image configuration that you would like your instances to use.
3. Select **Submit**. The Manage elastic instances page will be displayed, showing your new instances starting:
    a. A note will display stating that the elastic instances (and corresponding agents) are starting.
    

    b. Your elastic instances will then display with a status of Pending while they start up. This generally takes a few minutes.
    

    c. Once your elastic instances have started up, they will progress to Running status. An elastic agent process will then start up for each instance. They will display a status of Pending while they start.
    

    d. Once the elastic agents have started, they will display a status of Online.

Instance ⇢ Agent	Status	Up Time	Operations
Instance i-2204914b	Running	10 minutes	View | Shut Down
Elastic Agent on i-2204914b	Online		Disable

Notes

What if my elastic agent doesn't start? Bamboo has a set period of time that it waits for the agent to start on an elastic instance. If no response is received by the end of this time period, Bamboo will shut down the elastic instance.

You can configure this time period by modifying the following system property (default is 600):
`bamboo.agent.elastic.startupTimeoutSeconds`
Read Starting Bamboo for instructions on how to set a system property.

Scheduling your elastic instances

You can schedule the startup and shutdown of elastic instances in Bamboo. For example, you may wish to shut down all elastic instances on weekends or start up additional instances to help cope with job builds during regular busy periods.

Managing your elastic instance schedules

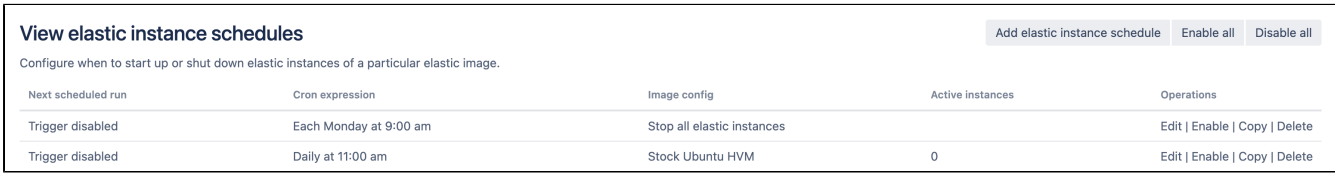
To manage your elastic instance schedules:

1. From the top navigation bar select > **Elastic Bamboo** > **Instance schedule**.
2. Do any of the following:

Task	Action
Add a new schedule	Select **Add elastic instance schedule** to create a schedule from new. Select **Copy** to use an existing schedule as a template. See the Adding a New Elastic Instance Schedule section below for further instructions.
Edit an existing schedule	Select **Edit** for an existing schedule. You can also **Delete** existing schedules.
Enable existing schedules	Select **Enable** for a particular schedule, or select **Enable all**.
Disable existing schedules	Select **Disable** for a particular schedule, or select **Disable all**.

You can also view the configuration for the elastic image that the instances will be created from, by clicking the image configuration name (e.g. Default) in the table of schedules.

Time displayed in Elastic Instance Schedules pages refers to the server time.



Adding a new elastic instance schedule

1. From the top navigation bar select > **Elastic Bamboo** > **Instance schedules**.
2. Select either **Add Elastic Instance Schedule** to create a schedule from new, or **Copy** for an existing schedule to use it as a template.

Enabled
Clear if you do not want this schedule to be enabled when you create it.

Trigger on
Select when this schedule should start:

- **Next Bamboo startup**
- **A cron schedule** — edit **Schedule** as required. For information on constructing cron expressions, see this FAQ.

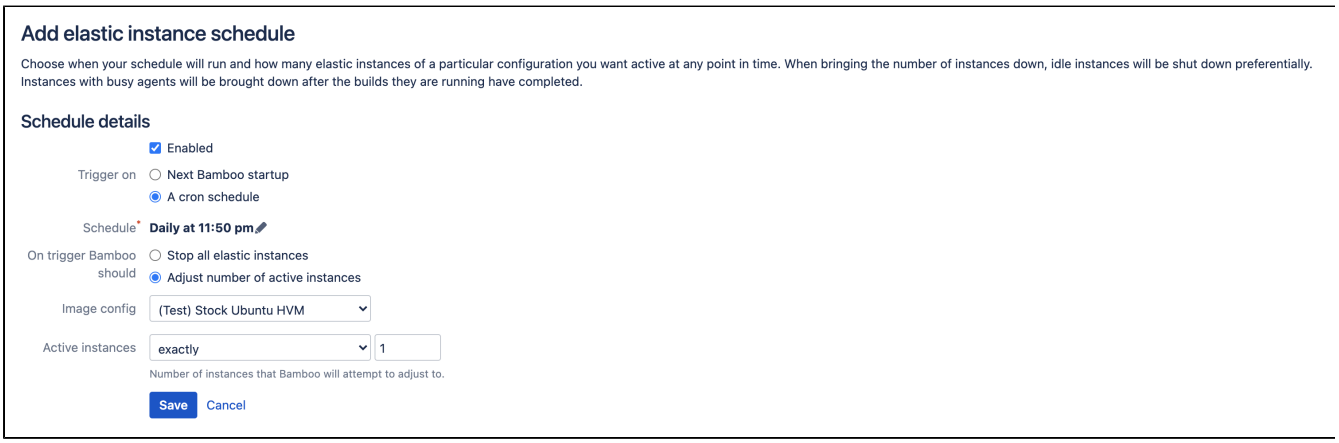
> Time displayed in Elastic Instance Schedules pages refers to the server time.

On trigger Bamboo should
Select the action Bamboo should perform:

- **Stop all elastic instances**
- **Adjust number of active instances**

Image config
Select which image the elastic instances should be started from. The elastic agents running on the instances will inherit the capabilities from the image.

Active instances
Select the logical operator and specify a value for the number of active instances.



Shutting down an elastic instance

We recommend that you shut down any elastic instances that are not being used. Amazon EC2 charge for the period of time that you have an instance running, so you can minimize your costs simply by shutting down instances with inactive agents. You should also shut down your elastic instances if you are going to restart your Bamboo server, otherwise you will orphan them from your Bamboo server.

If you have set up automated procedures using the Bamboo REST API to terminate agents (e.g. cron jobs), you can also configure Elastic Bamboo to automatically shut down instances after the agent processes terminate.

On this page:

Related pages:

- Managing your elastic instances

Shutting down an elastic instance

Before you begin:

- Please ensure that the agent on an elastic instance is not running a job build, before shutting down the instance. Any job builds running on the agent **will be abandoned** when you shut down the elastic instance.

To shut down an elastic instance:

1. From the top navigation bar select  > **Elastic Bamboo** > **Instances**.
2. Select **Terminate** for the instance that you wish to shut down (in the **Operations** column).
3. Select **Confirm**. In the Manage elastic instances screen, the elastic instance that you have shut down will show a Shutting down status for a few minutes, before it shuts down and disappears from this screen.

Shutting down all elastic instances

Before you begin:

- Please ensure that the agent on an elastic instance is not running a Job build, before shutting down the instance. Any Job builds running on the agent **will be abandoned** when you shut down the elastic instance.

To shut down all elastic instances:

1. From the top navigation bar select  > **Elastic Bamboo** > **Instances**.
2. Select **Terminate All Instances**.
3. Select **Confirm**. The Manage elastic instances screen will display again. The elastic instances will display Shutting down status for a few minutes, before they shut down and disappear from this screen.

Configuring automatic shutdown of instances after agent termination

To configure Elastic Bamboo to automatically shut down instances when agents are terminated:
Please refer to Configuring Elastic Bamboo and follow the instructions for setting the **Automatically shut down elastic instance when elastic agent process ends** option in the Elastic Bamboo global settings section.

Shutting down elastic instances using the AWS Console

We **strongly recommend** that you manage your instances using the Elastic Bamboo user interface. If your elastic instances become orphaned from your Bamboo server, you may need to shut your elastic instances down directly in the Amazon Web Services (AWS) console.

Your elastic instances can become orphaned from your Bamboo server, for example if you restart your Bamboo server without shutting down your elastic instances first.

Please refer to How do I shut down my elastic instances if I have restarted my Bamboo server for further details.

Managing your elastic agents

An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

- **To view your elastic agents,** see Viewing your elastic agents.
- **To view elastic agents that have terminated,** see Viewing your elastic agent usage history.
- **To configure your elastic agent's capabilities,** see Configuring elastic agent capabilities.
- **To disable an elastic agent,** see Disabling an elastic agent.

Viewing your elastic agents

An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

An elastic agent will always have an Idle status, (i.e. Idle or Idle (Disabled)). If you disable an elastic agent, the elastic instance will remain idle. However, if you shut down the elastic instance, then the elastic agents process is killed and will not appear in the remote agents list. Hence, an elastic agent will never have an Offline status.

Related pages:

- Managing your elastic agents

To view your elastic agents:

1. From the top navigation bar select > **Build resources** > **Agents**.

The agents for your Bamboo instance will be displayed. Any elastic agents that are running will be listed in the Remote agents section. The elastic agent name will be prefixed with Elastic agent, e.g. *Elastic Agent on i-2204914b*.

Viewing your elastic agent usage history

When you shut down an elastic instance, the agent process for that instance is killed. Consequently, the elastic agent will not display an offline status, but will be removed altogether from your available elastic agents in Bamboo.

However, information about these elastic agents is recorded in Bamboo and can be viewed on the Elastic agent history page.

Related pages:

- Managing your elastic agents

To view the history of an elastic instance that has been shut down:

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under **Elastic Bamboo**, select **Agent history**.
3. To view the usage history of the elastic agent, select the agent name, or **View** next to the agent. The Elastic agent history page (see screenshot) will show the following information:
    - Elastic instance — the elastic instance that the elastic agent ran in.
    - Last startup time — the last time that the elastic agent was started. This is based on the Bamboo server time.
    - Last shutdown time — the last time that the elastic instance was stopped. This is based on the Bamboo server time.
    - Uptime — the total time that the elastic agent was online.
    - Build History — this table lists the job builds run by the elastic agent and information about the job build, such as the status, duration, test results, etc. You can access the full results by selecting the build number.

Configuring elastic agent capabilities

An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

Note that elastic agents don't use the bamboo-capabilities.properties file. You can customize the capabilities of your elastic agents by configuring the capabilities on the relevant elastic image in Bamboo Administration.

You may want to configure the capabilities on your elastic image to force your job builds to run on particular elastic agents (e.g. running slow acceptance tests on your most powerful elastic agents). You may also need to configure the capabilities on any custom elastic images that you have created and/or associated with your Bamboo installation.

Note that adding a builder, JDK, or version control capability to the image does not install the actual builders, JDKs, or VCS modules on the image. Please take particular note of this, if you are adding capabilities to a custom image.

Related pages:

- Managing your elastic agents

To configure the capabilities on an elastic image:

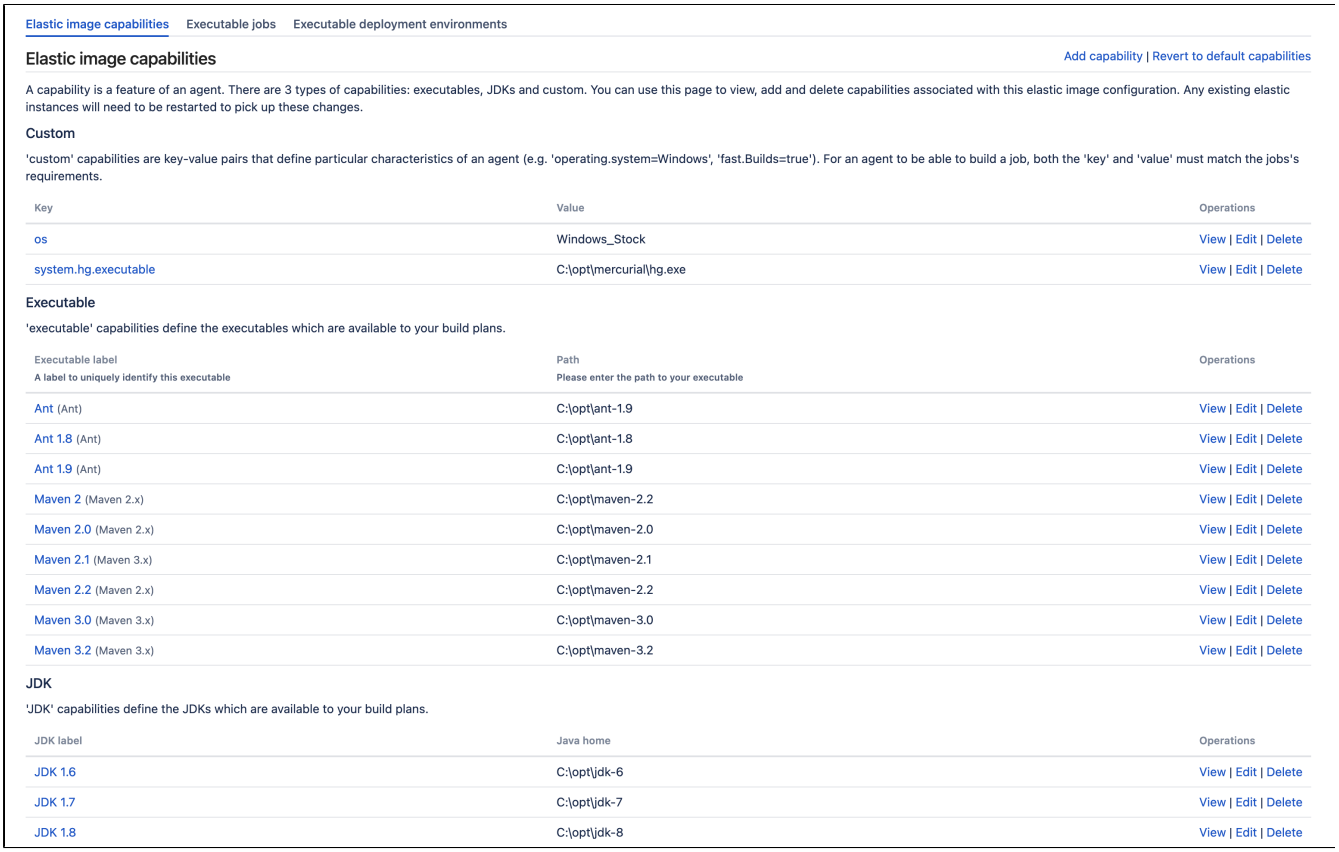
1. From the top navigation bar select  > **Elastic Bamboo** > **Image configurations**.
2. Select **Capabilities** (under Operations) for the relevant elastic image.
3. Use the Add capability panel at the end of the page to add new capabilities to the image. Please see the following pages for further information:
    - Defining a new executable capability
    - Defining a new JDK capability
    - Defining a new version control capability
    - Defining a new custom capability
    - Defining a new Docker capability

You can also edit, rename, or delete a capability from an elastic image. Please see the following pages for further information:

- Configuring capabilities
- Renaming a capability

You can also view the agents and elastic image configurations with a particular capability and the jobs with the related requirement by selecting **View** for the capability.

Any changes that you have made to elastic image capabilities will only be reflected in new agents started after the changes were made. You will need to restart any existing agents, if you want them to pick up your changes.



Elastic image capabilities | Executable jobs | Executable deployment environments

Elastic image capabilities

Add capability | Revert to default capabilities

A capability is a feature of an agent. There are 3 types of capabilities: executables, JDKs and custom. You can use this page to view, add and delete capabilities associated with this elastic image configuration. Any existing elastic instances will need to be restarted to pick up these changes.

Custom

'custom' capabilities are key-value pairs that define particular characteristics of an agent (e.g. 'operating.system=Windows', 'fast.Builds=true'). For an agent to be able to build a job, both the 'key' and 'value' must match the jobs's requirements.

Key	Value	Operations
os	Windows_Stock	View | Edit | Delete
system.hg.executable	C:\opt\mercurial\hg.exe	View | Edit | Delete

Executable

'executable' capabilities define the executables which are available to your build plans.

Executable label A label to uniquely identify this executable	Path Please enter the path to your executable	Operations
Ant (Ant)	C:\opt\ant-1.9	View | Edit | Delete
Ant 1.8 (Ant)	C:\opt\ant-1.8	View | Edit | Delete
Ant 1.9 (Ant)	C:\opt\ant-1.9	View | Edit | Delete
Maven 2 (Maven 2.x)	C:\opt\maven-2.2	View | Edit | Delete
Maven 2.0 (Maven 2.x)	C:\opt\maven-2.0	View | Edit | Delete
Maven 2.1 (Maven 3.x)	C:\opt\maven-2.1	View | Edit | Delete
Maven 2.2 (Maven 2.x)	C:\opt\maven-2.2	View | Edit | Delete
Maven 3.0 (Maven 3.x)	C:\opt\maven-3.0	View | Edit | Delete
Maven 3.2 (Maven 3.x)	C:\opt\maven-3.2	View | Edit | Delete

JDK

'JDK' capabilities define the JDKs which are available to your build plans.

JDK label	Java home	Operations
JDK 1.6	C:\opt\jdk-6	View | Edit | Delete
JDK 1.7	C:\opt\jdk-7	View | Edit | Delete
JDK 1.8	C:\opt\jdk-8	View | Edit | Delete

Disabling an elastic agent

An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

If you'd like to stop an elastic agent, you can disable it in Bamboo. This will abandon any job build it is running and prevent it from running any further job builds.

Note that disabling an elastic agent won't shut down the elastic instance it's running on (i.e. you will still be charged for the instance uptime). You can permanently stop an elastic agent and instance by shutting down the elastic instance.

The Bamboo server also supervises your elastic agents. If the Bamboo server detects that an elastic agent is offline, it will automatically terminate the elastic instance.

Related pages:

- Managing your elastic agents

To disable an elastic agent:

1. Navigate to the desired elastic agent, as described in Viewing your elastic agents.
2. Select **Disable** in the **Operations** column for the elastic agent. The elastic agent will display a status of Idle (Disabled).
   Re-enable the elastic agent by selecting **Enable**.

Elastic Bamboo FAQ

This page provides answers to common questions about running builds using Elastic Bamboo. If you are using Elastic Bamboo for the first time, we highly recommend that you read Getting started with Elastic Bamboo for instructions on setting up Elastic Bamboo and running your first build.

What job builds can I run on Elastic Bamboo?

You can run any of your job builds on any elastic agent (which in turn runs on an elastic instance), provided that the elastic agent's capabilities meet the job's requirements. An elastic agent inherits the capabilities of the elastic image it was created from. Hence, you can see which of your jobs can run on elastic agents by checking that your job's requirements match your elastic image's capabilities.

You can view your elastic image and the job builds that meet its requirements on the Agents and plans matrix.

On this page:

How do I run a plan build and its jobs on an elastic agent?

An elastic agent operates in a similar way to a non-elastic agent. The Bamboo server will determine if any job builds in the queue can be built on any of the available agents (including elastic agents), based on whether or not the capabilities of these agents meet the requirements of these jobs.

If an available elastic agent (like any other available agent) has capabilities which meet the requirements of a build in the build queue, the Bamboo server will assign the job build to that elastic agent.

If you do not have any free elastic agents running, you can configure Bamboo to automatically start up elastic instances whose elastic agents are capable of running job builds in the queue, or you can start up an appropriate elastic instance manually. (When an elastic instance is started, its elastic agent is also started, automatically.) For more information about starting elastic instances manually, refer to Starting an elastic instance.

If you do not use Bamboo's **Automatic Elastic Instance Management** feature and prefer to manage your elastic instances manually, then we strongly recommend that you **shut down** any elastic instances (running your elastic agents), when they are not in use. Minimizing unutilized elastic instance uptime will help reduce costs. Read Shutting down an elastic instance for instructions on how to shut down an elastic instance.

How do I automatically start or shut down elastic instances for job builds?

Bamboo can automatically start elastic instances based on demand from the build queue and shut them down once the elastic agents running on them have been idle for a specified period of time. For more information, please refer to the Automatic Elastic Instance Management section of the Configuring Elastic Bamboo topic.

While Bamboo's **Automatic Elastic Instance Management** feature is the easiest and most effective method of managing elastic instances in Bamboo, you can also manage elastic instances using the Bamboo REST API. For example, you could implement cron jobs to intelligently start and stop elastic instances, so that elastic agents are available at key times for your job builds.

How do I know whether my job build was run on an elastic agent?

The name of the image and elastic agent that ran a job build can be viewed as part of the build result. Please see the Viewing a build result page for more information.

How do I customize the capabilities of my elastic agents?

You may want to customize the capabilities of your elastic agents to suit certain jobs in your plans. For example, if you want to force certain job builds to only run on elastic agents, you can add a custom capability of `elastic=true` to your elastic agents and add the same requirement to these jobs.

To customize the capabilities for your elastic agents, you need to customize the capabilities of the image that they are created from. Read Configuring elastic agent capabilities for instructions.

How much does it cost to run a build?

As Elastic Bamboo usage varies from customer to customer, we cannot provide a definitive cost estimate for running a job build using Elastic Bamboo. We do provide high level guidelines for Elastic Bamboo costs, based on our own experience of using Elastic Bamboo at Atlassian, on the Elastic Bamboo Costs page.

You can significantly reduce the costs and time taken to run a job build by configuring Elastic Bamboo to use Automatic Elastic Instance Management and Amazon's Elastic Block Store (EBS).

What is EBS and how does it affect my job builds?

The Amazon Elastic Block Store (EBS) provides persistent storage volumes that can be attached to EC2 instances. Elastic Bamboo can use the EBS to store snapshots of relatively static build information, such as checkouts of source code and Maven repository data. You can choose a snapshot to create EBS volumes from. These volumes can then be attached to your elastic instances when they start up.

Can I use an Encrypted EBS Volume with an Elastic Agent?

While Bamboo does not support customizing the *RunInstance* command you may be able to achieve the desired results by using an Encrypted EBS enabled AMI. When the instance based on that AMI is launched it should also launch the encrypted EBS volume. For information about configuring the AMI to use the encrypted EBS volume, please see Amazon's documentation at Using encryption with EBS-backed AMIs.

Disabling Elastic Bamboo

If you don't want to execute Plan builds and their Jobs in the Amazon EC2 anymore, you can disable Elastic Bamboo for your Bamboo installation. Your AWS account details will be preserved when you disable Elastic Bamboo, so you can just enable it if you want to start using it again.

Related pages:

- Configuring Elastic Bamboo

Before you begin:

- Please ensure that you don't require your elastic agents before disabling Elastic Bamboo, as they will be **stopped immediately**.

To disable Elastic Bamboo:

1. From the top navigation bar select  > **Elastic Bamboo** > **Configuration**.
2. Select **Disable**. Elastic Bamboo will be disabled and a confirmation message will be displayed.

Quick filters

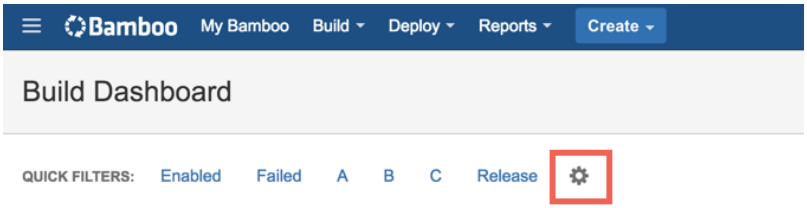
When opened in a viewport, the user will be redirected to: Quick filters for Bamboo.

Use quick filters for handy search shortcuts in your Bamboo build dashboard. Create filters based on configurable rules and never miss a build plan again.

Quick filters work only with plans displayed in Bamboo dashboard, which means that they don't include plan branches.

Configuration

Administrators can add, edit, and delete quick filters by clicking the cog icon in the quick filters menu:



The configuration view is also available from **Administration > Plans > Quick filters**.

Quick filters are available on the build dashboard for all users of a Bamboo instance.

Types of rules

Click a filter name in the build dashboard to display plans that match the rules assigned to the filter.

A plan is displayed only if it matches all the rules specified for a filter.

You can create filters with combinations of the following rules:

Rule type name	Description
By completion date	Displays plans that completed within a specific time frame. For example, you can display plans that completed in the last three days.
By label	Displays plans with a specific label. You can define multiple labels. The plan is display when it has at least one label specified in the rule. To assign a label to a plan, click the name of a plan to display the plan summary and go to **Actions > Modify plan label**.
By name	Displays plans with a specific name or plans that match a regular expression.
By project	Displays plans that are assigned to a specific project. You can select one or more projects.
By result status	Displays plans that have completed with a specific result. You can select from: • Successful • Failed

By status	Displays plans based with a specific status. You can select from: • Enabled • Disabled

Users and permissions

There are several options for managing your Bamboo users and groups:

- Manage locally in Bamboo.
- Manage with Atlassian's Jira applications or Crowd.
- Manage with an external user repository, such as LDAP.

On this page:

Note that the information on this page *doesn't* relate to application-level security for Bamboo – see Security inste ad.

Select a user management option

To select how you want to manage users in Bamboo:

1. From the top navigation bar select  > **Security** > **User directories**.
2. Select one of the user management options:

   Manage locally in Bamboo
   Managing users
   Managing groups

   Manage with Atlassian's Jira applications
   Allowing Other Applications to Connect to Jira applications for User Management

   Manage with Atlassian's Crowd
   Integrating Bamboo with Crowd

   Manage in a custom external user directory
   Integrating Bamboo with LDAP
3. Select **Save**.

About users and authors

An *author* is any person who checks in code to a repository that is associated with a Bamboo plan. An author need not be a Bamboo user.

Depending on your organization's requirements, you can configure Bamboo to grant access to non-users. However, only Bamboo users can:

- view the **My Bamboo** tab on the Dashboard.
- belong to a group.

About groups

Bamboo *groups* are used to specify which users will have global permissions and plan permissions. They can also be used to specify which users will receive notifications about a plan's build results. You can create and delete as many groups as you need. You will typically create at least one group per project.

A special group called **bamboo-admin** is automatically created when you install Bamboo. Members of this group have Bamboo administration rights.

Bamboo permissions

Bamboo permissions control access to plans, builds, and administration functions. See Managing permissions.

A *plan permission* is the ability to perform a particular operation on a plan and its jobs. For each plan, different permissions can be granted to particular groups and/or users. A *global permission* is the ability to perform a particular operation in relation to Bamboo as a whole.

Managing users

This page describes procedures for managing your Bamboo users locally in Bamboo.

For a brief overview of other options for managing your Bamboo users, see Users and permissions.

> ⚠ If you store a user's data in an external user directory like Crowd or LDAP, you should manage such users in these external directories and not in Bamboo.

Creating new user account

Follow this process if you store the user's data in Bamboo. If the user data is stored in an external user directory like Crowd or LDAP, go to that directory to perform this action.

Bamboo users can:

- view the **My Bamboo** tab on the Dashboard.
- belong to a group.

Depending on your organization's requirements, you can also configure Bamboo to grant access to non-users.

To create a Bamboo user:

1. From the top navigation bar select  > **Security** > **Users**.
2. Select **Create user**.
3. Complete the Add user form:

   Username
   Username can't be changed after the user is created.

   Full name
   User's full name.

   Email
   The address to which notifications will be sent.

   Password
   The user can easily change their password later.

   Instant Messaging address
   If no IM address is specified, Bamboo won't be able to recognize the user's *context* when interacting using IM.
4. Select **Save**.

Changing users' passwords or details

Follow this process if you store the user's data in Bamboo. If the user data is stored in an external user directory like Crowd or LDAP, go to that directory to perform this action.

To change a user's password or details:

1. From the top navigation bar select  > **Security** > **Users**.
2. Locate the user by typing part of their username, full name, or email.
3. Select **Edit** next to the user you want to modify.
4. Edit the user's details or password as necessary.
    - If you have configured SMTP email on your Bamboo server, the user will automatically receive an email containing their new password.
    - The user can easily change their password later.
5. Select **Save**.

Note that:

- Users who have forgotten their passwords can select the **Forgotten your password?** link on the Bamboo login screen. This will automatically generate a new password and email it to them (provided the Bamboo server has been configured to send SMTP email).
- Logged-in users can change their own password and details, as described in Managing your user profile.
- See Associating your author name with your user profile for information about **Source Repository Aliases**.

Granting administration rights to a user

Follow this process if you store the user's data in Bamboo. If the user data is stored in an external user directory like Crowd or LDAP, go to that directory to perform this action.

In Bamboo, there are two types of administrators:

- *Global administrators* — that is, people with the Admin *global* permission. These people can access the Bamboo **Administration** menu. They can also administer every plan.
- *Plan administrators* — that is, people with the Admin and Edit *plan* permissions. These people can administer a particular plan.

Grant global administration rights

To grant global administration rights to a user:

- Either grant the Admin global permission to the user explicitly (as described in Granting global permissions to users or groups);
  OR:
- Add the user to a *group* which has the Admin global permission (as described in Changing group members).

On this page:

Related pages:

- Managing users
- Granting global permissions to users or groups
- Changing group members
- Granting plan permissions in bulk

Grant plan administration rights

- Either grant the Admin and Edit plan permissions to the user explicitly (as described in Granting plan permissions in bulk);
  OR:
- Add the user to a *group* which has the Admin and Edit plan permissions (as described in Changing group members).

Deleting or deactivating a user

Follow this process if you store the user's data in Bamboo. If the user data is stored in an external user directory like Crowd or LDAP, go to that directory to perform this action.

Deleting a user removes their Bamboo user account. Deactivating a user prevents them from logging in to Bamboo.

Deleting a Bamboo user

Before you begin:

- Deleting a Bamboo user will not delete their *author* data — that is, their author statistics and code check-in comments will still exist in Bamboo.
- You cannot delete a user who has created labels or comments about build results. You may want to deactivate them instead.
- You cannot delete the user account with which you are currently logged in to Bamboo.

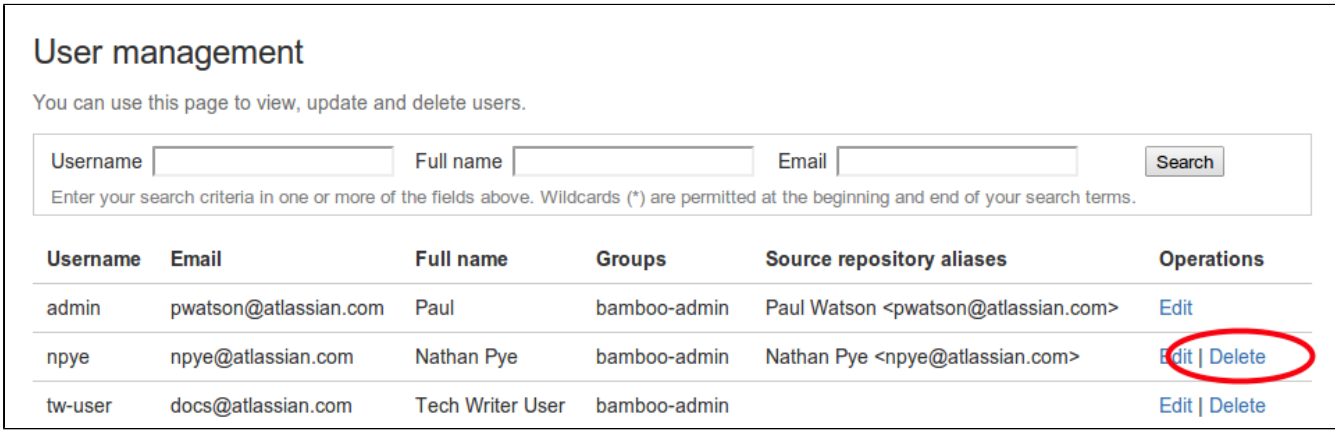
On this page:

Related pages:

- Managing users

To delete a Bamboo user:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Select **Users** in the left navigation panel.
3. Use the **Delete** link in the 'Operations' column.



Deactivating a Bamboo user

You can deactivate a user, which means that they won't receive any email or IM notifications and they won't be able to log in to your Bamboo instance.

Deactivated users can't recover their passwords, but they can still can trigger builds by committing to repositories.

To deactivate a Bamboo user:

1. Go to **Administration > User management**.
2. Find the user that you want to deactivate.
3. Click **Edit**.
4. Unselect the **Active user** check box:


5. Click **Save**.

Alternative scenario:

1. Go to **Administration > User management**.
2. Find the user that you want to deactivate.
3. Click **Edit**.
4. Enter a new password for the user.
   If you have configured SMTP email on your Bamboo server, the user will automatically receive an email containing their new password.
5. To get around the email problem, enter an invalid email address in the **Email** field, for example foobar@foobaremailaddress.foobar.
6. Delete the user's **Instant Messaging Address** so that he or she does not receive notifications on build events.
7. Click **Save**.

Invalidating active user sessions

Unless a user has selected the **Remember my login on this computer** checkbox when logging into Bamboo, by default, their session will expire after 30 minutes of inactivity. If you need to force the invalidation of all active user sessions for a particular user, you can do that from the Bamboo web interface or through the REST API.

> When a user changes their password, all their sessions except the one they used to request the password change will be automatically invalidated, and their `rememberme` token will be deleted from the database.

On this page:

Invalidating user sessions through the Bamboo web interface

To invalidate a user's all active sessions through the Bamboo web interface:

1. From the top navigation bar, select **Administration** > **User management**.
2. From the list on the **Users** page, select the user whose sessions you'd like to invalidate.
3. In the top-right corner of the **User details** page, select **Invalidate sessions**.
4. In the confirmation dialog, select **Confirm**.

Invalidating user sessions through the Bamboo REST API

To invalidate a user's all active sessions through the Bamboo REST API, call the following endpoint as an Administrator:

```
DELETE /rest/admin/latest/session/{username}
```

Replace `{username}` with the username of the account whose sessions you want to invalidate.

Invalidating persistent sessions

If you've enabled session persistence across Bamboo server restarts, sessions started before a restart can't be invalidated using any of the methods described on this page. In case you need to invalidate persistent sessions (potentially for security reasons), you can do so by deleting the file that stores session data and manually removing the `rememberme` token from the database.

To invalidate persistent sessions:

1. Shut down Bamboo.
2. Check the following directories for the `SESSIONS.ser` file and delete it from there:
   - `$CATALINA_BASE/work/<ENGINE_NAME>/<HOSTNAME>/<APP_NAME>`
   - `$CATALINA_BASE/work/Catalina/localhost/ROOT`
3. Filter the `rememberme_token` table in your database by the name of the user whose sessions you want to invalidate and delete all `rememberme` tokens associated with them.

Managing groups

Bamboo *groups* are used to specify which users will have global permissions and plan permissions. They can also be used to specify which users will receive notifications about a plan's build results. You can create and delete as many groups as you need. You will typically create at least one group per project.

A special group called **bamboo-admin** is automatically created when you install Bamboo. Members of this group have Bamboo administration rights.

Read more about managing groups for your users:

- Creating a group
- Deleting a group
- Changing group members

Creating a group

Bamboo *groups* are used to specify which users will have global permissions and plan permissions. They can also be used to specify which users will receive notifications about a plan's build results. You can create and delete as many groups as you need. You will typically create at least one group per project.

A special group called **bamboo-admin** is automatically created when you install Bamboo. Members of this group have Bamboo administration rights.

Related pages:

- Managing groups

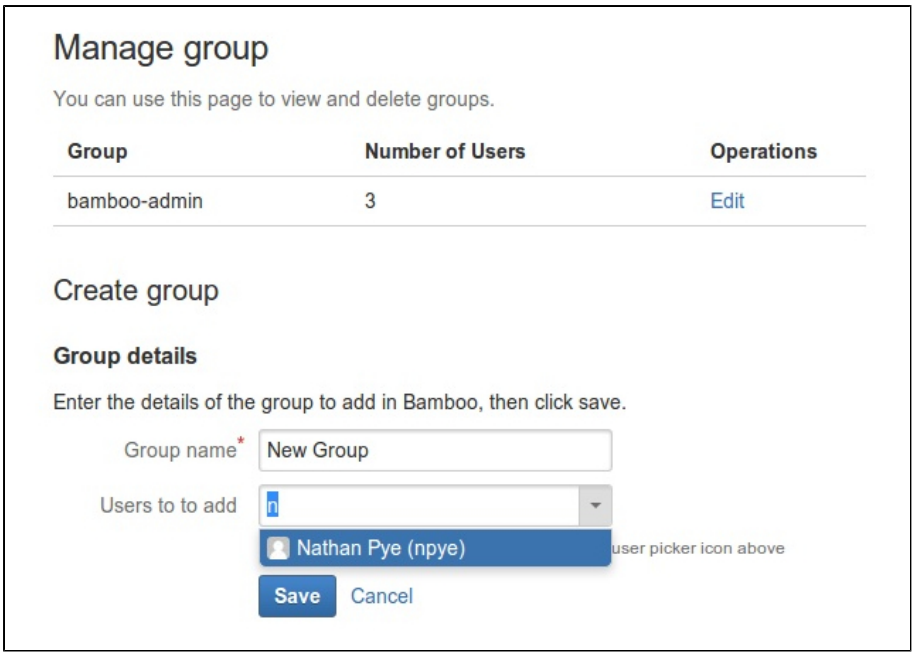
To create a group:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. In the left-hand navigation panel, click **Groups**.
3. Type a name for your new group into **Group Name**.

   > Group name cannot be changed after the group is created.
   >
   > In Bamboo, a group name can only contain characters from the following set: a-z, @, . , _ , - , 0-9. You can't create groups that contain invalid character. Similarly, if you try to import of groups that contain invalid characters from an external source, the import of that group will fail.

4. From the **Users to add** list, select your users. Hold <Ctrl> to select multiple users. You can also add or remove users from the group later if required.
5. Click **Save**.

Screenshot: Creating a Bamboo group



Deleting a group

To delete a group:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Groups** in the left navigation panel. The 'Manage Groups' screen will be displayed.
3. Click **Delete** for the relevant group, in the 'Operations' column.

Note that the *bamboo-admin* group cannot be deleted.

Changing group members

Bamboo *groups* are used to specify which users will have global permissions and plan permissions. They can also be used to specify which users will receive notifications about a plan's build results. You can create and delete as many groups as you need. You will typically create at least one group per project.

A special group called **bamboo-admin** is automatically created when you install Bamboo. Members of this group have Bamboo administration rights.

To change the members of a group:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Groups** in the left navigation panel. The 'Manage Groups' screen will be displayed.
3. Click **Edit** for the relevant group, in the 'Operations' column. The 'Edit Group Details' screen will be displayed. Users who already belong to the group are shown in blue; users who do not currently belong to the group are shown in white.
4. Press the <Ctrl> key and hold it while you select (or deselect) the users whom you want to add to (or remove from) the group.
5. Click **Save**.

Related pages:

- Managing groups

Connecting to external user directories

You can connect Bamboo to external user directories. This allows you to use existing users and groups stored in an enterprise directory, and to manage those users and groups in one place.

User management functions include:

- **Authentication:** determining which user identity is sending a request to Bamboo.
- **Authorization**: determining the access privileges for an authenticated user.
- **User management:** maintaining profile information in user's accounts.
- **Group membership:** storing and retrieving groups, and group membership.

It is important to understand that these are separate components of a user management system. You could use an external directory for any or all of the above tasks.

There are several approaches to consider when using external user directories wth Bamboo, described briefly below:

- Bamboo provides a "read-only" connection to external directories for user management. This means that users and groups, fetched from *any external directory*, can only be modified or updated in the external directory itself, rather than in Bamboo.
- Bamboo comes with an internal user directory, already built-in, that is enabled by default at installation.

LDAP

You should consider connecting to an LDAP directory server if your users and groups are stored in an enterprise directory. See Integrating Bamboo with LDAP for instructions.

Bamboo is able to connect to the following LDAP directory servers:

- Microsoft Active Directory
- Apache Directory Server (ApacheDS) 1.0.x and 1.5.x
- Apple Open Directory (Read-Only)
- Fedora Directory Server (Read-Only Posix Schema)
- Novell eDirectory Server
- OpenDS
- OpenLDAP
- OpenLDAP (Read-Only Posix Schema)
- Generic Posix/RFC2307 Directory (Read-Only)
- Sun Directory Server Enterprise Edition (DSEE)
- Any generic LDAP directory server

Crowd

You can connect Bamboo to Atlassian Crowd for user and group management, as well as for user authentication.

Crowd is an application security framework that handles authentication and authorization for your web-based applications. With Crowd you can integrate multiple web applications with multiple user directories, with support for single sign-on (SSO) and centralized identity management. See the Crowd Administration Guide.

You should consider connecting to Crowd if you want to use Crowd to manage existing users and groups in multiple directory types, or if you have users of other web-based applications.

See Integrating Bamboo with Crowd for configuration instructions.

Multiple directories

When Bamboo is connected directly to multiple user directories, where duplicate user names and group names are used across those directories, the effective group memberships that Bamboo uses for authorization can be determined using either of these two schemes:

- 'aggregating membership'
- 'non-aggregating membership'.

See Effective memberships with multiple directories for more information about these two schemes.

Note that:

- Aggregating membership is used by default for new installations of Bamboo.
- Authentication, for when Bamboo is connected to multiple directories, only depends on the mapped groups in those directories – the aggregation scheme is not involved at all.
- The directory order is significant during the authentication of the user, in cases where the same user exists in multiple directories. When a user attempts to log in, the application will search the directories in the order specified, and will use the credentials (password) of the *first occurrence of the user* to validate the login attempt.
- For inactive users, Bamboo only checks if the user is active in the first (highest priority) directory in which they are found for the purpose of determining authentication. Whether a user is active or inactive does not affect how their memberships are determined.
- When a user is added to a group, they are only added to the first writeable directory available, in priority order.
- When a user is removed from a group, they are only removed from the group in the first directory the user appears in, when non-aggregating membership is used. With aggregating membership, they are removed from the group in *all* directories the user exists in.
- When using Single Sign-On with Crowd and multiple Crowd directories:
  - signed-in users will be validated against the first Crowd directory the user is in
  - users that haven't signed in yet, but have a valid Crowd SSO cookie will be validated against all configured Crowd directories in order

A Bamboo admin can change the membership scheme used by Bamboo using the following commands:

- To change to *aggregating membership*, substitute your own values for `<username>`, `<password>` and `<base-url>` in this command:

```
curl -H 'Content-type: application/json' -X PUT -d '{"membershipAggregationEnabled":true}' -u
<username>:<password> <base-url>/rest/crowd/latest/application
```

- To change to non-*aggregating membership*, substitute your own values for `<username>`, `<password>` and `<base-url>` in this command:

```
curl -H 'Content-type: application/json' -X PUT -d '{"membershipAggregationEnabled":false}' -u
<username>:<password> <base-url>/rest/crowd/latest/application
```

Note that these operations are different from how you make these changes in Crowd. Note also that changing the aggregation scheme can affect the authorization permissions for your users, and how directory update operations are performed.

Connecting Bamboo to JIRA for user management

⚠ *This page does not apply to JIRA Software Cloud; you can't use JIRA Software Cloud to manage your Bamboo users.*

You can connect Bamboo to an existing Atlassian JIRA Software instance to delegate Bamboo user and group management, and authentication. Bamboo provides a "read-only" connection to JIRA Software for user management. This means that users and groups, fetched from JIRA Software, can only be modified or updated in that JIRA Software server, rather than in Bamboo.

Choose this option, as an alternative to Atlassian Crowd, for simple configurations with a limited number of users. Note that Bamboo can only connect to an instance running JIRA Software 4.3 or later.

Connecting Bamboo and JIRA Software is a 3-step process:

1. Set up JIRA Software to allow connections from Bamboo
2. Set up Bamboo to connect to JIRA Software
3. Set up Bamboo users and groups in JIRA Software

Also on this page:

⚠ You need to be an administrator in JIRA Software and a system administrator in Bamboo to perform the following tasks.

1. Setup JIRA Software to allow connections from Bamboo

1. Log in as a user with the 'JIRA Software Administrators' global permission.
2. For JIRA 4.3.x, select **Other Application** from the 'Users, Groups & Roles' section of the 'Administration' menu.
   For later versions, choose **Administration** > **Users** > **JIRA User Server.**
3. Click **Add Application**.
4. Enter the **application name** (case-sensitive) and **password** that Bamboo will use when accessing JIRA Software.
5. Enter the **IP address** of your Bamboo instance. Valid values are:
   - A full IP address, e.g. `192.168.10.12`.
   - A wildcard IP range, using CIDR notation, e.g. `192.168.10.1/16`. For more information, see the introduction to CIDR notation on Wikipedia and RFC 4632.
6. Click **Save**.
7. Define the directory order, on the 'User Directories' screen, by clicking the blue up- and down-arrows next to each directory. The directory order has the following effects:
   - The order of the directories is the order in which they will be searched for users and groups.
   - Changes to users and groups will be made only in the first directory where the application has permission to make changes.

2. Setup Bamboo to connect to JIRA Software

1. Log in to Bamboo as a user with 'Admin' permission.
2. In the Bamboo administration area click **User Directories** (under 'Security').
3. Click **Add Directory** and select **Atlassian JIRA**.
4. Enter settings, as described below.
5. Test and save the directory settings.

6. Define the directory order, on the 'User Directories' screen, by clicking the arrows for each directory. The directory order has the following effects:
    - The order of the directories is the order in which they will be searched for users and groups.
    - Changes to users and groups will be made only in the first directory where the application has permission to make changes.

3. Set up Bamboo users and groups in JIRA Software

In order to use Bamboo, users must be a member of the `Bamboo-users` group or have Bamboo global permissions. Follow these steps to configure your Bamboo groups in JIRA Software:

1. Add the `bamboo-users` and `bamboo-administrators` groups in JIRA Software.
2. Add your own username as a member of both of the above groups.
3. Choose one of the following methods to give your existing JIRA Software users access to Bamboo:
    - Option 1: In JIRA Software, find the groups that the relevant users belong to. Add those groups as members of one or both of the above Bamboo groups.
    - Option 2: Log in to Bamboo using your JIRA Software account and go to the administration area. Click **Global permissions** (under 'Security'). Assign the appropriate permissions to the relevant JIRA Software groups.

> Connecting Atlassian Bamboo to JIRA Software for user management is not sufficient, by itself, to allow your users to log in to Bamboo. You must also grant them access to Bamboo by using one of the above 2 options.
>
> We recommend that you use groups instead of individual accounts when granting permissions.

Server settings

Setting	Description
Name	A meaningful name that will help you to identify this Jira server in the list of directory servers. Examples: • `Jira Software` • `My Company Jira`
Server URL	The web address of your Jira server. Examples: • `http://www.example.com:8080` • `http://jira.example.com`
Application Name	The name used by your application when accessing the Jira server that acts as user manager. Note that you will also need to define your application to that Jira server, via the '**Other Applications**' option in the 'Users, Groups & Roles' section of the 'Administration' menu.
Application Password	The password used by your application when accessing the Jira server that acts as user manager.

Advanced settings

Setting	Description

Enable Nested Groups	Enable or disable support for nested groups. Before enabling nested groups, please check to see if nested groups are enabled on the JIRA server that is acting as user manager. When nested groups are enabled, you can define a group as a member of another group. If you are using groups to manage permissions, you can create nested groups to allow inheritance of permissions from one group to its sub-groups.
Enable Incremental Synchronization	Enable or disable incremental synchronization. Only changes since the last synchronization will be retrieved when synchronizing a directory.
Synchronization Interval (minutes)	Synchronization is the process by which the application updates its internal store of user data to agree with the data on the directory server. The application will send a request to your directory server every x minutes, where 'x' is the number specified here. The default value is 60 minutes.

Integrating Bamboo with Crowd

You can configure Bamboo to use Atlassian Crowd for user and group management, and for authentication and authorization.

Atlassian Crowd is an application security framework that handles authentication and authorization for your web-based applications. With Crowd you can integrate multiple web applications and user directories, with support for single sign-on (SSO) and centralized identity management. See the Crowd Administration Guide.

Connect to Crowd if you want to use Crowd to manage existing users and groups in multiple directory types, or if you have users of other web-based applications.

On this page:

To connect Bamboo to Crowd:

1. Log in as a user with 'Admin' permission.
2. In the Bamboo administration area, click **User Directories** (under 'Security').
3. Click **Add Directory** and select **Atlassian Crowd**.
4. Enter settings, as described below.
5. Test and save the directory settings.
6. Define the directory order, on the **Directories** tab, by clicking the blue up- and down-arrows next to each directory. The directory order has the following effects:
    - The order of the directories is the order in which they will be searched for users and groups.
    - Changes to users and groups will be made only in the first directory where the application has permission to make changes.

Server settings

Setting	Description
Name	A meaningful name that will help you to identify this Crowd server amongst your list of directory servers. Examples: • `Crowd Server` • `Example Company Crowd`
Server URL	The web address of your Crowd console server. Examples: • `http://www.example.com:8095/crowd/` • `http://crowd.example.com`
Application Name	The name of your application, as recognized by your Crowd server. Note that you will need to define the application in Crowd too, using the Crowd administration Console. See the Crowd documentation on adding an application.
Application Password	The password which the application will use when it authenticates against the Crowd framework as a client. This must be the same as the password you have registered in Crowd for this application. See the Crowd documentation on adding an application.

Crowd permissions

Bamboo offers **Read Only** permissions for Crowd directories. The users, groups and memberships in Crowd directories are retrieved from Crowd and can only be modified from Crowd. You cannot modify Crowd users, groups or memberships using the Bamboo administration screens.

Advanced settings

Setting	Description
Enable Nested Groups	Enable or disable support for nested groups. Before enabling nested groups, please check to see if the user directory or directories in Crowd support nested groups. When nested groups are enabled, you can define a group as a member of another group. If you are using groups to manage permissions, you can create nested groups to allow inheritance of permissions from one group to its sub-groups.
Enable Incremental Synchronization	Enable or disable incremental synchronization. Only changes since the last synchronization will be retrieved when synchronizing a directory. Note that full synchronization is always executed when restarting the application.
Synchronization Interval (minutes)	Synchronization is the process by which the application updates its internal store of user data to agree with the data on the directory server. The application will send a request to your directory server every x minutes, where 'x' is the number specified here. The default value is 60 minutes.

Single sign-on (SSO) with Crowd

Bamboo supports the following options for configuring Crowd SSO:

- Cookie-based single domain SSO, Learn how to configure cookie-based SSO
- Crowd SSO 2.0. Learn how to configure Crowd SSO 2.0

For more information, see Overview of SSO.

> Use only one configuration option at a time.

Using multiple directories

When Bamboo is connected to Crowd you can map Bamboo to multiple user directories in Crowd.

For Crowd 2.8, and later versions, there are two different membership schemes that Crowd can use when multiple directories are mapped to an integrated application, and duplicate user names and group names are used across those directories. The schemes are called 'aggregating membership' and 'non-aggregating membership' and are used to determine the effective group memberships that Bamboo uses for *authorization*. See Effective memberships with multiple directories for more information about these two schemes in Crowd.

Note that:

- *Authentication*, for when Bamboo is mapped to multiple directories in Crowd, only depends on the mapped groups in those directories – the aggregation scheme is not involved at all.
- For inactive users, Bamboo only checks if the user is active in the first (highest priority) directory in which they are found to determine *authentication*. The membership schemes described above are not used when Crowd determines if a user should have access to Bamboo.
- When a user is added to a group, they are only added to the first writeable directory available, in priority order.
- When a user is removed from a group, they are only removed from the group in the first directory the user appears in, when non-aggregating membership is used. With aggregating membership, they are removed from the group in *all* directories the user exists in.

An administrator can set the aggregation scheme that Bamboo uses when integrated with Crowd. Go to the **Directories** tab for the Bamboo instance in Crowd, and check **Aggregate group memberships across directories** to use the 'aggregating membership' scheme. When the checkbox is clear 'non-aggregating membership' is used.

Note that changing the aggregation scheme can affect the authorization permissions for your Bamboo users, and how directory update operations are performed.

Integrating Bamboo with LDAP

You can connect Bamboo to an existing LDAP user directory, so that your existing users and groups in an enterprise directory can be used in Bamboo. The LDAP directory is used for both user authentication and account management.

Bamboo is able to connect to the following LDAP directory servers:

- Microsoft Active Directory
- Apache Directory Server (ApacheDS) 1.0.x and 1.5.x
- Apple Open Directory (Read-Only)
- Fedora Directory Server (Read-Only Posix Schema)
- Novell eDirectory Server
- OpenDS
- OpenLDAP
- OpenLDAP (Read-Only Posix Schema)
- Generic Posix/RFC2307 Directory (Read-Only)
- Sun Directory Server Enterprise Edition (DSEE)
- Any generic LDAP directory server

On this page:

Synchronization when Bamboo is first connected to the LDAP directory

When you first connect Bamboo to an existing LDAP directory, the Bamboo is synchronized with the LDAP directory. User information, including groups and group memberships, is copied across to the Bamboo database.

Note that when Bamboo is connected to an LDAP directory, you cannot update user details in Bamboo. Updates must be done directly on the LDAP directory, perhaps using a LDAP browser tool such as Apache Directory Studio.

Option - Use LDAP filters to restrict the number of users and groups that are synchronized

You can use LDAP filters to restrict the users and groups that are synchronized with the Bamboo internal directory. You may wish to do this in order to limit the users or groups that can access Bamboo, or if you are concerned that synchronization performance may be poor.

For example, to limit synchronization to just the groups named "Bamboo_user" or "red_team", enter the following into the **Group Object Filter** field (see Group Schema Settings below):

```
(&(objectClass=group)(|(cn=Bamboo_user)(cn=red_team)))
```

For further discussion about filters, with examples, please see How to write LDAP search filters. Note that you need to know the names for the various containers, attributes and object classes in your particular directory tree, rather than simply copying these examples. You can discover these container names by using a tool such as Apache Directory Studio.

Authentication when a user attempts to log in

When a user attempts to log in to Bamboo, the username and password are passed to the LDAP directory for confirmation. If the password matches that stored for the user, LDAP passes a confirmation back to Bamboo, and Bamboo logs in the user. During the user's session, all authorizations (i.e. access to Bamboo resources such as repositories, reviews and administration screens) are handled by Bamboo, based on permissions maintained by Bamboo.

Connecting Bamboo

To connect Bamboo to an LDAP directory:

1. Log in as a user with 'Admin' permission.
2. In the Bamboo administration area, click **User Directories** (under 'Security').
3. Click **Add Directory** and select either **Microsoft Active Directory** or **LDAP** as the directory type.
4. Configure the directory settings, as described in the tables below.
5. Save the directory settings.
6. Define the directory order by clicking the arrows next to each directory on the 'User Directories' screen. The directory order has the following effects:
    - The order of the directories is the order in which they will be searched for users and groups.
    - Changes to users and groups will be made only in the first directory where the application has permission to make changes.

Server settings

Setting	Description
Name	Enter a meaningful name to help you identify the LDAP directory server. Examples: • `Example Company Staff Directory` • `Example Company Corporate LDAP`
Director y type	Select the type of LDAP directory that you will connect to. If you are adding a new LDAP connection, the value you select here will determine the default values for many of the options on the rest of screen. Examples: • `Microsoft Active Directory` • `OpenDS` • And more
Hostna me	The host name of your directory server. Examples: • `ad.example.com` • `ldap.example.com` • `opends.example.com`
Port	The port on which your directory server is listening. Examples: • `389` • `10389` • `636` (for example, for SSL)

Use SSL	Check this if the connection to the directory server is an SSL (Secure Sockets Layer) connection. Note that you will need to configure an SSL certificate to use this setting.
Username	The distinguished name of the user that the application will use when connecting to the directory server. Examples: • `cn=administrator,cn=users,dc=ad,dc=example,dc=com` • `cn=user,dc=domain,dc=name` • `user@domain.name` By default, all users can read the uSNChanged attribute; however, only administrators or users with relevant permissions can access the Deleted Objects container. The specific privileges required by the user to connect to LDAP are "Bind" and "Read" (user info, group info, group membership, update sequence number, deleted objects), which the user can obtain by being a member of the Active Directory's built-in administrators group. Note that the incremental sync will fail silently if the Active Directory is accessed by a user without these privileges. This has been reported as CWD-3093.
Password	The password of the user specified above. **Note:** Connecting to an LDAP server requires that this application log in to the server with the username and password configured here. As a result, this password cannot be one-way hashed - it must be recoverable in the context of this application. The password is currently stored in the database in plain text without obfuscation. To guarantee its security, you need to ensure that other processes do not have OS-level read permissions for this application's database or configuration files.

LDAP schema

Setting	Description
Base DN	The root distinguished name (DN) to use when running queries against the directory server. Examples: • `o=example,c=com` • `cn=users,dc=ad,dc=example,dc=com` • For Microsoft Active Directory, specify the base DN in the following format: `dc=domain1, dc=local`. You will need to replace the `domain1` and `local` for your specific configuration. Microsoft Server provides a tool called `ldp.exe` which is useful for finding out and configuring the the LDAP structure of your server.
Additional User DN	This value is used in addition to the base DN when searching and loading users. If no value is supplied, the subtree search will start from the base DN. Example: • `ou=Users`
Additional Group DN	This value is used in addition to the base DN when searching and loading groups. If no value is supplied, the subtree search will start from the base DN. Example: • `ou=Groups`

If no value is supplied for ***Additional User DN*** or ***Additional Group DN*** this will cause the subtree search to start from the base DN and, in case of a huge directory structure, could cause performance issues for login and operations that rely on login to be performed.

LDAP permission

Setting	Description
Read Only	LDAP users, groups and memberships are retrieved from your directory server and can only be modified via your directory server. You cannot modify LDAP users, groups or memberships via the application administration screens.
Read Only, with Local Groups	LDAP users, groups and memberships are retrieved from your directory server and can only be modified via your directory server. You cannot modify LDAP users, groups or memberships via the application administration screens. However, you can add groups to the internal directory and add LDAP users to those groups.

Advanced settings

Setting	Description
Enable Nested Groups	Enable or disable support for nested groups. Some directory servers allow you to define a group as a member of another group. Groups in such a structure are called *nested groups*. Nested groups simplify permissions by allowing sub-groups to inherit permissions from a parent group.
Manage User Status Locally	If true, you can activate and deactivate users in Crowd independent of their status in the directory server.
Filter out expired users	If true, user accounts marked as expired in Active Directory will be automatically removed. For cached directories, the removal of a user will occur during the first synchronization after the account's expiration date. **Note**: This is available in Embedded Crowd 2.0.0 and above, but not available in the 2.0.0 m04 release.
Use Paged Results	Enable or disable the use of the LDAP control extension for simple paging of search results. If paging is enabled, the search will retrieve sets of data rather than all of the search results at once. Enter the desired page size – that is, the maximum number of search results to be returned per page when paged results are enabled. The default is 1000 results.
Follow Referrals	Choose whether to allow the directory server to redirect requests to other servers. This option uses the node referral (JNDI lookup `java.naming.referral`) configuration setting. It is generally needed for Active Directory servers configured without proper DNS, to prevent a 'javax.naming.PartialResultException: Unprocessed Continuation Reference(s)' error.
Naive DN Matching	If your directory server will always return a consistent string representation of a DN, you can enable naive DN matching. Using naive DN matching will result in a significant performance improvement, so we recommend enabling it where possible. This setting determines how your application will compare DNs to determine if they are equal. • If this checkbox is selected, the application will do a direct, case-insensitive, string comparison. This is the default and recommended setting for Active Directory, because Active Directory guarantees the format of DNs. • If this checkbox is not selected, the application will parse the DN and then check the parsed version.

Enable Incremental Synchronization	Enable incremental synchronization if you only want changes since the last synchronization to be queried when synchronizing a directory. ⚠ Be aware that when using this option, the user account configured for synchronization must have read access to: • The `uSNChanged` attribute of all users and groups in the directory that need to be synchronized. • The objects and attributes in the Active Directory deleted objects container. If at least one of these conditions is not met, you may end up with users who are added to (or deleted from) the Active Directory not being respectively added (or deleted) in the application. This setting is only available if the directory type is set to "Microsoft Active Directory".

Update group memberships when logging in	This setting enables updating group memberships during authentication and can be set to the following options:
	
	• **Every time the user logs in**: during the authentication, the user's **direct** group memberships will be updated to match what's in the remote directory:
	  ◦ Remove the user from all groups that the user no longer belongs to in the remote directory.
	  ◦ Add the user to all the groups that the user belongs to in the remote directory. New groups with matching names and descriptions will be created locally if needed. The group will only contain the current user and other memberships will be populated when users who belong to the same group log in or when the synchronization happens.
	• **For newly added users only**: when a new user logs in for the first time, the user's **direct** group memberships will be updated to match what's in the remote directory.
	
	ⓘ Consider that the user's group memberships will be updated only if the user was created during the authentication.
	
	• **Never**: during the authentication, the user's group memberships won't change, even if the local state doesn't match what's in the remote.

Synchronization Interval (minutes)	Synchronization is the process by which the application updates its internal store of user data to agree with the data on the directory server. The application will send a request to your directory server every x minutes, where 'x' is the number specified here. The default value is 60 minutes.
Read Timeout (seconds)	The time, in seconds, to wait for a response to be received. If there is no response within the specified time period, the read attempt will be aborted. A value of 0 (zero) means there is no limit. The default value is 120 seconds.
Search Timeout (seconds)	The time, in seconds, to wait for a response from a search operation. A value of 0 (zero) means there is no limit. The default value is 60 seconds.
Connection Timeout (seconds)	This setting affects two actions. The default value is 10. • The time to wait when getting a connection from the connection pool. A value of 0 (zero) means there is no limit, so wait indefinitely. • The time, in seconds, to wait when opening new server connections. A value of 0 (zero) means that the TCP network timeout will be used, which may be several minutes.

User schema settings

Setting	Description

User Object Class	This is the name of the class used for the LDAP user object. Example: • `user`
User Object Filter	The filter to use when searching user objects. Example: • `(&(objectCategory=Person)(sAMAccountName=*))` More examples can be found in our knowledge base. See How to write LDAP search filters.
User Name Attribute	The attribute field to use when loading the username. Examples: • `cn` • `sAMAccountName` NB: In Active Directory, the 'sAMAccountName' is the 'User Logon Name (pre-Windows 2000)' field. The User Logon Name field is referenced by 'cn'.
User Name RDN Attribute	The RDN (relative distinguished name) to use when loading the username. The DN for each LDAP entry is composed of two parts: the RDN and the location within the LDAP directory where the record resides. The RDN is the portion of your DN that is not related to the directory tree structure. Example: • `cn`
User First Name Attribute	The attribute field to use when loading the user's first name. Example: • `givenName`
User Last Name Attribute	The attribute field to use when loading the user's last name. Example: • `sn`
User Display Name Attribute	The attribute field to use when loading the user's full name. Example: • `displayName`
User Email Attribute	The attribute field to use when loading the user's email address. Example: • `mail`
User Password Attribute	The attribute field to use when loading a user's password. Example: • `unicodePwd`
User Unique ID Attribute	The attribute used as a unique immutable identifier for user objects. This is used to track username changes and is optional. If this attribute is not set (or is set to an invalid value), user renames will not be detected — they will be interpreted as a user deletion then a new user addition. This should normally point to a UUID value. Standards-compliant LDAP servers will implement this as 'entryUUID' according to RFC 4530. This setting exists because it is known under different names on some servers, e.g. 'objectGUID' in Microsoft Active Directory.

Group schema settings

Setting	Description
Group Object Class	This is the name of the class used for the LDAP group object. Examples: • `groupOfUniqueNames` • `group`
Group Object Filter	The filter to use when searching group objects. Example: • `(&(objectClass=group)(cn=*))`
Group Name Attribute	The attribute field to use when loading the group's name. Example: • `cn`
Group Description Attribute	The attribute field to use when loading the group's description. Example: • `description`

Membership schema settings

Setting	Description
Group Members Attribute	The attribute field to use when loading the group's members. Example: • `member`
User Membership Attribute	The attribute field to use when loading the user's groups. Example: • `memberOf`
Use the User Membership Attribute, when finding the user's group membership	Check this if your directory server supports the group membership attribute on the user. (By default, this is the '`memberOf`' attribute.) • If this checkbox is selected, your application will use the group membership attribute on the user when **retrieving the list of groups to which a given user belongs**. This will result in a more efficient retrieval. • If this checkbox is not selected, your application will use the members attribute on the group ('`member`' by default) for the search. • If the **Enable Nested Groups** checkbox is selected, your application will ignore the **Use the User Membership Attribute** option and will use the members attribute on the group for the search.

Use the User Membership Attribute, when finding the members of a group	Check this if your directory server supports the user membership attribute on the group. (By default, this is the '`member`' attribute.) • If this checkbox is selected, your application will use the group membership attribute on the user when **retrieving the members of a given group**. This will result in a more efficient search. • If this checkbox is not selected, your application will use the members attribute on the group ('`member`' by default) for the search.

Testing LDAP or Active Directory connectivity with Paddle

Paddle is a tool that will test the LDAP or Active Directory settings in your `atlassian-user.xml`.

Using Paddle

You do not need to have Bamboo running to run this tool. The steps are:

1. Download into a directory where you have permissions to create files.
2. Copy your `atlassian-user.xml` into that directory - this is found in your .../{BAMBOO-HOME}/xml-data/configuration/ directory.
3. Run `java -jar paddle-x.x.jar` (where x.x is the version of Paddle you downloaded).

On this page:

Parameters

Paddle currently supports the following parameters:

Name	Example	Purpose
debug	`java -jar paddle-x.x.jar debug`	Prints DEBUG messages to the console as well as paddle.log.
limit	`java -jar paddle-x.x.jar limit=100`	Sets the limit on the number of results returned by user and group queries. Defaults to 10.

Sample output

This is an example of a successful run:

```
################################################################################################################
###############

LDAP Support Tool version 1.1

################################################################################################################
###############

Connection to LDAP/Active Directory Server at ldap://192.168.0.86:389 SUCCESSFUL.

----------------------------------------------------------------
TEST 1: Search and list 10 users
----------------------------------------------------------------

User: CN=Administrator
Member of:
        (1) CN=Schema Admins
        (2) CN=Enterprise Admins
        (3) CN=Domain Admins
        (4) CN=Group Policy Creator Owners

User: CN=Guest
        Does not belong to any LDAP groups.

User: CN=SUPPORT_388945a0
Member of:
        (1) CN=HelpServicesGroup
```

```
User: CN=IUSR_MALTSHOVEL
        Does not belong to any LDAP groups.

User: CN=IWAM_MALTSHOVEL
Member of:
        (1) CN=IIS_WPG

User: CN=ASPNET
        Does not belong to any LDAP groups.

User: CN=krbtgt
        Does not belong to any LDAP groups.

User: CN=John\, Smith
Member of:
        (1) CN=Domain Users
        (2) CN=Sales and Marketing

User: CN=Matt Ryall
Member of:
        (1) CN=Enterprise Admins
        (2) CN=Domain Admins

User: CN=Justin Koke
Member of:
        (1) CN=Domain Controllers
        (2) CN=Enterprise Admins

Found more than 10 results.

-----------------------------------------------------------------
TEST 2: Search and list 10 groups
-----------------------------------------------------------------

Group: CN=HelpServicesGroup
Members:
        (1) CN=SUPPORT_388945a0,CN=Users,DC=ad,DC=atlassian,DC=com

Group: CN=TelnetClients
        No members in this group.

Group: CN=IIS_WPG
Members:
        (1) CN=S-1-5-20,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com
        (2) CN=S-1-5-6,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com
        (3) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com
        (4) CN=IWAM_MALTSHOVEL,CN=Users,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005SQLBrowserUser$MALTSHOVEL
Members:
        (1) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005MSSQLServerADHelperUser$MALTSHOVEL
Members:
        (1) CN=S-1-5-20,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005SQLAgentUser$MALTSHOVEL$MSSQLSERVER
Members:
        (1) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005MSSQLUser$MALTSHOVEL$MSSQLSERVER
Members:
        (1) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005MSFTEUser$MALTSHOVEL$MSSQLSERVER
Members:
        (1) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005MSOLAPUser$MALTSHOVEL$MSSQLSERVER
Members:
        (1) CN=S-1-5-18,CN=ForeignSecurityPrincipals,DC=ad,DC=atlassian,DC=com

Group: CN=SQLServer2005NotificationServicesUser$MALTSHOVEL
        No members in this group.

Found more than 10 results.
```

Notes

Related Topics

Integrating Bamboo with LDAP

Managing permissions

Controlling access to build plans

You can use global permissions to control the users and groups that have access to build plans, and the actions they can perform.

Common global permissions tasks are:

- Granting plan permissions in bulk — control the users and groups that can perform actions on plans (e.g. edit, build, clone).
- Granting global permissions to users or groups — control the users and groups that can create plans, delete plans, and administer Bamboo.
- Allowing anonymous access to Bamboo — allow people not logged in to Bamboo to generate reports, and view plans and build results.

You can also change the permissions for an individual plan: see Configuring a plan's permissions.

Controlling access to the Bamboo server

Global security and permission properties allow a Bamboo system administrator to configure security- and permission-related properties that apply to Bamboo at a site-wide level.

Read more about configuring Bamboo's global security and permission properties:

- Allowing public signup
- Displaying full details about users
- Using Captcha for failed logins

Granting plan permissions in bulk

A *plan permission* is the ability to perform a particular operation on a plan and its jobs. For each plan, different permissions can be granted to particular groups and/or users.

- People who have the 'Admin' global permission can 'bulk edit' permissions for multiple plans at the same time, as described below. Note that this will overwrite any pre-existing plan permissions.
- People who have the 'Admin' plan permission for one or more plans, but do not have the 'Admin' global permission, can only edit one plan at a time, as described in Configuring a plan's permissions.

Note that it is recommended that you grant permissions to groups rather than to individual users.

To grant bulk plan permissions to a user or group:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. In the **Plans** section of left navigation panel, click **Bulk Edit Plan Permissions**.
3. Select the plans whose permissions you wish to edit, then click **Next** (at the bottom of the screen).
4. You can set plan permissions for the categories of users in the table below.
5. Select the check box for each permission that you wish to grant to the user or group.
6. Click **Save**.

Logged in Users
Users who are logged in to Bamboo.

Anonymous Users
Users who are not logged in to Bamboo.

User
A user already created in the Bamboo system.

To edit plan permissions for an existing user:

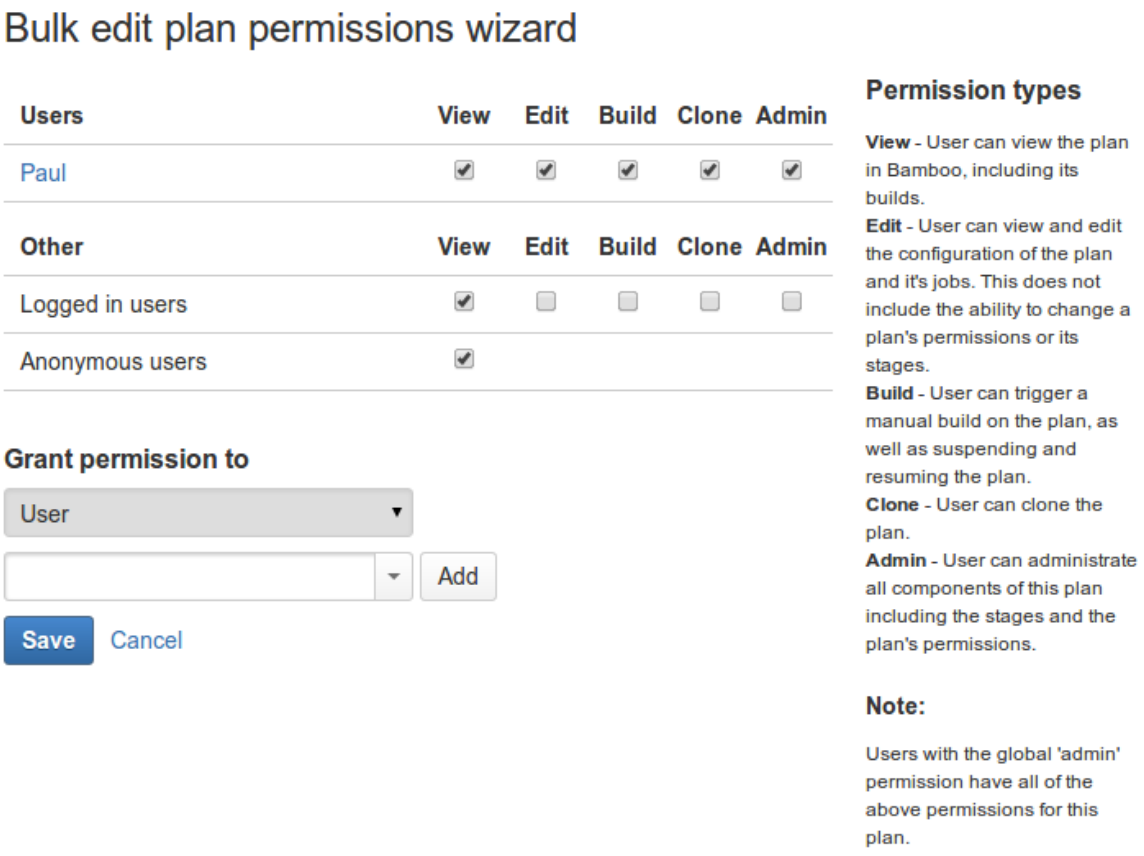
1. In the **Grant permission to** list, select **User**.
2. Type the username into the box, or click the icon to select from a list.
3. Click **Add**. The user will be added to the list on the screen, and you can then select permissions for them.

Group
A group already created in the Bamboo system.

To edit plan permissions for an existing group:

1. In the **Grant permission to** list, select **Group**.
2. Type the group name into the box.
3. Click **Add**. The group will be added to the list on the screen, and you can then select permissions for the group.

Screenshot: Bulk Edit Plan Permissions Wizard



Granting global permissions to users or groups

Global permissions control which users and groups have access to build plans and the Bamboo server, and what actions they can perform.

Note that if you remove *all* permissions for a user or group, that user or group will disappear from the **Permissions** tab for all plans.

Related pages:

- Configuring a plan's permissions
- Managing users

To change global permissions:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Global Permissions** in the left navigation panel, and then **Edit Global Permissions**.
3. You can set plan permissions for the categories of users in the table below.
4. Select (or clear) the check box for each permission that you wish to change for a user or group.
5. Click **Save**.

Logged in Users
Users who are logged in to Bamboo.

Anonymous Users
Users who are not logged in to Bamboo.

User
A user already created in the Bamboo system.

To edit plan permissions for an existing user:

1. In the **Grant permission to** list, select **User**.
2. Type the username into the box, or click the icon to select from a list.
3. Click **Add**. The user will be added to the list on the screen, and you can then select permissions for them.

Group
A group already created in the Bamboo system.

To edit plan permissions for an existing group:

1. In the **Grant permission to** list, select **Group**.
2. Type the group name into the box.
3. Click **Add**. The group will be added to the list on the screen, and you can then select permissions for the group.

You can grant the following global permissions:

Global permission	Description	Can be granted to

Access	Permission to view the Bamboo system. ⚠️ The ability to view build plans and build results is subject to individual plan permissions.	• a particular user • a particular group • all logged-in users • anonymous users
Create Plan	Permission to: • create new build plans • configure linked repositories	• a particular user • a particular group • all logged-in users
Create repository	Permission to: • create new repositories ⓘ Only users with **Admin** permissions on the *global* level can create repositories by default. *Plan* administrators can only select from linked repositories or must be granted the **Create repository** permission explicitly. See also: plan permissions.	• a particular user • a particular group
Admin	Permission to: • access the Bamboo Administration menu • create plans • delete plans • configure linked repositories The 'Admin' global permission also includes all plan permissions, for every plan.	• a particular user • a particular group

Screenshot: Global Permissions

Global permissions

You can edit your global application level permissions here. Permissions can be granted to specific users or groups. Please note these are global application permissions. For plan level permissions, please go to the plan configuration page.

Groups	Access	Create plan	Create repository	Admin
bamboo-admin	✓	✓	✓	✓

Other	Access	Create plan	Create repository	Admin
All logged in users	✓	✗	✗	
Anonymous users	✓			

Edit

Allowing anonymous access to Bamboo

Allowing *anonymous users* to access your Bamboo system means that people who aren't logged in to Bamboo will be able to perform functions such as generating reports, and viewing plans and build results — subject to individual plan permissions.

Note that people who aren't logged in to Bamboo do not have a 'My Bamboo' tab on their Dashboard.

To allow anonymous users to access Bamboo:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Global permissions** (under 'Security')
3. Check in the **Access** column for 'Anonymous users'.

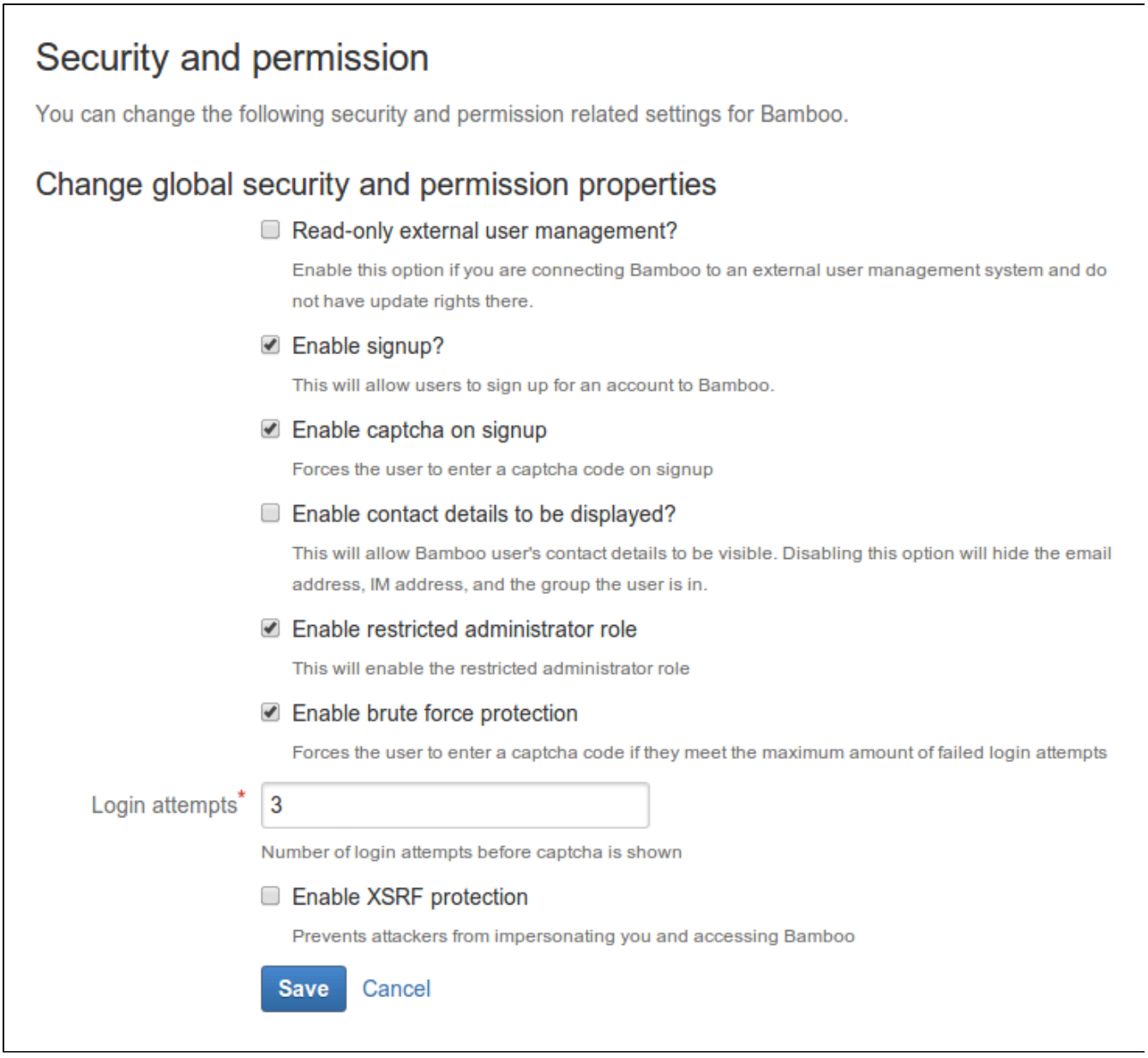
Anonymous users will now be able to access your Bamboo system. However, they will only be able to view plans and build results for plans where the 'Access' plan permission has been granted to 'Anonymous users'.

Allowing public signup

If you enable *signup* for your Bamboo system, visitors can create their own Bamboo user accounts. Public signup is enabled on your Bamboo site if you see the 'Signup' link at the top-right of the Bamboo user interface.

To enable (or disable) signup:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Security Settings** (under 'Security') in the left navigation panel to open the 'Global Security and Permission Properties' page.
3. Click **Edit** on this page.
4. Select, (or clear) the **Enable Signup?** check box.
5. Select **Enable Captcha On Signup** if you require an additional security measure to prevent brute force attacks.
6. Click **Save**.
7. Log out of Bamboo and verify that the top menu bar now contains (or does not contain) a **Signup** link.



Displaying full details about users

If you enable the display of contact details on your Bamboo system, the full contact details for all users, including email address, IM address, and group membership, will be visible to any visitors to Bamboo. The email addresses of administrators on the 'Contact Administrators' page will also be visible.

To enable (or disable) the display of contact details:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Security Settings** (under 'Security') in the left navigation panel to open the 'Global Security and Permission Properties' page.
3. Click **Edit** on this page.
4. Select (or clear) the **Enable contact details to be displayed?** check box.
5. Click **Save**.

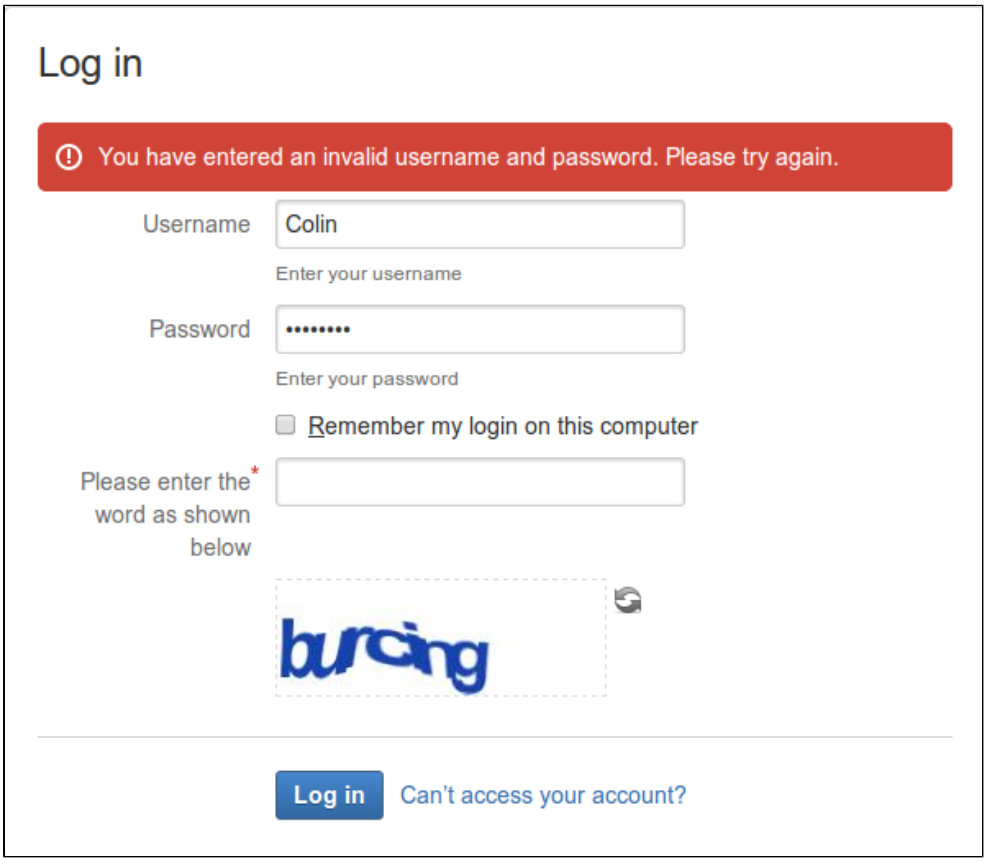
Using Captcha for failed logins

Captcha is a tool that prevents brute force attacks on the Bamboo login screen. A brute force attack occurs when an attacker uses malicious code to make automated, repeated login attempts on a Bamboo site with the aim of gaining access to that Bamboo site.

A Bamboo system administrator can configure Bamboo to block automated login attempts. Once a certain number of failed login attempts has been reached (the default is three) Bamboo's Captcha feature will be activated. When Captcha is activated, users will need to recognize a distorted picture of a word and must type the word into a text field. This is easy for humans to do, but very difficult for computers.

To enable (or disable) Captcha for Bamboo:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Security Settings** (under 'Security') in the left navigation panel to open the 'Global Security and Permission Properties' page.
3. Click **Edit** on this page.
4. Select (or clear) the **Enable Captcha** check box.
5. If required, specify the number of failed login attempts permitted by Bamboo before Captcha is activated. (This field is mandatory and requires a value of 1 or more.)
6. Click **Save**.



Managing authors

An *author* is any person who contributes to a build by checking-in code to a repository that is associated with a Bamboo plan. Bamboo extracts the author name from the code repository; an author need not be a Bamboo user.

Bamboo allows you to associate an author with a user. Association is with either the username or email address, and can be automatically or manually configured. This is useful for identifying who has made a particular commit, providing system notifications and apportioning blame. Author association also allows a user to quickly identify their commits on the MyBamboo tab.

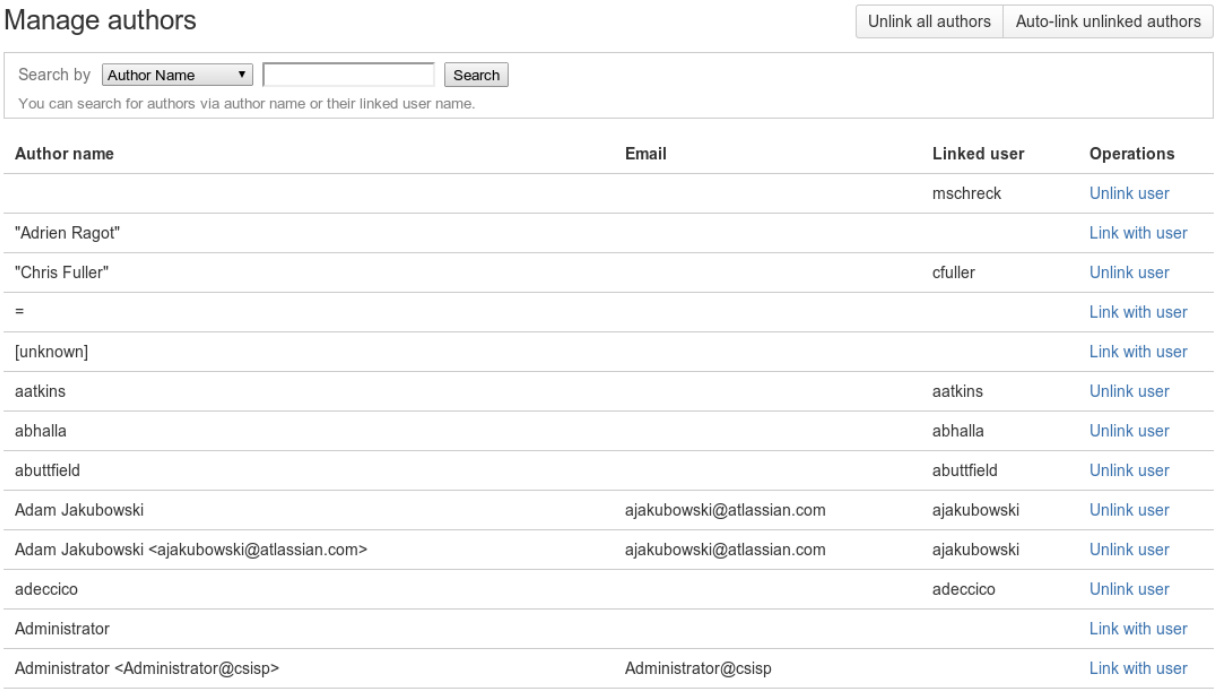
On this page:

Related pages:

- Managing permissions
- Associating your author name with your user profile

To manage Bamboo authors

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Select Authors from the side panel. the Manage Authors screen will display:



Author name	Email	Linked user	Operations
		mschreck	Unlink user
"Adrien Ragot"			Link with user
"Chris Fuller"		cfuller	Unlink user
=			Link with user
[unknown]			Link with user
aatkins		aatkins	Unlink user
abhalla		abhalla	Unlink user
abuttfield		abuttfield	Unlink user
Adam Jakubowski	ajakubowski@atlassian.com	ajakubowski	Unlink user
Adam Jakubowski <ajakubowski@atlassian.com>	ajakubowski@atlassian.com	ajakubowski	Unlink user
adeccico		adeccico	Unlink user
Administrator			Link with user
Administrator <Administrator@csisp>	Administrator@csisp		Link with user

3. On the 'Manage Authors' page you can perform any of the following actions:

   Search for author
   Search for a particular author using their repository author name

   Search for user
   Search for a particular author by their linked user name to see their author association

Link user
Link an author with their Bamboo user

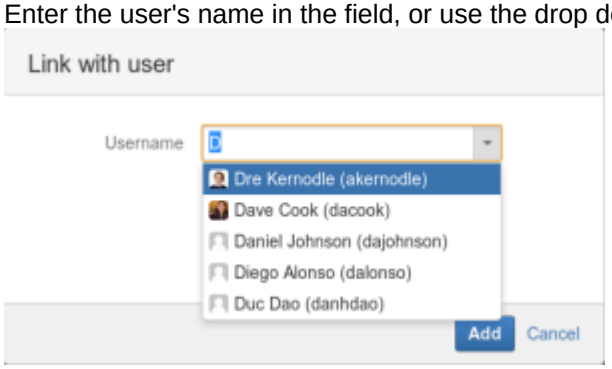
Unlink user
Unlink an author from their Bamboo user

Unlink all authors
Remove all existing author and user associations

Auto-link unlinked authors
Automatically associate any unlinked authors with a Bamboo user based on their Bamboo username or Email address

To associate an author with a user

1. From the Manage Authors screen, use the search tool to locate the author in question
2. Select the unlinked author and click **Link with user**
3. Enter the user's name in the field, or use the drop down menu to select a user:



4. Click **Add**

Note: You can link more than one author name to a Bamboo user name.

To disassociate an author with a user

1. From the Manage Authors screen, use the search tool to locate the author or username in question
2. Click **Unlink user**

Connect Bamboo to an external database

Bamboo can be connected to an external database. For details and instructions please see:

- Connect Bamboo to a PostgreSQL database
- Connect Bamboo to a MySQL database
  - Tomcat and external MySQL datasource example
- Connect Bamboo to an Oracle database
- Connect Bamboo to a Microsoft SQL Server database
  - Transition from jTDS to the Microsoft JDBC driver
- View database connection details
- Move data to a different database

- Troubleshooting Databases

Connect Bamboo to a PostgreSQL database

This page describes how to connect Bamboo to a PostgreSQL database.

Note that the JDBC driver for PostgreSQL is bundled with Bamboo. You do not have to download and install the driver.

See Supported platforms for other information about the versions of PostgreSQL supported by Bamboo.

On this page:

Related pages:

- Troubleshooting Databases

1. Configuring PostgreSQL

Accept remote TCP connections (remote PostgreSQL server only)

If you are connecting Bamboo to a remote PostgreSQL server (i.e. if your PostgreSQL server is not installed locally on your Bamboo server host system), you will need to configure your `data/postgresql.conf` and `data/pg_hba.conf` files to accept remote TCP connections from your Bamboo server's IP address.

The following PostgreSQL documentation contains information on the appropriate `listen_addresses` value in the `postgresql.conf` file as well as the `pg_hba.conf` file:

- PostgreSQL 8.2 documentation — Connections and Authentication

Once you have modified your `data/postgresql.conf` and `data/pg_hba.conf` files, you will need to restart PostgreSQL for your changes to take effect.

Creating a Bamboo database

```
sudo -s -H -u postgres
# Create the Bamboo user:
/opt/PostgreSQL/8.3/bin/createuser -S -d -r -P -E bamboouser
# Create the bamboo database:
/opt/PostgreSQL/8.3/bin/createdb -O bamboouser bamboo
exit
```

⚠ Creating a completely empty Bamboo database is recommended. Avoid using templates to create the database as some may insert default tables which can lead to conflicts when setting up Bamboo.

2. Connecting Bamboo to PostgreSQL

Bamboo provides two ways to connect to a PostgreSQL database — using JDBC or using a datasource. JDBC is generally simpler and is the recommended method.

Run the Setup wizard

For both methods, run the Setup Wizard and choose the **Custom Installation** option.

On the 'Choose a Database Configuration' page, choose **External Database**, select **PostgreSQL 8.2 and above** from the list and click **Continue**.

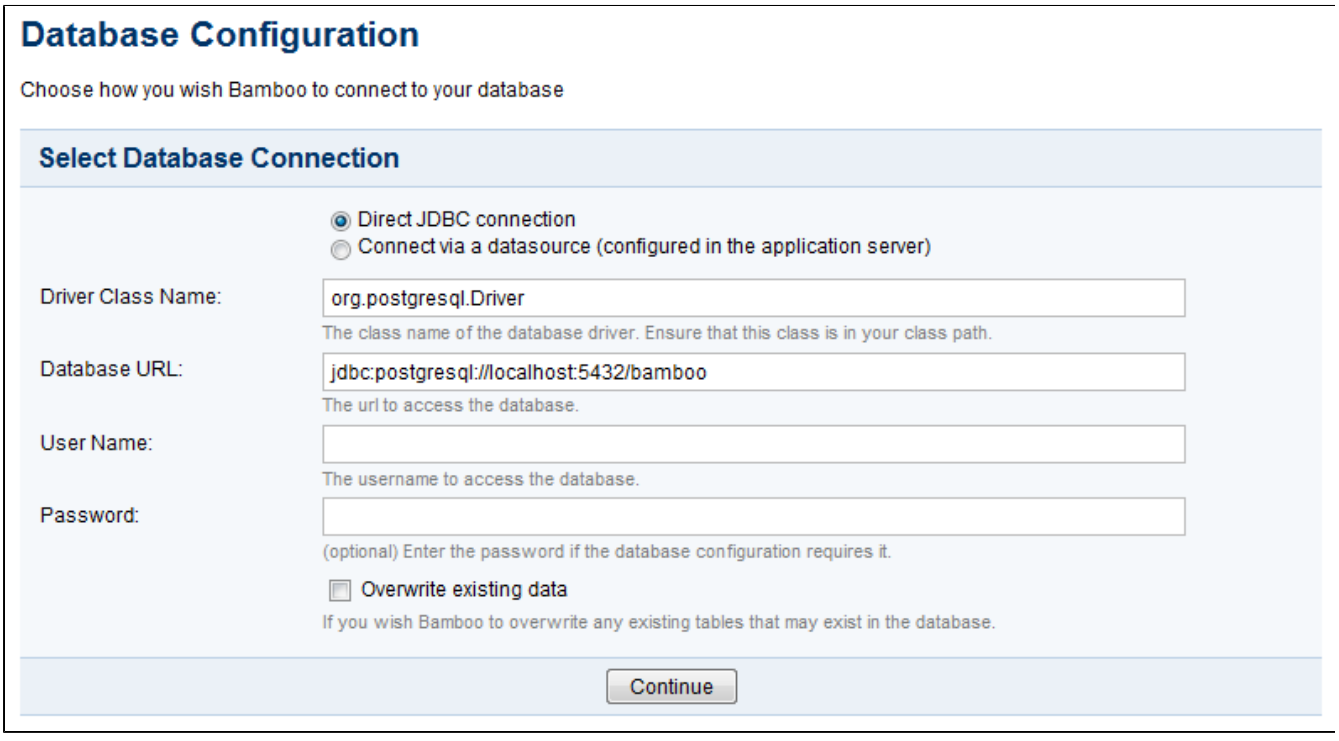
Choose one of the following:

Connecting using JBDC

On the 'Database Configuration' page of the Setup Wizard, ensure that **Direct JDBC connection** has been selected and make the following settings:

Setting	Description
Driver Class Name	Type `org.postgresql.Driver` (if different from the default).
Driver Class Name	Type the URL where Bamboo will access your database (if different from the default). For details about syntax, please refer to the Postgres JDBC driver documentation.
User Name	Type the username that Bamboo will use to access your database.
Password	Type the password (if required) that Bamboo will use to access your database.
Overwrite existing data	Select if you wish Bamboo to overwrite any tables that already exist in the database.

Screenshot 1: Setup JDBC Connection (PostgreSQL)



Connecting with a datasource

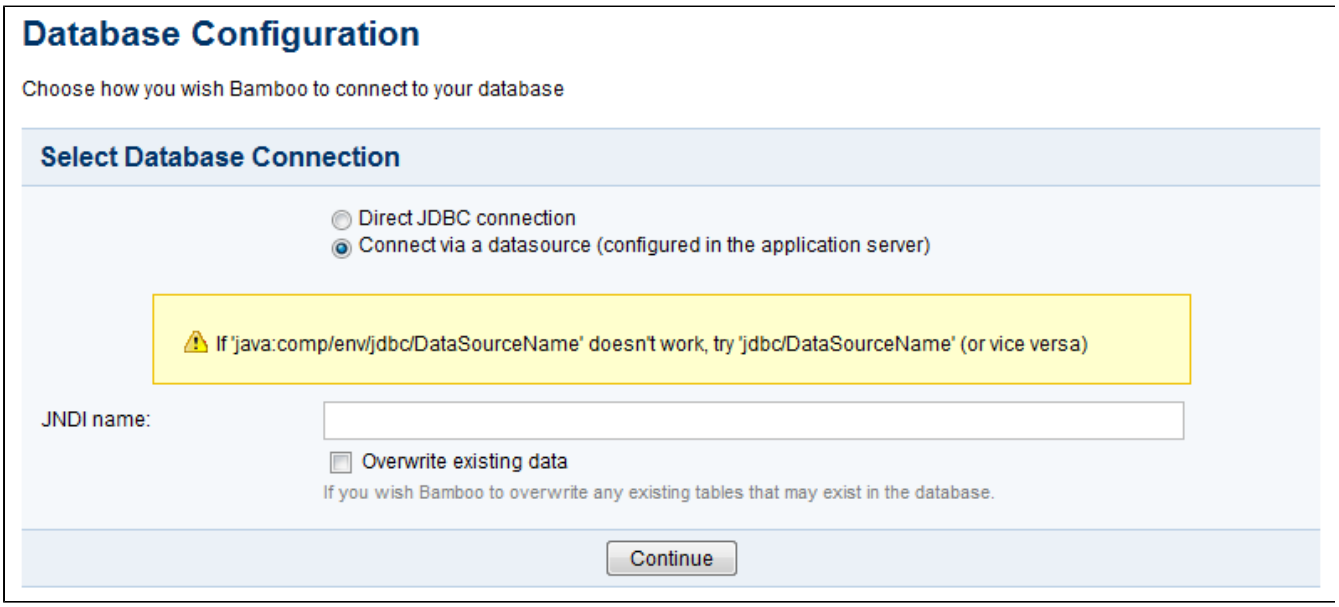
Configure a datasource in your application server (consult your application server documentation for details).

For details about the syntax to use for the JDBC database URL, please see the Postgres JDBC driver documentation.

On the 'Database Configuration' page of the Setup Wizard, choose **Connect via a datasource (configured in the application server)** and make the following settings:

Setting	Description
JNDI name	Type the JNDI name of your datasource, as configured in your application server. ⚠ If `java:comp/env/jdbc/DataSourceName` does not work, try `jdbc/DataSourceName` (and vice versa).
Overwrite existing data	Select if you wish Bamboo to overwrite any tables that already exist in the database.

Screenshot 2: Setup Datasource Connection



Connect Bamboo to a MySQL database

This page describes how to connect Bamboo to a MySQL database.

On this page:

Related pages:

Troubleshooting Databases

The JDBC driver for MySQL 5.1 (JDBC Connector/J 5.1) is **no longer** bundled with Bamboo. You **must** download and install the driver yourself.

See Supported platforms for other information about the versions of MySQL supported by Bamboo.

1. Creating and Configuring the MySQL database

For your external MySQL database to work well with Bamboo, it must be able to use the following:

- `utf8` or `utf8mb4` character set encoding instead of `latin1`
- `utf8_bin` or `utf8mb4_bin` collation
- the InnoDB storage engine
- (recommended, not required) lower_case_table_names=1

    Setting lower_case_table_names=1 might break other Atlassian applications. For more information, see the steps.

- global transaction isolation level as READ_COMMITTED
- Disable NO_AUTO_VALUE_ON_ZERO mode.

We also recommend that your MySQL database server is configured to use an InnoDB storage engine. Alternatively, you can configure Bamboo's JDBC connection to your MySQL database so that any tables that Bamboo creates in this database will be done using the InnoDB database engine.

A MySQL database administrator can easily create and configure a MySQL database for Bamboo by running the following MySQL commands:

```
mysql> CREATE DATABASE bamboo CHARACTER SET utf8 COLLATE utf8_bin;
mysql> GRANT ALL PRIVILEGES ON bamboo.* TO 'bamboouser'@'localhost' IDENTIFIED BY 'password';
mysql> FLUSH PRIVILEGES;
mysql> QUIT
```

```
mysql> CREATE DATABASE bamboo CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;
mysql> GRANT ALL PRIVILEGES ON bamboo.* TO 'bamboouser'@'localhost' IDENTIFIED BY 'password';
mysql> FLUSH PRIVILEGES;
mysql> QUIT
```

`bamboouser` — the user account name for the Bamboo MySQL database. This creates an empty MySQL database for Bamboo named `bamboo`, where:

- `localhost` — the host name of the MySQL database server

- `password` — the password for this user account

For more information about configuring character set encoding and collation for Bamboo MySQL databases, refer to the MySQL 5 documentation — Specifying Character Sets and Collations.

To verify if the NO_AUTO_VALUE_ON_ZERO mode is disabled, run this query on your MySQL server:

```
SELECT @@SQL_MODE, @@GLOBAL.SQL_MODE;
```

The first mode is the mode for the session, the second is the global settings for MySQL. If the global mode contains NO_AUTO_VALUE_ON_ZERO, it has to be removed as explained in the MySQL documentation.

2. Connecting Bamboo to the MySQL database

You can connect Bamboo to the MySQL in one of the following ways:

- using JDBC
- using a datasource

JDBC is generally simpler and we recommend using it.

Connect using JDBC

1. Download and install the JDBC driver

The JDBC drivers for MySQL Enterprise Server are *no longer* bundled with Bamboo (due to licensing restrictions). You need to download and install the driver yourself.

1. Download the MySQL Connector/J JDBC driver 5.1 or 8 from the download site.

   In MySQL 8, the `com.mysql.jdbc.Driver` class implementing `java.sql.Driver` in MySQL Connector/J was deprecated. Instead, you should use the `com.mysql.cj.jdbc.Driver` class.

2. Extract the downloaded zip/tar.gz file.
3. Copy the mysql-connector-java-XX.XX-bin.jar file from the extracted directory to the `<Bamboo installation directory>/lib` directory (create the `lib/` directory if it doesn't already exist). If you are using the Java Service Wrapper to start your Bamboo instance (`Bamboo/wrapper/run-bamboo start`), copy the mysql-connector-java-XX.XX-bin.jar file to `<Bamboo installation directory> /wrapper/lib directory.`
4. Stop Bamboo on Windows, Linux, or Mac.
5. Restart Bamboo on Windows, Linux, or Mac.

2. Connect Bamboo to a MySQL database using JDBC

1. Run the Setup Wizard.
2. On the 'Configure database' page, choose **MySQL** from the 'Database Type' dropdown menu, and then select **Continue**.
3. Ensure that **Direct JDBC connection** is selected and complete the following fields (as shown in the screenshot below):

   Driver Class Name
   Type:

   - com.mysql.cj.jdbc.Driver - for MySQL Connector/J 8.0 or later
   - com.mysql.jdbc.Driver - for MySQL Connector/J 5.1

Database URL
Type the URL where Bamboo will access your database (if different from the default). Your URL *must* include the `autoReconnect=true` flag.

- If you intend to use non-Latin characters in Bamboo, ensure that your URL includes the `useUnicode=true` and `characterEncoding=utf8` flags.
- If your MySQL database server is configured to use a storage engine other than InnoDB by default, ensure that your URL includes the `sessionVariables=storage_engine=InnoDB` flag.
  If you include all of these flags, your **Database URL** should look similar to:

```
jdbc:mysql://localhost/bamboo?
autoReconnect=true&useUnicode=true&characterEncoding=utf8&sessionVariables=storage_engine=InnoDB
```

> ⚠ If the `autoReconnect=true` flag is not specified, the MySQL JDBC driver will eventually time out and Bamboo will no longer be able to communicate with the database.

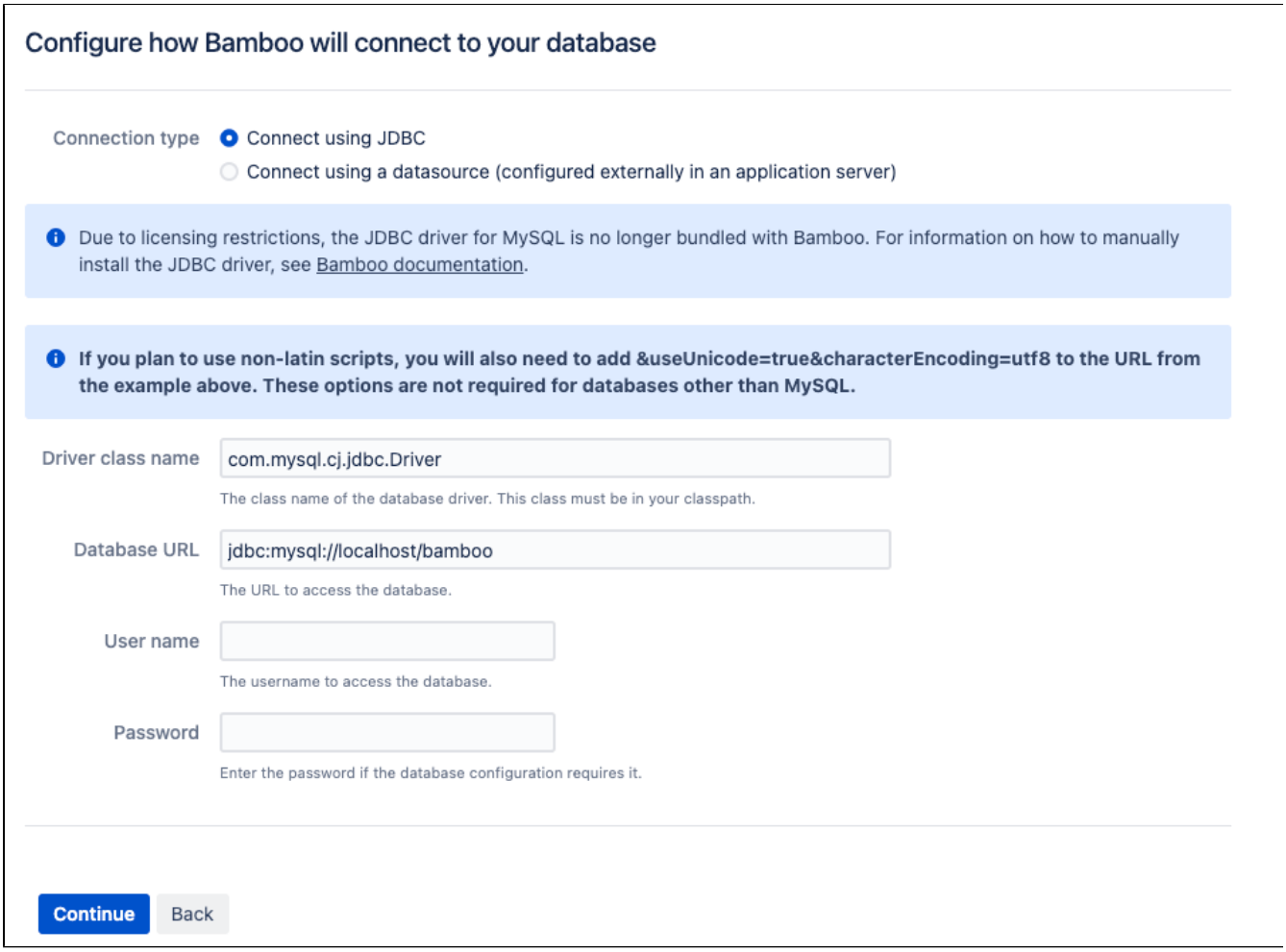
  For more information on the URL syntax, see the MySQL documentation.
- If you use MySQL Connector 8.0 with MySQL database 5.7, AO tables will not be created and Bamboo will not be able to start. To prevent this, include the `nullCatalogMeansCurrent=true` flag in your Database URL.
  Your Database URL should look similar to:

```
jdbc:mysql://instenv-2787-y7fe.c7uydxwwuprf.eu-west-1.rds.amazonaws.com:3306/bamboo?
autoReconnect=true&useUnicode=true&characterEncoding=utf8&nullCatalogMeansCurrent=true
```

User Name
Type the username that Bamboo will use to access your database. This is `bamboouser` defined in section 1 (above).

Password
Type the password (if required) that Bamboo will use to access your database. This is `password` defined in section 1 (above). Leave this field blank if a password for the database user account was not specified.

4. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
5. Select **Continue**.



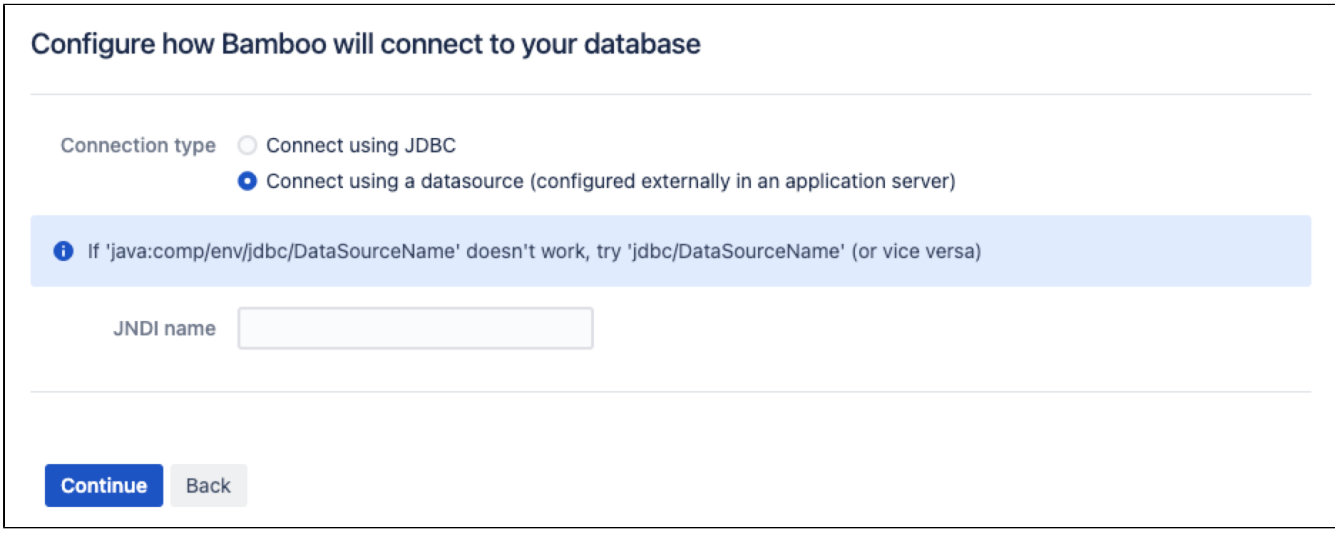
Connect using a datasource

1. Configure a datasource in your application server (consult your application server documentation for details). Please note the following:
    - Ensure that the JDBC URL which you configure in your application server includes the `autoReconnect=true`, `useUnicode=true` and `characterEncoding=utf8` flags, such that your database URL should look similar to: `jdbc:mysql://localhost/bamboo?autoReconnect=true&useUnicode=true&characterEncoding=utf8`
    - If your MySQL database server is configured to use a storage engine other than InnoDB by default, also include the `sessionVariables=storage_engine=InnoDB` flag in this URL.
    - *If the autoReconnect flag is not set, the MySQL JDBC driver will eventually time out and Bamboo will no longer be able to communicate with the database.*
      For more information on the URL syntax, see the MySQL documentation.
    - Datasource example: You can see an example of using Tomcat with a MySQL database as a datasource in the following document: Tomcat and external MySQL datasource example.
2. Run the Setup Wizard and choose the **Custom Installation** method.
3. Choose **External Database** > **MySQL** from the list and click **Continue**.
4. Choose **Connect via a datasource (configured in the application server)** (as shown in the screenshot below).
5. In the **JNDI name** field, type the JNDI name of your datasource, as configured in your application server.

    ⚠ *If `java:comp/env/jdbc/DataSourceName` does not work, try `jdbc/DataSourceName` (and vice versa).*

6. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
7. Click **Continue**.

Screenshot 2: Setup Datasource Connection



Tomcat and external MySQL datasource example

Add the DataSource Resource tag inside the Context tags of your context descriptor in the `server.xml` file located under <bamboo-installation-directory>/conf:

```
<Context .... >
<Resource name="jdbc/bamboo" auth="Container" type="javax.sql.DataSource"
        username="yourusername"
         password="yourpassword"
         driverClassName="com.mysql.jdbc.Driver"
         url="jdbc:mysql://localhost:3306/bamboo?autoReconnect=true"
         maxActive="100"
         maxIdle="7"
         validationQuery="Select 1" />
</Context>
```

Connect Bamboo to an Oracle database

This page describes how to connect Bamboo to an Oracle database.

Bamboo provides two ways to connect to an Oracle database: using JDBC or using a datasource. We recommend to use JDBC because this method is simpler.

See Supported platforms for other information about the versions of Oracle supported by Bamboo.

On this page:

Important

- For JDBC or JNDI connections, please ensure that the user connecting to the database will have total permissions. This includes the **DBMS_LOB package** and other resources available.

Related pages:

Troubleshooting Databases

Install Oracle

If you don't already have an operational Oracle server, download and install it now. See the Oracle documentation for instructions.

When setting up your Oracle server:

- **Character encoding** must be set to AL32UTF8 (this is the Oracle equivalent of Unicode UTF-8).
- **Collation** should be set to `BINARY`.

Configuring Oracle

1. Ensure that you have a database instance available for Bamboo (either create a new one or use an existing one).
2. Within that database instance, create a user whom Bamboo will connect (for example: **`bamboo-user`**). Remember this database user name, as it will be used to configure Bamboo's connection to this database.

   When you create a user in Oracle, Oracle will create a 'schema' automatically.

   ```
   create user bamboo-user identified by password;
   ```

3. Ensure that the user has the following permissions:

   ```
   grant connect, resource, create table to bamboo-user;
   ```

Connecting using JDBC

The JDBC drivers for Oracle Server are *no longer* bundled with Bamboo due to licensing restrictions. You need to download and install the driver yourself.

Download and install the JDBC driver

To download and install the JDBC driver:

1. Take a backup of the ojdbc8.jar file from the following location: `<Bamboo installation directory> /lib`
2. Download the Oracle JDBC driver ojdbc8.jar from the Oracle download site.
3. Copy the ojdbc8.jar file from the extracted directory to the `{{ <Bamboo installation directory> /lib }}directory`
   If the `lib/` directory doesn't exist, you can create it.

   If you are using the Java Service Wrapper, to start your Bamboo instance (`Bamboo/wrapper/run-bamboo start`), copy the ojdbc8.jar file to the `<Bamboo installation directory> /wrapper /lib` directory.
4. Stop Bamboo on Windows, Linux, or Mac.
5. Restart Bamboo on Windows, Linux, or Mac.

Connect Bamboo to an Oracle database using JDBC

To connect Bamboo to an Oracle database using JDBC:

1. Run the Setup Wizard and choose the **Custom Installation** method.
2. At the 'Choose a Database Configuration' step, choose **External Database** > **Oracle**.
3. Select **Direct JDBC connection** and complete the form:

   Driver Class Name
   Type: `oracle.jdbc.driver.OracleDriver`

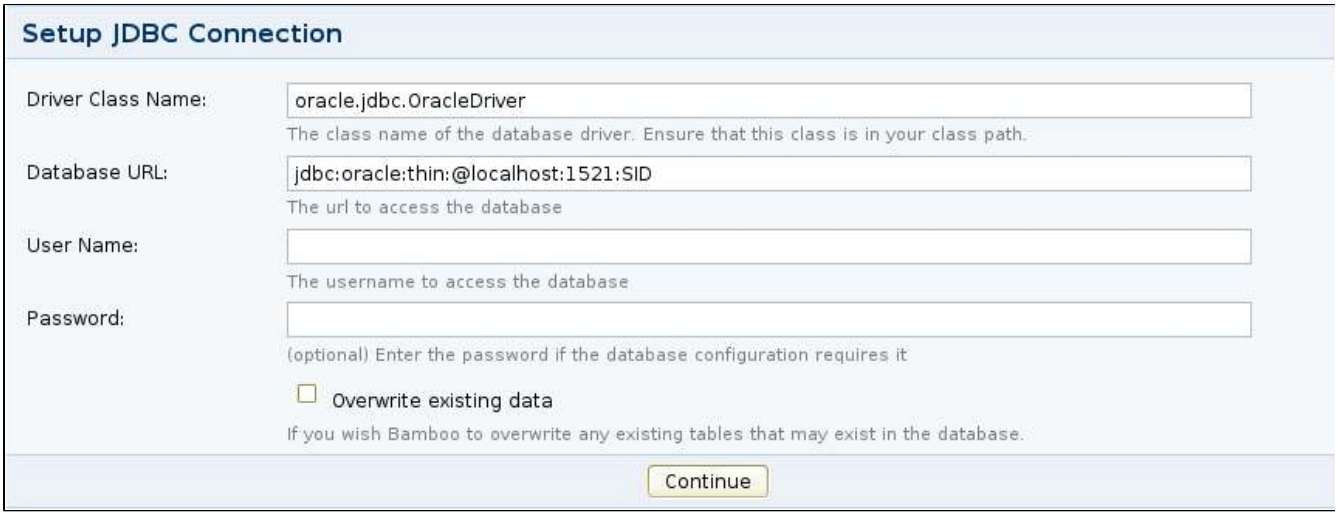
   Database URL
   Type the URL where Bamboo will access your database, e.g. `jdbc:oracle:thin:@localhost:1521:SID`. For syntax, please see the Oracle documentation.

   Username
   Type the username that Bamboo will use to access your database.

   Password
   Type the password that Bamboo will use to access your database.

4. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
5. Select **Continue**.

Screenshot: Setup JDBC Connection (Oracle)



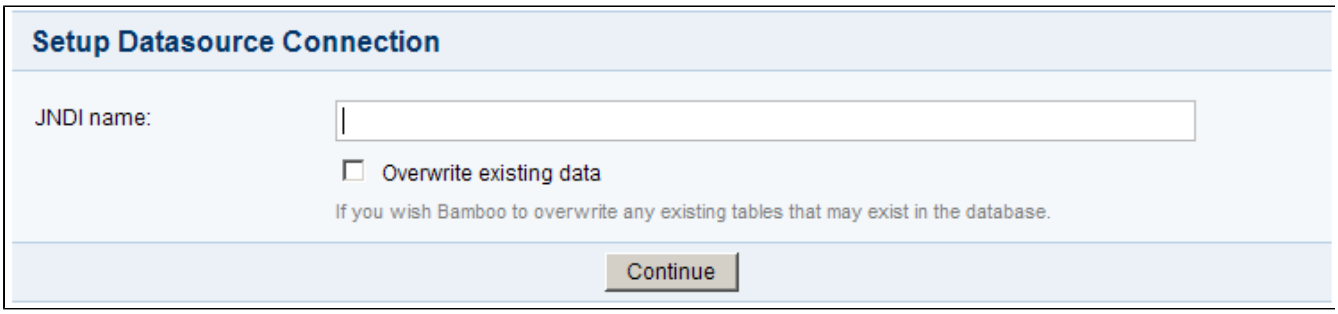
Connecting using a datasource

1. Configure a datasource in your application server (consult your application server documentation for details). For the syntax of the JDBC URL to use, please see the Oracle documentation.
2. Run the Setup Wizard and choose the **Custom Installation** method.
3. At the 'Choose a Database Configuration' step, choose **External Database** > **Oracle**.
4. Select **Connect using a datasource (configured in the application server)**.
5. In the **JNDI name** field, type the JNDI name of your datasource, as configured in your application server.

   ⚠ If `java:comp/env/jdbc/DataSourceName` doesn't work, try `jdbc/DataSourceName` (and vice versa).

6. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
7. Select **Continue**.

Screenshot Setup Datasource Connection



Connect Bamboo to a Microsoft SQL Server database

This page describes how to connect Bamboo to a Microsoft SQL Server database.

See Supported platforms for other information about the versions of SQL Server supported by Bamboo.

Note that the JDBC driver for SQL Server is bundled with Bamboo. You *do not* have to download and install the driver.

On this page:

Related pages:

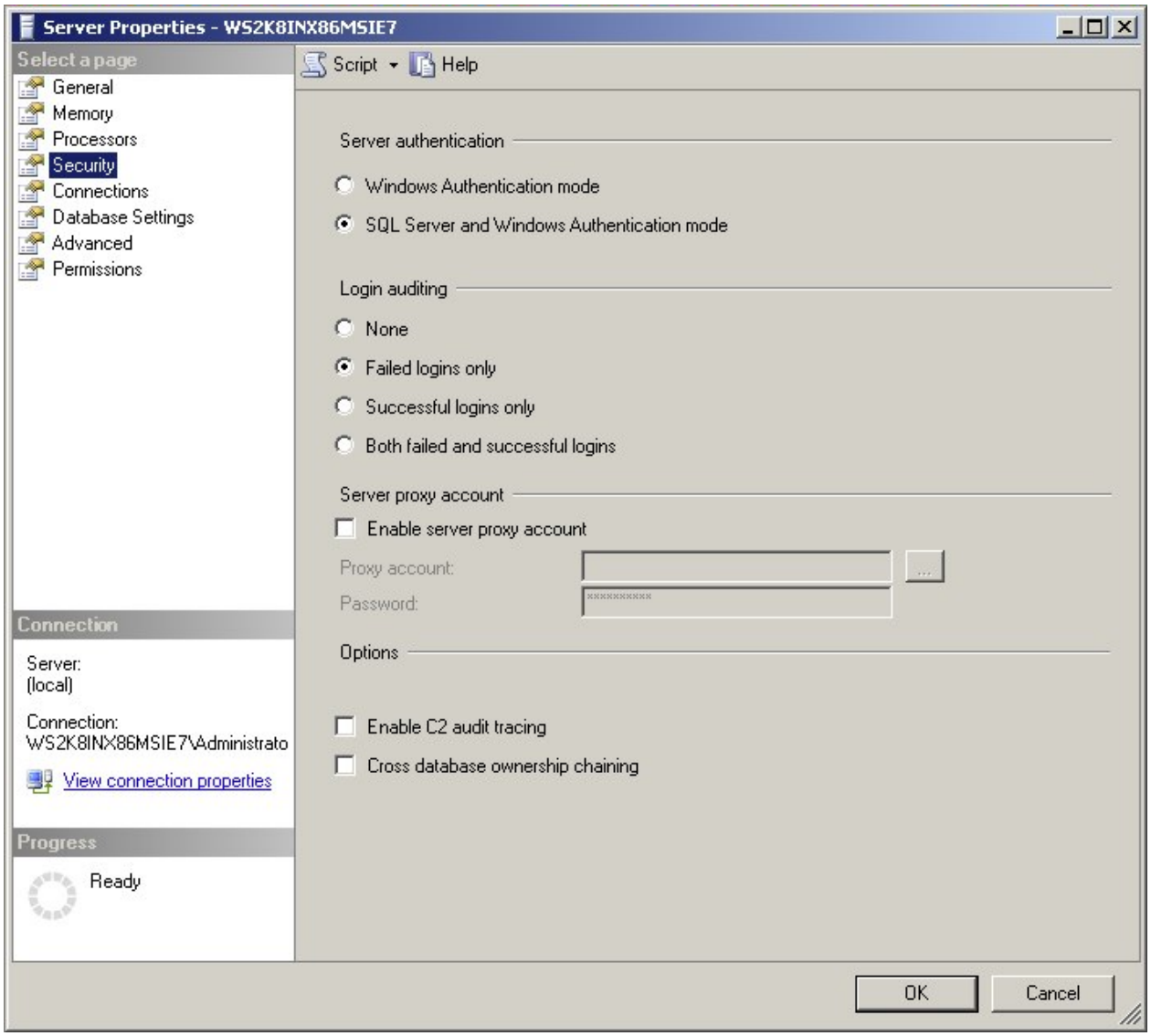
- Installing and upgrading
- Connect Bamboo to an external database
- Troubleshooting Databases

1. Configuring SQL Server

Before you connect Bamboo to a SQL Server, you need to configure SQL Server appropriately.

- **Change server authentication to 'SQL Server and Windows Authentication mode'** — On a typical SQL Server installation, **Windows Authentication mode** is the default security mode. However, if you try to connect to the database with a database user using this authentication mode, SQL Server will throw an error. You need to change the server authentication mode to **SQL Server and Windows Authentication mode** in SQL Server before you can connect Bamboo to SQL Server. See this MSDN article for instructions on how to do this.

Screenshot: Changing the SQL Server authentication mode



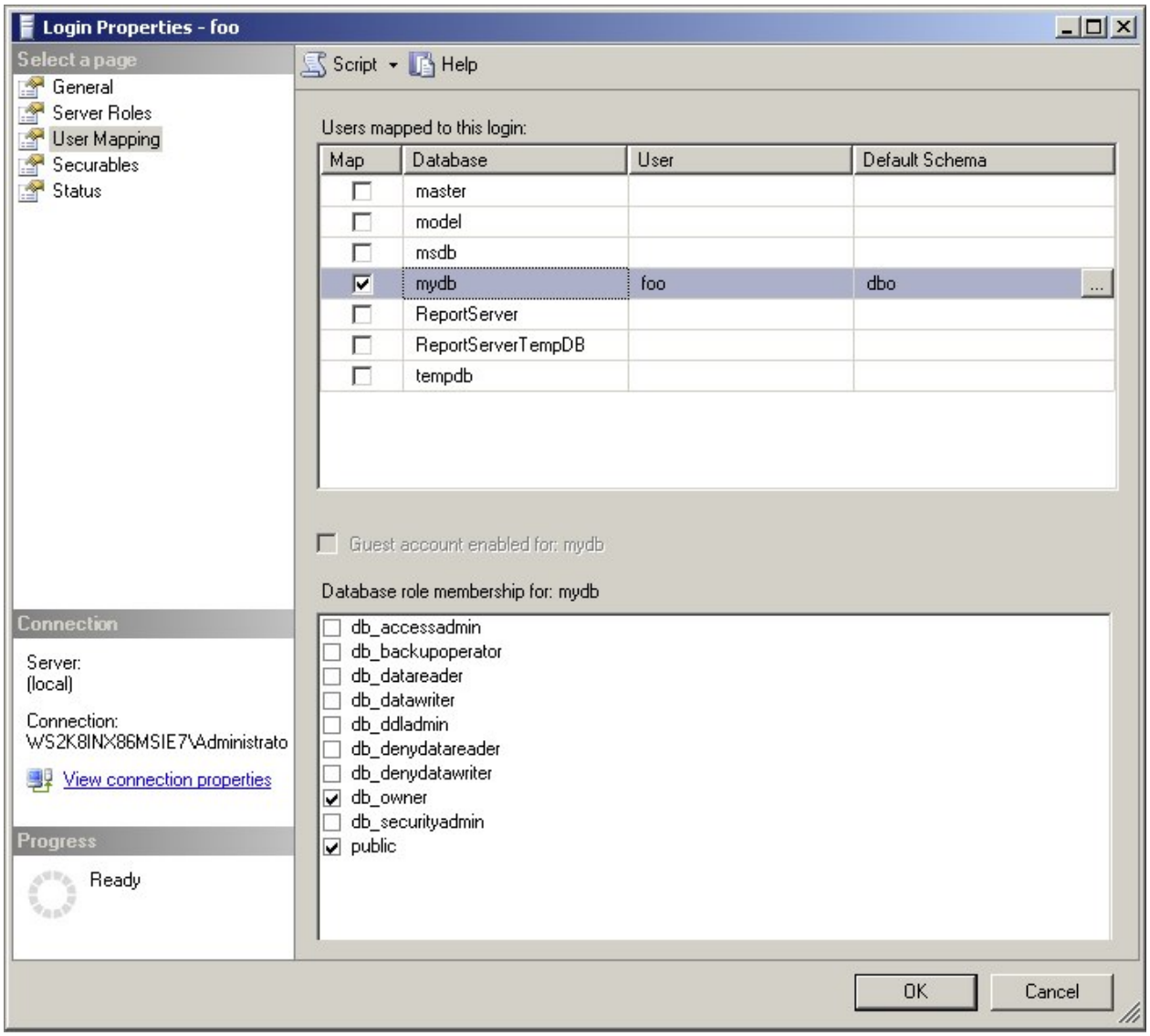
- **Configure your firewall to allow SQL Server access** — If you need to access SQL server through a firewall, you will need to configure your firewall appropriately. The following MSDN article describes how to configure a Windows firewall to allow SQL Server access, however the instructions are applicable to other firewalls: Configuring the Windows Firewall to Allow SQL Server Access.
- **Enable the TCP/IP protocol for your database instance** — You must enable the TCP/IP protocol for your SQL Server database instance by following the instructions in this MSDN article.

2. Creating your database

After configuring the SQL Server, you need to create the SQL database.

- **Create the database for Bamboo** — see this MSDN article for instructions.
- **Assign the 'db-owner' role on the database for the user that will access the Bamboo database** — the **'db_owner'** fixed database role allows the user to perform all configuration and maintenance activities on the database. You need to add this role to the Bamboo user used to access your database by updating the login properties for your database user in SQL Server. Read more about login properties for SQL Server.

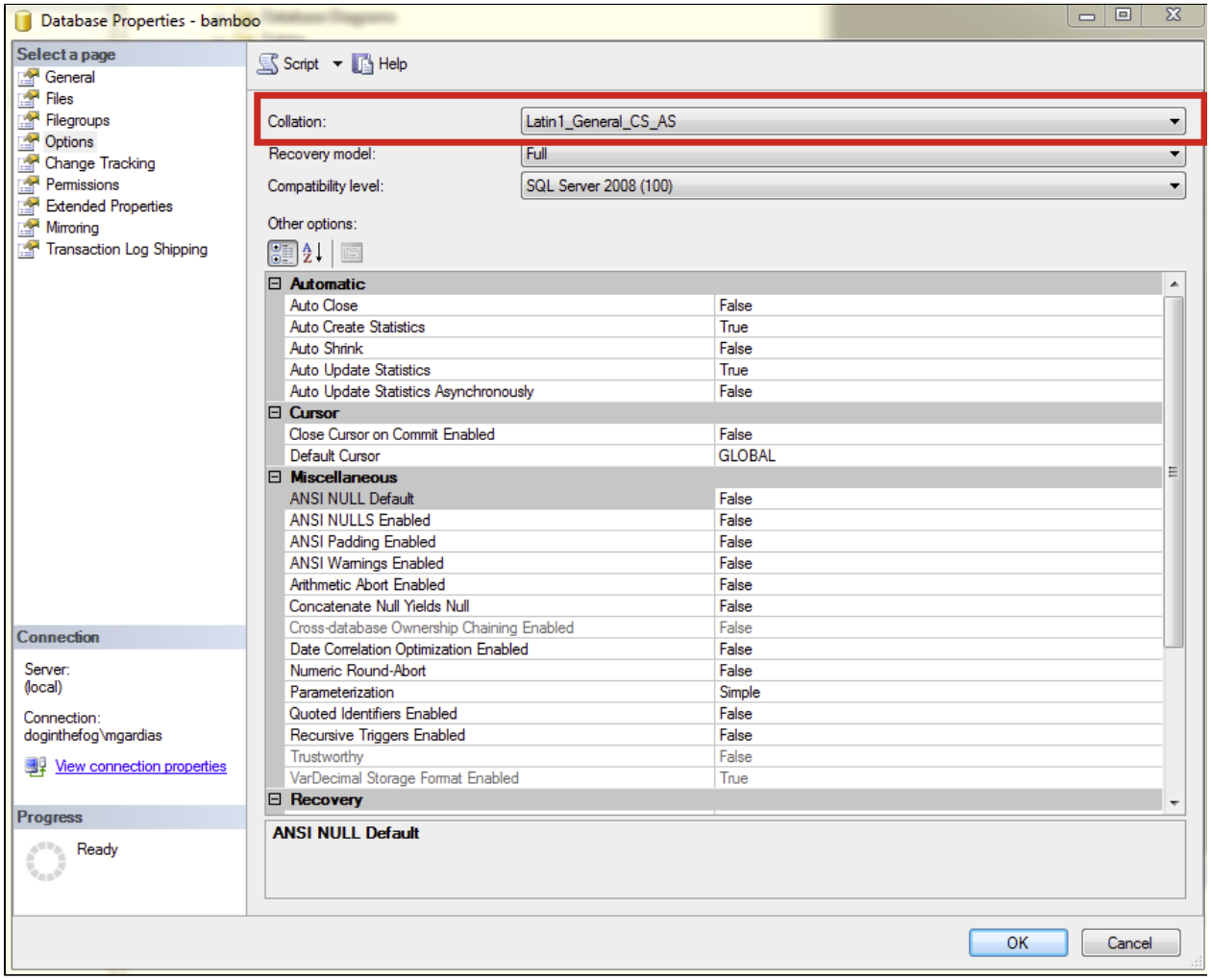
Screenshot: Adding the 'db_owner' database role to a database user in SQL Server



⚠ *Please ensure that you use a **SQL Server** user account to log into your database, not a Windows user account.*

- **Configure the database to use case-sensitive collation** — to make the SQL Server database respect case differences in the data it stores (which is required for Bamboo), ensure that you configure it using a case-sensitive collation option such as 'Latin1_General_CS_AS'. To access this feature in SQL Server Management Studio, right-click on the database name, select **Properties** from the resulting menu, then select the **Options** page.

Screenshot: Configuring the Bamboo database to use 'Latin1_General_CS_AS' collation



- **Configure the database to use the correct isolation level**— Ensure that the new database was set to use Read Committed with Row Versioning as its isolation level. You can apply the new isolation by executing the following query:

```
ALTER DATABASE <database name>
   SET READ_COMMITTED_SNAPSHOT ON
   WITH ROLLBACK IMMEDIATE;
```

To verify the changes, use this query which should result in '1':

```
SELECT sd.is_read_committed_snapshot_on
FROM sys.databases AS sd
WHERE sd.[name] = '<database name>';
```

3. Connecting Bamboo to SQL Server

Bamboo provides two ways to connect to a Microsoft SQL Server database — using JDBC or using a datasource. JDBC is generally simpler and is the recommended method.

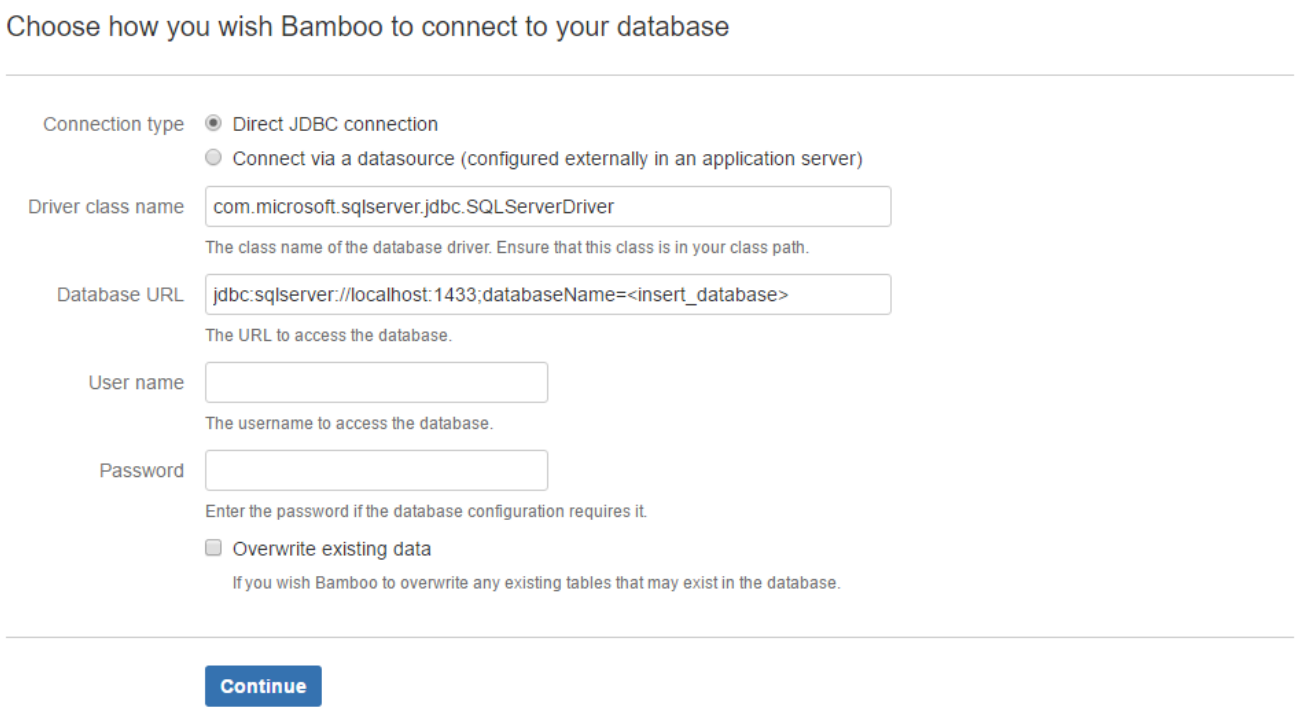
Connect to SQL Server using JBDC

1. Run the Setup Wizard and choose the **Custom Installation** method.
2. On the **Choose a Database Configuration** page, choose **External Database** > **Microsoft SQL Server** a nd click **Continue**.
3. Ensure that **Direct JDBC connection** has been selected and complete the following fields (as shown in the screenshot below):

Setting	Description
Driver Class Name	Type `com.microsoft.sqlserver.jdbc.SQLServerDriver` (if different from the default)
Database URL	The URL where Bamboo will access your database, e.g. `jdbc:sqlserver://localhost:1433;databaseName=bamboo` If you are connecting to a **Named Instance**, you will need to append *;instance=mssqlnamehere* to the connection string, where *mysqlnamehere* is the name of your named instance. For more details about syntax, please refer to the Microsoft SQL Server documentation.
Username	The username that Bamboo will use to access your database.
Password	The password that Bamboo will use to access your database.

4. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
5. Click **Continue**.

Screenshot: Set up JDBC connection

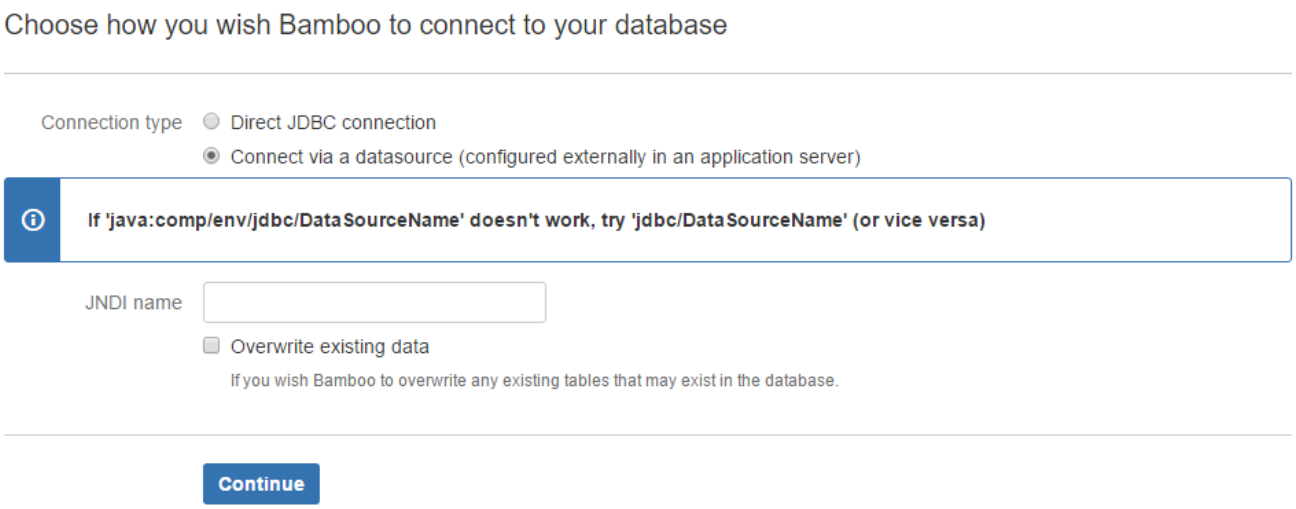


Connect to SQL Server using a datasource

1. Configure a datasource in your application server (consult your application server documentation for details).
   For details about the syntax to use for the SQL Server database URL, please refer to the Microsoft SQL Server documentation.
2. Run the Setup Wizard and choose the **Custom Installation** method.
3. On the 'Choose a Database Configuration' page, choose **External Database** > **Microsoft SQL Server** and click **Continue**.
4. Choose **Connect via a datasource (configured in the application server)**, as shown in the screenshot below.

5. In the **JNDI name** field, type the JNDI name of your datasource, as configured in your application server.
   ⚠️ If `java:comp/env/jdbc/DataSourceName` does not work, try `jdbc/DataSourceName` (and vice versa).
6. Select **Overwrite existing data** if you wish Bamboo to overwrite any tables that already exist in the database.
7. Click **Continue**.

Screenshot: Set up Datasource connection



Transition from jTDS to the Microsoft JDBC driver

This page describes how to change from using jTDS to using the Microsoft SQL Server JDBC driver to access Microsoft SQL Server.

What do I have to do?

Bamboo will try to automatically migrate the database configuration during upgrade. If that fails, the system will lock on startup. To resolve this, you need to manually update the driver class and URL.

How to proceed

In the Bamboo server home directory, `bamboo.cfg.xml` must be edited to change the JDBC driver and URL. The existing configuration should look similar to this:

```
<property name="hibernate.connection.driver_class">net.sourceforge.jtds.jdbc.Driver</property>
<property name="hibernate.connection.password">PASSWORD</property>
<property name="hibernate.connection.url">jdbc:jtds:sqlserver://127.0.0.1:1433/Bamboo</property>
<property name="hibernate.connection.username">bamboo_user</property>
<property name="hibernate.dialect">com.atlassian.bamboo.hibernate.SQLServerIntlDialect</property>
```

⚠ The JDBC URL above is in the format constructed by Bamboo when Connecting to SQL Server and will automatically be updated to a URL compatible with Microsoft's driver, with no change required on the administrator's part. If the URL contains additional properties, such as domain=, it will need to be manually updated.

To use Microsoft's SQL Server driver, the settings above would be updated to this:

```
<property name="hibernate.connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
<property name="hibernate.connection.password">your_password</property>
<property name="hibernate.connection.url">jdbc:sqlserver://localhost:1433;databaseName=bamboo</property>
<property name="hibernate.connection.username">username</property>
<property name="hibernate.dialect">com.atlassian.bamboo.hibernate.SQLServerIntlDialect</property>
```

The exact values to use in the new URL are beyond the scope of this documentation; they must be chosen based on the jTDS settings they are replacing.

Additional Information for the curious

The new JDBC driver class is: `com.microsoft.sqlserver.jdbc.SQLServerDriver`

The JDBC URL format for the jTDS driver is documented on SourceForge at http://jtds.sourceforge.net/faq.html#urlFormat.

The JDBC URL format for Microsoft's SQL Server driver is documented on MSDN at http://msdn.microsoft.com/en-us/library/ms378428.aspx, with documentation for additional properties at http://msdn.microsoft.com/en-us/library/ms378988.aspx.

Why change drivers?
Recent releases of Hibernate, which Bamboo uses to simplify its persistence layer, have introduced a requirement that the JDBC drivers and connection pools used be JDBC4-compliant. JDBC4 was introduced with Java 6.

The jTDS driver used by releases prior to Bamboo Server 6.0 is a JDBC3 driver, compatible with Java 1.3, and therefore cannot be used with newer versions of Hibernate. While jTDS 1.3.0 and 1.3.1 *claim* to implement JDBC4, and JDBC4.1, they actually don't. The new methods have been "implemented", but their implementations are all `throw new AbstractMethodError()`, which means they can't actually be *used*. (See an example here, on GitHub.)

Since jTDS 1.3.1 does not provide a functioning JDBC4 implementation, the decision was made to replace jTDS with Microsoft's own SQL Server driver. Microsoft's driver is actively maintained, where jTDS hasn't been updated since 2014 (and prior to the small round of updates done in 2014 it hadn't been updated for multiple years). Microsoft offers a full JDBC4.2 (Java 8) driver and supports all the features of SQL Server, including SQL Server 2016.

Bamboo attempts to automatically update jTDS JDBC URLs to values compatible with Microsoft's JDBC driver. However, for installations using custom JDBC URLs–for example, to use domain authentication–such automatic updating is not possible; the URL, which was manually entered, must be manually updated.

View database connection details

When you installed Bamboo, you would have set up a database connection by following one of these processes:

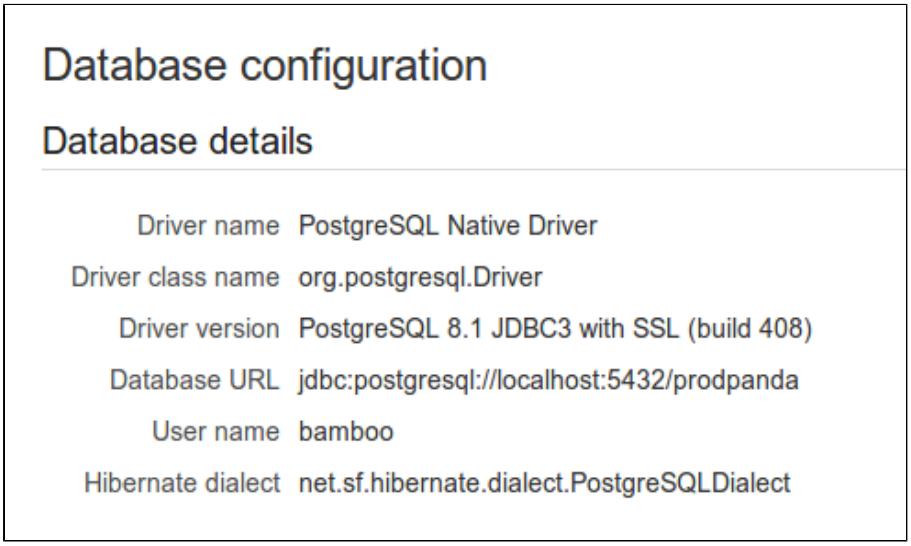
Once Bamboo is running, you can view the database configuration details as follows.

Related pages:

- Data and backups

To view your database connection details:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Database Configuration** in the left navigation column, under 'System'.



Move data to a different database

You can move data to a different database by installing a new Bamboo instance and updating the settings. Alternatively, if the database systems are the same or compatible, you can move the data manually.

In the initial Bamboo configuration, the database can be set to:

- an internal H2 database (not recommended for production environments) OR
- an external database.

Related pages:

- Data and backups

To move your Bamboo data to a different database:

> ⚠ The data import part of the process can take many hours for large instances. Please test the process to ensure Production outage windows are well-known for planned migration activities.

1. Export the data of the original Bamboo instance as described in Exporting data for backup.
2. Stop the original Bamboo instance.

   > ⓘ You may have to disable automatic Bamboo start if the instance was configured to run as, for example, a Windows service.

3. Install a new Bamboo instance as described in Installing and upgrading Bamboo.

   > ⓘ **Important**
   >
   > If you are installing a Bamboo instance on the same server, make sure that the new Bamboo instance doesn't have the same
   >
   > `<bamboo-install>` or `<bamboo-home>` paths as the original Bamboo instance. Using the same paths may result in data loss. For more information, see Locating important directories and files.

   > ⓘ The import process performed in later steps later can be very memory intensive depending on the size of your import. New installations of Bamboo will come with default Java heap allocation configurations which may not be sufficient to perform the import at an appropriate speed.
   >
   > If your Bamboo instance has been in use for long while before migrating and has become quite large, please be sure to tune / increase your Bamboo Java heap allocation before continuing: Configuring your system properties

4. Start the new Bamboo instance.
5. In the Setup Wizard:
   a. Make sure that the new **Configuration Directory**, **Build Data Directory** and **Build Working Directory** are not located in the same place as the original Bamboo instance directories.
   b. Select a new database:
      - Connect Bamboo to a PostgreSQL database
      - Connect Bamboo to a MySQL database
        - Tomcat and external MySQL datasource example
      - Connect Bamboo to an Oracle database
      - Connect Bamboo to a Microsoft SQL Server database
        - Transition from jTDS to the Microsoft JDBC driver
      - View database connection details
      - Move data to a different database

    c. Select **Import existing data** and specify the path to the file that you exported at the beginning of the procedure.
6. Once the data is ready, restart the new Bamboo instance.
7. Reindex the data as described in Reindexing data.
8. Verify that your build results and system settings are correct.

Alternative DB migration

If the database systems are:

- the same (for example, you are moving from PostgreSQL to another PostgreSQL) OR
- compatible (for example, you are moving from SQL Server 2005 to SQL Server 2008),

you can move the data manually. To migrate the data:

1. Stop the Bamboo instance that is using the source database.
2. Manually transfer the data.
3. Go to <bamboo-home> and open the `bamboo.cfg.xml` file.
4. Provide the properties of the new database.
5. Start the Bamboo instance.

Apps

An app is an installable component that supplements or enhances the functionality of Bamboo in some way. For example, the Jira Bamboo Plugin is an app that integrates Jira and Bamboo. Other apps are available for integrating Bamboo into the Visual Studio IDE, running arbitrary commands before or after builds, and accessing Atlassian support from the Bamboo interface.

Bamboo comes with many pre-installed apps called system apps. You can install more apps, either by acquiring them from the Atlassian Marketplace or by uploading them from your file system. This means that you can install apps that you have developed yourself. For information about developing your own apps for Bamboo, see the Bamboo Developer documentation.

On this page:

About the Universal Plugin Manager (UPM)

You administer apps for Bamboo using the Universal Plugin Manager (UPM). The UPM is itself an app that exposes app administration pages in the Bamboo Administration Console. UPM works across Atlassian applications, providing a consistent interface for administering apps in Bamboo, Jira, Confluence, Crucible, Fisheye or Bitbucket.

UPM comes pre-installed in recent versions of all Atlassian applications, so you do not normally need to install it yourself. However, like other apps, the UPM software is subject to regular software updates. For that reason before administering apps in Bamboo you should verify your version of the UPM and update it if needed.

Administering apps in Bamboo

You can update UPM, or any app, from the UPM's own app administration pages. Additionally, you can perform these tasks from the UPM administration pages:

- Install or remove apps
- Configure apps settings
- Discover and install new apps from the Atlassian Marketplace
- Enable or disable apps and their component modules

For information on performing these app administration tasks, see the Universal Plugin Manager documentation.

For app information specific to Bamboo, see these pages:

- Apps blacklist
- Enabling Clover for Bamboo

Apps blacklist

Outdated apps may break certain functionality in Bamboo. If Bamboo detects the presence of a non-working app, it will print a warning to its logs during startup and ask you to refer to this page.

For more information about why Bamboo printed a particular warning, please refer to a section below that is relevant to the app in question.

Experimental Bamboo Git Plugin

Since version 3.0, Bamboo is distributed with a fully supported version of the Bamboo Git Plugin.

The experimental Bamboo Git Plugin that was available before Bamboo 3.0 (and was not distributed with Bamboo) does not work with Bamboo 3.0 and later.

If you were using the experimental Bamboo Git Plugin, please remove the app from your Bamboo installation, and manually reconfigure each plan that was using it to use the Bamboo Git Plugin that is distributed with Bamboo.

Hung Build Killer

Starting from version 6.4, Bamboo is shipped with a built-in mechanism for monitoring builds. As a result, the Hung build killer becomes deprecated. If you're using the Hung build killer plugin and you want to upgrade to Bamboo 6.4 or higher, the Hung build killer is going to be disabled after the upgrade.

Enabling Clover for Bamboo

This page describes how to enable and configure Atlassian Clover or OpenClover app for a job in Bamboo.

When Bamboo is integrated with Clover, you can:

- View code-coverage details (i.e. the percentage of code covered by tests) for each build result
- View code-coverage trends for a job over a period of time
- View the code-coverage summary for the job.

Atlassian Blogs:

- Aggregated code coverage using Maven, Clover and Bamboo

As Clover integration (automatic and manual) produces instrumented classes, we recommend that you ensure that your job does not install them to production (for instance: 'mvn deploy' to public repository, 'scp' to an application server running on production, etc ...). Having instrumented code in such locations is usually not desired.

Common practices to ensure proper separation of instrumented and non-instrumented classes are:

- create a dedicated plan or job with Clover integration enabled
- enable automatic Clover integration for jobs running tests only (e.g. *"mvn verify"*)
- use different location of local artifact cache if you need to install artifacts (e.g. ~/.m2/repository-clover and *"mvn install"*)
- use different URL for uploading artifacts if necessary (e.g. a separate repository for *"mvn deploy"*)

Enabling automatic code coverage integration

Automatic integration works with Ant, Maven and Grails tasks.

1. Go to your job. See Configuring jobs.
2. Click the **Other** tab.
3. In the 'Would you like to view code coverage for this plan?' setting, check **Collect code coverage data for this job**.
4. Select **Automatically integrate a code coverage tool into this job**.
5. In the **Code coverage tool** option, select **Atlassian Clover** (you will need to provide a Clover license in *Administration > Manage Apps > Clover for Bamboo*) or **OpenClover** (no license is required).
6. Click **Save**

By default, HTML and XML reports are generated. Additionally, you can:

- Select **Generate a historical report** to compare the current coverage results with previous code coverage reports.
- Select **Generate a JSON report** to get the results in a format ready for embedding into applications or external report views.

When automatic code coverage integration is enabled, Bamboo:

- Creates an artifact named **Clover Report (System),** which is visible on the 'Artifacts' tab for the job.

and during every build:

- Extracts the Clover license (if set in the administration panel) into a temporary file and passes it to:
  - an Ant task as *-Dclover.license.path=/<bamboo-home>/xml-data/build-dir/<your-job>/.clover /clover.license*
  - a Maven task as *-Dmaven.clover.licenseLocation=/<bamboo-home>/xml-data/build-dir/<your-job>/.clover/clover.license*
- Enhances tasks by adding
  - Ant - targets like "*with.clover*", "*clover.report*"
  - Maven - goals like *"clover2:setup", "clover2:aggregate", "clover2:clover", "clover2:save-history";* it also adds *"verify"* phase if original command does not call *"compile"* or later phase
  - Grails - options like *"-clover.on"*
- Generates XML and HTML reports
- Generates statistics and charts for a plan summary

In order to protect you against publishing instrumented code, automatic Clover integration **will not** run if the Maven task runs the *"install"* or *"deploy"* phases. In such case, you will find no Clover report and a build log will contain an appropriate warning message. In order to get coverage reports for such job, either edit the Maven task to run the build till the "verify" phase (or earlier) or configure Clover manually. An alternative is to add `-Dmaven.clover.repositoryPollutionProtection=false` property to your Maven task.

Enabling manual code coverage integration

Manual code coverage integration works with any kind of task in which Clover can be called (Ant, Maven, Command, Grails). Use it when you already have Atlassian Clover or OpenClover integration configured to generate a report in your build scripts:

1. Go to your job. See Configuring jobs.
2. Click the **Other** tab.
3. In the 'Would you like to view code coverage for this plan?' settings, check **Collect code coverage data for this job**.
4. Select **I already integrated a code coverage tool in this job**.
5. In **Coverage report**, specify where Bamboo shall look for the XML report file generated. Specify the file path *relative to your plan's root directory*, e.g.:

   ```
   target/site/clover/clover.xml
   ```

6. Click **Save**.
7. In the **Artifacts** tab, click **Create artifact** and complete the form as follows

   Name
   This should begin with with "Clover Report".
   Location
   This should point to the HTML report directory (e.g. target/site/clover)
   Copy Pattern
   Use **/*.*

8. Configure Atlassian Clover or OpenClover in your build script so that it generates both XML and HTML reports. See quick start guides how to do this:
   - Ant: for Atlassian Clover and for OpenClover
   - Maven: for Atlassian Clover and for OpenClover
9. For Atlassian Clover configure the license in your build script or pass it as a proper task parameter in the job configuration:

a. Save the license key in a file (for example in /opt/bamboo/clover.license).
b. Pass the location of the license key to the build task:
    - Define it in the build script, or
    - Pass it as a Java property for the Ant/Maven task in the plan configuration.

build.xml

```
<project>
  <property name="clover.license.path" location="/opt/bamboo/clover.license"/>
</project>
```

or

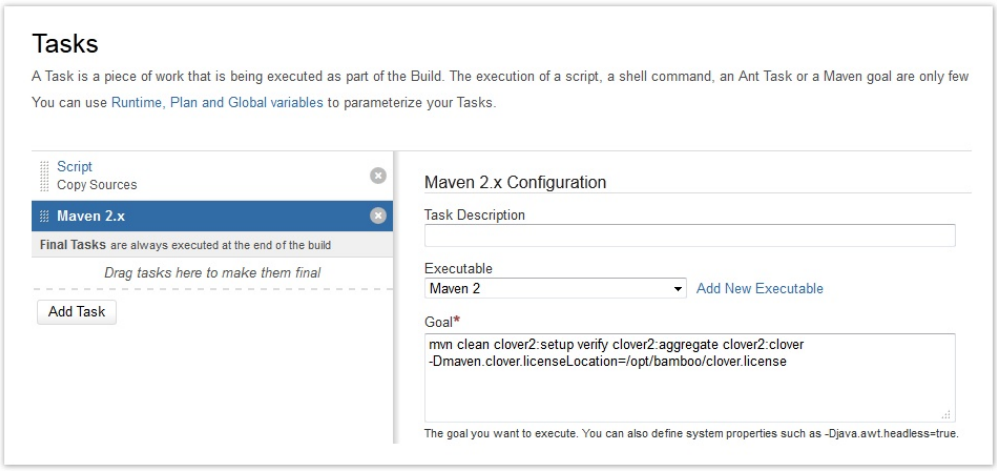
```
clean with.clover test clover.report -Dclover.license.path=/opt/bamboo/clover.license
```

pom.xml

```
<plugin>
    <groupId>com.atlassian.maven.plugins</groupId>
    <artifactId>maven-clover2-plugin</artifactId>
    <configuration>
        <licenseLocation>/opt/bamboo/clover.license</licenseLocation>
    </configuration>
</plugin>
```

or

```
mvn clean clover2:setup verify clover2:aggregate clover2:clover -Dmaven.clover.licenseLocation=/opt/bamboo
/clover.license
```



After every build, Bamboo will parse the Clover XML file and generate statistics and charts for a plan summary. The Plan summary and job summary pages will contain a **Clover** tab.

Browsing code coverage results

For more information on Clover HTML report and Clover statistics for a job, see Viewing the Clover code-coverage for a plan.

For more information on Clover code coverage summary for a plan, see Viewing the Clover code-coverage for a build.

For more information on Clover code coverage statistics across multiple plans, see Generating reports across multiple plans.

Limit the machines that Atlassian Clover runs on

If you have more remote agents than the number of machines for which Atlassian Clover is licensed, you can restrict the machines on which Clover runs by using capabilities:

1. For each of the EC2 images on which you would like to run builds with Atlassian Clover, add a capability such as `clover=true` to the configuration for the image.
   To do this, go to **Administration** > **Elastic Bamboo** > **Configuration**. Select the elastic image and click **Add Capability.**
2. Add a matching requirement, such as `clover=true` to the configuration for each job.
   To do this, go to **Actions** > **Configure Plan** > **Jobs**. Select the job where Clover runs and click **Require ments** and then **Add Extra Requirement**.

Troubleshooting

Using automatic Clover integration or adding a dependency to the *maven-clover2-plugin* manually is usually sufficient.

However, if your build spawns another JVM process (for example: unit tests executed in a forked JVM, tests in the container instantiated on the fly, tests calling code deployed on another server), you must manually add the dependency to the Clover JAR for these spawned processes.

See NoClassDefFoundError com_atlassian_clover/CoverageRecorder KB article.
In case you perform a build in a subdirectory (for instance, in the Maven Task configuration you have the *"Worki ng sub directory"* field set) and you have automatic Clover integration, you may need to correct the *Location* in the *"Clover Report (System)"* artifact. Otherwise, an HTML report may be empty as automatic Clover integration uses the default path (for instance, the *"target/site/clover"* in case of integration with Maven).

This issue has been fixed in Bamboo 5.7.
If you have a multi-module Maven project with dependencies between modules and use Automatic Clover integration, it can happen that an instrumented JAR of the dependent artifact will be taken for test execution in a build phase where Clover was not enabled yet. See BAM-13208 for more details. In such case, we recommend the following:

- create a separate Job in which automatic Clover integration is enabled
- create a Maven task in this job, which will do nothing (call the "clean" goal, for instance)
- Bamboo will automatically add Clover-related goals (clover2:setup verify clover2:aggregate clover2: clover)

This issue has been fixed in Bamboo 5.9.
In the build log you may see a warning like:

```
Failed to execute plugin 'Clover Results Collector' with error: No file matches
the specified pattern ...
```

The are several possible reasons, see this article for more details: Failed to execute plugin 'Clover Results Collector'.

Data and backups

For information on managing data and backups, see the following topics:

- Locating important directories and files
- Specifying Bamboo's working directory
- Reindexing data
- Specifying a backup schedule
- Exporting data for backup
- Importing data from backup
- Configuring global expiry
- Plan directory information REST API

Locating important directories and files

The information on this page describes how to find important Bamboo directories and files.

On this page:

Bamboo installation directory

When you installed Bamboo, you specified the location for the Bamboo installation directory, where Bamboo application files are stored. The default location depends on your operating system: Windows, Unix/Linux, or macOS.

The Bamboo installation directory contains the following files:

- `atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`
  This file tells Bamboo where to find the Bamboo home directory. The location of the containing directory is specified by the Bamboo administrator as described in the Bamboo installation guide. For more information, see Starting Bamboo.
- `bin/start-bamboo.sh`
  This is the startup file for Bamboo under Unix/Linux, Solaris, and Mac OS.
- `bin\start-bamboo.bat`
  This is the startup file for the Bamboo under Windows.
- `scripts/Triggers`
  This directory contains operational scripts used when configuring the repository to trigger a Bamboo build.
- `logs/*`
  This directory contains logs unless you have used the binary installer for Windows.

  Bamboo server logs are written to the root of the installation directory. Build logs are stored in the subdirectories under `<BAMBOO_HOME>/xml-data/builds` in Bamboo 7 and under `<SHARED_HOME>/builds` in Bamboo 8.

  For more information, see Build locations.

  If you used the binary installer for Windows, the log file will be located at `%USERPROFILE%\bamboo.log`.

  For Bamboo running as a Windows service, it's at `%WINDIR%\System32\Config\systemprofile\bamboo.log`.

- `atlassian-bamboo/WEB-INF/lib/`
  This directory is used when deploying Bamboo apps. It also contains other libraries required by Bamboo.
- `atlassian-bamboo/WEB-INF/classes/log4j2.properties`
  This is the Bamboo logging configuration file. You can also configure logging Bamboo UI by navigating to **> Log Settings** page, but the changes made on that page will be reverted by a Bamboo restart; however, changes made in the `log4j2.properties` file will persist even after a restart.

Bamboo local home directory

When you installed Bamboo, you specified the location for the Bamboo local home directory, where your Bamboo configuration data and build results are stored. The default location depends on your operating system: Windows, Unix/Linux, or macOS. This directory can grow quite large when managing large quantities of plans and builds.

Bamboo 8 introduced major changes to the structure of the home folder. The new organization is a requirement for using multiple nodes with Bamboo Data Center, but it affects every type of Bamboo installation, including Server and single-node Data Center setups. You can future-proof your tools by using the Plan directory information REST API.

Common locations and files in the local home directory

The Bamboo home directory contains the following common subdirectories and files:

- `bamboo.cfg.xml`
  This is Bamboo's core configuration file. It includes the configuration information for connecting to Bamboo's database.
- `database/`
  This directory contains Bamboo's embedded H2 database. The database contains plan configurations and some build results data. This directory is not present if an external database is used instead of the embedded H2.

  H2 is intended for evaluation purposes only and isn't recommended for production instances of Bamboo.

- `logs/*`
  This directory contains logs unless you have used the binary installer for Windows. The Bamboo server logs are written to the root of the installation directory. Build logs are stored in the subdirectories under `xml-data/builds/`.

  If you used the binary installer for Windows, the log file will be located at `%USERPROFILE%\bamboo.log`.

  For Bamboo running as a Windows service, it's at `%WINDIR%\System32\Config\systemprofile\bamboo.log`.

Build locations

The following table lists the build locations that you can find in the Bamboo local or shared home directories, depending on which version of Bamboo you are running. For a complete list of what has changed in the structure of the home directory, see Bamboo home migration.

		Bamboo 7.x	Bamboo 8.x
1	**Path**	`index/`	`<SHARED_HOME>/index/`
	Description	This directory contains the build results index. Removing or modifying files in this directory may corrupt build history. Rebuilding the search index from the global administration page (see Reindexing data) will completely regenerate the contents of this directory.	
2	**Path**	`artifacts/<PLAN_KEY>/shared/build-<BUILD_NUMBER>`	`<SHARED_HOME>/artifacts/<PLAN_KEY>/shared/build-<BUILD_NUMBER>`
	Description	This location is shared by all the stages of a plan. Stages will place artifacts here so that other stages from the same plan can have access to them. The `<BUILD_NUMBER>` always consists of a minimum of 5 digits with leading zeroes when necessary. For example, for build "42" the number will be "00042".	
3	**Path**	`xml-data/build-dir/<JOB_KEY>`	`<BAMBOO_HOME>/local-working-dir/<JOB_KEY>`

	Description	This location is known as the working directory. This is where Bamboo temporarily puts the checked-out files it's building. The path of this directory can be changed as described in Specifying Bamboo's Working Directory.	
4	**Path**	`xml-data/builds/`	`<SHARED_HOME>/builds`
	Description	This location is known as the build directory. This is where Bamboo stores build results, which are deleted as described in Configuring global expiry. Its contents can be backed up as described in Exporting data for backup.	
5	**Path**	`xml-data/builds/<JOB_KEY>/results`	`<SHARED_HOME>/builds/<JOB_KEY>/results`
	Description	Contains the build results for all the builds belonging to the plan identified by `<JOB_KEY>`. Each build result is an individual XML file. Don't edit these files or the corresponding information in the database may become corrupt.	
6	**Path**	`xml-data/builds/<JOB_KEY>/download-data`	`<SHARED_HOME>/builds/<JOB_KEY>/download-data`
	Description	Contains the logs for each build belonging to the plan identified by the `<JOB_KEY>`.	
7	**Path**	`xml-data/configuration`	`<SHARED_HOME>/configuration`
	Description	This is known as the configuration directory. It contains server-wide configuration data. Its contents can be backed up as described in Exporting data for backup.	

Bamboo agent home directory

When you installed your remote agents (if any), you specified the location for the Agent home directory, where the configuration data of agents is stored. The default name of this directory is `bamboo-agent-home`. This directory can grow quite large when managing significant numbers of plans and builds. The default `bamboo-agent-home` location depends on your operating system: Windows, Unix/Linux, or macOS.

The Bamboo agent home directory contains the following subdirectories and files:

- `bamboo-agent.cfg.xml`
  This contains configuration information about this remote agent. Most notably, it stores the agent id, which gets generated the first time this agent connects to the Bamboo server.
- `xml-data/build-dir/`
  This is where the agent will check out the files and perform builds (similar to the Bamboo server's `xml-data/build-dir/` directory)

Specifying Bamboo's working directory

The *Working Directory* is where Bamboo temporarily puts the checked-out files it is building. The location of this directory was specified using the Setup Wizard, can be viewed as described in Bamboo's system information, and can be changed as described below.

By default, this directory is located under the `xml-data` directory in the Bamboo home directory.

Each build's jobs have their own working directory relative to this configured *working directory:*

`xml-data/build-dir/JOB_KEY`

If **Concurrent Builds** are enabled, local agent builds will use the format:

xml-data/build-dir/AGENT_ID/JOB_KEY

To change the location of Bamboo's working directory:

1. Shut down Bamboo.
2. Open the <Bamboo-Home>/bamboo.cfg.xml file in a text editor. Find the following line -

```
....
    <property name="buildWorkingDir">/home/Bamboo-home/xml-data/build-dir</property>
....
```

3. Edit the Bamboo working directory to point to a new folder on disk.
4. **Save** the changes and restart Bamboo.
   Note: Bamboo will do a fresh checkout and perform a clean build of *all your plans*, once the directory is changed.

Reindexing data

About re-indexing

You will need to *re-index* your Bamboo build results data whenever you perform a data import. Re-indexing your data can also be helpful if your reports appear to be out-of-sync with your data. This may take a few minutes to complete (see System settings for an estimate of how long it will take).

Related pages:

- Data and backups

To re-index Bamboo's build results data:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Indexing** in the left navigation column, under 'System'.
3. Click **Perform a full reindex**.

Specifying a backup schedule

You can configure Bamboo to automatically create a backup each night, rather than doing a manual export every time.

Before you begin,

- Bamboo will be unavailable while the backup process completes. The export itself may take a long time to complete, depending on the number of builds and test. We recommend running your backups at a time of day or night when usage is low.
- Backups may require large amounts of disk space, depending on the number of builds and tests. Please make sure you have enough disk space in your desired backup location before proceeding.
- Bamboo will not export if plans are currently being built (see Using the Bamboo dashboard).

For large instances we recommend using native database and filesystem backup tools instead of the built in backup/export functionality. For more details, see Automating Bamboo backup operations. Reserve the use of the built-in tools for database type migrations only (e.g. MySQL to Postgre).

On this page:

Related pages:

- Data and backups
- Exporting data for backup
- Importing data from backup

Specifying a backup schedule

To specify a backup schedule:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Scheduled Backups** in the left navigation column (under 'System').

3. Click **Edit** to modify the schedule settings:

   Disable scheduled backups
   This check box must be cleared for automatic backups to be performed.

   Backup Artifacts
   Select if you want to include build artifacts in your scheduled backups.

   Backup path
   Specify the directory where you want to store your backups. Each backup will be stored as a single file. It may be necessary to modify the Bamboo `bamboo.paths.set.allowed` system property to do this.
   Note that:
   Bamboo restricts the editing of certain file path settings for security reasons (see Bamboo Security Advisory 2010-05-04). If you must configure Bamboo to permit modification to its file path settings, start Bamboo with the system property `-Dbamboo.paths.set.allowed=true`. The procedure for configuring a Bamboo system property is described on Starting Bamboo.
   Once you have configured your file path setting, we recommend removing or disabling the `bamboo.paths.set.allowed` system property and restarting Bamboo. If your Bamboo instance is accessible to anyone outside your organization, then this will minimize the risk of Bamboo being compromised by security-related attacks.

   Backup file prefix
   Specify the first part of the filename for all your backup files.

   Backup file date pattern
   Specify the date/time format for identifying your individual backup files. This will be appended to **Backup file prefix** to form the complete filename for your backup files.

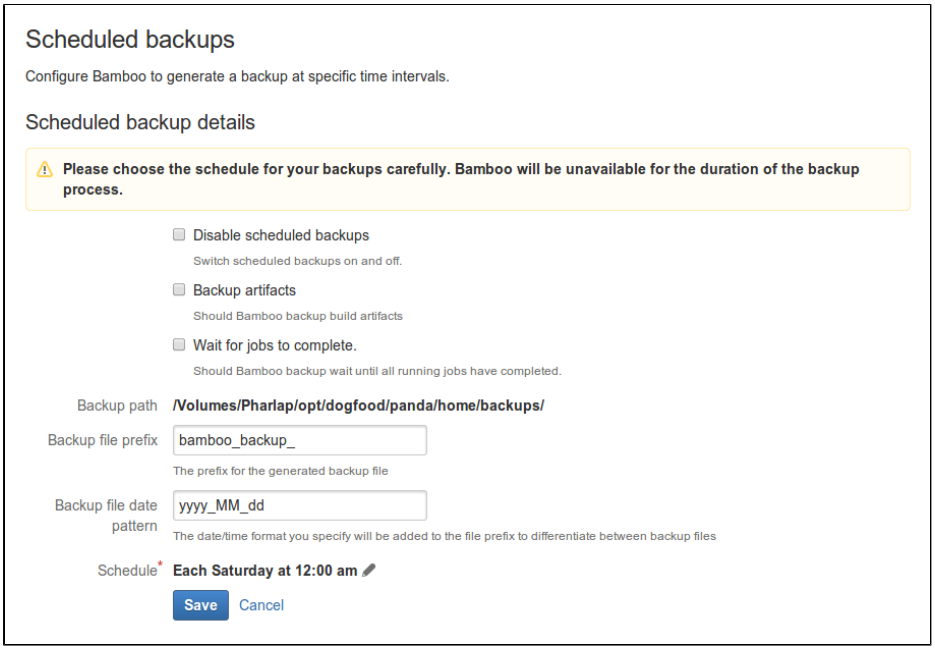
   Schedule
   Use the Schedule Editor to choose the frequency with which backups will be performed. See Cron-based scheduling for more information about the Schedule Editor.
4. Click **Save**. Your first backup will run when your server's clock matches the specified time.

Disabling a backup

If you disable schedule backups, your schedule details will be retained but no automatic backups will be performed.

To disable a scheduled backup:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Scheduled Backups** in the left navigation column. The 'Scheduled Backup Details' page will be displayed, showing details about the status of scheduled backups or any currently configured backup.
3. Click **Edit** to edit the current 'Scheduled Backup Details'.
4. Select the **Disable scheduled backups** check box.
5. Click **Save**.



Exporting data for backup

The instructions on this page describe how to export Bamboo data for backup.

Before you begin:

- Bamboo will be unavailable while the backup process completes. The export itself may take a long time to complete, depending on the number of builds and tests. We recommend running your backups at a time of day or night when usage is low.
- Backups may require large amounts of disk space, depending on the number of builds and tests. Please make sure you have enough disk space in your desired backup location before proceeding.
- Bamboo will not export if plans are currently being built.
- User management settings for Bamboo will be saved as part of the export. For information on user management in Bamboo, see Connecting to external user directories.
- **Export Directory Path** setting: Bamboo restricts the editing of certain file path settings for security reasons (see Bamboo Security Advisory 2010-05-04). If you must configure Bamboo to permit modification to its file path settings, start Bamboo with the system property `-Dbamboo.paths.set.allowed=true`. The procedure for configuring a Bamboo system property is described on Starting Bamboo.
  Once you have configured your file path setting, we recommend removing or disabling the `bamboo.paths.set.allowed` system property and restarting Bamboo. If your Bamboo instance is accessible to anyone outside your organization, then this will minimize the risk of Bamboo being compromised by security-related attacks.

  > For large instances we recommend using native database and filesystem backup tools instead of the built in backup/export functionality. For more details, see Automating Bamboo backup operations. Reserve the use of the built-in tools for database type migrations only (e.g. MySQL to Postgre).

Related pages:

- Data and backups
- Specifying a backup schedule
- Importing data from backup

To export data for backup:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Export** in the left navigation column (under 'System').
3. Complete the following settings:

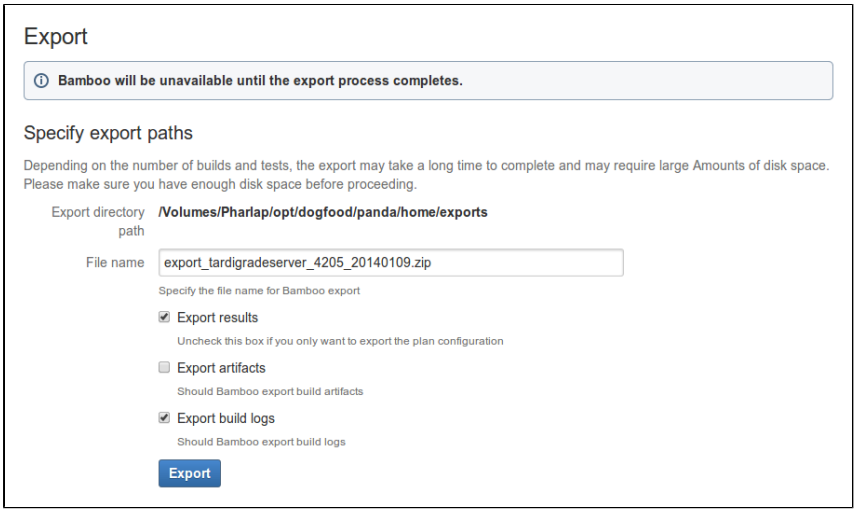
   Export Directory Path
   This can be configured – see the note above.

   File Name
   Edit the default name of the file to which Bamboo will export, if necessary.

   Export Results
   Clear this to export only the plan configurations.

   Export Artifacts
   Select to have Bamboo export build artifacts.

   Export Build Logs
   Select to have Bamboo export build logs.
4. Click the **Export**. Bamboo creates the export file in the location shown for **Export Directory Path**.



Importing data from backup

The instructions on this page describe how to import data from a Bamboo backup.

Before you begin:

- Bamboo will be unavailable until the import process is complete, which may take some time.
- The import process will *delete* your Bamboo installation and restore data from a previous export of Bamboo. This includes login data, so you will need to know an administration login in the Bamboo data to be imported.

- If you created your backup file using Bamboo 3.2 or later, importing the file will restore your user management settings. If you created your backup file using Bamboo 3.1 or earlier, importing the file will default your user management settings to 'Local users and groups' (i.e. user/group management in Bamboo). You may need to change your settings after the import.

- If you manage users externally (using LDAP or Crowd) and the Bamboo *internal* user repository (in the backup file) contains user names that duplicate user names in the external repository, you will not be able to import from the backup file.
- **Backup Directory Path**: Bamboo restricts the editing of certain file path settings for security reasons (see Bamboo Security Advisory 2010-05-04). If you must configure Bamboo to permit modification to its file path settings, start Bamboo with the system property `-Dbamboo.paths.set.allowed=true`. The procedure for configuring a Bamboo system property is described on Starting Bamboo.
  Once you have configured your file path setting, we recommend removing or disabling the `bamboo.paths.set.allowed` system property and restarting Bamboo. If your Bamboo instance is accessible to anyone outside your organization, then this will minimize the risk of Bamboo being compromised by security-related attacks.

Related pages:

- Data and backups
- Specifying a backup schedule
- Exporting data for backup

To import data from backup:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Import** in the left navigation column (under 'System').
3. Complete the following settings:

   File Path
   The absolute path to the data file that Bamboo should import. For example, "/opt/bamboo/bamboo-home/export.zip" on UNIX-based operating systems.

   Backup data
   Highly recommended. Bamboo will not import data unless it is able to successfully export data first.

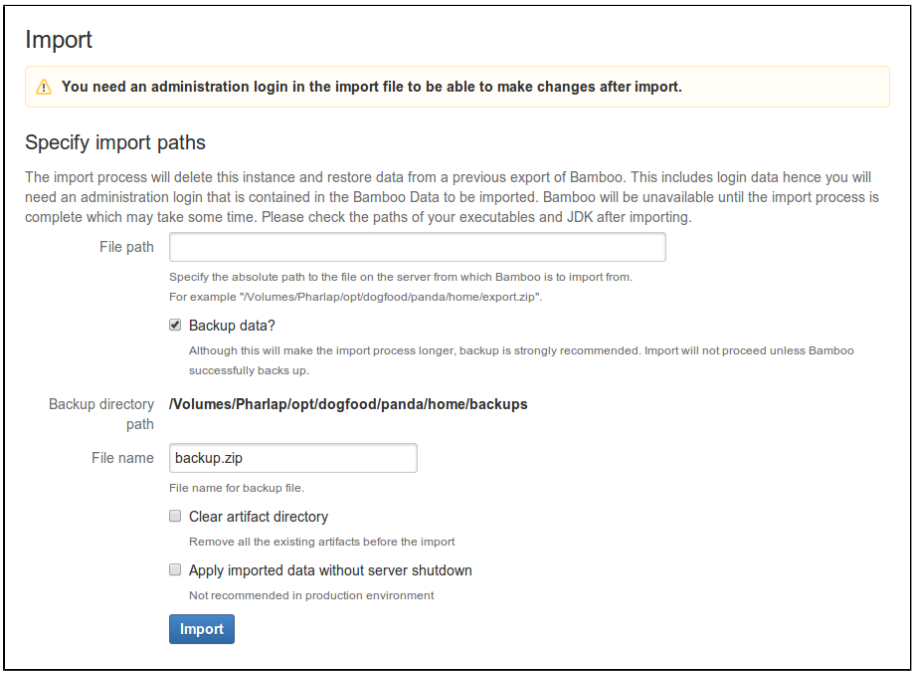
   Backup Directory Path
   This can be configured – see the note above.

   File Name
   The file to which Bamboo will export its data.

   Clear artifact directory
   Delete all existing build artifacts before the import.

   Apply imported data without server shutdown
   Not recommended in a production environment.
4. Click **Import**.
5. After the import is complete,
   - check the paths of your builders and JDK.

- index your data.



Configuring global expiry

Global expiry allows you to manage the timing for when build and deployment artifacts should be deleted from your Bamboo system.

You may want to consider doing this for the following reasons:

- Build and deployment artifacts can be large, and so consume storage on your system. Your system may run out of disk space if artifacts no longer in active use are retained indefinitely.
- Large numbers of builds and deployments clutter the Bamboo user interface, and may reduce performance, making it slower to work with Bamboo.

See this Atlassian blog post for a discussion of using build expiry and labels.

Global expiry applies to all build plans and deployment projects, and is generally the easiest way to manage artifacts expiry in Bamboo.

However, note that:

- You can configure build expiry for individual build plans to override the global expiry settings. You can *not* yet override the global expiry configuration for particular deployment projects.
- You can also delete the results of a plan build manually.

A Bamboo administrator can configure global expiry for both build and deployment artifacts as described below. Bamboo has built-in functionality to remove workspaces from remote agents for plans that no longer exist.

Configure global expiry

Ensure that you back up any build results data before their expiry date is reached.

To enable and configure global expiry:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **Expiry** (under 'Plans') in the left-hand navigation panel.
3. If necessary, enable deployment expiry. *Note that this can not be reversed – see the Bamboo 5.7 upgrade notes.*
4. Click **Edit**.

5. Configure global expiry using the following settings:

   Complete build & deployment results...
   All build results data (including artifacts and build logs), and deployment results and release artifacts, are deleted.

   Build and release artifacts
   Only user-defined artifacts are deleted.

   Build and deployment result logs
   Only build logs and deployment result logs are deleted. Note that logs can be excluded from expiry based on size.

   Expire after
   Specifies the age (days, weeks or months) that build and deployment results must reach before they are deleted.
   For example, specify '24 months' to keep results created in the last two years.

   Maximum builds to keep
   Specifies the maximum number of results you want to keep.

   Minimum builds to keep
   Specifies the minimum number of results you want to keep.
   For example, specify '50' to keep the latest 50 build results, even if they are older than the age specified with **Expire after**.

   Keep builds with the following labels
   Specifies the build labels (not plan labels or job labels) applied to builds for which you want to keep build results, regardless of the Expire after and Minimum builds to keep settings. Note that builds can be labeled either manually or automatically.

   Minimum deployments to keep
   Specifies the minimum number of successful deployments to keep, even if they are older than the age specified with Expire after. The minimum value is 2.
6. Click the icon to the right of 'Schedule' to set when the expiry event will be triggered. You can specify a cron expression if required. See this FAQ for help constructing cron expressions.
7. Click **Save**.

The global expiry *event* runs periodically (as determined by the expiry **Schedule)**, regardless of whether you disable or enable expiry for your build and deployment results. When this event occurs, your build and deployment results will be expired according to the global and plan settings you have made.

Calculating the expiry date

This section outlines how the ages of build or deployment results are calculated so as to determine when they should be expired.

Build results and all logs

The ages of build results, build logs, and deployment logs are simply calculated from their respective creation dates.

If the age of the build result or log is equal to or greater than the **Expire after** age, then it is deleted when the expiry event occurs (assuming build results or logs are configured for deletion).

Note that logs can be excluded from expiry based on size.

Build and deployment artifacts

The ages of build and deployment artifacts are calculated as follows:

- If there is no release associated with the build result, then use the build result creation date.

- Otherwise, if the build result has never been deployed, then use the creation date for the latest release that refers to it.
- Otherwise, use the creation date for the latest deployment.

If the age of the build or deployment artifact is equal to or greater than the **Expire after** age, then it is deleted when the expiry event occurs (assuming artifacts are configured for deletion).

Plan directory information REST API

An upcoming Bamboo release will make changes to the on-disk directory structure for BAMBOO_HOME. The changes are required for the improvement of the robustness of some Bamboo features.

As the use cases for this endpoint are somewhat different to the typical usage of Bamboo REST API functionality and the information disclosed is relatively low-risk, we have decided to make the access control strategy configurable using a system property.

For more information about system properties, see Starting Bamboo.

Plan directory information property details

The `bamboo.plan.directory.info.rest` is a system property with the following settings:

`disabled (default)`

The plan directory information REST API is disabled and all requests will be rejected

`local`

The plan directory information REST API is available without authentication to any request originating from localhost

`anonymous`

The plan directory information REST API is accessible anonymously

`authenticated`

The plan directory information REST API is accessible to any authenticated request

`authenticated-admin`

The plan directory information REST API is accessible to any request authenticated as an administrator

API Usage

The API is available at `/rest/api/latest/planDirectoryInfo/{planKey}`. For example:

`GET /rest/api/latest/planDirectoryInfo/PROJ-PLAN`

```
{"results":
  [{
    "planName": "Plan name",
    "isBranchBuild": false,
    "artifact_plan_roots": ["/opt/bamboo-home/artifacts/PROJ-PLAN"],
    "build_log_job_roots": {
      "PROJ-PLAN-JOB1":["/opt/bamboo-home/xml-data/builds/PROJ-PLAN-JOB1"],
      "PROJ-PLAN-JOB2":["/opt/bamboo-home/xml-data/builds/PROJ-PLAN-JOB2"]
    }
  }]
}
```

If no build exists that matches the provided key, an empty list is returned for the results.

artifact_plan_roots contains a list of directories that contain artifacts for the plan.

build_log_job_roots returns a map of job keys to the directory arrays. That is, each job in the plan is mapped to a list of directories that contain logs and build results for that build.

Bamboo 5.9 will only ever return single-item lists, but future versions of Bamboo will make changes to the on-disk directory layout and may return lists with multiple entries.

Security

As a distributed application, Bamboo's security is important. This page contains links to security-related information in the Bamboo documentation.

Security advisories

For information on how to report a security vulnerability in Bamboo and our policy on security advisories and patches, read Bamboo security advisories. A full list of security advisories that we have previously issued is also available on that page.

Bamboo permissions

For information on Bamboo's internal security model, i.e. user management and permissions, please see Users and permissions.

Remote agent security considerations

Note the following security implications when enabling remote agents for Bamboo:

- Encryption needs to be enabled on JMS and HTTP connections. The following data is encrypted:
  - login credentials for version control repositories (JMS)
  - build logs (JMS)
  - build artifacts (HTTP)

  See Securing your remote agents.

- Agent authorization should be enabled, see Agent authentication for more information. If it's not enabled, unauthorized parties will be allowed to install new remote agents, compromising the version control repository credentials.
- Agent secure token should be enabled. If it's not enabled, malicious users can send multiple approval requests for rogue agents which could lead to one of them being mistakenly accepted by a Bamboo administrator. See Security token verification.

As with all services, we strongly recommend that you do not open up agent JMS communication port on a public or untrusted network unless you want to use it. Creating remote agents is Disabling and enabling remote agents support by default.

Bamboo configuration

The following pages contain information on how to configure Bamboo features that can permit/forbid access to the Bamboo application.

- Agent authentication
- Bamboo cookies
- Best practices for Bamboo security
- Securing Bamboo against potential SSRF attacks
- Securing your remote agents
- Serialization protection methods
- Configuring XSRF protection
- Managing trusted keys
- System-wide encryption
- Repository-stored Bamboo Specs security
- Encrypting database password
- Encrypting passwords in server.xml
- Requiring personal access token expiration

Other security resources

- Bamboo security advisories
- Users and permissions
- Securing Bamboo against potential SSRF attacks

- Securing your remote agents
- Securing your repository connection
- Elastic Bamboo Security
- Configuring a plan's permissions

Agent authentication

Bamboo provides ways to verify that remote agents are allowed to connect to the Bamboo server. This provides improved security for sensitive information in Bamboo.

- Bamboo prevents unknown remote agents from connecting to the Bamboo server.
- Remote agents need to be manually approved by an administrator before they can communicate with the Bamboo server in any way.
- You can enable security token verification for additional level of safety.

Remote agent authentication (the manual agent approval) doesn't interfere with security token verification an both features can be enabled or disabled independently.

Note that Elastic agents do not have to be approved.

On this page:

Related pages:

- Bamboo remote agent installation guide
- Disabling and enabling remote agents support
- Configuring agents

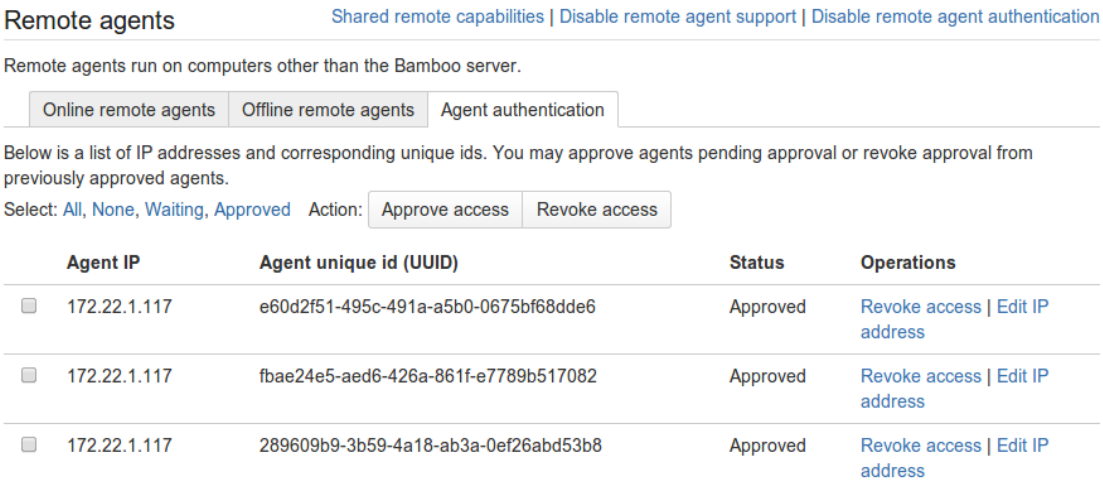
Authenticating remote agents

To enable agent authentication:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Then select **Agents** (under 'Build Resources').
3. Click **Enable Remote Agent Authentication**, and then **Confirm**.

Now you can approve access for a particular remote agent. To do this, click on the **Agent Authentication** tab (under 'Remote Agents').

See Bamboo remote agent installation guide for details about installing a remote agent.



Security token verification

Enable token verification to ask all remote agents to provide the token during the initial contact with the Bamboo server. Once you enable the verification, all agents that try to connect to Bamboo without the token are rejected before leaving any trail in Bamboo. By default, the feature is disabled for Bamboo Server.


This feature doesn't affect elastic agents.

Enabling security token verification

To enable security token verification, go to **Bamboo administration** > **Build resources** > **Agents**.

When you enable the verification, all agents that are already authenticated and connected continue to work. In other words, no running builds should be stopped or broken when the feature gets enabled. However, on server restart or agent restart each agent is required to have a correct token.

There are problems with backward compatibility. If the feature is enabled, old agents (from older Bamboo versions) will not be able to connect. Users need to download the new agent JAR.

Viewing the current security token

To view the current token, go to **Bamboo administration** > **Build resources** > **Agents** > **Install remote agent** page.

Each time the feature gets enabled, a new security token is generated, which means that disabling and re-enabling security token verification can be used to reset the token.

Notes

- If the agent's IP address changes, perhaps because DHCP is being used, then you will have to reapprove the agent when it next tries to connect using that different IP address.
- If you know the IP range of the remote agents, then you can allow the remote agents to connect from specific subsets. It is possible to work with wildcard characters to approve a range of IP addresses provided these remote agents keep the same UUID after a restart. Go to **Agents** > **Agent Authentication** > **Edit remote agent authentication IP**, where you can use wildcard characters to match multiple IP addresses.
- If you revoke access for a connected agent, the agent will remain connected and will continue to run. However, if the agent is subsequently restarted, it will not be able to connect.
- If you enable remote agent authentication, having previously revoked access for connected agents and disabled remote agent authentication, then you get the option to approve access for all connected agents at once. If you don't approve this, the agents stay connected and continue to run, but you will need to manually approve them when they next try to connect.

Bamboo cookies

Bamboo uses Seraph, an open source framework, for HTTP cookie authentication.

Authentication cookies

Bamboo uses two cookies:

- The JSESSIONID cookie is created by the application server and used for session tracking purposes.
- The 'remember me' cookie, `seraph.bamboo`, is generated by Bamboo when the user selects the **Remember me** checkbox on the login page.

You can read about cookies on the Wikipedia page.

On this page:

The 'Remember Me' cookie

The 'remember me' cookie is a long-lived HTTP cookie. This cookie can be used to authenticate an unauthenticated session. Bamboo generates this cookie when the user selects the **Remember me** checkbox on the login page.

Cookie key and value

By default, the cookie key is `seraph.bamboo`. This key is defined in the `BAMBOO-INSTALLATION/webapp/WEB-INF/classes/seraph-config.xml` file, in the `login.cookie.key` parameter.

The cookie contains a unique identifier plus a securely-generated random string.

Use of cookie for authentication

When a user requests a web page, if the request is not already authenticated via session-based authentication or otherwise, Bamboo will match the 'remember me' cookie (if present) against the token stored for the user in the Bamboo database (if present).

If the random string matches the value stored in the database and the cookie has not expired, the user is authenticated.

Life of 'Remember Me' cookies

You can configure the maximum age of the cookie. To do that you will need to modify the `BAMBOO-INSTALLATION/webapp/WEB-INF/classes/seraph-config.xml` file and insert the following lines below the other `init-param` elements:

```
<init-param>
    <param-name>autologin.cookie.age</param-name>
    <param-value>2592000</param-value><!-- 30 days in seconds -->
</init-param>
```

Other cookie usage

There are several cookies in Bamboo that are used for storing basic presentation states, such as the number of log lines to show, which tab was previously selected etc. They are:

Cookie	Purpose
AJS.conglomerate.cookie	Track which general tabs are open and closed
BAMBOO-AGENT-FILTER	Date range to show the builds for agents
BAMBOO-BUILD-FILTER	Date range to show the builds
BAMBOO-LOG-REFRESH	Log refresh interval in seconds
BAMBOO-MAX-DISPLAY-LINES	Maximum # of lines to show on the live logs page
atlassian.bamboo.dashboard.tab.selected	Which tab is selected on the dashboard
bamboo.author.view	Which tab is selected on the Authors tab
bamboo.build.groupby.type	Which time group-by period is used in the reports
bamboo.dash.display.toggle	The ids of the projects that are expanded on the dashboard

Best practices for Bamboo security

The best way to harden a system is to look at each of the involved systems individually. Contact your company's security officer or department to find out what security policies you should be using. There are many things to consider, such as the configuration of your underlying operating systems, application servers, database servers, network, firewall, routers, etc. It would be impossible to outline all of them here.

This page contains guidelines on good security practices, to the best of our knowledge.

On this page:

Configuring the web server

Please refer to the following guides for system administrators:

- How to configure Apache to lock down the administration interface to those people who really need it. See Using Apache to limit access to the Confluence administration interface for guidance.
- How to reduce the risk of brute force attacks: Enabling or Disabling Captcha for Failed Logins.

Configuring the application server

See the following system administrator guide for general hints on the application server level:

- Tomcat security best practices

Configuring the application

The way you set up Bamboo roles, permissions and processes makes a big difference in the security of your Bamboo site.

Below are some more Bamboo-specific items to consider. None of these provides 100% security. They are measures to reduce impact and to slow down an intruder in case your system does become compromised.

- Restrict the number of users with powerful roles or group memberships. If only one department should have access to particularly sensitive data, then do restrict access to the data to those users. Do not let convenience over-rule security. Do not give all staff access to sensitive data when there is no need.
- Put documented procedures in place for the case of employees leaving the company.
- Perform security audits regularly. Know who can help in case a security breach occurs. Perform 'what if' planning exercises. ('What is the worst thing that could happen if a privileged user's password were stolen while he's on vacation? What can we do to minimize damage?').
- Make sure the Bamboo database user (and all datasource database users) only has the amount of database privileges it really needs.
- Monitor your binaries. If an attacker compromises an account on your system, he will usually try to gain access to more accounts. This is sometimes done by adding malicious code, such as by modifying files on the system. Run routine scripts that regularly verify that no malicious change has been made.
- Disable Bamboo from serving HTML and JavaScript artifacts. Allowing Bamboo to do this creates an XSS vulnerability where HTML and JavaScript artifacts can be executed on the user's browser. Go to **Security settings** (under 'Security') in the Bamboo admin area, and *clear* the **Resolve artifacts**

content type by extension checkbox. Such artifacts will then be returned as plain text resources and the user's browser will handle them as simple text.
- Bamboo Server share permissions and accesses rights with with local agents. Keep in mind that by using local agents in your environment, you're giving other Bamboo users access to sensitive information you might be storing on the server.

Configuring system admin access

Below are some things to consider specifically related to the system admin role:

- Keep the number of Bamboo administrators extremely low. For example, 3 system administrator accounts should be the maximum.
- The administrators should have separate Bamboo accounts for their administrative roles and for their day to day roles. If John Doe is an administrator, he should have a regular user account without administrator access to do his day to day work (such as configuring build plans). This could be a 'john.doe' account. In addition, he should have an entirely separate account (that cannot be guessed by an outsider and that does not even use his proper name) for administrative work. This account could be 'jane smith' – using a username that is so obscure or fake that no outsider could guess it. This way, even if an attacker singles out the actual person John Doe and gets hold of his password, the stolen account would most likely be John's regular user account, and the attacker cannot perform administrative actions with that account.
- Lock down administrative actions as much as you can. If there is no need for your administrators to perform administrative actions from outside the office, then lock down access to those actions to known IP adresses, for example. See Using Apache to limit access to the Confluence administration interface for guidance.

Further precautions

As another precaution:

- Regularly monitor the above requirements. There are many things that could start out well, but deteriorate over time:
  - A system may start out with just 3 administrators, but over the course of a year this could grow to 30 administrators if no one prevents expansion.
  - Apache administration restrictions may be in place at the start of the year, but when the application server is migrated after a few months, people may forget to apply the rules to the new system.

Again, keep in mind that the above steps may only be a fraction of what could apply to you, depending on your security requirements. Also, keep in mind that none of the above rules can guarantee anything. They just make it harder for an intruder to move quickly.

Securing Bamboo against potential SSRF attacks

Attackers may use server-side request forgery (SSRF) vulnerabilities to access or modify data and resources that are not directly accessible from outside of your network.

We've been able to determine the following possible attack vectors against Bamboo:

- The `/rest/api/latest/repository/testConnection` endpoint allows scanning internal services of the victim's host. This enables the attacker to identify services through port enumeration and discover private files through file enumeration.
- A harmful webhook set up by an attacker that allows them to exploit an SSRF vulnerability to scan and read internal files on the victim's host.

If you have any non-public services accessible from the machine hosting your Bamboo instance, we recommend that you enable authentication for those services to protect your network against unauthorized access.

Securing your remote agents

This page applies to remote agents and not elastic agents. Elastic agents are secured automatically by the Bamboo server and no additional steps are required.

We **strongly recommend** that you do not enable remote agent installation without securing them on any Bamboo instance accessible from a public or untrusted network. Creating remote agents is disabled by default. If you choose to enable your remote agents without securing them, go to Security to understand the security implications.

You can secure your remote agents by configuring them to use SSL (Secure Sockets Layer). This protocol provides a secure mechanism for communication between your Bamboo server and remote agents. The information below describes how to configure your remote agents to use SSL.

Note that this solution doesn't cover artifact transfer between your remote agents and the server. To secure them, enable HTTPS (HTTP over SSL) access for Bamboo. See:

- Securing Bamboo with Tomcat using SSL
- Securing your Atlassian applications with Apache using SSL

On this page:

Related pages:

- Security
- Agent authentication
- Bamboo remote agent installation guide
- Disabling and enabling remote agents support
- Configuring agents
- Knowledge Base articles

Configure your Bamboo server to use SSL

To instruct your Bamboo server to start using SSL so that agents will be able to authenticate the server, you need to modify the addresses used for communication between the agent and the server.

To configure your Bamboo server to use SSL:

If you are setting up Bamboo for the first time:

Launch the Bamboo Setup Wizard and change the protocol of the **Broker URL** to SSL.
i.e. `ssl://host:port/`

Setting up **Broker URL** during the installation doesn't change the **Broker client URL** to the same protocol. You can change the Broker client URL either directly in the Bamboo GUI (> **System** > **General configuration**) or in the bamboo.cfg.xml file. Restart Bamboo to run it with the updated setup.

If you are configuring an existing installation of Bamboo:

1. Shut down your Bamboo server and agents.

2. Change the protocol of your **Broker URL** and **Broker client URL** in the **bamboo.cfg.xml** file to SSL. Note, do not change the address of this URL.

   ```
   <property name="bamboo.jms.broker.uri">ssl://myhost:myport</property>
   <property name="bamboo.jms.broker.client.uri">failover:(ssl://myhost:myport)<
   /property>
   ```

   For Bamboo 6.9 and later, add the `socket.verifyHostName=false` option to the `bamboo.jms.broker.client.uri` property, as in the example below:

   ```
   <property name="bamboo.jms.broker.client.uri">failover:(ssl://bamboo:54663?socket.
   verifyHostName=false&wireFormat.maxInactivityDuration=300000)?
   initialReconnectDelay=15000&maxReconnectAttempts=10</property>
   ```

3. Start up the Bamboo server.
4. Start up the Bamboo agents. If your agents do not start up, please check that you have set up your certificates correctly.

Special considerations/troubleshooting

On a standard Bamboo installation, the above steps are sufficient to secure your agents. After they're done, Bamboo will automatically set up the key/trust stores and distribute certificates to the agents the moment the first time the agent connects to the server.

The automatic keystore management can be enabled or disabled by adding -Dbamboo.manage.jms.ssl=true /false to the server command line. When this variable is present, Bamboo will not decide whether to run automatic key management.

The following files are used by automatic key management:

- The Agent stores the keystore and truststore in BAMBOO_AGENT_HOME/xml-data/configuration /jmsclient.ks and BAMBOO_AGENT_HOME/xml-data/configuration/jmsclient.ts, respectively.
- Server stores the keystore in BAMBOO_HOME/xml-data/configuration/broker.ks

To force generation of new keystores and truststores, simply remove these files. They will be regenerated on the next restart.

Serialization protection methods

For security/compatibility reasons, you can control the way Java classes are filtered during deserialisation. This is particularly important for agent-server communication.

The filtering can be either whitelist- or blacklist-based.

See also

- **Bamboo developer documentation**

⚠ The whitelist is the only recommended option for XStream serialisation. Blacklist (the former default) is scheduled for removal and should only be considered as a temporary fix in case of problems with the whitelist.

You can disable serialization security completely by setting the *bamboo.security.serialization.disable* system property. This is not recommended for security reasons.

You can set up the serialization protection methods in **Bamboo administration > Security > Security settings**.

Serialization	Description	Options
XStream	Agent - server messaging	• whitelist (default) • blacklist **(insecure)** • strict blacklist **(insecure)**
Bandana	Bamboo custom storage mechanism that can be used by plugins	• blacklist • strict blacklist (default)

Overview of options

The recommended option: whitelist

Whitelist has three sources:

- **bundled with Bamboo** (can't be modified),
- a list of whitelisted classes can be added into **Bamboo home directory,**
- **plugin vendors** can define certain classes as allowed.

A whitelist has higher priority than a blacklist. If a class is blacklisted by Bamboo, but is whitelisted anywhere (by a plugin or via bamboo home directory settings), then even if we're using the blacklist security setting, the class will still be allowed to be serialized/deserialized.

For more information about how to add classes to the whitelist or implement a plugin module, see Bamboo developer documentation.

Blacklist (insecure)

Blacklists are provided by Bamboo and can't be modified by plugin vendors or administrators.

Strict blacklist (insecure)

Strict blacklist restricts a bit more classes then the blacklist. Nevertheless, it's still considered insecure and it can cause problems with some of the plugins.

Configuring XSRF protection

To prevent users being tricked into unintentionally submitting malicious data, Bamboo uses XSRF security protection.

Atlassian supported plugins have been updated to support XSRF. XSRF protection is enabled by default for Atlassian Cloud customers and new customers for Bamboo Server, however, if you are using a plugin that is not yet compatible with this security feature, you can disable it.

Please carefully consider the security risks before you disable XSRF protection in your Bamboo installation.

Read more about XSRF (Cross Site Request Forgery) at wikipedia.

To configure XSRF protection:

1. Click the  icon in the Bamboo header and choose **Overview**.
2. Choose **Security settings** in the left-hand panel.
3. Choose **Edit**.
4. Uncheck **Enable XSRF protection** to disable XSRF protection or check it to enable XSRF protection.
5. Choose **Save**.

Related pages:

- Security
- Best practices for Bamboo security

XSRF protection was introduced in Bamboo 5.3, and is enabled automatically for all existing and new Atlassian Cloud users. *Existing* Bamboo Server users can enable XSRF protection by following the instructions above and checking **Enable XSRF protection.**

Is my Bamboo server already protected against XSRF attacks?

Customers upgrading...	XSRF protection
... an existing installation of Bamboo 5.2, and earlier, to Bamboo 5.3, and later.	XSRF protection is NOT enabled by default. You can enable XSRF protection using the instructions on this page.
... a new installation of Bamboo Server 5.3, and later.	XSRF protection IS enabled by default.

Managing trusted keys

By default, Bamboo accepts communication from all repository hosts that authenticate with SSH. You can secure communication between Bamboo and repositories by setting up trusted key management.

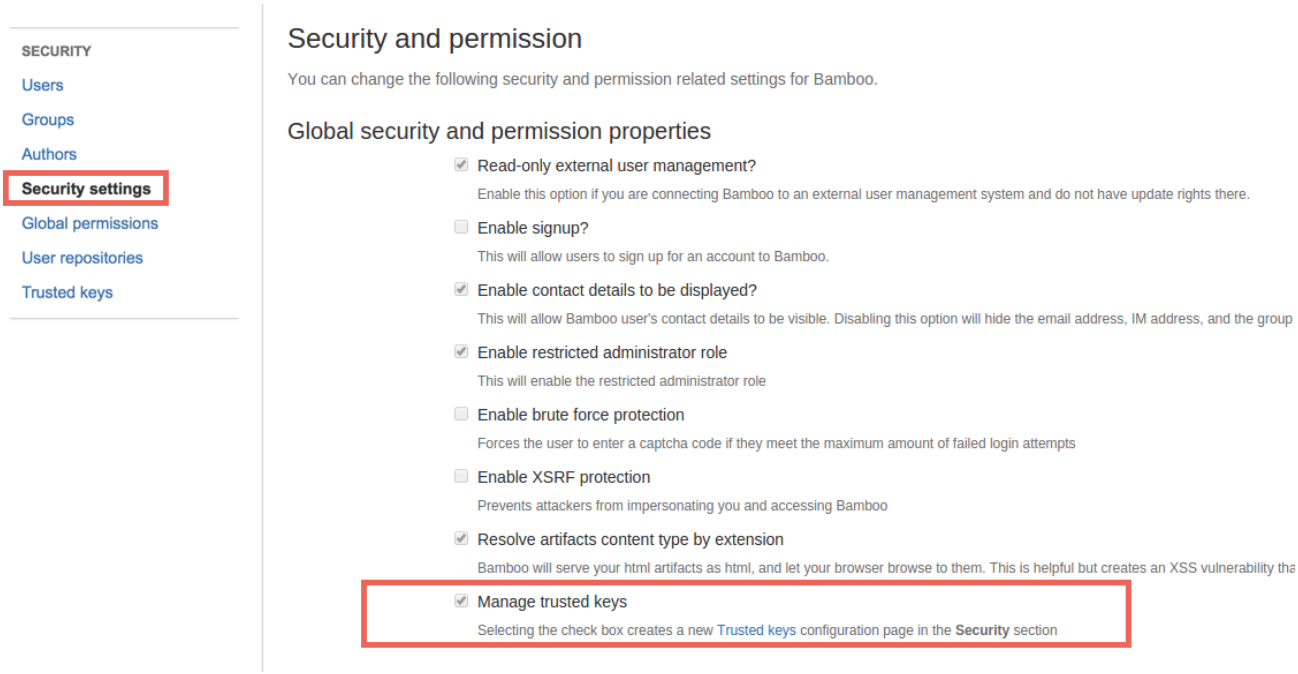
- Prevent Bamboo from connecting to unauthorized services via SSH.
- Manually authorize SSH key of the repository hosts upon first connection.
- Automatically authorize repository hosts that were added to the trusted keys list.

⚠ Trusted keys management is available only for restricted administrators.

Enabling trusted keys management in Bamboo

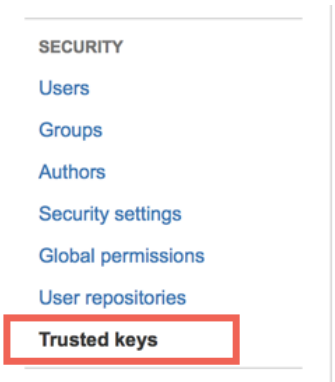
To enable SSH key management:

1. Go to **Administration > Security > Security settings > Global security and permission properties**.
2. Select the **Manage trusted keys** check box:

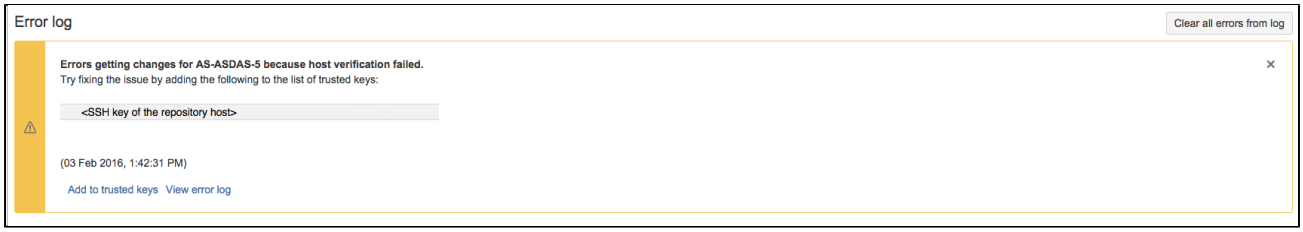


Results

- The **Trusted keys** configuration page is now available in **Administration > Security**:



- When Bamboo initiates the SSH connection with a repository host for the first time, you can decide whether to authorize the connection:



Adding and deleting trusted keys in Bamboo

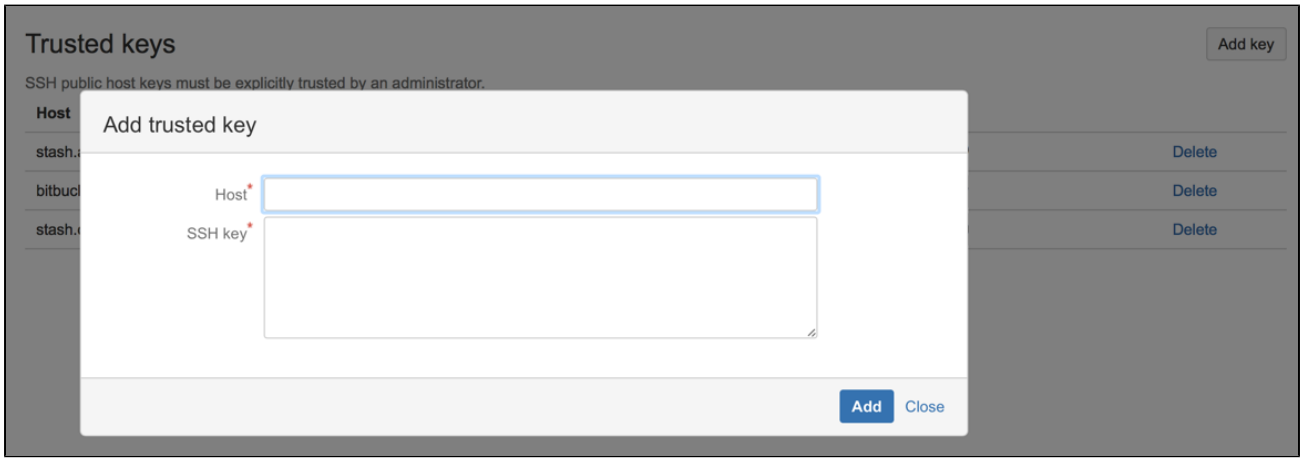
You can manage the authorized public SSH keys in the **Trusted keys** page.

To add a trusted key:

1. Go to **Administration > Security > Trusted keys**.
2. Specify the host URL, for example:

```
bitbucket.org
```

3. Paste the **public** key that you generated for your repository host and click **Add**.



System-wide encryption

As a CI/CD system, Bamboo stores sensitive data used to authenticate to external systems, such as VCS's, issue trackers and deployment targets. To protect this data, Bamboo uses a central encryption service.

Data encrypted at rest

The following data is encrypted:

- variables that include keywords such as "secret" and "password". These variables will also be obfuscated in the UI,
- shared credentials,
- credentials stored in the repository configuration (keys, passwords and passphrases).

This data is encrypted in the database and in the backups.

Encryption of data in transit

Bamboo relies on transport-level encryption for security of data in transit.

In the case of remote agents, this means that Bamboo must be configured with SSL for the JMS and web interfaces. In case of elastic agents, the encrypted tunnel (automatically set up by Bamboo) provides security out of the box.

Manual encryption

Bamboo 6.9 and later allows you to manually encrypt your sensitive data and later use it in repository-stored Bamboo Specs. For more information see Bamboo Specs encryption.

If you're a Bamboo administrator, you can enable/disable and configure the sensitive data encryption feature by going to  > **Security** > **Security settings** and changing the **System-wide encryption** section.

Encryption algorithm

The data is encrypted with AES algorithm using a key length of 256 bits. Both the key and the initialization vector are automatically generated using a secure random source when first used.

Key storage

The encryption key is stored in the database and on the filesystem. Both the filesystem and the database key parts are required to perform successful decryption.

The key part stored on your filesystem is located under `BAMBOO-HOME/xml-data/configuration/cipher`.

Data recovery

In case a part of your key is lost, your credentials will no longer be available and nothing can be done to recover them.

Repository-stored Bamboo Specs security

To modify security setting for repository-stored Bamboo Specs:

1. Go to  > **Security Settings**.
2. Choose from the following settings:

Enable Repository Stored Specs	This is a global toggle to enable/disable processing of Bamboo Specs projects stored in Bitbucket Server repositories. This feature allows to manage plans and deployments using configuration stored as code in your VCS. Once you enable this feature, you can select which repositories contain Bamboo Specs to be processed, see Enabling repository-stored Bamboo Specs.
Process Bamboo Specs in Docker	This is a global toggle to enable/disable processing of Bamboo Specs in Docker containers. This feature isolates the Bamboo Specs process from your Bamboo instance. By default, this will use the `atlassian/bamboo-specs-runner` image matching your version of Bamboo. Notes: The default Docker Image will be downloaded from Docker Hub Docker needs to be installed and running as the Bamboo user on the Bamboo server for this feature to function Process Bamboo Specs in Docker is only compatible with versions of docker specified at the Bamboo supported platforms documentation.

Encrypting database password

To add extra security to your Bamboo instance, you can encrypt the database password that is stored in the configuration file used by Bamboo to access your database. We've prepared different encryption methods for basic and advanced users. Additionally, you can create your own encryption based on our Cipher interface.

This solution is an obfuscation, which doesn't assure full security. Bamboo still needs to use the plain text password to connect to your database, so the configuration will contain all the information needed to decrypt the password. An attacker could act like Bamboo to obtain the password. We recommend that you secure the server where Bamboo and the database reside.

Basic encryption

This method uses a Base64 cipher, which is a simple obfuscation. It's recommended for users who don't want to store passwords in plain text, or have to meet specific requirements to encode them. See Basic database password encryption.

Advanced encryption

This method allows you to choose an algorithm to encrypt a database password. It provides more security as you don't have to store the encrypted password anywhere in the configuration file, which makes it difficult to find and decrypt. See Advanced database password encryption.

Encryption with custom Cipher

If you have extra requirements for storing the password, you can create your own Cipher based on our implementation and examples. To do this, you will need Java knowledge and some basic knowledge of Maven.

Basic database password encryption

To add extra security to your Bamboo instance, you can encrypt the database password that is stored in the configuration file used by Bamboo to access your database. This method uses a Base64 cipher, which is a simple obfuscation. It's recommended for users who don't want to store passwords in plain text, or have to meet specific requirements to encode them.

⚠ This solution is an obfuscation, which doesn't assure real security. Bamboo still needs to use the plain text password to connect to your database, so the configuration will contain all the information needed to decrypt the password. An attacker could act like Bamboo to obtain the password. We recommend that you secure the server where Bamboo and the database reside.

To encrypt your database password:

Step 1: Encrypt your password:

1. Go to `<Bamboo-installation-directory>/tools/atlassian-password`.
2. Run the following command to encrypt your password. Additionally, you can use optional arguments described below.

```
java -cp "./*" com.atlassian.db.config.password.tools.CipherTool
```

- **-- silent -s**: limits logging to minimum
- **-- help -h**: prints a help message with all parameters
- **-- mode -m**: defines what to do with the password, either **encrypt** or **decrypt**. If omitted, 'encrypt' will be used.
- **-- password -p**: plain text password. If omitted, you'll be asked to enter it. We recommend that you omit this parameter so your password is not stored in the history.

```
main DEBUG [db.config.password.DefaultCipherProvider] Initiate cipher provider class: com.atlassian.
db.config.password.ciphers.base64.Base64Cipher
main DEBUG [password.ciphers.base64.Base64Cipher] Initiate Base64Cipher
main DEBUG [password.ciphers.base64.Base64Cipher] Encrypting data...
main DEBUG [password.ciphers.base64.Base64Cipher] Encryption done.
Success!
For Jira (...)

For Bamboo, set the following properties in bamboo.cfg.xml:

<property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.base64.
Base64Cipher</property>
<property name="hibernate.connection.password">YmFtYm9v</property>

and restart then instance.
```

Step 2: Add the encrypted password to `bamboo.cfg.xml`:

1. Go to Bamboo home directory and back up the `bamboo.cfg.xml` file. Move the backup to a safe place outside of your Bamboo server.
2. Edit the `bamboo.cfg.xml` by adding the following tag:

```
<property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.base64.
Base64Cipher</property>
```

3. Replace content of the `<property name="hibernate.connection.password">` tag with the password encrypted by CLI. For example:

```
<property name="hibernate.connection.password">YmFtYm9v</property>
```

4. Restart Bamboo.

To decrypt your database password:

To decrypt the password, extend the command with the **-m decrypt** parameter:

```
java -cp "./*" com.atlassian.db.config.password.tools.CipherTool -m decrypt
```

When asked for a password, provide the encrypted one from your `bamboo.cfg.xml` file.

Advanced database password encryption

To add extra security to your Bamboo instance, you can encrypt the database password that is stored in the configuration file used by Bamboo to access your database. In this advanced method, you can use the Cipher algorithm that allows you to choose the algorithm used to encrypt your password. It provides more security as you don't have to store the encrypted password anywhere in the configuration file, which makes it difficult to find and decrypt.

This solution is an obfuscation, which doesn't assure real security. Bamboo still needs to use the plain text password to connect to your database, so the configuration will contain all the information needed to decrypt the password. An attacker could act like Bamboo to obtain the password. We recommend that you secure the server where Bamboo and the database reside.

Before you begin

Prepare a JSON object which contains all arguments required to encrypt your password using the following information:

Field	Description
`plainTextPassword`	Password in plain text.
`algorithm`	You can choose one of the following algorithms: • AES/CBC/PKCS5Padding • DES/CBC/PKCS5Padding • DESede/CBC/PKCS5Padding
`algorithmKey`	The algorithm key must correspond with the algorithm chosen above: • AES • DES • DESede

```
{"plainTextPassword":"yourPassword","algorithm":"AES/CBC/PKCS5PADDING","
algorithmKey":"AES"}
```

To encrypt your database password:

Step 1: Encrypt the password:

1. Go to `<Bamboo-installation-directory>/tools/atlassian-password`.
2. Run the following command to encrypt your password. You can also use optional parameters described below.

   ```
   java -cp "./*" com.atlassian.db.config.password.tools.CipherTool -c com.atlassian.db.config.password.
   ciphers.algorithm.AlgorithmCipher
   ```

   - **-- silent -s**: limits logging to minimum
   - **-- help -h**: prints a help message with all parameters
   - **-- mode -m**: defines what to do with the password, either **encrypt** or **decrypt**. If omitted, 'encrypt' will be used.
   - **-- password -p**: JSON object with required arguments. If omitted, you'll be asked to enter it. We recommend that you omit this parameter so your password is not stored in the history.
3. When prompted, provide the required arguments in a JSON object.

Step 2: Secure the generated files:

1. Secure the generated files:

Move the files generated by the tool to a secure place, and change them to read-only. Bamboo needs to be able to access and read those files to decrypt your password and connect to the database.

The following files have been generated:

- `javax.crypto.SealedObject_[timestamp]` - file with the encrypted password.
- `javax.crypto.spec.SecretKeySpec_[timestamp]`- key used to encrypt your password. You will need this file to decrypt your password.
- `java.security.AlgorithmParameters_[timestamp]`- Algorithm parameters used to encrypt your password. You will need this file only if you wanted to recreate an encrypted password.

Step 3: (*optional*) Store file paths as environment variables:

You can store paths to the generated files as environment variables. If the paths aren't present in the `bamboo.cfg.xml` file, Bamboo will automatically look for them in the specific environment variables. In this way, file paths will not be stored in the `bamboo.cfg.xml` file, making it difficult to locate the files used for encryption.

1. Store the two of the generated files as environment variables:

```
com_atlassian_db_config_password_ciphers_algorithm_javax_crypto_spec_SecretKeySpec
com_atlassian_db_config_password_ciphers_algorithm_javax_crypto_SealedObject
```

```
export com_atlassian_db_config_password_ciphers_algorithm_javax_crypto_spec_SecretKeySpec=/home
/bamboo/javax.crypto.spec.SecretKeySpec_123456789
export com_atlassian_db_config_password_ciphers_algorithm_javax_crypto_SealedObject=/home/bamboo
/javax.crypto.SealedObject_123456789
```

2. Edit the output from Step 1 and remove paths to the files. The final output should look similar to the following JSON object:

```
<property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.
algorithm.AlgorithmCipher</property>
<property name="hibernate.connection.password">{}</property>
```

Step 4: Adding the encrypted password to `bamboo.cfg.xml`:

1. Go to Bamboo home directory and back up the `bamboo.cfg.xml` file. Move the backup to a safe place outside of your Bamboo server.
2. In the `bamboo.cfg.xml` file, replace the content of the `<property name="hibernate.connection.password">` tag with the output JSON object. Depending on whether you used environment variables or not, adjust the JSON object to one of the following examples:
   - If you stored file paths as environment variables, remove the paths from the output. It should look like the following example:

     ```
     <property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.
     algorithm.AlgorithmCipher</property>
     <property name="hibernate.connection.password">{}</property>
     ```

   - If you didn't use environment variables and want to stick to file paths in the `bamboo.cfg.xml` file, make sure you updated them after moving the files to a secure place. The output should look like the following example:

```
<property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.
algorithm.AlgorithmCipher</property>
<property name="hibernate.connection.password">{"sealedObjectFilePath":"javax.crypto.
SealedObject_1621327067811","keyFilePath":"javax.crypto.spec.SecretKeySpec_1621327067777"}<
/property>
```

- You need to additionally escape the file paths and change double quotes (") surrounding the path to single quotes (') to avoid JSON parsing errors. The paths should look like the following example:

```
<property name="jdbc.password.decrypter.classname">com.atlassian.db.config.password.ciphers.
algorithm.AlgorithmCipher</property>
<property name="hibernate.connection.password">{"sealedObjectFilePath":'C:\\bamboo\\javax.
crypto.SealedObject_123456789',"keyFilePath":"'C:\\bamboo\\javax.crypto.spec.
SecretKeySpec_123456789'"}</property>
```

3. Restart Bamboo.

To decrypt your database password:

1. Run the encryption command with the **-m decrypt** parameter:

```
java -cp "./*" com.atlassian.db.config.password.tools.CipherTool -c com.atlassian.db.config.password.
ciphers.algorithm.AlgorithmCipher -m decrypt
```

2. When asked for the JSON object, provide the one from your `bamboo.cfg.xml` file.

```
{"sealedObjectFilePath":"/home/bamboo/javax.crypto.SealedObject_123456789","keyFilePath":"/home
/bamboo/javax.crypto.spec.SecretKeySpec_123456789"}
```

```
{}
```

Recreating an encrypted password

When you lose the encrypted password and encrypt the plain text password once again, the new encrypted password will look differently. This is not an issue, as it will still represent the same plain text password. However, in some cases, you might want to keep the consistency, for example by having the same encrypted password for all Bamboo Data Center nodes.

To encrypt the password in the exact same way as you did before, you will need the key used to encrypt the original password and the algorithm parameters. Both of these were generated by the encryption tool and saved in the following files:

- **Key:** `javax.crypto.spec.SecretKeySpec_[timestamp]`
- **Algorithm parameters:** `java.security.AlgorithmParameters_[timestamp]`

Once you've located these files, you can point the encryption tool to their location by using two extra fields in the JSON object. Below you can find the description of these fields and a sample JSON object.

Field	Description
`keyFilePath`	Path to a file that contains the key used to encrypt your original password, e.g. `javax.crypto.spec.SecretKeySpec_[timestamp]`. If you stored the file path as environment variable, you can omit this parameter.
`algorithmParametersFilePath`	Path to a file that contains the algorithm parameters used to encrypt your original password, e.g. `java.security.AlgorithmParameters_[timestamp]`.

Example of a JSON object with all fields:

```
{"plainTextPassword":"yourPassword", "algorithm":"AES/CBC/PKCS5PADDING", "algorithmKey":"AES",
"algorithmParametersFilePath":"java.security.AlgorithmParameters_123456789", "keyFilePath":"javax.crypto.
spec.SecretKeySpec_123456789"}
```

To encrypt the password, follow the steps in Step 1, and use the JSON object with they key and algorithm parameters.

Encrypting database password with custom Cipher

If you have extra requirements for storing the password, you can create your own Cipher based on our implementation and examples.

⚠ This solution is an obfuscation, which doesn't assure real security. Bamboo still needs to use the plain text password to connect to your database, so the configuration will contain all the information needed to decrypt the password. An attacker could act like Bamboo to obtain the password. We recommend that you secure the server where Bamboo and the database reside.

To encrypt your database password:

Step 1: Create a Maven project and get API dependencies:

1. Get 'api' and 'base' dependencies:
    a. Go to `<bamboo_installation_directory>/atlassian-bamboo/WEB-INF/lib.`
    b. Copy the following jar files:
        - **password-cipher-api-<version>.jar**: This file contains the API.
        - (optional) **password-cipher-base-<version>.jar**: This file contains sample implementation.
2. Create a Maven project.
3. Go to `resources`, and create a new folder `libs`.
4. Copy the jar files to the `libs` folder.
5. Use the following `pom`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/xsd/maven-
4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId><your_group_ID></groupId>
   <artifactId><your_artifact_ID></artifactId>
   <version><your_version></version>
   <properties>
      <maven.compiler.source>1.8</maven.compiler.source>
      <maven.compiler.target>1.8</maven.compiler.target>
   </properties>
   <repositories>
      <repository>
         <id>local-maven-repo</id>
         <url>file:///${project.basedir}/libs</url>
      </repository>
   </repositories>
   <build>
      <resources>
         <resource>
            <directory>src/main/resources/libs</directory>
            <excludes>
               <exclude>*</exclude>
            </excludes>
            <filtering>false</filtering>
         </resource>
      </resources>
   </build>
   <dependencies>
      <dependency>
         <groupId>com.atlassian.db.config</groupId>
         <artifactId>password-cipher-api</artifactId>
         <version><api_version></version>
         <scope>provided</scope>
      </dependency>
      <dependency>
         <groupId>com.atlassian.db.config</groupId>
         <artifactId>password-cipher-base</artifactId>
         <version><base_version></version>
         <scope>provided</scope>
      </dependency>
   </dependencies>
</project>
```

Step 2: Implement the Cipher interface

The Cipher interface contains only two methods — encrypt and decrypt. Decrypt will be called during Bamboo startup, which means that long running tasks can affect the startup time. Encrypt will not be called by Bamboo, as it's only used in the encryption tool.

You can use Base64Cipher and AlgorithmCipher as examples**.**

Step 3: Test your implementation

The encryption tool, described in Basic encryption and Advanced encryption, uses the same code as Bamboo to decrypt the password. You can use it to test your implementation.

Assuming that CLI and your jar is in the same folder:

```
java -cp "./*" com.atlassian.db.config.password.tools.CipherTool -c your.package.here.ClassName
```

Step 4: Make your lib available to Bamboo

Bamboo must be able to access your lib. Your class will be initiated using reflection. Put the lib in the following directory:

```
<Bamboo_installation_directory>/atlassian-bamboo/WEB-INF/lib
```

After upgrading Bamboo, you'll need to copy your lib to the Bamboo installation directory again.

Encrypting passwords in server.xml

To add extra security to your Bamboo instance, you can encrypt passwords that you use to configure Connectors in the Tomcat's server.xml file.

Before you begin

> ⚠ This solution is an obfuscation, which doesn't assure real security. Bamboo still needs to use the plain text password to connect to your database, so the configuration will contain all the information needed to decrypt the password. An attacker could act like Bamboo to obtain the password. We recommend that you secure the server where Bamboo and the database reside.

Bamboo provides the following protocols that extend Tomcat protocols with support for password encryption.

Bamboo protocol	Tomcat protocol on which Bamboo protocol is based	Attributes for which password encryption is supported
`com.atlassian.bamboo.tomcat.utils.Http11NioProtocolWithPasswordEncryption`	`Http11NioProtocol`	• `KeystorePass` • `KeyPass` • `SSLPassword` • `TruststorePass`
`com.atlassian.bamboo.tomcat.utils.Http11Nio2ProtocolWithPasswordEncryption`	`Http11Nio2Protocol`	• `KeystorePass` • `KeyPass` • `SSLPassword` • `TruststorePass`
`com.atlassian.bamboo.tomcat.utils.Http11AprProtocolWithPasswordEncryption`	`Http11AprProtocol`	• `KeystorePass` • `KeyPass` • `SSLPassword` • `TruststorePass`
`com.atlassian.bamboo.tomcat.utils.AjpNioProtocolWithPasswordEncryption`	`AjpNioProtocol`	• `secret`
`com.atlassian.bamboo.tomcat.utils.AjpNio2ProtocolWithPasswordEncryption`	`AjpNio2Protocol`	• secret
`com.atlassian.bamboo.tomcat.utils.AjpAprProtocolWithPasswordEncryption`	`AjpAprProtocol`	• secret

Encrypting a single password

1. Go to `<Bamboo-installation-directory>/lib`.
2. Run the following command:

```
java -jar atlassian-bamboo-tomcat-utils-<your bamboo version>.jar
```

3. Enter your password when prompted.

   The encryption tool will generate two files: `encryptedPassword` and `encryptionKey.` Move those files to a safe location. You can also rename the files if you want.

Encrypting multiple passwords for one Connector

If you want to encrypt more than one password for a single Connector, you must use the same encryption key for all passwords. After you encrypt you first password, use the generated `encryptionKey` to encrypt the subsequent password by passing path to the key to the encryption tool:

```
java -jar atlassian-bamboo-tomcat-utils-*.jar /path/to/encryptionKey
```

The encryption tool will generate only the `encryptedPassword` file.

Using encrypted passwords in Connector configuration

To use encrypted passwords in Connector configuration, you need to set up the following properties:

- `protocol` - use on of the Bamboo protocols described above
- `bambooEncryptionKey` - specify a path to the `encryptionKey` file

Then you can use path to a proper `encryptedPassword` file in place of plain text password in the Connector configuration.

For example, configuration of a `Http11Nio2` Connector with encrypted keystore and key passwords might look similarly to this:

```
<Connector
    protocol="com.atlassian.bamboo.tomcat.utils.Http11Nio2ProtocolWithPasswordEncryption"
    port="8443"

    (...)

    keystoreFile="/var/secrets/keystore/keystore"
    keystorePass="/var/secrets/keystore/encryptedKeystorePass"
    keyPass="/var/secrets/keystore/encryptedKeyPass"

    bambooEncryptionKey="/var/secrets/encryptionKey"
/>
```

Note that only one `bambooEncryptionKey` is specified, and both `keystorePass` and `keyPass` had to be encrypted with the same key.

Requiring personal access token expiration

By default, when a user creates a personal access token, they can optionally set an expiration period. As a system admin, you can make setting a token expiry a requirement and control the global maximum validity period for tokens in days. The expiration requirement will apply to all existing and future tokens.

Learn more about personal access tokens

1. In the upper-right corner of the screen, select **Administration** > **Overview**.
2. Under the **Security** section of the vertical menu, select **Security settings**.
3. On the **Security and permission** page, find the **Personal access tokens expiration** section.
4. For **Expiry required**, select **Yes**.
5. Enter a value for **Max days until expiry**.
6. Select **Save**.

Advanced actions

This section describes the administrative actions that are performed from outside of the Bamboo administration console.

Integrating Bamboo with Apache HTTP server

Securing Bamboo with Apache using SSL

Securing Bamboo with Tomcat using SSL

Running Bamboo as a Windows service

Disabling SSH access to elastic instances

Integrating Bamboo with Apache HTTP server

When opened in a viewport, the user will be redirected to: Proxying Atlassian server applications with Apache HTTP Server (mod_proxy_http).

This page explains how to establish a network topology in which Apache HTTP Server acts as a reverse proxy for Bamboo. Typically, such a configuration would be used when Bamboo is installed in a protected zone 'behind the firewall', and Apache HTTP Server provides a gateway through which users outside the firewall can access Bamboo.

Be aware that Bamboo does not need to run behind a web server, since it is capable of serving web requests directly; to secure Bamboo when run in this way see Securing Bamboo with Tomcat using SSL. Otherwise, if you want to install Bamboo in an environment that incorporates Apache HTTP Server, keep on reading.

About using Apache software

This section has general information pertaining to the use of Apache HTTP Server and Apache Tomcat. It is important that you read this section before proceeding to the steps that follow.

Configuring Tomcat 7

Bamboo ships with an instance of Tomcat 7, the configuration of which is determined by the contents of the `server.xml` file, which can be found in the `conf` directory immediately under the Bamboo installation directory. Note that any changes that you make to the `server.xml` file will be effective upon starting or re-starting Bamboo.

You may also find it helpful to refer to the Apache Tomcat 7.0 Proxy Support HowTo page.

On this page:

Configuring Apache HTTP Server

Since Apache HTTP Server is not an Atlassian product, Atlassian does not guarantee to provide support for its configuration. You should consider the material on this page to be for your information only; use it at your own risk. If you encounter problems with configuring Apache HTTP Server, we recommend that you refer to the Apache HTTP Server Support page.

You may find it helpful to refer to the Apache HTTP Server Documentation, which describes how you can control Apache HTTP Server by changing the contents of the `httpd.conf` file. The section on Apache Module mod_proxy is particularly relevant. Note that any changes you make to the `httpd.conf` file will be effective upon starting or re-starting Apache HTTP Server.

This document relates to Apache HTTP Server version 2.4.2; the configuration of other versions may differ.

Step 1: Configure the Tomcat Connector

Find the normal (non-SSL) `Connector` directive in Tomcat's `server.xml` file, and add the **`scheme`** , **`proxyName`** , and **`proxyPort`** attributes as shown below. Instead of `mycompany.com`, set the `proxyName` attribute to the domain name that Apache HTTP Server will be configured to serve. This informs Bamboo of the domain name and port of the requests that reach it via Apache HTTP Server, and is important to the correct operation of the Bamboo functions that construct URLs.

```
<Connector port="8085"
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    useBodyEncodingForURI="true"
    redirectPort="8443"
    compression="on"
    compressableMimeType="text/html,text/xml,text/plain,text/css,application/json,application/javascript,
application/x-javascript"
    scheme="http"
    proxyName="mycompany.com"
    proxyPort="80" />
```

Note: Apache HTTP Server's `ProxyPreserveHost` directive is another way to have the hostname of the incoming request recognized by Bamboo instead of the hostname at which Bamboo is actually running. However, the `ProxyPreserveHost` directive does not cause the scheme to be properly set. Since we have to alter Tomcat's `Connector` directive anyway, we recommend that you stick with the above-described approach, and don't bother to set the `ProxyPreserveHost` in Apache HTTP Server.

For more information about configuring the Tomcat Connector, refer to the Apache Tomcat 7.0 HTTP Connector Reference.

Step 2: Change Bamboo's base URL

After re-starting Bamboo, open a browser window and log in using an administrator account. Go to the Bamboo administration area and click **Server settings** (under 'Settings'), and change **Base URL** to match the proxy URL (the URL that Apache HTTP Server will be serving).

Step 3 (optional): Set a context path for Bamboo

By default, Bamboo is configured to run with an empty context path; in other words, from the 'root' of the server's name space. In that default configuration, Bamboo is accessed at:

```
http://localhost:8085/
```

It's perfectly fine to run Bamboo with the empty context path as above. Alternatively, you can set a context path by changing the `Context` directive in Tomcat's `server.xml` file:

```
<Context path="/bamboo" docBase="${catalina.home}/atlassian-bamboo" reloadable="false" useHttpOnly="true">
    ....
</Context>
```

If you do set a context path, it is important that the same path be used in Step 5, when setting up the `ProxyPass` and `ProxyPassReverse` directives. You should also append the context path to Bamboo's base URL (see Step 2).

Step 4: Enable `mod_proxy` and `mod_proxy_http` in Apache HTTP Server

In the `mod_proxy` documentation, you will read that `mod_proxy` can be used as a forward proxy, or as a reverse proxy (gateway); you want the latter. Where the `mod_proxy` documentation mentions '*origin server*', it refers to your Bamboo server. Unless you have a good reason for doing otherwise, load `mod_proxy` and `mod_proxy_http` dynamically, using the LoadModule directive; that means un-commenting the following lines in the `httpd.conf` file:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

Experienced administrators may be aware of the Apache Connector module, `mod_jk`. Atlassian does not recommend use of the `mod_jk` module with Bamboo, since it has proven itself to be less reliable than `mod_proxy`.

Step 5: Configure `mod_proxy` to map requests to Bamboo

To configure `mod_proxy` for use with Bamboo, you need to use the `ProxyPass` and `ProxyPassReverse` directives in Apache HTTP Server's `httpd.conf` file as follows:

```
ProxyPass        / http://localhost:8085/ connectiontimeout=5 timeout=300
ProxyPassReverse / http://localhost:8085/
```

Suppose Apache HTTP Server is configured to serve the `mycompany.com` domain; then the above directives tell Apache HTTP Server to forward web requests of the form `http://mycompany.com/*` to the Tomcat connector (Bamboo) running on port `8085` on the same machine.

> The `connectiontimeout` attribute specifies the number of seconds Apache HTTP Server waits for the creation of a connection to Bamboo.
>
> The `timeout` attribute specifies the number of seconds Apache HTTP Server waits for data to be sent to Bamboo.

If you set up a context path for Bamboo in Step 3, you'll need to use that context path in your `ProxyPass` and `ProxyPassReverse` directives. Suppose your context path is set to "`/bamboo`", the directives would be as follows:

```
ProxyPass        /bamboo http://localhost:8085/bamboo connectiontimeout=5 timeout=300
ProxyPassReverse /bamboo http://localhost:8085/bamboo
```

If Bamboo is to run on a different domain and/or different port, you should use that domain and/or port number in the `ProxyPass` and `ProxyPassReverse` directives; for example, suppose that Bamboo will run on port `9900` on `private.mycompany.com` under the context path `/bamboo`, then you would use the following directives:

```
ProxyPass        /bamboo http://private.mycompany.com:9900/bamboo connectiontimeout=5 timeout=300
ProxyPassReverse /bamboo http://private.mycompany.com:9900/bamboo
```

Step 6: Configure `mod_proxy` to disable forward proxying

If you are using Apache HTTP Server as a reverse proxy only, and not as a forward proxy server, you should turn forward proxying off by including a `ProxyRequests` directive in the `httpd.conf` file, as follows:

```
ProxyRequests Off
```

Step 7: Allow proxying to Bamboo from everywhere

Strictly speaking, this step is unnecessary because access to proxied resources is unrestricted by default. Nevertheless, we explicitly allow access to Bamboo from any host so that this policy will be applied regardless of any subsequent changes to access controls at the global level. Use the `Proxy` directive in the `httpd.conf` file as follows:

```
<Proxy *>
    Order Deny,Allow
    Allow from all
</Proxy>
```

The `Proxy` directive provides a context for the directives that are contained within its delimiting tags. In this case, we specify a wild-card url (the asterisk), which applies the two contained directives to all proxied requests.

The `Order` directive controls the order in which any `Allow` and `Deny` directives are applied. In the above configuration, we specify "`Deny,Allow`", which tells Apache HTTP Server to apply any `Deny` directives first, and if any match, the request is denied unless it also matches an `Allow` directive. In fact, "`Deny,Allow`" is the default; we include it merely for the sake of clarity. Note that we specify one `Allow` directive, which is described below, and don't specify any `Deny` directives.

The `Allow` directive, in this context, controls which hosts can access Bamboo via Apache HTTP Server. Here, we specify that all hosts are allowed access to Bamboo.

Step 8 (optional): Configure Apache HTTP Server for SSL

If you want to set up SSL access to Bamboo, take steps 8(a) to 8(d) below. When you are finished, users will be able to make secure connections to Apache HTTP Server; connections between Apache HTTP Server and Bamboo will remain unsecured (not using SSL). If you don't want to set up SSL access, you can skip this section entirely.

Note: It would be possible to set up an SSL connection between Apache HTTP Server and Tomcat (Bamboo), but that configuration is very unusual, and not recommended in most circumstances.

Step 8(a): Configure the Tomcat Connector for SSL

Find the normal (non-SSL) `Connector` directive in Tomcat's `server.xml` file, and change the **`redirectPort`**, **`scheme`** and **`proxyPort`** attributes as follows:

```
<Connector port="8085"
     protocol="HTTP/1.1"
     connectionTimeout="20000"
     useBodyEncodingForURI="true"
     redirectPort="443"
     compression="on"
     compressableMimeType="text/html,text/xml,text/plain,text/css,application/json,application/javascript,
application/x-javascript"
     secure="true"
     scheme="https"
     proxyName="mycompany.com"
     proxyPort="443" />
```

The `redirectPort` directive causes Tomcat-initiated redirections to secured resources to use the specified port. Right now, the Bamboo configuration of Tomcat does not involve Tomcat-initiated redirections, so the change to `redirectPort` is redundant. Nevertheless, we suggest that you change it as directed above for the sake of completeness.

Step 8(b): Set up a virtual host in Apache HTTP Server

Un-comment the following LoadModule directive in Apache HTTP Server's `httpd.conf` file:

```
LoadModule ssl_module modules/mod_ssl.so
```

Add the following directives to the `httpd.conf` file:

```
Listen 443
<VirtualHost *:443>
    SSLEngine On
    SSLCertificateFile    "/usr/local/apache2/conf/server.crt"
    SSLCertificateKeyFile "/usr/local/apache2/conf/server.key"
    ProxyPass        / http://localhost:7990/ connectiontimeout=5 timeout=300
    ProxyPassReverse / http://localhost:7990/
</VirtualHost>
```

The `Listen` directive instructs Apache HTTP Server to listen for incoming requests on port 443. Actually, we could omit that directive in this case, since Apache HTTP Server listens for `https` requests on port 443 by default. Nevertheless, it's good to make one's intentions explicit.

The `VirtualHost` directive encloses a number of child directives that apply only and always to requests that arrive at port 443. Since our `VirtualHost` block does not include a `ServerName` directive, it inherits the server name from the main server configuration.

The `SSLEngine` directive toggles the use of the SSL/TLS Protocol Engine. In this case, we're using it to turn SSL on for all requests that arrive at port 443.

The `SSLCertificateFile` directive tells Apache HTTP Server where to find the PEM-encoded certificate file for the server.

The `SSLCertificateKeyFile` directive tells Apache HTTP Server where to find the PEM-encoded private key file corresponding to the certificate file identified by the `SSLCertificateFile` directive. Depending on how the certificate file was generated, it may contain a RSA or DSA private key file, making the `SSLCertificateKeyFile` directive redundant; however, Apache strongly discourages that practice. The recommended approach is to separate the certificate and the private key. If the private key is encrypted, Apache HTTP Server will require a pass phrase to be entered when it starts up.

The `ProxyPass` and `ProxyPassReverse` directives should be set up in manner described in Step 5.

For more information about the support for SSL in Apache HTTP Server, refer to the Apache SSL/TLS Encryption manual. In addition, you will find lots of relevant information in the `<apache directory>/conf/extra/httpd-ssl.conf` file, which is included in the standard Apache distribution.

Step 8(c): Create SSL certificate and key files

In Step 8(b), you specified `server.crt` and `server.key` as the certificate file and private key file respectively. Those two files must be created before we can proceed. This step assumes that OpenSSL is installed on your server.

Generate a server key file:

```
openssl genrsa -des3 -out server.key 1024
```

You will be asked to provide a password. Make sure that the password is strong because it will form the one real entry point into the SSL encryption set-up. **Make a note of the password because you'll need it when starting Apache HTTP Server later**.

Generate a certificate request file (`server.csr`):

```
openssl req -new -key server.key -out server.csr
```

Generate a self-signed certificate (`server.crt`):

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

The above command generates a self-signed certificate that is valid for one year. You can use the certificate signing request to purchase a certificate from a certificate authority. For testing purposes though, the self-signed certificate will suffice. Copy the certificate file and private key file to the locations you specified in Step 8(b).

```
cp server.key /usr/local/apache2/conf/
cp server.crt /usr/local/apache2/conf/
```

Step 8(d): Update the base URL for 'https'

Open a browser window and log into Bamboo using an administrator account. Go to the Bamboo administration area and click **Server settings** (under 'Settings'). Change **Base URL** to use 'https'.

Using a self-signed certificate

There are two implications of using the self-signed certificate:

- When you access Bamboo in a web browser, you can expect a warning to appear, alerting you that an un-trusted certificate is in use. Before proceeding you will have to indicate to the browser that you trust the certificate.
- When you perform a git clone operation, SSL verification will fail.

The SSL verification error message will look something like this:

```
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify
failed while accessing  https://justme@mycompany/git/TP/test.git
```

It's easy to fix. Turn SSL verification off for individual git operations by setting the `GIT_SSL_NO_VERIFY` environment variable. In Unix, you can set the variable in-line with git commands as follows:

```
GIT_SSL_NO_VERIFY=true git clone  https://justme@mycompany/git/TP/test.git
```

In Windows you have to set the variable in a separate shell statement:

```
set GIT_SSL_NO_VERIFY=true
git clone  https://justme@mycompany/git/TP/test.git
```

Once you have purchased and installed a signed certificate from a certificate authority, you will no longer have to include the `GIT_SSL_NO_VERIFY` modifier.

A note about application links

When an application link is established between Bamboo and another Atlassian product (e.g. Jira), and Bamboo is operating 'behind' Apache HTTP Server, the link from the other product to Bamboo must be via the proxy URL; that is, the 'reciprocal URL' from, say Jira, to Bamboo must comport with the proxy name and port that you set at Step 1.

Troubleshooting

- On **Fedora Core 4,** people have reported 'permission denied' errors when trying to get mod_proxy (and mod_jk) working. Disabling SELinux (`/etc/selinux/config`) apparently fixes this.
- Some users have reported problems with user sessions being hijacked when the mod_cache module is enabled. If you have such problems, disable the **mod_cache** module. Note that this module is enabled by default in some Apache HTTP Server version 2 distributions.
- In general, if you are having problems:

    1. Ensure that Bamboo works as expected when running directly from Tomcat on http://localhost:8085/bamboo
    2. Watch the log files (usually in /var/log/httpd/ or /var/log/apache2/). Check that you have a **LogLevel** directive in your httpd.conf, and turn up logging **('LogLevel debug')** to get more info.

Securing Bamboo with Apache using SSL

If you want to set up SSL access to Bamboo, follow steps 1 to 4 below. When you are finished, users will be able to make secure connections to Apache HTTP Server; connections between Apache HTTP Server and Bamboo will remain unsecured (not using SSL).

Note:

- The steps on this page would normally be performed after Integrating Bamboo with Apache HTTP Server.
- It would be possible to set up an SSL connection between Apache HTTP Server and Tomcat (Bamboo), but that configuration is very unusual, and not recommended in most circumstances.

Step 1: Configure the Tomcat Connector for SSL

Find the normal (non-SSL) `Connector` directive in Tomcat's `server.xml` file, and change the **`redirectPort`**, **`scheme`** and **`proxyPort`** attributes as follows:

On this page:

Related pages:

- Integrating Bamboo with Apache HTTP Server
- Securing Bamboo with Tomcat using SSL

```
<Connector port="8085"
     protocol="HTTP/1.1"
     connectionTimeout="20000"
     useBodyEncodingForURI="true"
     redirectPort="443"
     compression="on"
     compressableMimeType="text/html,text/xml,text/plain,text/css,application/json,application/javascript,
application/x-javascript"
     secure="true"
     scheme="https"
     proxyName="mycompany.com"
     proxyPort="443" />
```

The `redirectPort` directive causes Tomcat-initiated redirections to secured resources to use the specified port. Right now, the Bamboo configuration of Tomcat does not involve Tomcat-initiated redirections, so the change to `redirectPort` is redundant. Nevertheless, we suggest that you change it as directed above for the sake of completeness.

Step 2: Set up a virtual host in Apache HTTP Server

Un-comment the following LoadModule directive in Apache HTTP Server's `httpd.conf` file:

```
LoadModule ssl_module modules/mod_ssl.so
```

Add the following directives to the `httpd.conf` file:

```
Listen 443
<VirtualHost *:443>
    SSLEngine On
    SSLCertificateFile    "/usr/local/apache2/conf/server.crt"
    SSLCertificateKeyFile "/usr/local/apache2/conf/server.key"
    ProxyPass        / http://localhost:8085/ connectiontimeout=5 timeout=300
    ProxyPassReverse / http://localhost:8085/
</VirtualHost>
```

The `Listen` directive instructs Apache HTTP Server to listen for incoming requests on port 443. Actually, we could omit that directive in this case, since Apache HTTP Server listens for `https` requests on port 443 by default. Nevertheless, it's good to make one's intentions explicit.

The `VirtualHost` directive encloses a number of child directives that apply only and always to requests that arrive at port 443. Since our `VirtualHost` block does not include a `ServerName` directive, it inherits the server name from the main server configuration.

The `SSLEngine` directive toggles the use of the SSL/TLS Protocol Engine. In this case, we're using it to turn SSL on for all requests that arrive at port 443.

The `SSLCertificateFile` directive tells Apache HTTP Server where to find the PEM-encoded certificate file for the server.

The `SSLCertificateKeyFile` directive tells Apache HTTP Server where to find the PEM-encoded private key file corresponding to the certificate file identified by the `SSLCertificateFile` directive. Depending on how the certificate file was generated, it may contain a RSA or DSA private key file, making the `SSLCertificateKeyFile` directive redundant; however, Apache strongly discourages that practice. The recommended approach is to separate the certificate and the private key. If the private key is encrypted, Apache HTTP Server will require a pass phrase to be entered when it starts up.

The `ProxyPass` and `ProxyPassReverse` directives should be set up in the manner described in Step 5 of the Integrating Bamboo with Apache HTTP server page.

For more information about the support for SSL in Apache HTTP Server, refer to the Apache SSL/TLS Encryption manual. In addition, you will find lots of relevant information in the `<apache directory>/conf/extra/httpd-ssl.conf` file, which is included in the standard Apache distribution.

Step 3: Create SSL certificate and key files

In Step 2, you specified `server.crt` and `server.key` as the certificate file and private key file respectively. Those two files must be created before we can proceed. This step assumes that OpenSSL is installed on your server.

Generate a server key file:

```
openssl genrsa -des3 -out server.key 2048
```

You will be asked to provide a password. Make sure that the password is strong because it will form the one real entry point into the SSL encryption set-up. **Make a note of the password because you'll need it when starting Apache HTTP Server later**.

Generate a certificate request file (`server.csr`):

```
openssl req -new -key server.key -out server.csr
```

Generate a self-signed certificate (`server.crt`):

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

The above command generates a self-signed certificate that is valid for one year. You can use the certificate signing request to purchase a certificate from a certificate authority. For testing purposes though, the self-signed certificate will suffice. Copy the certificate file and private key file to the locations you specified in Step 2.

```
cp server.key /usr/local/apache2/conf/
cp server.crt /usr/local/apache2/conf/
```

Step 4: Update the base URL for 'https'

Open a browser window and log into Bamboo using an administrator account. Go to the Bamboo administration area and click **Server s ettings** (under 'Settings'). Change **Base URL** to use 'https'.

Using a self-signed certificate

There are two implications of using the self-signed certificate:

- When you access Bamboo in a web browser, you can expect a warning to appear, alerting you that an un-trusted certificate is in use. Before proceeding you will have to indicate to the browser that you trust the certificate.
- When you perform a git clone operation, SSL verification will fail.

The SSL verification error message will look something like this:

```
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify
failed while accessing   https://justme@mycompany/git/TP/test.git
```

It's easy to fix. Turn SSL verification off for individual git operations by setting the `GIT_SSL_NO_VERIFY` environ ment variable. In Unix, you can set the variable in-line with git commands as follows:

```
GIT_SSL_NO_VERIFY=true git clone   https://justme@mycompany/git/TP/test.git
```

In Windows you have to set the variable in a separate shell statement:

```
set GIT_SSL_NO_VERIFY=true
git clone   https://justme@mycompany/git/TP/test.git
```

Once you have purchased and installed a signed certificate from a certificate authority, you will no longer have to include the `GIT_SSL_NO_VERIFY` modifier.

Securing Bamboo with Tomcat using SSL

This page in intended for administrators setting up Bamboo for a small team. It describes how to enable HTTPS (HTTP over SSL) access for Tomcat, the webserver distributed with Bamboo, using a self-signed certificate. You should consider doing this, and making secure access mandatory, if Bamboo will be internet-facing and usernames, passwords and other proprietary data may be at risk.

If you are setting up a production instance you should consider using a CA certificate, briefly described below.

Note that you can set up Bamboo to run behind a web server, such as Apache HTTP Server. To secure Bamboo with HTTPS, when Apache HTTP Server acts as a reverse proxy for Bamboo, see Integrating Bamboo with Apache HTTP Server.

Please note that Atlassian Support will refer SSL-related support to the issuing authority for the certificate. The documentation on this page is for reference only.

On this page:

Related pages:

- Integrating Bamboo with Apache HTTP Server

1. Generate a self-signed certificate

Self-signed certificates are useful where you require encryption but do not need to verify the website identity. They are commonly used for testing and on internal corporate networks (intranets).

Users may receive a warning that the site is untrusted and have to "accept" the certificate before they can access the site. This usually will only occur the first time they access the site.

The following approach to creating a certificate uses Java's keytool, for Java 1.6. Other tools for generating certificates are available.

To generate a self-signed certificate:

- Log in with the user account that Bamboo will run under, and run the following command:

  Windows

  ```
  "%JAVA_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA
  ```

  Linux, MacOS and Unix

  ```
  $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
  ```

  This will create (if it doesn't already exist) a new `.keystore` file located in the home directory of the user you used to run the keytool command.

Note the following:

- When running the keytool command you will be prompted with: `What is your first and last name?`

  You **must** enter the **fully qualified hostname** of the server running Bamboo. This is the name you would type in your web browser after 'http://' (no port number) to access your Bamboo installation. The qualified host name should match the base URL you have set in Bamboo (without the port number).

- The keytool utility will also prompt you for two passwords: the keystore password and the key password for Tomcat.

  You **must** use the same value for both passwords, and the value **must** be either:

  - "changeit", which is the default value Tomcat expects, or
  - any other value, but you must also specify it in `conf/server.xml` by adding the following attribute to the `<Connector/>` tag: `keystorePass="<password value>"`

2. Configure HTTPS in Tomcat

To configure HTTPS in Tomcat:

- Edit `conf/server.xml` and, at the bottom, before the `</Service>` tag, add this section (or uncomment it if it already exists) and add the following attribute to the `<Connector/>` tag: `keystoreFile="<location of keystore file>"`:

```
<Connector port="8443"
               maxHttpHeaderSize="8192"
               SSLEnabled="true"
               maxThreads="150"
               minSpareThreads="25"
               maxSpareThreads="75"
               enableLookups="false"
               disableUploadTimeout="true"
               useBodyEncodingForURI="true"
               acceptCount="100"
               scheme="https"
               secure="true"
               clientAuth="false"
               sslProtocol="TLS"
               keystoreFile="/path/to/.keystore" />
```

  This enables SSL access on port 8443 (the default for HTTPS is 443, but 8443 is used instead of 443 to avoid conflicts).

Exporting the self-signed certificate

If Bamboo will run as the user who ran the `keytool --genkey` command, *you do not need to export the certificate*.

You may need to export the self-signed certificate, so that you can import it into a different keystore, if Bamboo will not be run as the user executing `keytool --genkey`. You can do so with the following command:

Windows

```
"%JAVA_HOME%\bin\keytool" -export -alias tomcat -file file.cer
```

Linux, MacOS and Unix

```
$JAVA_HOME/bin/keytool -export -alias tomcat -file file.cer
```

If you generate the certificate as one user and run Bamboo as another, you'll need to do the certificate export as the generating user and the import as the target user.

Requesting a CA certificate

Digital certificates that are issued by trusted 3rd party CAs (Certification Authorities) provide verification that your website does indeed represent your company.

When running Bamboo in a production environment, you will need a certificate issued by a CA, such as VeriSign , DigiCert or Thawte. The instructions below are adapted from the Tomcat documentation.

First, you will generate a local certificate and create a 'certificate signing request' (CSR) based on that certificate. You then submit the CSR to your chosen certificate authority. The CA will use that CSR to generate a certificate for you.

1. Use Java's `keytool` utility to generate a local certificate, as described in the section above.
2. Use the `keytool` utility to generate a CSR, replacing the text `<MY_KEYSTORE_FILENAME>` with the path to and file name of the `.keystore` file generated for your local certificate:

   Windows

   ```
   "%JAVA_HOME%\bin\keytool" -certreq -keyalg RSA -alias tomcat -file certreq.
   csr -keystore <MY_KEYSTORE_FILENAME>
   ```

   Linux, MacOS and Unix

   ```
   $JAVA_HOME/bin/keytool -certreq -keyalg RSA -alias tomcat -file certreq.csr -
   keystore <MY_KEYSTORE_FILENAME>
   ```

3. Submit the generated file called `certreq.csr` to your chosen certificate authority. Refer to the documentation on the CA's website to find out how to do this.
4. The CA will send you a certificate.
5. Import the new certificate into your local keystore. Assuming your certificate is called "file.cer" whether obtained from a CA or self-generated, the following command will add the certificate to the keystore:

   Windows

   ```
   "%JAVA_HOME%\bin\keytool" -import -alias tomcat -file file.cer
   ```

   Linux, MacOS and Unix

   ```
   $JAVA_HOME/bin/keytool -import -alias tomcat -file file.cer
   ```

Troubleshooting

Here are some troubleshooting tips if you are using a self-signed key created by keytool, or a CA certificate, as described above.

When you enter "https://localhost:8443/" in your browser, if you get a message such as "Cannot establish a connection to the server at localhost:8443", look for error messages in your `logs/catalina.out` log file. Here are some possible errors with explanations:

SSL + Apache + IE problems

Some people have reported errors when uploading attachments over SSL using IE. This is due to an IE bug, and can be fixed in Apache by setting:

```
BrowserMatch ".MSIE." \
        nokeepalive ssl-unclean-shutdown \
        downgrade-1.0 force-response-1.0
```

Google has plenty more on this.

Can't find the keystore

```
java.io.FileNotFoundException: /home/user/.keystore (No such file or directory)
```

This indicates that Tomcat cannot find the keystore. The keytool utility creates the keystore as a file called `.keystore` in the current user's home directory. For Unix/Linux the home directory is likely to be `/home/<username>`. For Windows it is likely to be `C:\User\<UserName>`.

Make sure you are running Bamboo as the same user who created the keystore. If this is not the case, or if you are running Bamboo on Windows as a service, you will need to specify where the keystore file is in `conf/server.xml`. Add the following attribute to the connector tag you uncommented:

```
keystoreFile="<location of keystore file>"
```

Incorrect password

```
java.io.IOException: Keystore was tampered with, or password was incorrect
```

You used a different password than "changeit". You must either use "changeit" for both the keystore password and for the key password for Tomcat, or if you want to use a different password, you must specify it using the `keystorePass` attribute of the Connector tag, as described above.

Passwords don't match

```
java.io.IOException: Cannot recover key
```

You specified a different value for the keystore password and the key password for Tomcat. Both passwords must be the same.

Wrong certificate

```
javax.net.ssl.SSLException: No available certificate corresponds to the SSL cipher suites which are enabled.
```

If the Keystore has more than one certificate, Tomcat will use the first returned unless otherwise specified in the SSL Connector in `conf/server.xml`.

Add the `keyAlias` attribute to the Connector tag you uncommented, with the relevant alias, for example:

```
<Connector port="8443"
                maxHttpHeaderSize="8192"
                maxThreads="150"
                minSpareThreads="25"
                maxSpareThreads="75"
                enableLookups="false"
                disableUploadTimeout="true"
                useBodyEncodingForURI="true"
                acceptCount="100"
                scheme="https"
                secure="true"
                clientAuth="false"
                sslProtocol="TLS"
                keystoreFile="/opt/local/.keystore"
                keystorePass="removed"
                keyAlias="tomcat"/>
```

Using Apache Portable Runtime

APR uses a different SSL engine, and you will see an exception like this in your logs

```
SEVERE: Failed to initialize connector [Connector[HTTP/1.1-8443]]
LifecycleException:  Protocol handler initialization failed: java.lang.Exception:
No Certificate file specified or invalid file format
```

The reason for this is that the APR Connector uses OpenSSL and cannot use the keystore in the same way. You can rectify this in one of two ways:

Use the Http11Protocol to handle SSL connections

Edit the server.xml so that the SSL Connector tag you just uncommented specifies the Http11Protocol instead of the APR protocol:

```
<Connector port="8443"
               protocol="org.apache.coyote.http11.Http11Protocol"
               maxHttpHeaderSize="8192"
               SSLEnabled="true"
               keystoreFile="${user.home}/.keystore"
               maxThreads="150"
               enableLookups="false"
               disableUploadTimeout="true"
               acceptCount="100"
               scheme="https"
               secure="true"
               clientAuth="false"
               sslProtocol="TLS"
               useBodyEncodingForURI="true" />
```

Configure the Connector to use the APR protocol

This is only possible if you have PEM encoded certificates and private keys. If you have used OpenSSL to generate your key, then you will have these PEM encoded files - in all other cases contact your certificate provider for assistance.

```
<Connector port="8443"
               maxThreads="200"
               scheme="https"
               secure="true"
               SSLEnabled="true"
               SSLCertificateFile="${user.home}/certificate.pem"
               SSLCertificateKeyFile="${user.home}/key.pem"
               clientAuth="optional"
               SSLProtocol="TLSv1"/>
```

Enabling client authentication

To enable client authentication in Tomcat, ensure that the value of the `clientAuth` attribute in your `Connector` element of your Tomcat's `server.xml` file is `true`.

```
<Connector
        ...
        clientAuth="true"
        ... />
```

For more information about `Connector` element parameters, please refer to the 'SSL Support' section of the Tomcat 6.0 documentation.

Wrong certificate type

If the certificate from the CA is in PKSC12 format, add the `keystoreType` attribute to the SSL Connector in `conf/server.xml`.

```
keystoreFile="/opt/local/wildcard_atlassian_com.p12"
keystorePass="removed"
keystoreType="PKCS12"/>
```

Certificate chain is incomplete

If the root certificate and intermediary certificate(s) aren't imported into the keystore before the entity/domain certificate, you will see the following error:

```
[root@dev atlas]# /usr/java/jdk1.7.0_17/bin/keytool -import -alias tomcat -file my_entity_cert.crt
Enter keystore password:
keytool error: java.lang.Exception: Failed to establish chain from reply
```

Most likely, the CA sent a compressed file containing several certificates. The import order matters so you must import the root certificate first, followed by one or many intermediate certificates, followed lastly by the entity /domain certificate. There are many resources online that provide guidance for certificate installation for Tomcat (Java-based) web servers using keytool.

Disabling SSH access to elastic instances

By default, SSH (Secure Shell) access is enabled for elastic instances, the first time that you use Elastic Bamboo. Access rules for the Amazon Elastic Compute Cloud (EC2) are managed by 'security groups' in the Amazon Web Services Console. You can disable SSH access for your elastic instances by changing the EC2 access rules to remove the '**SSH**' Connection Method from the '**elasticbamboo**' security group.

For instructions on changing the EC2 access rules for Elastic Bamboo, please read the Elastic Bamboo Security document.

Changing Bamboo's root context path

There are various reasons why you may wish to change Bamboo's context path. Two of those are:

- You are running Bamboo behind a proxy.
- You have another Atlassian application, or Java web application, available at the same hostname and context path as Bamboo, and are experiencing login problems.

Related pages:

- Integrating Bamboo with Apache HTTP Server
- Login and session conflicts with multiple Atlassian applications

Upgrade Note

Since the manual steps of this process modify your Bamboo server, you will need to repeat Steps 1-6 each time you upgrade.

Changing the context path for Bamboo:

1. Navigate to the directory where you are running Bamboo from. This is *the install directory that you extracted Bamboo to*, not Bamboo home.
2. Stop Bamboo. This can be done using /`bin/stop-bamboo.bat` on Windows or /`bin/stop-bamboo.sh` on OSX or Linux.
3. Edit `conf/server.xml` and find the element below:

   ```
   <Context path="" docBase="${catalina.home}/atlassian-bamboo" reloadable="false" useHttpOnly="true"/>
   ```

   Update the `path` attribute to reflect the context path that you want Bamboo to be accessible at, e.g. "/bamboo":

   ```
   <Context path="/bamboo" docBase="${catalina.home}/atlassian-bamboo" reloadable="false" useHttpOnly="true"/>
   ```

   Then save the file.

4. Start Bamboo using /`bin/start-bamboo.bat` on Windows or /`bin/start-bamboo.sh` on OSX or Linux.

   Bamboo should now be available at the same host as before under the new context path. For example a server that was at http://localhost:8085 will now be reachable at http://localhost:8085/bamboo.
5. Once Bamboo has started, go to the administration area and click **General Configuration** (under 'System'). Add the new context path to your base URL:

   ```
   https://my-bamboo-hostname:8085/bamboo
   ```

6. Click **Save**.

Bamboo + Apache

Note that if you are running Bamboo behind Apache:

- You will need to make sure that the host or context path that Bamboo is exposed on is not also being used by another web application that is listening on a different port.
- If you have updated the Bamboo context path using the steps outlined above, you will need to update your Apache configuration, as described in Integrating Bamboo with Apache HTTP Server.

Collecting analytics for Bamboo

We are continuously working to improve Bamboo. Data about how you use Bamboo helps us do that. We have updated our Privacy Policy so that we may collect usage data automatically unless you disable collection. The data we collect includes information about the systems on which your installation of Bamboo is operating and the features you use in Bamboo.

For more details, see our Privacy Policy, in particular the 'Analytics Information from Downloadable Products' section.

See also our End User Agreement.

How to change data collection settings?

You can opt in to, or out of, data collection at any time. A Bamboo admin can change the data collection settings by going to  **> Analytics**.

How is data collected?

We use the Atlassian Analytics plugin to collect event data in Bamboo. Analytics logs are stored locally and then periodically uploaded to a secure location.

Bamboo Instance Health check

Bamboo provides a set of tools that you can use to monitor the health of your instance, as well as to identify the root cause when the instance is not performing as expected.

It's recommended that you look at the status of the health check tools after you install Bamboo or when you need to troubleshoot your setup.

To access the health check tools, go to  **> Overview > System > Troubleshooting and support tools.** T hen choose the Instance health tab.

> ⚠ Health checks visibility is dependent on your instance setup. For example, MySQL health checks are visible only if you're using a MySQL database.

Instance Health is a functionality provided by a built-in Troubleshooting and support tools plugin. For more information, see Instance Health.

Bamboo Instance Health check types

Bamboo Embedded database
Checks if the instance is connected to an HSQL database.

Bamboo MySQL Max Allowed Packet
Checks if the max_allowed_packet variable in your MySQL database is appropriate.

Bamboo MySQL Character Set
Checks if the character set used by the tables, columns and database defaults in your MySQL database is correct.

Bamboo MySQL Collation
Checks if the collation used by the tables, columns and database defaults in your MySQL database is correct.

Bamboo MySQL InnoDB Log File Size
Checks if the innodb_log_file_size variable in your MySQL database is appropriate.

Bamboo Embedded database

This check verifies if the instance is connected to an H2 database.

The H2 database is provided for evaluating Bamboo and is not supported as a production database. To keep your data safe, migrate to a production database once you finish the evaluation and before moving the instance to production.

For more information about how to move to a supported database, see Move data to a different database.

Bamboo MySQL Max Allowed Packet

This check verifies if the max_allowed_packet variable in your MySQL database is appropriate.

If the packet size limit is too small, it may cause problems with saving build results to the database.

For more information about the packet size limit, see:

- Unable to Save Build Results to the Database due to Error 'Packet for query is too large'
- MySQL Packet Too Large

Bamboo MySQL Character Set

This check verifies the correctness of the character set that is used by tables, columns, and database defaults in your MySQL database.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server.

For more information about character sets, see MySQL Character Set Support.

How to Fix the Collation and Character Set of a MySQL Database

What is Collation?

Collation determines how results are sorted and ordered. In newer versions of Atlassian applications, collation changes may become more strict - i.e, an application requires a certain collation. You must ensure your database has the correct collation for the application it will be used with.

Collation in MySQL can be complicated because you can have a separate collation set at:

1. The database level
2. The table level
3. The column level

Additionally, information inside a column may be encoded incorrectly as well - causing the data in that column to be displayed incorrectly.

Setup Guides for MySQL

To setup your MySQL database correctly, see the following resources for each product:

- Bamboo
- Confluence
- Crowd
- Fisheye / Crucible
- JIRA
- Bitbucket Server

⚠ Always back up your data before performing any modifications to the database. If possible, test any alter, insert, update, or delete SQL commands on a staging server first.

ⓘ You may wish to add all the ALTER TABLE statements to a single file for easier execution.

Changing the Database Collation

Change `yourDB` to suit your database name:

```
ALTER DATABASE yourDB CHARACTER SET utf8 COLLATE utf8_bin
```

Changing Table Collation

The following query will produce a series of `ALTER TABLE` statements, which you must then run against your database. Change `yourDB` to suit your database name:

```
SELECT CONCAT('ALTER TABLE ',  table_name, ' CHARACTER SET utf8 COLLATE utf8_bin;')
FROM information_schema.TABLES AS T, information_schema.`COLLATION_CHARACTER_SET_APPLICABILITY` AS C
WHERE C.collation_name = T.table_collation
AND T.table_schema = 'yourDB'
AND
(
    C.CHARACTER_SET_NAME != 'utf8'
    OR
    C.COLLATION_NAME != 'utf8_bin'
);
```

Changing Column Collation

The following queries (one for `varchar` columns, and one for non-`varchar` columns) will produce a series of `ALTER TABLE` statements, which you must then run against your database. Change `yourDB` to suit your database name:

```
SELECT CONCAT('ALTER TABLE `', table_name, '` MODIFY `', column_name, '` ', DATA_TYPE, '(',
CHARACTER_MAXIMUM_LENGTH, ') CHARACTER SET UTF8 COLLATE utf8_bin', (CASE WHEN IS_NULLABLE = 'NO' THEN ' NOT
NULL' ELSE '' END), ';')
FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA = 'yourDB'
AND DATA_TYPE = 'varchar'
AND
(
    CHARACTER_SET_NAME != 'utf8'
    OR
    COLLATION_NAME != 'utf8_bin'
);
```

```
SELECT CONCAT('ALTER TABLE `', table_name, '` MODIFY `', column_name, '` ', DATA_TYPE, ' CHARACTER SET UTF8
COLLATE utf8_bin', (CASE WHEN IS_NULLABLE = 'NO' THEN ' NOT NULL' ELSE '' END), ';')
FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA = 'yourDB'
AND DATA_TYPE != 'varchar'
AND
(
    CHARACTER_SET_NAME != 'utf8'
    OR
    COLLATION_NAME != 'utf8_bin'
);
```

Dealing with Foreign Key Constraints

It may be necessary to ignore foreign key constraints when making changes to a large number of columns. You can use the `SET FOREIGN_KEY_CHECKS` command to ignore foreign key constraints while you update the database.

```
SET FOREIGN_KEY_CHECKS=0;

-- Insert your other SQL Queries here...

SET FOREIGN_KEY_CHECKS=1;
```

It didn't work, what should I do ?

UTF8MB4/UTF8MB4_BIN

In some cases when you have emojis on commits, it may be needed to change some columns to utf8mb4/utf8mb4_bin.

Verify the current charset and collation:

```
USE NAME-OF-BAMBOO-DB;
SELECT @@character_set_database, @@collation_database;
```

The workaround to MySQL is to change those column to utf8mb4/utf8mb4_bin.

Columns that can show this issue are:

- *commit_files.commit_file_name*
- *commit_files.commit_file_reivision*
- *deployment_version_commit.commit_comment_clob*
- *user_commit.commit_comment_clob*

Follow those steps:

1. Stop Bamboo
2. Backup
3. Run the commands below

```
ALTER TABLE commit_files MODIFY COMMIT_FILE_NAME VARCHAR(1000) CHARACTER SET utf8mb4 COLLATE
utf8mb4_bin;

ALTER TABLE commit_files MODIFY COMMIT_FILE_REIVISION VARCHAR(128) CHARACTER SET utf8mb4
COLLATE utf8mb4_bin;

ALTER TABLE deployment_version_commit MODIFY COMMIT_COMMENT_CLOB LONGTEXT CHARACTER SET
utf8mb4 COLLATE utf8mb4_bin;

ALTER TABLE user_commit MODIFY COMMIT_COMMENT_CLOB LONGTEXT CHARACTER SET utf8mb4 COLLATE
utf8mb4_bin;
```

4. Change the bamboo.cfg.xml file to: `jdbc:mysql://[host]/[database]?autoReconnect=true&useUnicode=true&characterEncoding=UTF-8`
5. Start Bamboo

Verify the current char set and collation:

```
USE NAME-OF-BAMBOO-DB;
SELECT @@character_set_database, @@collation_database;
```

Bamboo MySQL Collation

The check verifies the correctness of the collation used by tables, columns, and database defaults in your MySQL database.

Incorrect collation may result in data loss, incorrect results, unwanted sorting orders, and poor performance.

For more information, see MySQL Collation Implementation Types.

Bamboo MySQL InnoDB Log File Size

This check verifies if the innodb_log_file_size variable in your MySQL database is appropriate.

A fail might be caused either by the maximum allowed packet size of the MySQL server being too small or when the InnoDB log file is too small (sometimes both).

For more information, see:

- MySQLSyntaxErrorException: Row size too large
- MySQL Limits on Table Column Count and Row Size

Lockout recovery process

This page describes how to recover administrator access for Bamboo 6.6 and later.

As an administrator, you may find yourself locked out of Bamboo and unable to log in. This can happen for various reasons, including:

- The external user directory server is not accessible (because the network is down, or the directory is down, or the directory has been moved to another IP address).
- The admin password has been forgotten or lost.
- The Bamboo instance is not configured properly and then restarted.

To regain your access to Bamboo:

1. Add the `"-Datlassian.recovery.password=temporarypassword"` Java property.
    a. For operating system and installation specific instructions for configuring a Java property for Bamboo, please see: Configuring your system properties
    b. **Linux Example**: Edit the `<Bamboo_installation_directory>\bin\setenv.sh` file and add the `"-Datlassian.recovery.password=temporarypassword"` value to the `JVM_SUPPORT_RECOMMENDED_ARGS` property.
    The property value must not be blank, and should look like this when you've done that:

    ```
    # Occasionally Atlassian Support may recommend that you set some specific JVM arguments.
    # You can use this variable to do that. Simply uncomment the below line and add any required
    # arguments. Note however, if this environment variable has been set in the environment of the
    # user running this script, uncommenting the below will override that.
    #
    JVM_SUPPORT_RECOMMENDED_ARGS=-Datlassian.recovery.password=temporarypassword
    ```

    Here we are using `temporarypassword` but you should use your own value.
2. Restart your Bamboo instance.
3. Log in to Bamboo using the `recovery_admin` username and the temporary password specified in Step 1.
4. Repair your Bamboo configuration.

    > ⚠ In the recovery mode, Bamboo creates an additional account with administrative privileges to allow you to fix your configuration. These privileges are removed when Bamboo restarts without the recovery mode. We strongly recommend that you do not perform any additional actions while Bamboo is in recovery mode.

5. Confirm your ability to log in with your usual admin profile.
6. Shut down Bamboo and remove the `atlassian.recovery.password` argument.
7. Start Bamboo again.

Fallback authentication in Bamboo

Alternatively, from Bamboo 8.1 or onwards Data Center, if SSO is the primary authentication method and for some reason, it fails, we can enable username and password authentication.

1. Enable username and password authentication with a REST call:

    ```
    curl -vvv -k -L -u <admin_username> -X PATCH <BambooURL>/rest/authconfig/1.0/sso \
        -H 'Content-Type: application/json'\
        -d '{"enable-authentication-fallback": true}'
    ```

2. Go to `<BambooURL>/userlogin!doDefault.action?auth_fallback` to display the Bamboo login page.

You can also restore username and password authentication by performing the following REST call:

```
curl -vvv -k -L -u <admin_username> -X PATCH <BambooURL>/rest/authconfig/1.0/sso \
    -H 'Content-Type: application/json'\
    -d '{"show-login-form": true}'
```

Bamboo Specs

Configuration as code is now available in Bamboo! We called this feature Bamboo Specs. Learn more about this feature that lets you store build plans configuration as code.

Why configuration as code?

Consider storing your build plan configuration as code for easier automation, change tracking, validation, and much more. You can read about the details in What is configuration as code?

What's in the package?

✔ Bamboo goodies

- Bamboo Specs library with an API for writing configuration as code
- Bamboo Specs Runner Maven plugin for easier plan deployments

✔ High-level language for configuration

YAML can get the job done, but we know that enterprise users need something much more powerful. That's why we decided to use a simple Java-based plan description language:

- Enjoy highlighting, syntax checks, and code autocompletion.
- Validate when you compile and run offline tests.
- Use high-level language features like modularization or libraries.

If you're not familiar with Java, don't worry. Our onboarding process will bootstrap you directly into a working environment and we have made sure that the plan definitions will be familiar to users of other languages such as Python, C++ or C#.

✔ Configuration in a language of your choice

The Bamboo Specs library is written in Java. It means that you can write your code in any high-level JVM language that interoperates with Java, for example Groovy, Scala, or Kotlin.

✔ Docs and more docs

We're still working on our documentation, but progress is more important than perfection, so we're sharing the first versions with you.

Bamboo Specs reference
Concepts explained with examples. We really like this one, check it out!

Bamboo Specs API reference
Our API. Documented 🙂

Best practices

Because we already have some recommendations!

Supported scenarios
Make the best use out of Bamboo Specs and our Support.

OK, I'm sold. Where do I start?

Easy. We've prepared some short tutorials for you.

Start with Java | Start with YAML

Quick links

- Tutorial: Create a simple plan with Bamboo Java Specs
- Bamboo Specs reference documentation
- Best practices
- What is configuration as code?

What is configuration as code?

Configuration as code allows the entire configuration of Bamboo plans to be stored as source code. It moves the managing of plans from the Bamboo UI to the developer's integrated development environment (IDE). This approach brings a lot of benefits.

In a world of UI-driven configuration

Prior to **Bamboo 6.0**, the only way to manage projects and builds plans was via the web UI, and few REST endpoints with limited functionality. It required the user to manually create plans and add stages, jobs, and tasks to them. The user had to define source code repositories, triggers, credentials, artifacts, and much more. Although Bamboo provided shortcuts for some actions, such as cloning an existing job or plan, sharing repositories or plan branches feature, it still required a lot of effort from the Bamboo administrator to manage it.

Advantages of configuration as code

Automation and standardisation

As configuration is written as source code, you can use all best development practices to optimise it, such as: creating reusable definitions of plans, parameterisation, using loops to create lots of different entities like plans, jobs, or repositories.

It is especially crucial for large Bamboo instances with hundreds of plans. Also, in the micro-services world, it's quite common to have similar build plans.

Versioning of changes

You can store configuration code in a version control system, such as Git, to see who changed what and when in your Bamboo environment. You can use tags to mark versions that have been published to Bamboo. You can use branches to isolate changes under construction and to work in parallel streams without affecting your production Bamboo instance.

Traceability of changes

If source code is versioned and properly managed (e.g. tagged), you can track which changes have been applied to your Bamboo server. Analysing code differences (e.g. via *git diff*) is quite often more convenient and efficient that reading audit logs in Bamboo.

Smooth promotion of changes from test to production

In case you use two Bamboo servers - a production and test instance - it's a lot easier to promote changes using configuration as code. In the "UI world", you had to click through many UI pages, test that everything works and next tediously repeat the same steps on the production instance. With configuration as code you can simply deploy plans to a test instance, verify changes and then deploy to the production instance just by changing the target URL.

Keeping build environment in sync with a product

It's quite common to keep a build configuration in the same source code repository as the product being built. As your product evolves, so does the environment needed to build it. Thanks to this synchronisation, you can always set up a proper environment, no matter whether you want to build the latest commit from the master branch or a bug fix branch created a few years ago.

Coding assistance and validation

Editing build plans in an IDE (such as Eclipse or IntelliJ IDEA) allows to you use IDE features such as: code autocompletion, parameter tool tips, pop-ups with JavaDoc, code refactoring, searching for usages of a given method/object and many more. You can also quickly perform offline validation by compiling and running the code.

Enabling repository-stored Bamboo Specs

Storing Bamboo Specs in a repository allows you to keep your project configuration together with the code and automatically publish any code changes. It also gives you access to the history of plan specifications and makes it easy to revert to a particular moment in time.

> Repository-stored Bamboo Specs are natively supported in Bitbucket Data Center and Bitbucket Cloud repositories. To use repository-stored Bamboo Specs with other types of repositories (including Git, GitHub, and Subversion), set up webhooks to trigger Bamboo to scan for changes to Specs.

Before you begin

Make sure that:

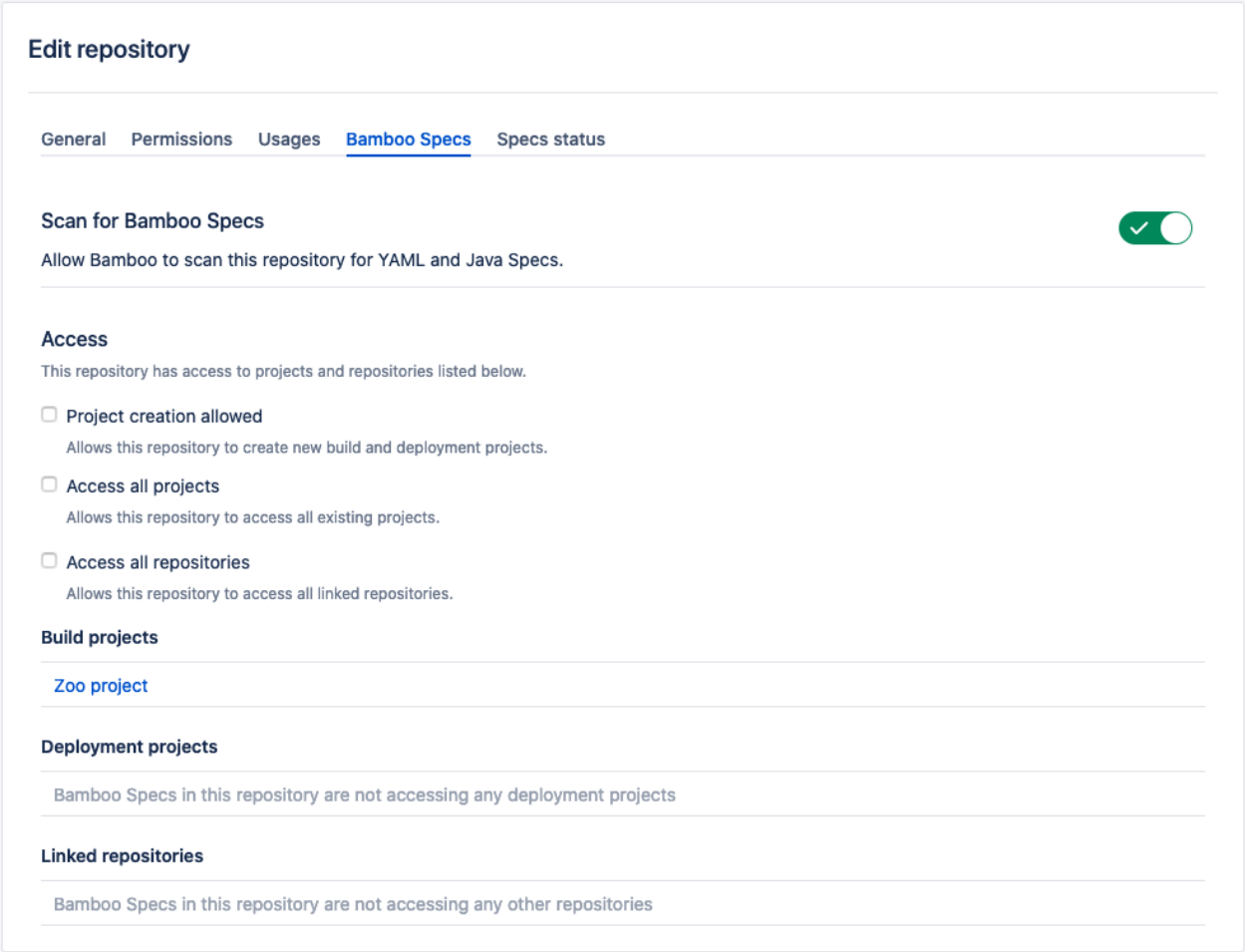
- native Git is installed
- you have access to the Maven central repository

Enable repository-stored Bamboo Specs in Bitbucket Data Center and Bitbucket Cloud

To enable repository-stored Bamboo Specs:

1. In the upper-right corner of the screen, select **Administration** > **Linked repositories**.
2. Select your repository.
3. In the **Bamboo Specs** tab, enable **Scan for Bamboo Specs**.

   > Bamboo Specs from this repository will be able to modify your plans and deployments in Bamboo. Make sure that write permissions to this repository are properly set in Bitbucket Data Center or Bitbucket Cloud as any commit to this repository will refresh Bamboo configuration.



4. In the **Access** section, select which projects Bamboo Specs can access and decide if Bamboo Specs should be allowed to create new build and deployment projects.

   Available only to users with global 'Create' permission. Use this option if you want to be able to create new build projects and deployment projects from Bamboo Specs. Bamboo Specs will be allowed to access and modify projects thus created.
   Available only for Bamboo administrators. Use this option if you want to have a repository managing multiple build plans and deployment projects in your Bamboo instance.
   You must have project administrator or Bamboo administrator permissions to add a build project. Yo u can't add new projects here.

   To add a Build project:
   i. From the Bamboo header, select **Projects**.
   ii. Select your project.
   iii. In the top-right corner, click **Project settings**.
   iv. In the sidebar, select **Bamboo Specs repositories**.
   v. Select your repository and select **Add**.

   > In case a plan downloads artifacts from another project (the Artifact Downloader task) or triggers builds of plans in another project (the Dependencies tab on the Plan configuration page), you have to grant access to these projects as well.

   You must have project administrator or Bamboo administrator permissions to add a deployment project. You can't add new projects here.

   To add a Deployment project:
   i. From the Bamboo header, select **Deploy** > **All Deployment Projects.**
   ii. Select your project.
   iii. In the top-right corner, select **[...]** > **Edit project**.

iv. Select **Bamboo Spect repositories**.
v. Select your repository and select **Add**.
Now, Bamboo Specs from this repository will be able to modify this deployment project and environments. Once you have added your repositories, you can see them listed in the Projects section in Linked repositories.

Enable project-level repositories using repository-stored Bamboo Specs

If the repository is defined within a project, it can be used to create/update plans within that project. For configurations outside the project, linked repositories need to be used.

BAM-21632 - Allow project-level repositories using RSS to publish deployment projects
GATHERING INTEREST

Use webhooks with other repository types

Webhooks allow repositories other than Bitbucket Data Center and Bitbucket Cloud to communicate with Bamboo (including Git, GitHub, and Subversion).

The following is an example of a webhook request using curl for a Git repository:

```
#!/bin/bash
/usr/bin/curl -X POST -H "X-Atlassian-Token: no-check" http://[BAMBOO_URL]/bamboo/rest/api/latest/repository
/scan\?repositoryId\=[REPOSITORY_ID]
```

Once you set up a webhook for a repository, it sends the HTTP request to Bamboo with every new commit. This HTTP request, in turn, triggers Bamboo Specs scan repository to see if there are any changes to Specs. If Bamboo detects any changes in a repository, it automatically updates necessary plans and deployments. Learn more about setting up webhooks.

Bamboo Java Specs

We've written down some details about how configuration as a code in Java works in Bamboo.

Bamboo uses high-level language for configuration

YAML can get the job done when you want to define your plan quickly but we know that enterprise users sometimes need something much more powerful. That's why we decided to pick Java as the default language for creating Bamboo Specs.

When using Java, you get the following features:

- syntax checking and highlighting while editing
- code autocompletion - IDE "knows" what is available (whereas a YAML file is just text)
- code refactoring
- code analysis - IDE helps you find usages of a given method or object
- code validation by the compiler - you can easily spot any spelling mistakes
- offline and online code validation by the *Bamboo Specs* runner
- API versioning and deprecation (via JavaDoc's '@since' and '@deprecated' tags)
- language features such as loops, modularisation, libraries, etc.

Bamboo allows you to write configuration as code in a language of your choice

The Bamboo Specs library, which provides an API to write configuration as code, has been written in Java. Thus the most natural is to use Java to write the configuration as well and this is a language that Atlassian will officially support for Bamboo Specs.

However, if you are familiar with another JVM language and have experience with how to integrate it with Java classes, you can use any language of your choice. Good examples are Groovy, Scala, and Kotlin (we performed smoke tests with Groovy and Kotlin and they worked fine).

We also provide a *Spec Runner* Maven plugin, which eases the deployment of plans.

For more information, see Bamboo Specs reference.

Create a Bamboo Specs project using Maven Archetype

You can create Bamboo Specs projects using our Maven Archetype.

On this page

Related links

- Tutorial: Create a simple plan with Bamboo Java Specs
- Bamboo Specs reference
- Bamboo Specs API reference

Basic usage

You can quickly create a basic Bamboo Specs project using the following command:

```
mvn archetype:generate -DarchetypeGroupId=com.atlassian.bamboo -DarchetypeArtifactId=bamboo-specs-
archetype \
  -DarchetypeVersion=6.0.0
```

This command runs in the interactive mode. You will be asked to provide the following parameters for the project:

- groupId
- artifactId
- a version
- a package prefix

The command creates a directory that has the name specified in the artifactId parameter.

You can also run the command in the batch mode and provide all necessary parameters:

```
mvn archetype:generate -B \
  -DarchetypeGroupId=com.atlassian.bamboo -DarchetypeArtifactId=bamboo-specs-archetype \
  -DarchetypeVersion=6.0.0 \
  -DgroupId=com.my.company -DartifactId=my-bamboo-casc -Dversion=1.0.0 -Dpackage=com.my.company
```

Additional options

- *-Dtemplate=minimal* - creates a minimal template (with no repositories, stages, or artifacts)

Importing created project into IDE

You can easily import the project into IDE, such as Eclipse or IntelliJ IDEA.

1. Run Eclipse.
2. From the main menu, select **File** > **Import...**
3. In the Import dialog, click **Maven > Existing Maven Projects** and click **Next**.
4. Click **Browse**, select your folder, and click **Open**.
5. Click **Finish**.
   Eclipse creates a new project and downloads necessary dependencies.

1. Run IntelliJ IDEA.
2. From the main menu, click **File** > **Open** and select the *pom.xml* file.
3. Click **Open as Project**.
   IntelliJ IDEA creates a new project and downloads necessary dependencies.

Troubleshooting

Unable to add module

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-archetype-plugin:3.0.0:generate
 (default-cli) on project p4:
 org.apache.maven.archetype.exception.InvalidPackaging:
 Unable to add module to the current project as it is not of packaging type 'pom' -> [Help 1]
```

The archetype found pom.xml file in the current directory and failed to add the module to it.

Run the archetype from a directory which does not contain the pom.xml file.

See next: Best practices

Creating deployment projects in Bamboo Specs

Bamboo Specs can be used to create and update deployment projects.

Creating deployment projects in Bamboo Specs is similar to that of updating a plan. The main difference is the class you need to define in your Bamboo Specs code:

```
private Deployment createDeployment() {
    return new Deployment(new PlanIdentifier("PROJECTKEY", "PLANKEY"),
                "Deployment project name")
                .releaseNaming(new ReleaseNaming("release-1")
                        .autoIncrement(true))
                .environments(new Environment("Test environment")
                        .tasks(new ScriptTask().inlineBody("echo Hello world!")));
}
```

Once you defined this class, instruct Bamboo Specs to publish your deployment to deployment server, just like you do with plans:

```
Deployment myDeployment = createDeployment();
bambooServer.publish(myDeployment);
```

Exporting existing plan configurations to Bamboo Specs

To ease migration of your Bamboo plans to configuration as code, we have prepared the export feature.

> Note: This will not export historical data for the plans.

In order to export an existing build plan to Bamboo Specs source code:

1. If you don't have Bamboo Specs project, create one.
2. Go to your build plan and select **Actions > Configure plan.**
3. On the plan configuration page, select **Actions > View plan as Java Specs.**
4. Copy generated Java code to your code editor by putting it in the *PlanSpec#createPlan* method for instance.

To export an existing deployment plan to Bamboo Specs source code:

1. If you don't have Bamboo Specs project, create one.
2. Go to your deployment project and select **... > Edit project.**
3. On the deployment project configuration page, select **... > View project as Java Specs.**
4. Copy generated Java code to your code editor by putting it in the *PlanSpec#createPlan* method for instance.

The code is ready to run, with a few exceptions:

- `.credentials` file needs to exist in the directory where the Bamboo Specs are run
- manually created branches are not exported - you can add them manually in the UI

We advise you to cross-check the generated code with the Bamboo Specs reference and to compare the original with the generated plan.

Sensitive data such as passwords or keys are exported as values encrypted using Bamboo's System-wide encryption algorithm. You will see **BAMSCRT@0@**... exported in place of the secret string. For details on generating encrypted secrets for new values, see Bamboo Specs encryption.

You will find equivalent classes for the vast majority of Bamboo concepts in the generated code, such as: plans, stages, jobs, tasks of different kinds, permissions, and deployments. The export feature also handles tasks not known to Bamboo (e.g. third party plugins or a plugin written by you) via a generic *AnyTask* class.

You may want to refactor the code generated, for instance: split code into smaller methods, extract common parts or merge code from several Bamboo plans into one Bamboo Spec project.

Best practices

There's a couple of things that we find important.

On this page

Related links

- Tutorial: Create a simple plan with Bamboo Java Specs
- Create a Bamboo Specs project using Maven Archetype

Keep your project in version control system

Keeping your code in VCS, such as Git or Mercurial will give you a lot of benefits, such as:

- traceability - you can track who, when and why changed your build environment
- comparison - you can easily compare two different configurations to analyse what changed, which helps when troubleshooting
- separation of work - you can use branches to prepare new versions of your environment
- remaining in sync with the product.

Keep information from which URL a given plan was created

With hundreds of plans and separate configuration projects it's very valuable to know which plan is being managed by CASC and which via UI, and which repository manages a given plan in Bamboo.

To stay on top of things, use plan description field for this purpose, e.g.:

"This plan is being managed via configuration-as-code. Modify <vcs url> project to update the plan."

Keep in mind that your plan configuration is source code as any other

Apply all best development practices when maintaining it. Avoid copy-and-paste, extract common methods or components, use parameterization, modularization.

Go back: Bamboo Specs

Tutorial: Create a simple plan with Bamboo Java Specs

This guide will help you understand how to create plans with Bamboo Java Specs. You'll create a simple project and execute it to create a plan in Bamboo.

On this page

Related links

- Bamboo Specs reference
- Bamboo Specs API reference
- Create a Bamboo Specs project using Maven Archetype

Before you begin

Make sure you have the following installed:

- JDK 8 or higher
- Maven 3.2 or higher
- Eclipse or IntelliJ IDEA IDE

Note: You may use a different IDE, but tutorial provides examples for the two above

- Bamboo 6.0 or higher

Step 1: Create a project base with Maven

To create a base of the project, execute the following Maven archetype:

```
mvn archetype:generate -B \
    -DarchetypeGroupId=com.atlassian.bamboo -DarchetypeArtifactId=bamboo-specs-archetype \
        -DarchetypeVersion=6.2.1 \
    -DgroupId=com.atlassian.bamboo -DartifactId=bamboo-specs -Dversion=1.0.0-SNAPSHOT \
    -Dpackage=tutorial -Dtemplate=minimal
```

where:

Property	Description

• *archetypeGroupId* • *archetypeArtifactId* • *archetypeVersion*	Maven uses a set of identifiers to uniquely identify a project and specify how the project artifact should be packaged: • archetypeGroupId - Bamboo Specs archetype's groupId. Must be set to `com.atlassian.bamboo` • archetypeArtifactId - Bamboo Specs archetype's artifactId. Must be set to `bamboo-specs-archetype` • archetypeVersion - Bamboo Specs archetype's version. It should match the version of Bamboo you're using.
• *groupId* • *artifactId* • *version*	Your Project identifiers: • groupId - ID of the project's group (eg. base name of your company); it can have an arbitrary value; • artifactId - ID of the project's artifact; it should be set to `bamboo-specs`, so the project will be created in the *bamboo-specs* directory; you can change this value, but then you must manually rename the output directory to *bamboo-specs;* • version - version of your project; it can have an arbitrary value.
• *package*	Prefix of the Java package that you want to use for the project.
• *template*	Type of code in which the project is generated. For the purpose of this tutorial we're using *minimal*.

The project is created in the `bamboo-specs` directory:

```
cd bamboo-specs
```

Step 2: Import the project into IDE

You can now import the project into your integrated development environment (IDE).

1. Run Eclipse.
2. In the main menu, go to **File** > **Import**.
3. In the **Import** dialog, select **Maven** > **Existing Maven Projects**.
4. Select **Next**.
5. Select **Browse.**
6. Select the `bamboo-specs-tutorial` directory, and select **Open.**
7. Select **Finish**.

Eclipse will create a new project and download necessary dependencies (it may take a while).

1. Run IntelliJ IDEA.
2. In the main menu, go to **File** > **Open**.
3. Select the `pom.xml` file.
4. Select **Open as project**.

IntelliJ IDEA will create a new project and download necessary dependencies, which might take a while.

If you want to see how PlanSpec.java files is structured, go to src/main/java/tutorial/PlanSpec.java. Your file should have the following structure:

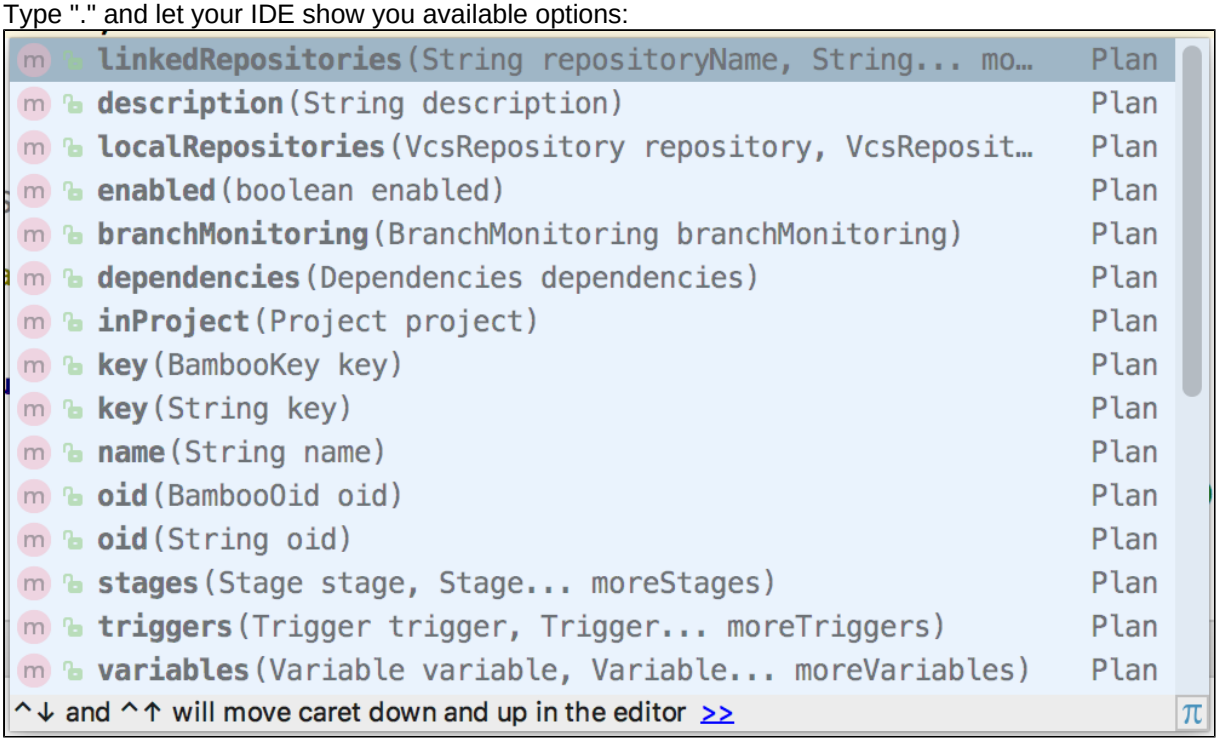
PlanSpec	The name of the class. You can use any class name.
BambooSpec	The file is annotated. The annotation is used by the *spec-runner* Maven plugin to find classes containing Bamboo plans.
main	With the *main* method you can run the project as any other Java application.
BambooServer	The project uses the *BambooServer* class to publish plans with password authentication. The username and password are read from the `.credentials` file which is located in the current working directory.

Step 3: Define a job with a script task

1. In the **createPlan** method put a cursor after the **description** method call:

```
private Plan createPlan() {
    return new Plan(
            project(),
            "Plan Name", "PLANKEY")
            .description("Plan created from (enter repository url of your plan)");
}
```

2. Type "." and let your IDE show you available options:



3. Select the **stages** method and add the **new Stage ("Stage 1")** constructor call inside the method's argument (you need to add the import statement for Stage class).
4. Add a job to the stage using the **jobs** method and **new J o b()** constructor (add the import statement too). Name the job **Build & run** and use **RUN** for a key:

```
private Plan createPlan() {
    return new Plan(
            project(),
            "Plan Name", "PLANKEY")
            .description("Plan created from (enter repository url of your plan)")
            .stages(
                    new Stage("Stage 1")
                            .jobs(new Job("Build & run", "RUN")));
}
```

5. Let's add a task to the job. Type **.tasks()** and declare a new **ScriptTask()** inside as shown below (add the import statement too). Call **.inlineBody** on the **ScriptTask()**.

```
private Plan createPlan() {
    return new Plan(
            project(),
            "Plan Name", "PLANKEY")
            .description("Plan created from (enter repository url of your plan)")
            .stages(
                    new Stage("Stage 1")
                            .jobs(new Job("Build & run", "RUN")
                                    .tasks(
                                            new ScriptTask().inlineBody("echo Hello world!"))));
}
```

You can always open a JavaDoc dialog to learn more about given method or class:

To download JavaDocs, hold the **Ctrl/Cmd** key, place the mouse cursor over a method or class name, and select **open declaration**.

It opens a source editor with a decompiled class file. Eclipse immediately starts downloading sources and JavaDoc JARs in the background and updates editor as soon as it completes.

To display JavaDocs place the mouse cursor over a class or method name.
To download JavaDocs hold the **Ctrl/Cmd** key and select the method or class name.

It opens a source editor with the decompiled class file. Select the **Download sources** link. IDEA will download sources and JavaDoc JARs.

To display JavaDocs place the cursor over a method or class name and press **Ctrl+J** to open a quick documentation pop-up.

Step 4: Validate Bamboo Specs offline

You can perform offline validation before deploying a plan to Bamboo. Let's try it out by running a unit test.

```
mvn test
```

1. In the **Package Explorer** view, right-click on `src/test/java/tutorial/PlanSpecTest.java`.
2. Select **Run as** > **JUnit test**.

1. In the **Project** view, right-click on the *src/test/java/tutorial/PlanSpecTest* class.
2. Select **Run 'PlanSpecTest'** .

Test fails with the following stack trace:

```
com.atlassian.bamboo.specs.api.exceptions.PropertiesValidationException:
Plan or job / Name: can not contain any of those characters: [", &, ', <, >, \] but it is 'Build & run'
    at com.atlassian.bamboo.specs.api.validators.common.ImporterUtils.checkNoErrors(ImporterUtils.java:
44)
    ...
    at tutorial.PlanSpec.createPlan(PlanSpec.java:42)
    at tutorial.PlanSpecTest.checkYourPlanOffline(PlanSpecTest.java:12)
```

where:

`Plan or job / Name`	A path to an invalid element

`can not contain ...`
 	
`but it is ...`
	Expected and actual value

`PlanSpec.java:42`	The source line containing the error

As you can see, the validation fails because the name of the job contains an invalid **&** character. Let's remove it. Your code should look like this now:

```
private Plan createPlan() {
    return new Plan(
            project(),
            "Plan Name", "PLANKEY")
            .description("Plan created from (enter repository url of your plan)")
            .stages(
                    new Stage("Stage 1")
                            .jobs(new Job("Run", "RUN")
                                    .tasks(
                                            new ScriptTask().inlineBody("echo Hello world!"))));
}
```

Run the test again to make sure that it passes this time.

Step 5: Publish Bamboo Specs to the Bamboo server

- Make sure that your Bamboo instance is up and running.
- If you're not running Bamboo on your local machine (http://localhost:8085), change the *bambooUrl* variable in the *main* method.
- We're assuming that you have an **administrator** account with username **admin** and password **admin**. If you want to use other credentials, you need to update the `.credentials` file located in the root directory of the project.

When you're sending a plan, Bamboo validates it.

The pom.xml contains the publish-specs profile which executes the *spec-runner* Maven plugin. So just type:

```
mvn -Ppublish-specs
```

1. In the **Package Explorer** view, right-click on the PlanSpec class.
2. Select **Run as** > **Java application**.

1. In the **Project** view, right-click on the **PlanSpec** class.
2. Select **Run 'PlanSpec.main()'**.

The console output looks like this:

```
Publishing plan PLANKEY
Result OK: http://localhost:8085/browse/PRJ-PLANKEY
```

⚠ For more verbose logging, add -Dbamboo.specs.log.level=DEBUG program argument when running Bamboo Specs.

Step 6: Check the results

1. Go to your Bamboo instance.
2. Open the plan that you created.
3. Go to **Actions** > **Configure plan**.
4. Check whether the stage contains a job with the *Hello world* Script task.
5. Select **Run** > **Run plan** to execute the build.
6. Find the "Hello World!" message in the logs.

Having configuration written as code using Bamboo Specs you can very easily manage all your build plans in Bamboo.

This is a very convenient method of managing large Bamboo instances with huge number of plans, publishing plans on Bamboo test instances before promoting changes to production, and tracking configuration changes in version control system.

Read What is Configuration as Code? to learn more about the benefits of using Bamboo Specs.

Next steps

Here are some resources that can help you with writing your own Bamboo Specs:

- Create a Bamboo Specs project using Maven Archetype
- Exporting existing plan configurations to Bamboo Specs
- Creating deployment projects in Bamboo Specs
- Bamboo Specs reference
- Bamboo Specs API reference

Tutorial: Bamboo Java Specs stored in Bitbucket Server

This guide will show you how you can store Bamboo Specs in a Git repository on Bitbucket Data Center and Server. This approach lets you automatically build and execute Bamboo Specs on every push you make to a Git repository.

Before you begin

- Make sure you have the required software installed:
  - Bamboo 6.2 or laterCreate a simple plan with Bamboo Java Specs
  - Bitbucket Data Center and Server 7.5 or later
  - JDK 8 or higher
  - Maven 3.2 or higher
- Set up an application link between Bamboo and Bitbucket Data Center and Server. See Integrating Bamboo with Bitbucket Data Center.
- If you're not familiar with Bamboo Specs, make sure you read our introductory tutorial: Tutorial: Create a simple plan with Bamboo Java Specs.

Step 1: Create a Git repository in Bitbucket Sever and clone it locally

1. In Bitbucket Data Center and Server, open the **Projects** page.
2. Select **Create project**.
3. Enter `Bamboo` for project name and key and select **Create project**.
4. Select **Create repository**.
5. Give your new repository the name `tutorial` and select **Create repository**.

You've just created a new empty repository. Use the *git clone* command to create a clone on your computer. For example:

```
git clone http://admin@localhost:7990/scm/bamboo/tutorial.git
cd tutorial
echo "Hello Bamboo Specs" > greet.txt
git add greet.txt
git commit -m "Setup master branch"
git push
```

Step 2: Create a linked repository in Bamboo

1. Open Bamboo and go to  > **Linked repositories**.
2. Select **Add repository.**
3. Select a **Bitbucket Server/Stash** repository type.
4. Select a name for your repository.
5. From the Server dropdown, select your Bitbucket Data Center and Server.
6. In the **Web repository** section, select the `Bamboo / tutorial` repository from the Web repository drop-down.
7. Select **Save repository**.

Your new repository is created and you can start using it in Bamboo.

Step 3: Enable processing of Bamboo Specs in your repository

By default, Bamboo won't look for Bamboo Specs in the Git repository until you explicitly tell it to do so. Let's do it now:

1. Go to  > **Linked repositories**.

2. Select your repository.
3. In the Bamboo Specs tab, enable **Scan for Bamboo Specs**.

> In this tutorial we simply grant access to all projects in the Bamboo instance. You can fine-tune project access. See Enabling repository-stored Bamboo Specs.

Now, Bamboo is ready to execute Bamboo Specs when the relevant code it committed to the repository. Let's create some code.

Step 4: Create Bamboo Specs project using Maven

1. Go to the empty Git repository you cloned in step 1:

```
cd tutorial
```

2. Use the Maven archetype to create a project template. For the purpose of this tutorial, type:
Note: You must create Bamboo Specs in the `bamboo-specs` directory, under the repository root.

```
mvn archetype:generate -B \
  -DarchetypeGroupId=com.atlassian.bamboo -DarchetypeArtifactId=bamboo-specs-archetype \
  -DarchetypeVersion=6.2.1 \
  -DgroupId=com.my.company -DartifactId=bamboo-specs \
  -Dversion=1.0.0-SNAPSHOT -Dpackage=com.my.company
```

where:

Property	Description
`archetypeGroupId`	Bamboo Specs archetype's groupId. Must be set to `com.atlassian.bamboo`
`archetypeArtifactId`	Bamboo Specs archetype's artifactId. Must be set to `bamboo-specs-archetype`
`archetypeVersion`	Bamboo Specs archetype's version. It should match the version of Bamboo you're using.
`groupId`	ID of the project's group (eg. base name of your company); it can have an arbitrary value.
`artifactId`	ID of the project's artifact; it should be set to `bamboo-specs`, so the project will be created in the *bamboo-specs* directory; you can change this value, but then you must manually rename the output directory to *bamboo-specs*.
`version`	The version of your project; it can have an arbitrary value.
`package`	The prefix of the Java package that you want to use for the project.

Your project is created. You can open it in an IDE, such as Eclipse or IDEA, if you want to see how the project is set up. For more information on the code structure, take a look at our tutorial: Create a simple plan with Bamboo Specs.

Step 5: Commit and push code changes to Bitbucket Data Center and Server

1. Create a new project in Bamboo with the name "**Project Name**" and key "**PRJ**".
2. Go to **Project settings** > **Bamboo Specs repositories** and select the linked repository created in Step 2.

3. Add created `bamboo-specs` directory to VCS and push changes to the server:

```
git add bamboo-specs
git commit -m "Initial commit of Bamboo Specs"
git push
```

As soon as you push your code changes to Bitbucket Data Center and Server, Bamboo will get notified about a new commit available.

Bamboo will checkout your project, compile it, and execute Bamboo Specs in a sandbox environment.

Execution of Bamboo Specs will create or update configuration of plans or deployment projects accordingly.

Step 6: Check to see if the plan was created

1. Open your Bamboo instance.
2. From the header, select **Build** > **All build plans**.
3. Open the project and plan you've just created.

> All configuration options are disabled because entire plan configuration is now managed by Bamboo Specs from your Bitbucket repository.

4. Select **Run plan** to execute the build.
5. Find the "Hello World!" message in the logs.

> Regardless whether your Bamboo Specs were processed successfully or not, you'll receive an email with status of you your Bamboo Specs execution.

Next steps

Here are some resources that can help you with writing your own Bamboo Specs:

- Create a Bamboo Specs project using Maven Archetype
- Exporting existing plan configurations to Bamboo Specs
- Creating deployment projects in Bamboo Specs
- Bamboo Specs reference

Bamboo YAML Specs

As an alternative to using Bamboo Java Specs, Bamboo 6.3+ allows you to create simple plans using Bamboo YAML Specs in no time. Just use one of the templates we provide and you're ready to start committing your files to a repository.

How it works

Bamboo YAML Specs is another Bamboo Specs format supported by Bamboo next to Bamboo Specs Java. Once you have defined your plan/deployment configuration in YAML, you need to allow your repository to scan for Bamboo Specs.

Bamboo always looks for YAML Specs first. If it doesn't find YAML Specs, Bamboo tries to find and process Java Specs.

In the repository you specified, Bamboo looks for `bamboo-specs/bamboo.yml` or `bamboo-specs/bamboo.yaml` files to fetch your configuration. Linked repository needs to have permissions to create plans within given project in order to process YAML definition and create a plan.

A new plan created using Bamboo YAML Specs does not have any explicit permissions granted - administrators and logged in users will have access to it. You can use YAML Specs to define plan-level permissions. We advise you to use project-level permissions. Contrary to Java Specs, YAML Specs can't create projects.

In Bamboo YAML Specs, artifacts are shared by default. Artifacts are downloaded between stages, e.g user Defines Stage 1 with Artifact A, Stage 2 with Artifact B and Stage 3. This means that Stage 2 will download Artifact A from Stage 1 and Stage 3 will download both Artifact A and B from previous stages.

In Bamboo YAML Specs, artifacts are also required by default. This means that a build fails if the artifact can't be published.

YAML Format

Bamboo Specs YAML allows for a multiple entities definition in a single YAML file and accepts the following format:

```
---
version: 2
plan:
  project-key: MARS
  key: ROCKET
  name: Build the rockets

# List of plan's stages and jobs
stages:
  - Build the rocket stage:
    - Build

#Job definition
Build:
  tasks:
    - script:
        - mkdir -p falcon/red
        - echo wings > falcon/red/wings
        - sleep 1
        - echo 'Built it'
    - test-parser:
        type: junit
        test-results: '**/junit/*.xml'
  # Job's requirements
  requirements:
     - isRocketFuel
  # Job's artifacts. Artifacts are shared by default.
  artifacts:
     - name: Red rocket built
       pattern: falcon/red/wings
```

```
---
version: 2
deployment:
  name: Deploy Rocket
  source-plan: MARS-ROCKET

release-naming:
  next-version-name: 0.${bamboo.buildNumber}

environments:
  - QA

QA:
  tasks:
    - clean
    - artifact-download:
        destination: /
    - script:
        - echo 'Hello space'
```

Default settings

Here's the default settings of plans created using YAML Specs.

These settings cannot be changed:

- notifications are sent to committers and watchers of the plan when plan fails
- default repository is one with YAML Specs file. All plan branches will use it.

These settings can be changed:

- Bamboo YAML plan check-out starts with checking out a repository in which it's defined. But if first task is checkout task which uses another linked repository then checkout from repo with YAML Specs file will be skipped.
- artifacts sharing is turned on
- YAML plans use Bitbucket Server triggers
- only Bamboo administrators can see plan configuration and run builds

- logged in users can view plan builds
- plan branches are created automatically with plan brach expiry set for 30 days. These settings are configurable.

Read more

- Bamboo Specs Reference

Bamboo Specs YAML format

Bamboo Specs YAML allows for a multiple entities definition in a single YAML file and accepts the following format:

```
---
version: 2
plan:
  project-key: MARS
  key: ROCKET
  name: Build the rockets

# List of plan's stages and jobs
stages:
  - Build the rocket stage:
    - Build

#Job definition
Build:
  tasks:
    - script:
        - mkdir -p falcon/red
        - echo wings > falcon/red/wings
        - sleep 1
        - echo 'Built it'
    - test-parser:
        type: junit
        test-results: '**/junit/*.xml'
  # Job's requirements
  requirements:
     - isRocketFuel
  # Job's artifacts. Artifacts are shared by default.
  artifacts:
     - name: Red rocket built
       pattern: falcon/red/wings
```

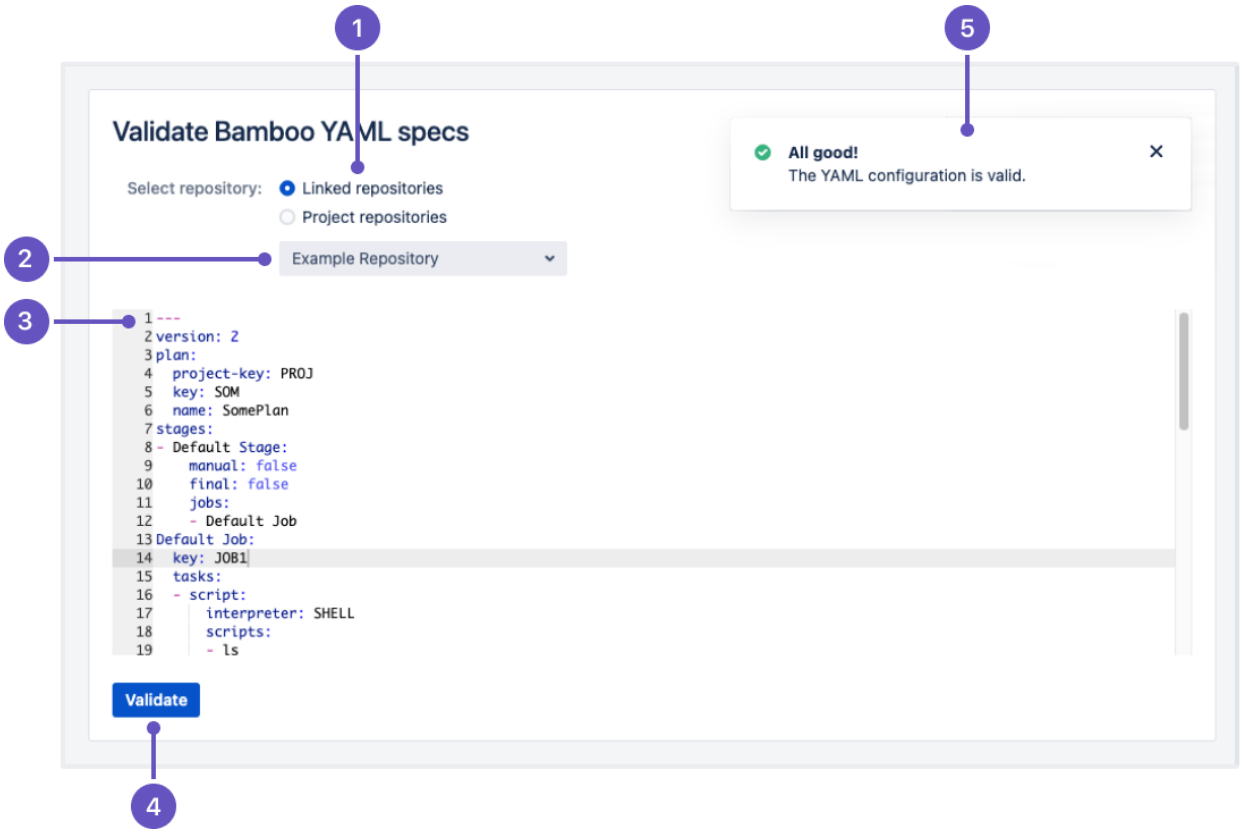
```
---
version: 2
deployment:
  name: Deploy Rocket
  source-plan: MARS-ROCKET

release-naming:
  next-version-name: 0.${bamboo.buildNumber}

environments:
  - QA

QA:
  tasks:
    - clean
    - artifact-download:
        destination: /
    - script:
        - echo 'Hello space'
```

Validating YAML Specs

Instead of verifying or troubleshooting your Bamboo YAML Specs configuration by repeatedly pushing changes to the source code, you can use the built-in validator utility. The utility analyzes your Bamboo YAML Specs configuration against a linked or project-specific repository and checks for syntax errors following the Bamboo YAML Specs format.

On this page:

The validator is available from the **Specs** menu in Bamboo's top navigation bar. Here's what it looks like:



1. Source code repository type selector
2. Repository selection menu
3. YAML Specs input field
4. **Validate** button
5. Result flag

How to use the Bamboo YAML Specs validator

To validate your Bamboo YAML Specs configuration:

1. From the source code repository type selector, select either **Linked repositories** or **Project repositories**.

   If you see the *No available repositories* message, it means that Bamboo couldn't find any repositories with Specs enabled or you don't have permission to access them.

2. From the repository selection menu, select the repository to validate your YAML Specs against.
3. Paste your YAML Specs configuration into the YAML Specs input field.
4. Select **Validate**.

Results

If your configuration is valid, you'll see the following notification:

> ***All good!***
>
> *The YAML configuration is valid.*

Otherwise, you'll see the following error message followed by the details of what went wrong:

> ***The YAML configuration is invalid***
>
> *There's something wrong with the YAML configuration. Fix the errors and try again.*
>
> *(…)*

In that case, you'll need to fix the errors identified during validation and retry.

Tutorial: Bamboo Specs YAML stored in Bitbucket Server

This guide will show you how you can store Bamboo Specs in a Git repository on Bitbucket Data Center and Server. This approach allows to automatically build and execute Bamboo Specs on every push you make to a Git repository.

Related links

- Link to Bamboo Specs YAML reference

Before you begin

- Make sure you have the following software installed:
  - Bamboo 6.9 or later
  - Bitbucket Data Center and Server 7.5 or later
- Set up an application link between Bamboo and Bitbucket Data Center and Servre. See Integrating Bamboo with Bitbucket Data Center.

Step 1: Create a Git repository in Bitbucket Data Center and Server and clone it locally

1. In Bitbucket Data Center and Server, open the **Projects** page.
2. Click **Create project**.
3. Select `Bamboo` for project name and key and click **Create project**.
   You see that the project has no repositories.
4. Click **Create repository**.
5. Give your new repository the name `tutorial`, and click **Create repository**.

You've just created a new empty repository. Use the *git clone* command to create a clone on your computer. For example:

```
git clone http://admin@localhost:7990/scm/bamboo/tutorial.git
```

Step 2: Create Bamboo Specs YAML config

1. Go to the empty Git repository you cloned in step 1:

   ```
   cd tutorial
   mkdir bamboo-specs
   cd bamboo-specs
   echo > bamboo.yml
   ```

2. Use any of the templates we've prepared for your in Bamboo YAML Specs Reference or write YAML definition on your own with Bamboo Specs YAML format./

   ⚠ It's important to save your Bamboo Specs YAML definition in the ${repo-home}/bamboo-specs /bamboo.yml or bamboo.yaml file under the repository root.

```
---
version: 2
plan:
  project-key: MARS
  key: ROCKET
  name: Build the rocket
stages:
  - Build hull:
    - Build

Build:
  tasks:
    - script:
      - echo 'Hello World!'
```

3. Add created `bamboo-specs` directory to VCS and push changes to the server:

```
git add bamboo-specs
git commit -m "Initial commit of Bamboo Specs"
git push
```

Step 3: Create a new Project in Bamboo

1. Open Bamboo and go to **Create** > **Create Project**.
2. Fill in Project name, e.g. **Mars**.
3. If not auto-generated, fill in the Project Key – eg: **MARS** – this will be referenced in the YAML file.
4. Click **Save**.

Step 4: Enable processing of Bamboo Specs in your repository

By default Bamboo will not look for Bamboo Specs in the Git repository until your explicitly tell it to do so. Let's do it now:

1. Go to **Specs** > **Set up Specs repository**.
2. Select your project **Mars**.
3. Select **Link new repository**
4. Select a **Bitbucket Server / Stash** repository type.
5. Choose a name for your repository.
6. From the Server drop-down, select your Bitbucket Data Center and Server.
7. Select the `Bamboo / tutorial` repository from the Repository drop-down.
8. Click **Confirm**.

Your new repository is created and you can start using it in Bamboo. Bamboo will checkout your repository, process it and create plan.

Execution of Bamboo Specs will create or update configuration of plans accordingly.

In this tutorial we simply grant access to one project in the Bamboo instance. You can fine-tune project access, see Enabling repository-stored Bamboo Specs how to do this.

Step 5: Check if plan was created

1. In Bamboo from the header, select **Build > All build plans**.
2. Open the project and plan you've just created.

> All configuration options are disabled because entire plan configuration is now managed by Bamboo Specs from your Bitbucket repository.

3. Click **Run plan** to execute the build.
4. Find the "Hello World!" message in the logs.

> Regardless whether your Bamboo Specs were processed successfully or not, you'll receive an email with status of you your Bamboo Specs execution.

Next steps

Here are some resources that can help you with writing your own Bamboo YAML Specs:

- Bamboo YAML Specs Reference

Bamboo Specs reference documentation

We're still working on our documentation, but progress is more important than perfection, so we're sharing the first versions with you.

On this page

Related links

- Tutorial: Create a simple plan with Bamboo Java Specs
- Create a Bamboo Specs project using Maven Archetype
- Best practices

Bamboo Specs reference

This documentation set explains main concepts behind Bamboo Specs. We've added a lot of examples for you to use in your configuration files.

Bamboo Specs reference

Bamboo Specs API reference

Have a look at our Bamboo Specs API docs for the details of packages and classes that you can use.

Bamboo Specs API reference

Bamboo Specs troubleshooting

We've gathered answers to the most common problems with Bamboo Specs.

On this page:

I committed Bamboo Specs but nothing happened

This may happen for a number of reasons. Here's how to resolve this problem:

- Make sure you pushed your code to Bitbucket Server. It's trivial, but it really happens.
- Make sure that Bitbucket Server and Bamboo are connected with an application link. See Linking to another application.
- Make sure that Bamboo Specs processing is enabled in Bamboo. See Repository-stored Bamboo Specs security.
- Make sure that Bamboo Specs are enabled for a repository you pushed to. See Enabling repository-stored Bamboo Specs.
- Make sure that Bamboo Specs have access to projects or deployment projects you want to modify. To do this, select the **Access all projects** checkbox in your Bitbucket Server linked repository settings. See Enabling repository-stored Bamboo Specs.

Bamboo Specs compilation fails

In case the compilation fails during the first execution of Bamboo Specs, no plans or deployment projects are created or updated. As a consequence, Bamboo is unable to associate the Bamboo Specs with any plan, so you won't find an error log in any of the existing plans.

In case compilation fails not the first time, you can find the Specs execution error on the build results page for the related plans.

> In both cases, a committer of the change will receive an email with details of the error.

Compilation may fail due to errors in the source code or because the `pom.xml` file has been sanitized by Bamboo.

Errors in the source code

Check out the repository on your computer and build it yourself (use the `mvn compile` command or import it into your IDE) to locate the error. These are usually typos in the code, wrong project dependencies, or an outdated parent `pom.xml` version.

The pom.xml file has been sanitized by Bamboo

By default, Bamboo Specs are executed using the default `<BAMBOO_INSTALL_DIR>/atlassian-bamboo/WEB-INF/classes/bamboo-specs-pom.xml` file. If you've added extra dependencies to the `pom.xml` file in your Bamboo Specs project, Bamboo will copy those dependencies to the default file and skip all other tags. Additionally, Bamboo prints the content of the effective `pom.xml` file to the Specs execution log for troubleshooting.

However, even if your Bamboo Specs project compiles and runs correctly on your computer, it may fail in Bamboo. If that's the case, make sure that:

- you're not using any extra plugins and you're not relying on executing tests; it's enough to compile your project using `maven-resources-plugin` and `maven-compiler-plugin` only
- your `pom.xml` inherits from `com.atlassian.bamboo:bamboo-specs-parent`

Furthermore, you can disable the sanitization of the pom.xml file by setting the following system property:

```
-Dbamboo.repository.stored.specs.pom.sanitization.enabled=false
```

Learn how to configure system properties

Bamboo Specs compilation succeeded but the log shows that not everything was built

The repository-stored Bamboo Specs feature requires that your project in the `/bamboo-specs` directory consists of only one Maven module. We don't support multi-module builds. So in case you put some Bamboo Specs classes in sub-modules of `bamboo-specs`, they simply won't be built.

Bamboo Specs fail to import configuration

The committer of the change will receive an email with details of the error. You can also look for Specs execution errors on the build results page of the related plan or plans. The most typical reasons are:

Insufficient permissions

Make sure that Bamboo Specs has access to projects or deployment projects you want to modify. It applies also to dependent projects, for instance:

- child plans triggered after a build - see plan's dependencies section,
- artifacts downloaded from a plan from another project.

Validation errors

Bamboo validates your plan configuration for correctness. It reports an error if any constraint is violated, such as an invalid project key, a reference to a non-existing repository or plan, etc.

Incompatible versions

The Bamboo Specs version in your `pom.xml` needs to match the Bamboo Server version of Bamboo Specs. If an error occurs, update your `pom.xml` Bamboo Specs version.

The repository in which you defined a plan has no permissions to access a project

1. Go to  > **Build resources** > **Linked repositories**.
2. Select your repository.
3. Select the **Bamboo Specs** tab.
4. Copy the webhook URL.
5. In the repository you want to use for storing Bamboo Specs, go to your repository settings.
6. Find webhook-specific configuration.
7. Paste in the URL Bamboo provided you with.

A webhook to allow your repository to communicate with Bamboo isn't set up

1. From the Bamboo header select **Projects**, and find your project.
2. Select **Project settings** > **Bamboo Specs repositories**.
3. Select your repository and select **Add**.
4. Go to the settings of the repository you want to use for storying Bamboo Specs.
5. Find webhook-specific configuration.
6. Paste in the URL Bamboo provided you with.

Bamboo can't connect to Docker to execute Bamboo Specs

Make sure that Docker is running. Alternatively, disable Bamboo Specs processing in Docker security settings, see Repository-stored Bamboo Specs security.

Bamboo Specs - supported scenarios

Bamboo Specs provides the capability to programmatically configure Bamboo using either Java or YAML.

Bamboo Specs typical use case scenarios:

- Migration of existing plans that were created via the Bamboo UI
- New plan configurations created with Bamboo Specs directly and that contain little if any glue code.
- New plan configurations created with Bamboo Specs, or existing plans that are migrated to Specs, which contain a significant amount of glue code which is complex.

What Atlassian support can help you out with:

- Migration of existing plans that were created using the UI to Bamboo Specs with minimal glue code, preferably with minor if any changes to the code that is provided by the View Plan as Specs feature
- New plan configurations that are created with Bamboo Specs which contain a minimal amount of glue code, similar to what is generated by View Plan as Specs.

What is glue code?

By *glue code* we understand the Java code that is not part of the Bamboo Specs API and which is necessary for the Specs Java code to run as a Java application. The glue code can be minimal or implement extra functionality using heavy code edits and possibly include package/project structure changes

Bamboo Specs projects that contain a large amount of complex glue code fall outside supported scenarios. For such cases, we recommend starting from simple Bamboo Specs, preferably produced by the View Plan as Specs with a minimal amount of glue code. Then gradually add the complex code and perform as many intermediary tests as possible to be able to detect errors sooner rather than later.

Below is a list of Atlassian validated Bamboo Specs code:

- code generated by View Plan as Bamboo Specs from within the Bamboo UI
- code from the official documented tutorials
- sample code snippets described in the latest Bamboo Specs reference, which show the use of the API.

Audit log for plans managed repository-stored Bamboo Specs

When plan is managed by Repository Stored Specs, Bamboo doesn't keep individual property changes on plan configuration caused by repository commits. Instead of it record at plan's Audit log is placed

```
Plan has been updated using Repository Stored Bamboo Specs and the VCS revision HASH_ID
```

This behaviour might be changed by System property.

```
bamboo.rss.audit.logs.disabled = false
```

Pay attention that setting this value to **false** might cause performance issues for large Bamboo instances with massive plan updates by Repository Stored Specs.

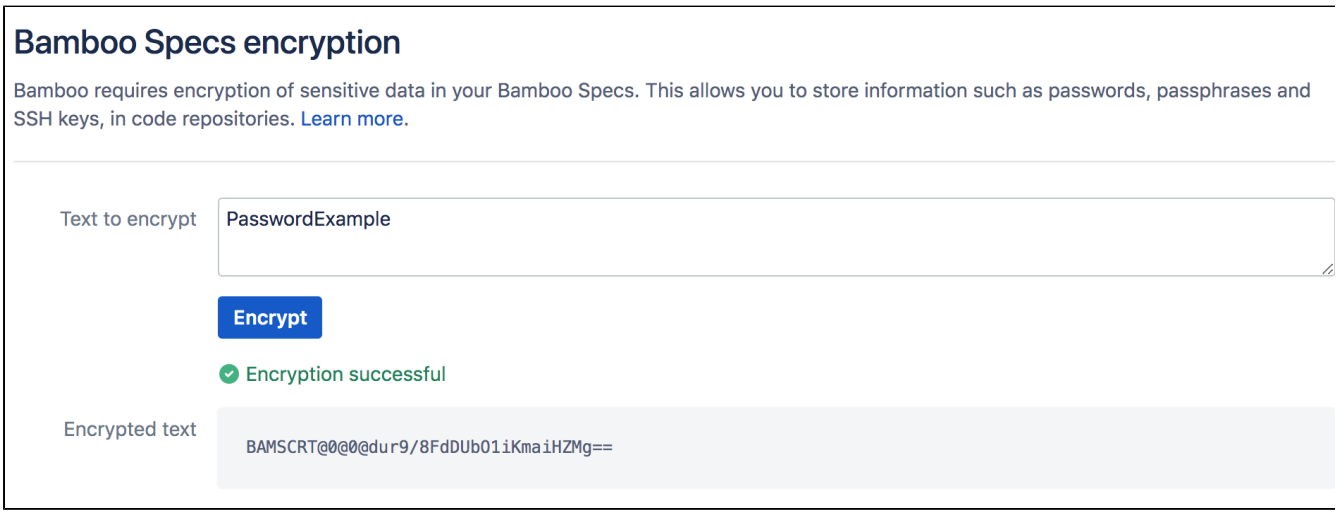
Bamboo Specs encryption

When working on your repository-stored Bamboo Specs, you might need to use sensitive data like passwords, pass phrases, or other access credentials. To make sure your data remains secure, you can use Bamboo sensitive data encryption before storing the confidential information in your code repository.

To encrypt your data

1. From the top navigation bar, select **Specs** > **Sensitive data encryption**.
2. Paste the content you want to encrypt in the text box.
3. Click **Encrypt**.

Your content is now encrypted. You can copy it and use it in your Bamboo Specs safely.



Where to use the encrypted data

You can use the manually encrypted data in your Bamboo Specs for configuring access to repositories, defining secret plan variables, working with credentials in various tasks, and in a handful of other places. Refer to the documentation or JavaDocs of individual locations to learn whether encrypted content is supported.

> Secret variables are determined by their name. To be considered secret and to support encrypted content, a variable name needs to contain one of the following words: "password", "passphrase", "secret", "sshkey".

Troubleshooting
If you're using Bamboo 6.9 or later, sensitive data encryption is enabled by default. If you can't see it, contact your administrator.

If you're a Bamboo administrator, you can enable/disable and configure the sensitive data encryption feature.

Go to  > **Security** > **Security settings** and change the **System-wide encryption** option.

Repository-stored Specs thread permission

Symptom

By **design of thread security**, Bamboo does not permit the access of thread externally.

When a repository stored spec tries to perform an operation that requires thread access, an exception similar to this is thrown.

Exception

```
oracle.jdbc.driver.OracleDriver registerMBeans
WARNING: Error while registering Oracle JDBC Diagnosability MBean.java.security.AccessControlException:
access denied ("javax.management.MBeanServerPermission" "createMBeanServer")
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
        at java.security.AccessController.checkPermission(AccessController.java:884)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
        at com.atlassian.bamboo.specs.maven.sandbox.AbstractThreadPermissionVerifier.checkPermission
(AbstractThreadPermissionVerifier.java:18)
        at com.atlassian.bamboo.specs.maven.sandbox.BambooSpecsSecurityManager.checkPermission
(BambooSpecsSe      curityManager.java:37)
        at java.lang.management.ManagementFactory.getPlatformMBeanServer(ManagementFactory.java:465)
        at oracle.jdbc.driver.OracleDriver.registerMBeans(OracleDriver.java:365)
        at oracle.jdbc.driver.OracleDriver$1.run(OracleDriver.java:241)
        at java.security.AccessController.doPrivileged(Native Method)
        at oracle.jdbc.driver.OracleDriver.<clinit>(OracleDriver.java:237)
        at Main.main(Main.java:43)
```

Workaround

Bamboo Repository-Stored Specs can be set to not use the Security Manager using the following JVM argument:

-Dbamboo.repository.stored.specs.security.manager.enabled=false

Bamboo FAQ

Bamboo FAQ

Answers to commonly raised questions about configuring and using Bamboo:

- Glossary
  - activity log
  - agent
  - agent-specific capability
  - artifact
  - authors in Bamboo
  - build
  - build activity
  - build duration
  - build log
  - build queue
  - build result
  - build telemetry
  - capability
  - child
  - committer
  - custom capability
  - default repository
  - elastic agent
  - elastic Bamboo
  - elastic block store
  - elastic image
  - elastic instance
  - executable
  - favorites
  - global permission
  - job
  - label
  - local agent
  - parent
  - permission
  - plan
  - plan permission
  - projects in Bamboo
  - queue
  - reason
  - remote agent
  - remote agent supervisor
  - requirement
  - shared capability
  - stage
  - Stock images
  - task
  - triggering
  - watcher
- Contributing to the Bamboo documentation
- How to Prevent Password Auto-completion in Bamboo

Need more help?

Do you have a question, or need help with Bamboo? Please create a support request.

Browse our Bamboo Developer FAQ.

You may also like to check out the forums:

- Bamboo General Forum
- Bamboo Developers Forum

Usage FAQ

Can multiple plans share a common 3rd-party directory

For example, you might have three repository directories, say, A, B, and C, where A is a common 3rd-party library. A is used across projects.

At this stage, Bamboo doesn't support having multiple checkout directories per build plan. However, you can work around this by setting these three directories up as separate Bamboo build plans - P_A, P_B and P_C.

To make this work, you will also need to specify as an argument to your build scripts for P_B and P_C the location of A, which will be something like this:
../Plan_key_for_A/

Using a set up like this, your library module (A) should only be checked out once across the Bamboo instance.

See also:
Triggering a build when another build finishes

Changing Bamboo database settings

The Bamboo database configuration is persisted in the `<Bamboo-Home>/bamboo.cfg.xml` file. You can change the database settings by editing this file, as detailed in the instructions below:

Changing the Bamboo database username and password.

If you want to change the database username and password, edit the following line,

```
    <property name="hibernate.connection.password">YOUR_PASSWORD</property>
    <property name="hibernate.connection.username">YOUR_USERNAME</property>
```

Changing the Bamboo database URL

If you want to change the database URL, edit the following line,

```
    <property name="hibernate.connection.url">DATABASE_URL</property>
```

⚠️ *You need to restart the Bamboo application server for the changes to take effect. If you have any elastic agents running, ensure that they are shut down before you restart the Bamboo server. If you do not shut down your elastic instances before restarting, they will continue to run and become orphaned from your Bamboo server.*

Deploying Multiple Atlassian Applications in a Single Tomcat Container

Deploying multiple Atlassian applications in a single Tomcat container is **not supported**. We do not test this configuration and upgrading any of the applications (even for point releases) is likely to break it. There are also a number of known issues with this configuration:

- You may not be able to start up all of the applications in the container, due to class conflicts (in 3rd party libraries bundled with our application) that result from the Atlassian applications sharing a single JVM in the Tomcat container.
- You will not be able to determine the startup order of the applications. Hence, you may experience problems such as JIRA starting before Crowd, rather than vice versa.
- Memory problems are also common as one application may allocate all of the memory in the Tomcat JVM to itself, starving the other applications.

We also do not support deploying multiple Atlassian applications to a single Tomcat container for a number of practical reasons. Firstly, you must shut down Tomcat to upgrade any application and secondly, if one application crashes, the other applications running in that Tomcat container will be inaccessible.

Finally, we recommend not deploying *any other applications* to the same Tomcat container that runs the Atlassian application, especially if these other applications have large memory requirements or require additional libraries in Tomcat's `lib` subdirectory.

How Bamboo processes task arguments and passes them to OS shell

Parsing arguments

When executing different Tasks, Bamboo attempts to tokenize value entered in Arguments field. The general rules are:

- white characters are argument separators,
- single and double quotes are used to preserve white characters in arguments.

This particular string

```
clean install -DpartiallyQuotedArgument1="Partially Quoted Argument" 'Fully Quoted Argument'
```

would be tokenized as

```
clean
install
-DpartiallyQuotedArgument1="Partially Quoted Argument"
'Fully Quoted Argument'
```

Each line here represents a single argument that will be passed to shell.

Passing arguments to shell

Bamboo generally doesn't modify tokenized arguments before passing them to shell with one exception:

- on non-Windows OS arguments that are fully enclosed in single or double quotes will be stripped from those quotes.

This particular string

```
clean install -DpartiallyQuotedArgument1="Partially Quoted Argument" 'Fully Quoted Argument'
```

would be passed to Windows shell as

```
clean
install
-DpartiallyQuotedArgument1="Partially Quoted Argument"
'Fully Quoted Argument'
```

but to Unix shell as

```
clean
install
-DpartiallyQuotedArgument1="Partially Quoted Argument"
Fully Quoted Argument
```

FAQ

I want to pass double quotes with my argument to Unix shell.

Try this

```
'"Only external quotes will be stripped and double quotes will be preserved when passing this to Unix
shell"'
```

My Maven Task doesn't work when I specify multiple targets in argument field

Make sure you haven't quoted the whole contents of Arguments field:

```
"clean install"
```

You should simply delete quotes

```
clean install
```

Securing your repository connection

About this page

This page shows how to secure your bamboo server to source repository connection.

Subversion

svn+ssh

In your build plan you must specify the absolute path to the repository when using svn+ssh, for example `svn+ssh://<svnhost>/absolute/path/to/repository/root/your/module`

Using a key pair

They key pair is shared between your bamboo agent box (the bamboo server box in case of local agents) and the repository server box. Your repository configuration allows you to specify the location of a private key file that must be stored on the agent box.

The key pair has to be in PKCS12/OpenSSH format and the private key must be passphrase protected, otherwise a runtime exception is thrown by JDK security engine while opening the user key.

Linux and related

1. On the repository box generate the keypair
   ```
   ssh-keygen -t rsa
   ```
2. add public key to ~/.ssh/authorized_keys
   ```
   cat id_rsa.pub >> ~/.ssh/authorized_keys
   ```
3. copy the private key to all the agent boxes **into a directory that is common to all agents (remote and local)** e.g. /var/keys/ssh/id_rsa

> **For windows agents**
>
> Store the private key file in the same location **on the drive that the agent is started from**. For example you start your agent with
>
> ```
> d:\bamboo-agent > java -jar atlassian-bamboo-agent-installer-xxx.jar ....
> ```
>
> Then the key file must be in `d:\var\keys\ssh\id_rsa`

Windows

Private key should always be in OpenSSH format. On windows usually "putty" (plink) program is used that uses keys in its proprietary format
(PPK - putty private key), this format is not supported by bamboo. The PuttyGen program may be used on Windows to convert key in PPK format to
OpenSSH.

How to add the public key to the windows version of `~/.ssh/authorized_keys` <<<< comment needed

Trouble shooting

You can test the svn+ssh connection from the command line.
First you need to tell the svn command line client which key file to use:

```
$ export SVN_SSH="ssh -i /absolute/path/to/private/key"
```

Then you can test the connection with

```
$ svn list svn+ssh://<svn-server>/Absolute/Path/To/Repository/[Module]
```

Changing the remote agent heartbeat interval

Remote agents periodically send a "heartbeat" signal to the Bamboo server. This is vital for tracking whether your remote agents are online or offline. The remote heartbeat is *asynchronous*, which means that if a remote agent goes offline and comes back online again it will reconnect instead of being shut down (as long as the same server is available).

However, you may wish to adjust the time parameters for the remote agent heartbeat, particularly if you have a lot of network activity already.

There are three configurable parameters on the bamboo server for the remote agent heartbeat:

bamboo.agent.heartbeatInterval=60
The frequency of the heartbeat signal from remote agents. The value is in seconds.

The default value is 60 seconds.

bamboo.agent.heartbeatTimeoutSeconds=600
How long the Bamboo server will wait before it times out an agent that it hasn't received a heartbeat signal from. A remote agent that has been timed out will be marked as 'Offline'. Any builds being run by agents which have timed out will be abandoned. The value is in seconds.

The default value is 600 seconds.

bamboo.agent.heartbeatCheckInterval=30
How often Bamboo checks for agents that have exceeded the heartbeat timeout specified in `bamboo.agent.heartbeatTimeoutSeconds`. The value is in seconds.

The default value is 30 seconds.

The parameters above are passed as JVM parameters. We recommend to add these options using the variable JVM_SUPPORT_RECOMMENDED_ARGS defined in the file <Bamboo install directory>/bin/setenv.sh.

See Configuring Bamboo on startup for instructions on how to change a Bamboo system property.

Cloning a Bamboo instance

You can clone an existing Bamboo instance by getting a new Bamboo instance in the same version and using the setup of the existing one.

You may want to transfer a snapshot of your current production Bamboo instance to a test server as permitted in the license agreement.

Cloning Bamboo can be a step in preparation for migrating to another database or for upgrading.

- If you are using Jira or Crowd for user management, the URL of the Bamboo server may change when you clone the Bamboo instance, in which case you will need to edit that setting for the Bamboo application in Jira/CROWD to match the new URL.
- When cloning your Bamboo instance using the export/import process, you must re-install all of your apps manually.

On this page:

License

Development licenses are available for any Commercial or Academic license. Create one or contact us for help.

Cloning a Bamboo instance to a new server

To clone a Bamboo instance to a new server:

1. Export/Backup your original Bamboo instance.
2. Install **the same version** of Bamboo on the new server.

   If you are cloning a Bamboo instance on the same server, make sure that the original Bamboo instance doesn't have the same `<bamboo-install>` installation path as the new Bamboo instance.

3. On the new server, in the new `<bamboo-install>` directory, go to `webapp/WEB-INF/classes/` and open the `bamboo-init.properties` file. In the `bamboo-init.properties` file, set the new `<bamboo-home>` path.

   If you are cloning a Bamboo instance on the same server, make sure that the new Bamboo instance doesn't have the same `<bamboo-home>` path as the original Bamboo instance.

4. Start the new Bamboo instance and import the existing export /backup data prepared in Step 1.

The installation path is referred to as `<bamboo-install>` and points to the directory into which you extracted the Bamboo package during the installation. It is different from the `<bamboo-home>` path which points to the directory where Bamboo data is stored.

Alternative cloning scenario

If your current instance has grown too large and export/import does not work, you can still clone your instance by using an alternative backup and restore strategy.
The approach is to clone the <bamboo-home> content and make it available to the new Bamboo instance:

1. Stop the original Bamboo instance.
2. Create a backup:

embedded DB	external DB
Compress the original <bamboo-home> directory into a .zip file. The embedded database is included in the directory.	• Compress the original <bamboo-home> directory into a .zip file. • Create a database backup with the native tools.

> You can reduce the size of a compressed <bamboo-production-home>file by deleting the xml-data/build-dir directory that contains working copies of the checked-out sources.

   For more information about migrating databases, see Move data to a different database.
3. Restart the original Bamboo instance.
4. Install **the same version** of Bamboo on the new server.

> If you are cloning a Bamboo instance on the same server, make sure that the new Bamboo instance doesn't have the same <bamboo-install> installation path as the original Bamboo instance.

5. Transfer the compressed original <bamboo-home> directory to the new server where you installed the new Bamboo instance.
6. Replace the content of the new <bamboo-home> directory with the unzipped <home-directory> content.
7. On the new server, in the new <bamboo-install> directory, go to webapp/WEB-INF/classes/ and open the bamboo-init.properties file. In the bamboo-init.properties file, set the new <bamboo-home> path.

> If you are cloning a Bamboo instance on the same server, make sure that the new Bamboo instance doesn't have the same <bamboo-home> path as the original Bamboo instance.

8. (External DB only) Create a new database for the cloned instance and import the external database backup.
9. In the new <bamboo-home> directory, open:

   - bamboo.cfg.xml
   - xml-data/configuration/administration.xml

   and change the server names/IP addresses according to the new location.
10. (External DB only) Go to the new <bamboo-home> directory, open the bamboo.cfg.xml file, and enter the new database connection details and credentials.

> Before starting Bamboo, depending on your motivation for cloning, you may also want to ensure any customizations that were made in the previous Bamboo instance's installation directory are also copied to the clone. Examples of customizations include proxy settings, additional JVM arguments and JVM memory allocation tweaks. These customizations most commonly exist in:
>
> - <bamboo-install>/bin/setenv.sh
> - <bamboo-install>/conf/server.xml

11. Start the new Bamboo instance.

Next steps

- (Optional) You can upgrade the new Bamboo instance.
- If the new server has different locations for:
  - JDKs
  - Ant
  - Maven
  - Perforce
  - Msbuild tools

  adjust the settings in the server capabilities settings to match the locations on the new server.

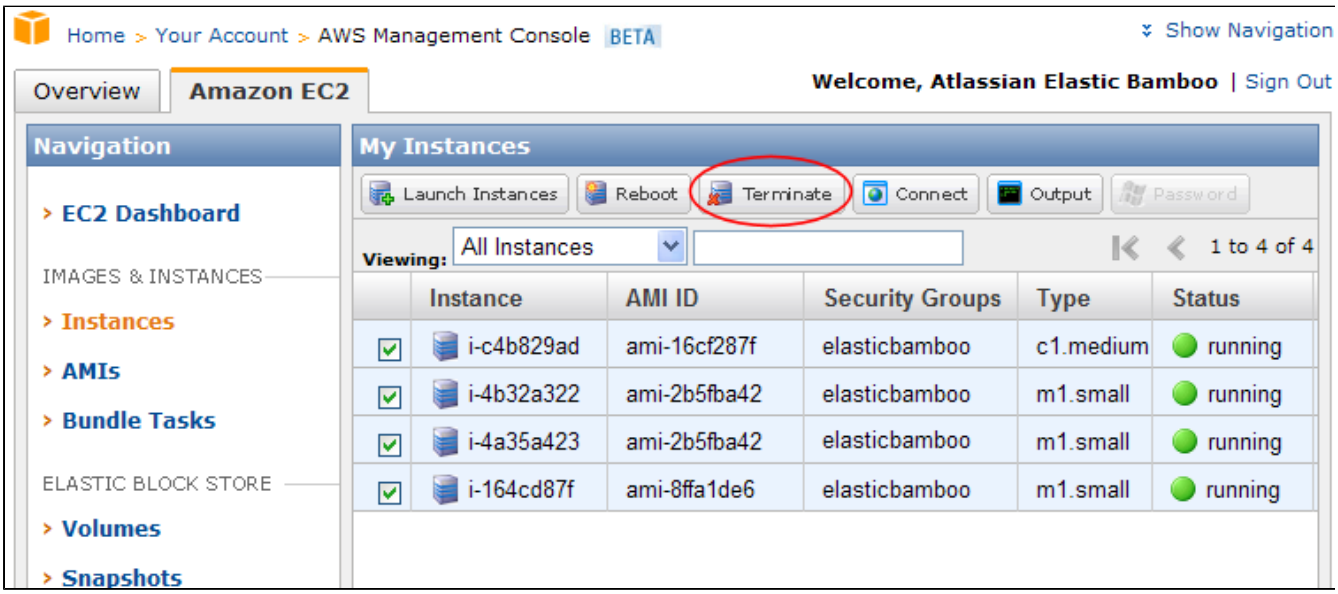
How do I shut down my elastic instances if I have restarted my Bamboo server

If you restart your Bamboo server without shutting down your elastic instances first, your elastic instances will continue to run. Your elastic instances will also be orphaned from your Bamboo server, and you will not be able to shut them down via Bamboo after your Bamboo server has restarted. You will need to terminate them via the Amazon Web Services (AWS) Console.

To shut down an elastic instance via the AWS Console:

1. Log in to the AWS Console. The 'Amazon EC2' tab of the console should display.
2. Click the **Instances** link under the 'Images & Instances' section of the left navigation column. Your EC2 instances should be displayed.
3. Check the checkbox next to the instances that need to be terminated in the 'My Instances' panel. In most cases, it should be all instances unless you are running Elastic Bamboo on multiple Bamboo servers.
4. The buttons at the top of the 'My Instances' panel should become enabled. Click **Terminate** to terminate your instances.

Screenshot: Shutting down an elastic instance via the AWS Console



How do I stop the Bamboo server from automatically configuring my remote agent's capabilities

The Bamboo server automatically detects and populates the capabilities that a remote agent should be configured with upon agent start up. If you have modified the agent capabilities, they will be reset by the server's automatic capability detection when the agent is next restarted.

You can override this by adding the following flag, "`-DDISABLE_AGENT_AUTO_CAPABILITY_DETECTION=true`", to the Bamboo server. Read Starting Bamboo for information on how to do this.

JUnit parsing in Bamboo

Bamboo can parse any test output that conforms to standard JUnit XML format. The implementation of this is pretty simple — Bamboo looks for specific tags in the JUnit XMl output.

A failed JUnit XML report, that is successfully parsed by Bamboo.

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuite errors="0" tests="3" time="0.391" failures="1"
           name="com.atlassian.bamboo.repository.perforce.PerforceSyncCommandTest">
    <properties>
        <property value="Java(TM) 2 Runtime Environment, Standard Edition" name="java.runtime.name"/>
        <property value="UnicodeBig" name="sun.io.unicode.encoding"/>
    </properties>
    <testcase time="0.001" name="testGeneratesCorrectP4CommandLine"/>
    <testcase time="0" name="testGettersReturnExpectedStuff"/>
    <testcase time="0.164" name="testUsingPerforceWhenNoFilesHaveChanged">
        <failure type="junit.framework.AssertionFailedError"
                 message="Should not have any errors. [Perforce client error:,         Connect to server
failed; ">
            junit.framework.AssertionFailedError: Should not have any errors. [Perforce client error:,
Connect to server
            failed; check $P4PORT., TCP connect to keg failed., keg: host unknown.] expected:&lt;0&gt; but
was:&lt;4&gt;
            at junit.framework.Assert.fail(Assert.java:47)
            at junit.framework.Assert.failNotEquals(Assert.java:282)
            at junit.framework.Assert.assertEquals(Assert.java:64)
            at junit.framework.Assert.assertEquals(Assert.java:201)
            at  com.atlassian.bamboo.repository.perforce.PerforceSyncCommandTest.
testUsingPerforceWhenNoFilesHaveChanged(PerforceSyncCommandTest.java:60)
            at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
            at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
            at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
            at java.lang.reflect.Method.invoke(Method.java:585)
            at junit.framework.TestCase.runTest(TestCase.java:154)
            at junit.framework.TestCase.runBare(TestCase.java:127)
            at junit.framework.TestResult$1.protect(TestResult.java:106)
            at junit.framework.TestResult.runProtected(TestResult.java:124)
            at junit.framework.TestResult.run(TestResult.java:109)
            at junit.framework.TestCase.run(TestCase.java:118)
            at junit.framework.TestSuite.runTest(TestSuite.java:208)
            at junit.framework.TestSuite.run(TestSuite.java:203)
            at sun.reflect.GeneratedMethodAccessor17.invoke(Unknown Source)
            at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
            at java.lang.reflect.Method.invoke(Method.java:585)
            at org.apache.maven.surefire.battery.JUnitBattery.executeJUnit(JUnitBattery.java:242)
            at org.apache.maven.surefire.battery.JUnitBattery.execute(JUnitBattery.java:216)
            at org.apache.maven.surefire.Surefire.executeBattery(Surefire.java:215)
            at org.apache.maven.surefire.Surefire.run(Surefire.java:163)
            at org.apache.maven.surefire.Surefire.run(Surefire.java:87)
            at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
            at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
            at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
            at java.lang.reflect.Method.invoke(Method.java:585)
            at org.apache.maven.surefire.SurefireBooter.runTestsInProcess(SurefireBooter.java:313)
            at org.apache.maven.surefire.SurefireBooter.run(SurefireBooter.java:221)
            at org.apache.maven.test.SurefirePlugin.execute(SurefirePlugin.java:371)
            at org.apache.maven.plugin.DefaultPluginManager.executeMojo(DefaultPluginManager.java:412)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeGoals(DefaultLifecycleExecutor.
java:534)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeGoalWithLifecycle
(DefaultLifecycleExecutor.java:475)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeGoal(DefaultLifecycleExecutor.
java:454)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeGoalAndHandleFailures
(DefaultLifecycleExecutor.java:306)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.executeTaskSegments
(DefaultLifecycleExecutor.java:273)
            at org.apache.maven.lifecycle.DefaultLifecycleExecutor.execute(DefaultLifecycleExecutor.java:
140)
            at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:322)
            at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:115)
            at org.apache.maven.cli.MavenCli.main(MavenCli.java:256)
            at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

```
            at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
            at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
            at java.lang.reflect.Method.invoke(Method.java:585)
            at org.codehaus.classworlds.Launcher.launchEnhanced(Launcher.java:315)
            at org.codehaus.classworlds.Launcher.launch(Launcher.java:255)
            at org.codehaus.classworlds.Launcher.mainWithExitCode(Launcher.java:430)
            at org.codehaus.classworlds.Launcher.main(Launcher.java:375)
        </failure>
        <system-out>
                PerforceSyncCommand.command: /usr/local/bin/p4
        </system-out>
    </testcase>
</testsuite>
```

Click here to download the XML report.

A passed JUnit XML report, that is successfully parsed by Bamboo.

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuite errors="0" skipped="0" tests="1" time="0.045" failures="0" name="com.atlassian.bamboo.labels.
LabelManagerImplTest">
  <properties>
    <property value="Java(TM) 2 Runtime Environment, Standard Edition" name="java.runtime.name"/>
    <property value="/usr/java/jdk1.5.0_07/jre/lib/i386" name="sun.boot.library.path"/>
    <property value="1.5.0_07-b03" name="java.vm.version"/>
    <property value="Sun Microsystems Inc." name="java.vm.vendor"/>
    <property value="http://java.sun.com/" name="java.vendor.url"/>
    <property value=":" name="path.separator"/>
    <property value="Java HotSpot(TM) Client VM" name="java.vm.name"/>
    <property value="sun.io" name="file.encoding.pkg"/>
    <property value="US" name="user.country"/>
    <property value="unknown" name="sun.os.patch.level"/>
    <property value="Java Virtual Machine Specification" name="java.vm.specification.name"/>
    <property value="/opt/bamboo-data/bamboohome/xml-data/build-dir/BAM-MAIN" name="user.dir"/>
    <property value="1.5.0_07-b03" name="java.runtime.version"/>
    <property value="sun.awt.X11GraphicsEnvironment" name="java.awt.graphicsenv"/>
    <property value="/opt/bamboo-data/bamboohome/xml-data/build-dir/BAM-MAIN/bamboo-core" name="basedir"/>
    <property value="/usr/java/jdk1.5.0_07/jre/lib/endorsed" name="java.endorsed.dirs"/>
    <property value="i386" name="os.arch"/>
    <property value="/tmp" name="java.io.tmpdir"/>
    <property value="Sun Microsystems Inc." name="java.vm.specification.vendor"/>
    <property value="Linux" name="os.name"/>
    <property value="/opt/java/tools/maven2/bin/m2.conf" name="classworlds.conf"/>
    <property value="ISO-8859-1" name="sun.jnu.encoding"/>
    <property value="/usr/java/jdk1.5.0_07/jre/lib/i386.." name="java.library.path"/>
    <property value="Java Platform API Specification" name="java.specification.name"/>
    <property value="49.0" name="java.class.version"/>
    <property value="HotSpot Client Compiler" name="sun.management.compiler"/>
    <property value="2.6.15-1.1833_FC4smp" name="os.version"/>
    <property value="/home/bamboo" name="user.home"/>
    <property value="Australia/Sydney" name="user.timezone"/>
    <property value="sun.print.PSPrinterJob" name="java.awt.printerjob"/>
    <property value="ISO-8859-1" name="file.encoding"/>
    <property value="1.5" name="java.specification.version"/>
    <property value="bamboo" name="user.name"/>
    <property value="/opt/java/tools/maven2/boot/classworlds-1.1.jar" name="java.class.path"/>
    <property value="1.0" name="java.vm.specification.version"/>
    <property value="32" name="sun.arch.data.model"/>
    <property value="/usr/java/jdk1.5.0_07/jre" name="java.home"/>
    <property value="Sun Microsystems Inc." name="java.specification.vendor"/>
    <property value="en" name="user.language"/>
    <property value="mixed mode, sharing" name="java.vm.info"/>
    <property value="1.5.0_07" name="java.version"/>
    <property value="/usr/java/jdk1.5.0_07/jre/lib/ext" name="java.ext.dirs"/>
    <property value="Sun Microsystems Inc." name="java.vendor"/>
    <property value="/opt/java/tools/maven2" name="maven.home"/>
    <property value="/home/bamboo/.m2/repository" name="localRepository"/>
    <property value="/" name="file.separator"/>
    <property value="http://java.sun.com/cgi-bin/bugreport.cgi" name="java.vendor.url.bug"/>
    <property value="little" name="sun.cpu.endian"/>
    <property value="UnicodeLittle" name="sun.io.unicode.encoding"/>
    <property value="" name="sun.cpu.isalist"/>
  </properties>
  <testcase time="0.045" name="testBAM1436"/>
</testsuite>
```

Click here to download the XML report.

Click here for the AntXmlResultParser.java file which contains the Bamboo code for parsing JUnit XML output.

For those interested in the XUint XML Schema, please see this document.

Known issues with CVS in Bamboo

Bamboo uses CVS `rlog` command - this lets you perform a CVS update on your local working directory without checking out your project.

> **CVS Error logging in Bamboo**
>
> Currently, if the server throws an error during a CVS build in Bamboo versions 2.0.x, the application will hang with no indication of any checkout/update problems. There is an open Jira issue tracking this problem.
>
> In order to further debug any CVS issues, you will need to turn up the CVS logging by passing in the `-DcvsClientLog=system` system argument to Bamboo.

1) Incompatibility with CVS servers 1.11.1 and below

Support for the `rlog` command 1.11.1p and performing a CVS `rlog` command returns the following error:

```
-cvs [rlog aborted]: server does not support rlog
```

2) Incompatibility with CVS server version 1.11.x when using "." to denote the root module to be checked out.

The CVS `rlog` command fails if you are using CVS version 1.11.x, with the following error.

```
INFO   | jvm 1    | 2008/05/15 14:19:10 | E cvs: recurse.c:642: do_recursion: Assertion `strstr
(repository, "/./") == ((void *)0)' failed.
INFO   | jvm 1    | 2008/05/15 14:19:10 | error
```

Please upgrade your CVS version to 1.12.x to get around this issue.

3) CVS Checkout format

Due to prior issues, Bamboo will checkout all files (including text files) from the CVS server as binary, however post Bamoo 2.1.2 this behavior can be changed via a system parameter. To do this restart Bamboo with the following parameter (*if you have any elastic agents running, ensure that they are shut down before you restart the Bamboo server. If you do not shut down your elastic instances before restarting, they will continue to run and become orphaned from your Bamboo server*).

```
-DCVS_CHECKOUT_BINARY_FORMAT=false
```

Post 2.1.5 this has been replaced with a more flexible option

```
-DCVS_CHECKOUT_FORMAT=BINARY
```

Option	Command Options	Behavior
BINARY (Default)	-b	forces all files to be checked out in binary and won't convert any line endings
TEXT	-kv	forces all files to be checked out as text and converts all line endings (even Binary files)
NONE		lets CVS decide whether or not to convert line endings

For further reference, on configuring Bamboo start-up options see this document

Monitor Memory usage and Garbage Collection in Bamboo

These are some recommended settings to enable Garbage Collection logging and Heap Dump when Bamboo runs on Out Of Memory.

Parameters

Please add the following parameters to Bamboo. Make sure to select the correct Java version.

Java 8:

```
-Xloggc:<bamboo-home>/logs/atlassian-bamboo-gc-%t.log -XX:-OmitStackTraceInFastThrow -XX:
+PrintConcurrentLocks -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=<bamboo-home>/logs/atlassian-bamboo-
heapdump-%t.hprof -XX:+PrintTenuringDistribution -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:
+PrintGCTimeStamps -XX:+PrintGCApplicationStoppedTime -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -
XX:GCLogFileSize=10M -verbose:gc
```

Java 11:

```
-Xlog:gc*,gc+age=trace:file=<bamboo-home>/logs/atlassian-bamboo-gc-%t.log:time,level,tags:filesize=10M,
filecount=5 -XX:-OmitStackTraceInFastThrow -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=<bamboo-home>
/logs/atlassian-bamboo-heapdump-%t.hprof
```

Note: Remember to substitute `<bamboo-home>` with a correct Bamboo Home path on your server.

GC log file location

The garbage collection traces and the heap dumps are in `<bamboo-install>/logs/atlassian-bamboo-gc-%t.log`. Where `%t`, is the time the file was created.

Additional Note

The `-XX:+PrintGCTimeStamps` flag, prints when GCs happen relative to the start of the application. That's applicable to Java 8. On Java 11, the timestamps are implicit by `-Xlog:gc*`

Some helpful links:

http://blogs.atlassian.com/developer/2007/10/plugging_leaks_in_confluence.html

http://www.oracle.com/technetwork/articles/javase/gcportal-136937.html

Moving Bamboo-Home of an agent

To move an agent's Bamboo-Home -

1. Move the Bamboo-Home of the agent, to the intended location.
2. Edit the <Bamboo-Agent-Home>/bamboo-agent.cfg.xml file, find the following line -

```
<buildWorkingDirectory>/YOUR_OLD_DIRECTORY/xml-data/build-dir</buildWorkingDirectory>
```

3. Point the working directory to the new Bamboo-Home.
4. Start your Agent with -Dbamboo.home=your_new_agent_home and point to your new Bamboo-Agent-Home.

Performing a thread dump

If Bamboo stops responding, or is performing poorly, you should create a thread dump to help Atlassian determine the cause of the problem.

This will show the state of each thread in the JVM, including a stack trace and information about what locks that thread is holding and waiting for.

Linux (and Solaris and other Unixes) Users

Find the process id of the JVM and issue the command:

Use the **ps** command to get list of all processes.

```
kill -3 <pid>
```

Note: This will not kill your server (so long as you included the "-3" option, no space in between).
The thread dump will be printed to Bamboo's standard output.

Please note that some application servers (like tomcat) redirect stdout (to catalina.out for instance).

Jstack (any Platform with an JAVA JDK)

Sun JDK 1.5 and above ship with native tool called `jstack` to perform thread dump. To use the tool find the Proccess ID and execute the command:

```
jstack <ProcessID>
```

When running jstack, please be sure to run it as the same user that runs the process you're targeting for thread dumps.

If you run your Atlassian product via wrapper (as a service) on Windows, you may encounter this error, ' Not enough storage is available to process this command'. See the suggestions in this KB article for workarounds.

Java VisualVM (any Platform with an JAVA JDK)

Oracle JDK has a native tool `jvisualvm` to perform thread dumps (and much more). To use the tool execute the command:

```
jvisualvm
```

Find Bamboo process ({{com.atlassian.bamboo.server.Server}}) and execute "Thread Dump" option available from a context menu.

Thread Dump Tools

- Samurai
- Thread Dump Analyzer TDA

Restoring passwords to recover admin users

This page describes the procedures if:

- you are unable to log in as an administrator.
- you have forgotten your password and don't have Mail Server configured,
- you want to replace administrator passwords manually.

If you're using the embedded database, you can access Bamboo's H2 through a Database Administration tool, Java GUI, or the command line. Check out how to access Bamboo's H2 database

⚠ Before you follow the instructions below, we recommend to back up your database. By that, you'll be able to restore your data if errors appear.

Identifying existing local admin users

To identify any existing local admin users on the Bamboo Internal Directory:

1. Connect to the Bamboo database and run the following statement to identify the ID of the Bamboo Internal User Directory:

```
select id from cwd_directory where directory_name = 'Bamboo Internal Directory'
```

2. Take note of the ID returned by the query.
3. Run the following statement to list the users that are part of that directory, replacing `<DIRECTORY_ID>` with the ID we got from the previous step.

```
select * from cwd_user where directory_id = '<DIRECTORY_ID>' order by id
```

Usually, the first user returned by the query would be the local admin created when Bamboo was first installed. However, we recommend checking the first few records to make sure you've identified the correct user.

Reset password

Once you have identified the local admin username, you can reset it if you don't remember its password. Connect to the Bamboo database and run the following statement. Replace the username and directory ID retrieved by the previous steps.

```
update cwd_user

set credential = 'x61Ey612Kl2gpFL56FT9weDnpSo4AV8j8+qx2AuTHdRyY036xxzTTrw10Wq3+4qQyB+XURPWx1ONxp3Y3pB37A==' 

where user_name = '<USERNAME>'

and directory_id = '<DIRECTORY_ID>'
```

The local admin password will now be `admin`.

If you want to change the password:

1. Go to **Bamboo Administration** > **User Management Search** > **User Directories**
2. Reorder the list of user directories so that the Bamboo Internal Directory comes first on the list. By that, you'll ensure that the internal admin will be used to log in instead of an external user with the same username.
3. Open Bamboo in an incognito window and log in with the admin username you got from the database (usually `admin`). The password is `admin`.
4. Go to **Profile icon** > **Profile** > **Change password** and set up a new password.
5. Log out and log in again via the incognito window to ensure everything works as expected.

6. Return to the standard browser window where you reordered the user directories list. You can now return to the original order (assuming Crowd was first on the list).

Send Errors to stderr - Script Builder in Visual Studio WinXP to build Solutions Files

To display an Error Summary for erroneous builds in bamboo build summary is not available for the Script Builder - going through the build logs seems tedious.
There is a section named "Error summary" which collects all errors during the build process that are printed to **st derr**. For example a build script

```
#!/bin/bash
echo "ERROR build xyz failed" >&2
```

would print this message into the build summary section. It is up to you to insert the appropriate messages into your build script.

Problem

The actual problem is devenv.com/msbuild
not being very helpful: both build tools only append to **stdout** stream,
even in the case of warnings/errors during the build.

Solution

I solved the issue by writing a simple Ruby script that invokes the build tool and filters the stdout stream for any warnings and errors via regexp; the matching warning/error lines are
then echoed to **stderr** and Bamboo picks them up nicely.

build script.ry

```
pipe = IO.popen("devenv.com #{$*[0]} /Rebuild ")
errors = 0
warnings = 0
while line = pipe.gets
if line =~ /^.* : .* error .*$/
$stderr.puts line
errors += 1
elsif line =~ /^.* : warning .*$/
$stderr.puts line
warnings += 1
else
$stdout.puts line
end
end
exit errors > 0 ? 1 : 0
```

Related Pages

Knowledge Base - (BSP-1381) Script Builder Display build errors in Error Summary

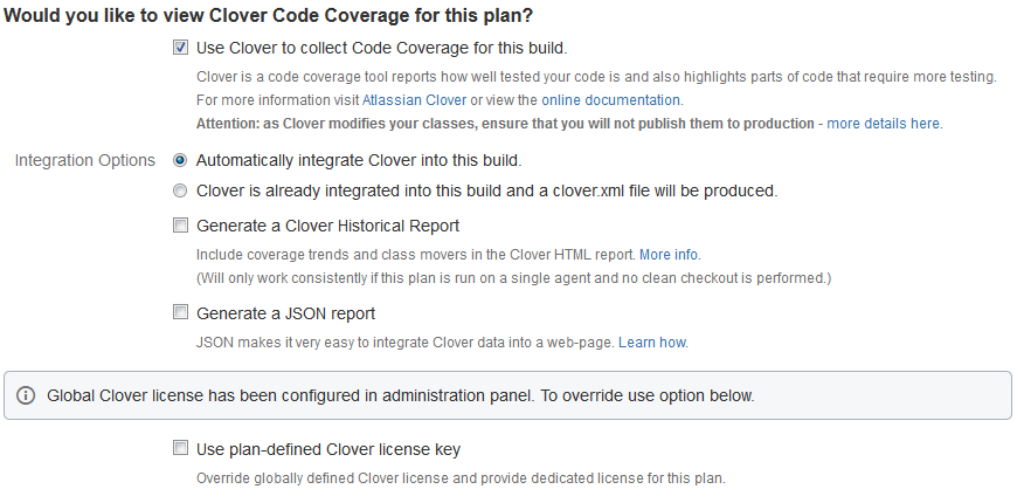
Using Bamboo with Clover

Clover is now available as an open source project. Learn more

Getting Started

One-click Clover Integration

Clover reports can be activated in the Builder configuration screen. Please see Automatic Clover integration or further details.



To configure Clover activity refer to Clover Reference Guides for your builder:

- Clover for Ant
- Clover for Maven 2

Classic Clover Integration

To use Clover with Bamboo, you need to:

1. Integrate Clover with your build and ensure that HTML and XML reports are generated:
    - Clover-for-Ant Installation Guide
    - Clover-for-Maven 2 and 3 Installation Guide
2. Ensure that there are tests present in your build plan that generate test results in JUnit test report format.
3. Configure where Bamboo can find Clover reports:
    - see Enabling the Clover add-on # Manual Clover integration

For further details, please see Configuring tasks.

Common Problems

Q: I have managed to get Clover statistics displayed in numerical form for each build, but the graphs do not show a history of these statistics?
A: The history of Clover is displayed over time periods (e.g. a day, a week, a month), and the minimum data point is per day. The Clover coverage will not display data that is less than a day old.

Q: Will the Bamboo/Clover integration run on failed builds?
A: Before Bamboo version 1.2.1, Bamboo would only report Clover coverage for successful builds. As of Bamboo 1.2.1, Bamboo will report Clover coverage regardless of the build outcome.

Getting gcov results in Clover coverage summary

Clover is now available as an open source project. Learn more

This feature is not officially supported by Atlassian. It is being maintained by open source community, feel free to contribute.

Description

Clover does not support code coverage for C/C++. However, it is possible to display C/C++ coverage statistics on "Clover" tab on "Job Summary" and "Plan Summary" pages. In order to get this working:

- create a task in which gcov is used and produces coverage file
- create a task in which python script (see references below) converts gcov data to clover.xml file
- enable Clover on Miscellaneous tab on Job Configuration page
    - enable "Use Clover to collect code coverage for this build"
    - select option "Clover is already integrated into this build and a clover.xml will be produced."
    - enter path to clover.xml file

References

Source code for Python script performing conversion is kept in Mercurial bamboo-gcov-plugin repository on bitbucket.org:

```
hg clone ssh://hg@bitbucket.org/atlassian/bamboo-gcov-plugin
```

Discussion about Clover schema on Atlassian Answers:

- https://answers.atlassian.com/questions/68875/clover-xml-schema

Working with Java libraries

Due to licensing restrictions, we are not allowed to re-distribute native Java libraries through our maven2 public repositories.

If you are developing plugins for Bamboo or building Bamboo from source, you might need `javax.mail` and `javax.transaction:jta:jar` for Bamboo to build successfully. The relevant POMs for this look something like this:

```
.......
<dependency>
  <groupId>javax.mail</groupId>
   <artifactId>mail</artifactId>
   <version>1.3.2</version>
   <scope>compile</scope>
 </dependency>
<dependency>
  <groupId>jta</groupId>
   <artifactId>jta</artifactId>
   <version>1.0.1</version>
   <scope>compile</scope>
</dependency>
.......
```

Before building, please install the Oracle JAR's into your local Maven2 repositories by following the instructions below.

To install the `javax.mail` jar into your local Maven2 repository:

1. Download the `javax.mail` jar from the Oracle website.
2. Install it on our your local machine by entering the following command in a terminal:

```
mvn install:install-file -DgroupId=javax.mail -DartifactId=mail -Dversion=1.3.3 -Dpackaging=jar -
Dfile=YOUR/PATH/TO/FILE
```

To install `javax.transaction:jta:jar` into your local Maven2 repository:

1. Download the javax.transaction:jta:jar from the Oracle website.
2. Install it on your local machine by entering the following command in a terminal:

```
mvn install:install-file -DgroupId=javax.transaction -DartifactId=jta  -Dversion=1.0.1B -
Dpackaging=jar -Dfile=/path/to/file
```

Bamboo indicates that my Ant or Maven builds failed, even though they were successful

⚠ Please note this Bamboo functionality relates only to the Maven Task and Ant Task outputs.

If your plan's build logs indicate that your Maven or Ant builds are passing but Bamboo is reporting them as failed (or vice-versa), it could be that:

- Bamboo is not finding 'BUILD SUCCESS' in your build logs
- Bamboo is finding 'BUILD FAILED' in your build logs when it should not be doing so. (This marker is not used in Maven.)
- Your builds are returning a non-zero return code. (For example, the build log will indicate `Build process for 'ABC Application - XYZ Build' returned with return code = 1`.)

If your builds produce atypical or non-standard output, you can make Bamboo check for text other than 'BUILD SUCCESS' or 'BUILD FAILED' in your build logs. An additional system property is available to specify how far back in the logs Bamboo checks for these text markers.

System Property	Description	Default Value
`atlassian.bamboo.builder.successMarker`	Specifies the text (or string) that Bamboo looks for in the build log to determine if the build was successful	BUILD SUCCESS
`atlassian.bamboo.builder.failedMarker`	Specifies the text (or string) that Bamboo looks for in the build log to determine if the build failed	BUILD FAILED
`SUCCESS_MESSAGE_LINES`	Specifies the number of lines from the end of the builder log in which to check for the values of `atlassian.bamboo.builder.successMarker` or `atlassian.bamboo.builder.failedMarker`.	250

For instructions on how to configure a system property, please refer to the Starting Bamboo page.

DRAFT - Usage FAQ

Raising a request with Atlassian Support

If you encounter any problems when setting up or using Bamboo, please let us know — we're here to help!

You may want to search the following first:

- the Atlassian Answers site (the Bamboo forum), where Atlassian staff and Bamboo users can answer your questions.
- the Bamboo Knowledge Base.

If you've found a bug in Bamboo, or want to request a feature or improvement, raise a ticket in the Bamboo project of our public issue tracker. Try searching for similar issues - voting for an existing issue is quicker, and avoids duplicates.

If you still need assistance, please raise a support request, either from within Bamboo or on the Atlassian Support site, as described in the following sections.

Providing as much information as possible about your Bamboo installation with your initial request will help our Support Engineers to give you a faster and more complete response.

On this page:

Raising a Support Request from within Bamboo

This method depends on having a mail server configured for Bamboo that supports large zip file attachments.

1. Log in to Bamboo (as a System Administrator) and go to the admin area.
2. Click **Troubleshooting and support tools** (under 'System') then **Get help**.
3. Choose **Contact Technical Support or Report a Bug**.
4. Provide as much information as possible in the Description, including steps to replicate the problem, and any error messages that are appearing on the console or in the logs. For performance issues, please include profiling logs. See the section below about information you should provide.
5. Click **Send**.

This will produce a zip file containing the information categories selected from the list and will email this to Atlassian Support. You will receive an email advising you of details of the Support Request that was automatically created, and you will receive emailed updates about progress on your issue. You can also see the status of your request directly by visiting the Atlassian Support System.

Raising a Support request yourself at Atlassian Support

1. Log in to Bamboo (as a System Administrator) and go to the admin area.
2. Click **Troubleshooting and support tools** (under 'System') then **Create support zip**.
3. Select information categories to include in the zip file with **Customize zip**.
4. Click **Create zip**.

The zip file is created in the home directory of the Bamboo server, for example `<Bamboo_HOME>\export\Bamboo_support_2013-11-17-20-49-18.zip`.

When you now go to Atlassian Support and create a Support Request, you can attach the Support Zip file to the request.

Please provide as much information as possible in the request, including steps to replicate the problem, and any error messages that are appearing on the console or in the logs. For performance issues, please include profiling logs. See the section below about information you should provide.

Information you should provide

In addition to the logs and configuration information that you can include in the Support Request zip file, the following information can help to give you a faster response:

Environment details

- Bamboo version
- Java version (for example OpenJDK 1.8.0 JDK)
- Operating system (for example, Windows 7, Mac OS X 10.6.8)
- Database type (for example, MySQL) and version
- Browsers and versions
- Network topology - is Bamboo running behind a reverse proxy? Is that secured using HTTPS (SSL)?

Configuration

- Java settings, including JVM_MINIMUM_MEMORY, JVM_MAXIMUM_MEMORY

Logs

You may need to adjust the logging level, or enable profiling in Bamboo, in order to get more detailed logs. See Bamboo debug logging.

- Debug logs – Bamboo debug logs can be found in `<Bamboo_HOME>/log`.
- Profiling logs – Bamboo profiling logs can help with analyzing performance issues and can be found in `<Bamboo_HOME>/log`.

Performance factors

- Number of users
- CPU spec, number of cores, whether hyperthreading is enabled
- RAM and cache sizes

Integrations

- Other Atlassian applications (and their versions)
- Which build servers are integrated with Bamboo, if any?
- Are Application Links configured?

Support Policies

Welcome to the support policies index page. Here, you'll find information about how Atlassian Support can help you and how to get in touch with our helpful support engineers. Please choose the relevant page below to find out more.

- Bamboo Support Policy
- New Features Policy
- Finding Your Bamboo Support Entitlement Number (SEN)

To request support from Atlassian, please raise a support issue in our online support system. To do this, visit support.atlassian.com, log in (creating an account if need be) and create an issue under Bamboo. Our friendly support engineers will get right back to you with an answer.

Bamboo Support Policy

This page contains details about the scope of Bamboo Support.

On this page:

Build Failures

Atlassian will provide Troubleshooting Guide(s) and documentation to help customers resolve Bamboo-related issues.

Ultimately, users are responsible for the administration and maintenance of their build systems and infrastructure.

However, if the root cause of the problem is partially or wholly related to Bamboo, we will create a Bug Report or Feature request to address the issue.

> ⚠ Any bug or feature request reported during the course of investigation is subject to Atlassian's Bug Fixing and New Features Policies, as outlined in the Atlassian Support Offerings document.

Distributed Builds

The pre-requisites outlined in the Technical Overview section of Troubleshooting Guide must be met for server /agent communication to work.

If Atlassian determines that a customer's agent connectivity or communication problem results from a network or environmental factor, it is the customer's responsibility to address this problem and keep their network maintained.

EC2

Atlassian does not support custom elastic images (custom AMIs) and recommends using an EBS volume to customize your image if desired. While we are happy to assist with issues related to the elastic agent, we can not help troubleshoot modifications to the Stock images which are not directly related to Bamboo functionality.

Plugins

Atlassian offers support for certain third party plugins as listed in our supported plugins list. For unsupported plugins, issues should be raised with the provider of the plugin.

The following can be classified as being third-party plugins:

- Integration with repositories other than Subversion, CVS and Perforce.
- Third party builders, test and code coverage tools other than what is shipped with Bamboo.

Each plugin's supported status is listed on its page in the Plugin Exchange.

New Features Policy

> This policy does not apply to bugs. See our Server Bug Fix Policy or Cloud Bug Fix Policy to learn about our approach to bug fixing.

How we choose what to implement

There are many factors that influence our product roadmaps and determine the features we implement. When making decisions about what to prioritize and work on, we combine your feedback and suggestions with insights from our support teams, product analytics, research findings, and more. This information, combined with our medium- and long-term product and platform vision, determines what we implement and its priority order.

How to track when features are implemented

Cloud products

We're continuously improving and updating our Cloud products. To see the latest changes, take a look at the Atlassian Cloud release notes blog.

Data Center products

When a new feature or improvement is scheduled, we'll update the fix version on the relevant Jira issue to indicate the earliest product version that will include the change. This update often happens close to the product release date.

For a summary of changes, see the release notes for your product:

- Jira Software | Jira Service Desk | Jira platform | Advanced Roadmaps for Jira
- Confluence | Questions for Confluence | Team Calendars for Confluence
- Bitbucket | Bamboo | Fisheye | Crucible

Server products

We're simplifying our self-managed offerings and sharpening our focus to our cloud and Data Center products. This means we've discontinued new feature development in our server product line. Learn more about these changes

We'll still be offering bug fixes for server customers with active maintenance. For details, see our Atlassian Data Center and Server bug fix policy.

Product roadmaps

We publish a public roadmap for Jira Cloud products, Confluence Cloud, Bitbucket Cloud, and our Cloud Platform. This lets you know what's coming soon and what we're thinking about for future updates.

The Atlassian Cloud release notes blog and Bitbucket Cloud blog may also contain information on upcoming changes.

We don't provide specific release dates for upcoming changes.

Feature and improvement suggestions

Feedback from customers like you helps us shape and improve Atlassian products for all teams. We're continuously learning, analyzing and interviewing customers to make our products more intuitive and user-friendly, and to meet feature requirements. We encourage you to share your feedback and how you're using Atlassian products through Atlassian Community, jira.atlassian.com, or directly within our Cloud products. Learn more about our plans and the latest changes via our Cloud Roadmap or Data Center Roadmap.

We get a large number of suggestions and feature requests. Your comments and votes on suggestions help us understand what you're passionate about and how you want our products to support you and your team. The most helpful information you can provide us when commenting on issues is how a particular suggestion would help you. If you describe your use-case to us, and how the suggested change would benefit you and your team, it lets us gain a much deeper understanding of the need behind the suggestion.

Suggestions often have an impact on what we work on, even if we ultimately choose not to implement a suggestion exactly as it's described. Our ultimate goal is to understand what you and all of our customers need and to create products that meet those needs. Occasionally, that'll mean implementing a suggestion as described, but it usually means working to understand the need behind the suggestion and how we can meet that need for as many users as possible.

While we endeavor to update and respond to popular suggestions, the volume we receive means there will often be occasions when we can't provide an update or response. We don't provide any compensation or credit for feature suggestions that we implement.

Join the conversation on Atlassian Community

Our Product Managers regularly post articles about new features and changes to the Atlassian Community. You can comment on these posts, ask questions, and discuss with our PMs and other Atlassian users.

Release terminology for Data Center and server products

- **Platform release** (example: 4.0) contains significant or breaking changes. For example changes or removal of existing APIs, significant changes to the user experience, or removal or a major feature.
- **Feature release** (example: 4.6) can contain new features, changes to existing features, changes to supported platforms (such as databases, operating systems, Git versions), or removal of features. These were previously referred to as 'major' releases by most products.
- **Bugfix release** (example: 4.6.2) can contain bug fixes and stability and performance improvements. Depending on the nature of the fixes they may introduce minor changes to existing features, but do not include new features or high-risk changes, so can be adopted quickly. We recommend regularly upgrading to the latest bugfix release for your current version. These were previously referred to as 'maintenance' releases by most products.

In addition to the three main release types, a feature release can also be designated a **Long Term Support release** (formerly known as an Enterprise release), which means it will receive bug fixes for a longer period of time than a standard feature release.

Long Term Support releases

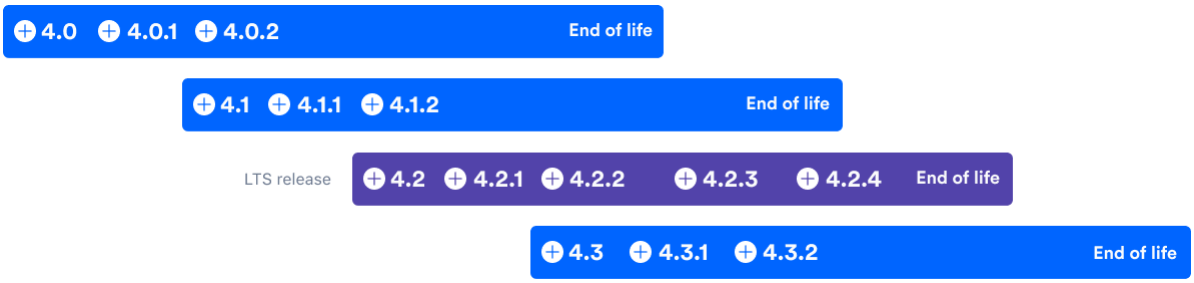
Long Term Support releases (formerly known as Enterprise releases) are for Server and Data Center customers who prefer to allow more time to prepare for upgrades to new feature versions but still need to receive critical bug fixes. If you only upgrade to a new feature version about once a year, a Long Term Support release may be a good fit for your organization. For Jira Software, Confluence, Bitbucket, and Bamboo we will:

- Designate a feature release as a Long Term Support release, at least every 12 months.
- Backport critical security fixes, as outlined in our current security bug fix policy, and fixes relating to stability, data integrity, or critical performance issues.
- Make bug fix releases available for the designated version until it reaches end-of-life.
- Provide a change log of all changes between one Long Term Support release and the next to make upgrading easier.

Not all bug fixes will be backported. We'll target the bugs and regressions that we deem most critical, focusing on stability, data integrity, or performance issues. There may also be some fixes that we choose not to backport due to risk, complexity, or because the fix requires changes to an API, code used by third-party apps (also known as add-ons), or infrastructure that we would usually reserve for a platform release.

For Jira Software Data Center customers, we'll endeavor to allow zero downtime upgrades between one Long Term Support release and the next Long Term Support release, but can't guarantee that downtime will not be required, depending on the nature of the changes. The change log will indicate if a zero downtime upgrade will be available.

In the example below, version 4.2 has been designated a Long Term Support release. The number of bug fix releases and timing illustrated below is just an example, your product's release cadence may differ.



Long Term Support changes for Server customers

If you have a Server license, you'll only be eligible to upgrade to versions released prior to February 15, 2024 PT, when we officially end support for our Server product line. If you intend to move your instance to Cloud or Data Center, contact our Support team.

Further reading

See Atlassian Support Offerings for more support-related information.

Finding Your Bamboo Support Entitlement Number (SEN)

Your Support Entitlement Number (SEN) is required when raising a support request in our Support system: http://support.atlassian.com.

See How to find your Support Entitlement Number (SEN) in the Support space for more general information about how Atlassian Support uses this number.

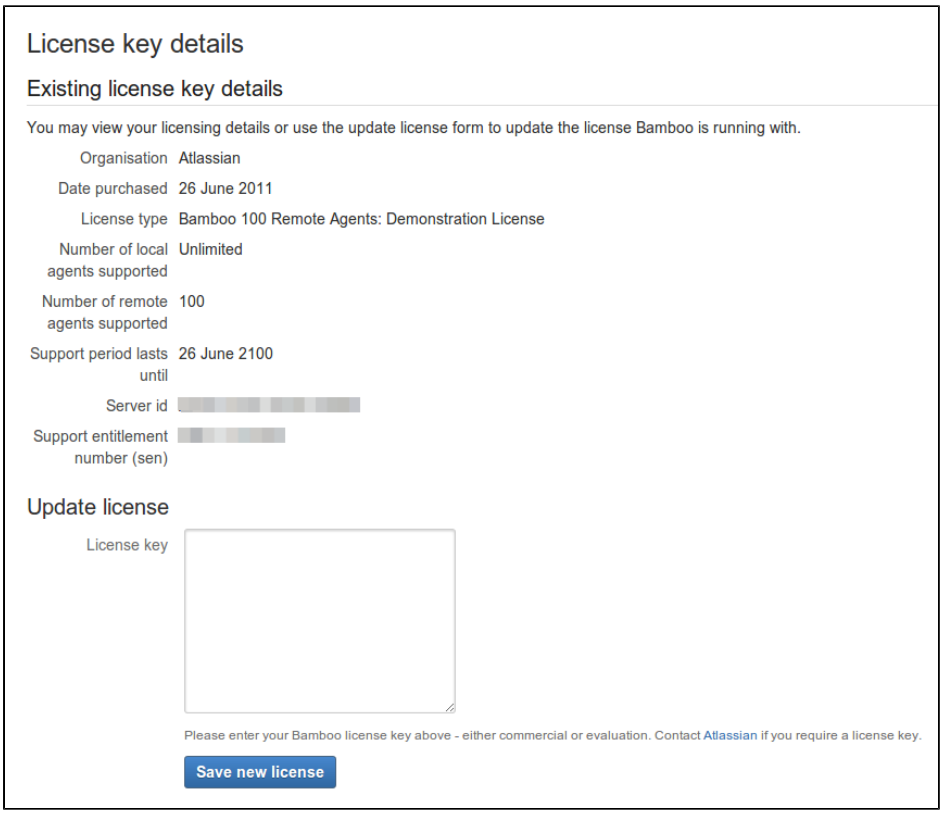
The three ways of finding your SEN are described below.

On this page:

Method 1 — Check the Bamboo Administration Interface

To find your SEN in the Bamboo administration interface:

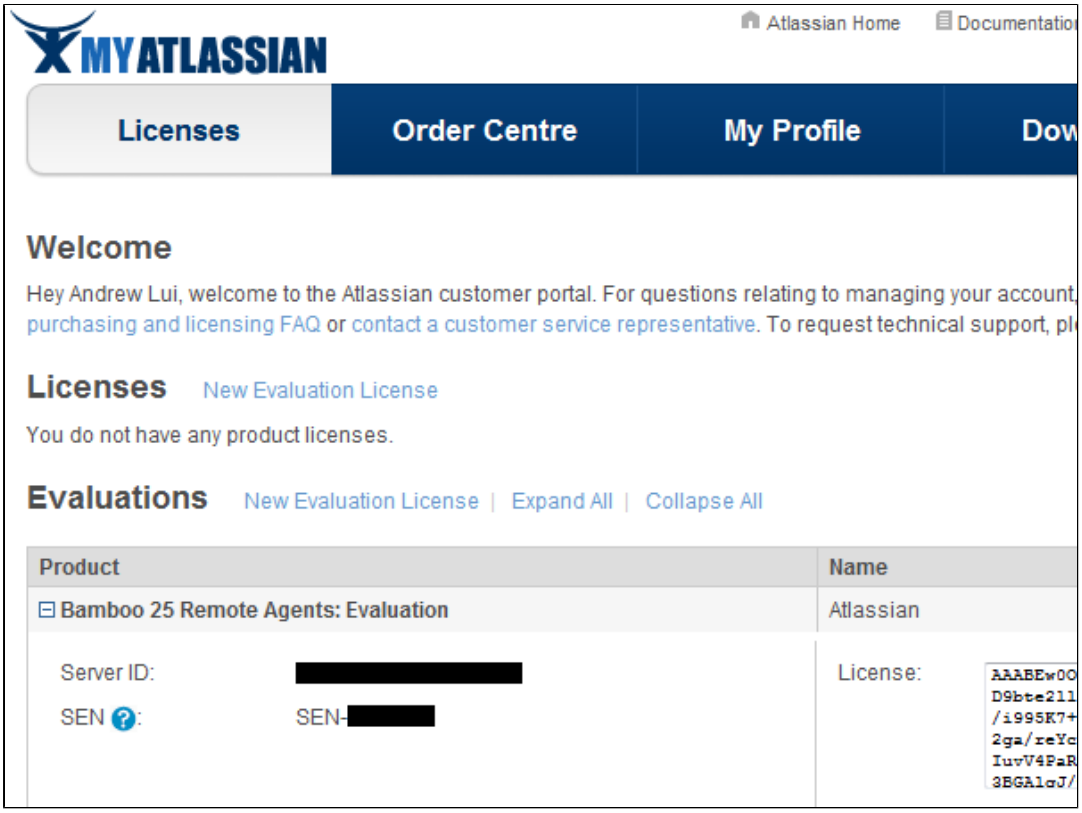
1. Click the  icon in the Bamboo header and choose **Overview**.
2. Click **License Details** in the left navigation panel (under 'System'). The SEN is shown, as in the screenshot below:



Method 2 — Check my.atlassian.com

To find your SEN via my.atlassian.com:

1. Log into my.atlassian.com as the Account Holder or Technical Contact for your Bamboo product.
2. The SEN will be shown, as per the screenshot below:



Method 3 — Check your Atlassian Invoice

Your Support Entitlement Number (SEN) appears on the third page of your Atlassian Invoice.

Bamboo resources

Resources for Evaluators

- Free Trial
- Feature Tour

Resources for Administrators

- Bamboo forum at Atlassian Answers
- Bamboo Knowledge Base
- Bamboo FAQ
- Guide to Installing an Atlassian Integrated Suite
- The big list of Atlassian gadgets

Resources for Developers

- Bamboo Developer Documentation
- API documentation
- Developer topics on Atlassian Answers

Downloadable Documentation

- Bamboo documentation in PDF, HTML or XML formats

Plugins

- Atlassian Marketplace

IDE Connectors

- Use the Atlassian Connector for Eclipse or the Atlassian Connector for IntelliJ IDEA to work with your Bamboo builds right there in your development environment. Do you use Jira, Crucible or Fisheye too? With the connector you can manage your issues and code reviews within your IDE, or move quickly between the IDE and a Fisheye view of your source repository. **Hint:** The Atlassian IDE Connectors are free.

Support

- Atlassian Support
- Support Policies

Training

- Atlassian Training

Forums

- Bamboo forum at Atlassian Answers
- Bamboo developers forum

Mailing Lists

- Visit http://my.atlassian.com to sign up for mailing lists relating to Atlassian products, such as technical alerts, product announcements and developer updates.

Feature Requests

- Issue Tracker and Feature Requests for Bamboo

Glossary

activity log

agent

agent-specific capability

artifact

authors in Bamboo

build

build activity

build duration

build log

build queue

build result

build telemetry

capability

child

committer

custom capability

default repository

elastic agent

elastic Bamboo

elastic block store

elastic image

elastic instance

executable

favorites

global permission

job

label

local agent

parent

permission

plan

plan permission

projects in Bamboo

queue

reason

remote agent

remote agent supervisor

requirement

shared capability

stage

Stock images

task

triggering

watcher

activity log

Every *plan* has an *activity log*. An *activity log* is a temporary display of the latest output from the plan's most recent *build log*.

agent

A Bamboo *agent* is a service that can run job builds. There are the following types of Bamboo agents:

- *local agents* run as part of the Bamboo server.
- *remote agents* run on computers, other than the Bamboo server, that run the remote agent tool.
- elastic agents run in the Amazon Elastic Compute Cloud (EC2).

Local agents run in the Bamboo server's process, i.e. in the same JVM as the server. Each remote agent runs in its own process, i.e. has its own JVM.

Each agent has a defined set of capabilities and can only run builds for jobs whose requirements match the agent's capabilities.

agent-specific capability

An *agent-specific capability* is a capability that applies to one agent only. Note that the value of an agent-specific capability will override the value of a *shared capability* of the same name (if one exists).

See

Agents and capabilities and

Configuring capabilities for more information.

artifact

Artifacts are files created by a *job build* (e.g. JAR files). Artifact definitions are used to specify which artifacts to keep from a build and are configured for individual jobs.

See Sharing artifacts.

authors in Bamboo

An *author* is any person who checks in code to a repository that is associated with a Bamboo plan. An author need not be a Bamboo user.

See Generating reports on selected authors.

build

A build is the execution of either a plan or a job. The execution of a plan is referred to as a plan build and that of a job is a job build.

build activity

Build activity is the number of builds that occur in a given period of time.

build duration

Build *duration* is the total time taken to execute a *plan* measured in seconds or minutes - from the time the plan is dispatched till the plan is finished and the build results are processed.

Variations in a plan's build *duration* can be observed over time.

build log

Every *build* has a *build log*. A *build log* is a permanent record of all the output generated by compiling the *job's* source-code and executing the tests.

build queue

The Bamboo *build queue* controls the sequence of *builds*. When a plan submits a build to the build queue, the build will wait in the build queue until a suitable agent is available to run the build.

The build queue is displayed on the **Build Activity** tab of the Dashboard.

build result

Every completed build has a *build result*:

- Successful — the code compiled, with or without errors, and all tests completed successfully.
- Failed — either the code did not compile, or at least one test failed.
- Incomplete — the build was not completed, e.g. it may have been stopped manually.

Additionally,

- if the build result is Failed, and the previous build result was Successful, the build is labeled as Broken.
- if the build result is Successful, and the previous build result was Failed, the build is labeled as Fixed.

build telemetry

Build telemetry is the insight provided by Bamboo's dynamic reports, charts and collation of build metrics. Build telemetry helps identify trends across build plans and across authors — not just focusing on the results of a single build.

capability

A *capability* is a feature of an agent. A capability can be defined on an agent for:

- an executable (e.g. Maven)
- a JDK
- a Version Control System client application (e.g. Git)
- a custom capability. This is a key-value property which defines a particular characteristic of an agent (e.g. 'operating.system=WindowsXP' or 'fast.builds=true').

Capabilities typically define the path to an executable that has already been installed, and must be defined in Bamboo before Bamboo or its agents can make use of those.

Capabilities can be defined specifically for an agent, or they can be shared between either all local agents or all remote agents. Note that the value of an agent-specific capability overrides the value of a shared capability of the same name (if one exists).

See Configuring capabilities for more information.

child

A *child* is a plan which gets triggered when another plan completes a build.See Setting up plan build dependencies.

committer

A *committer* is the Bamboo user(s) who committed code to a particular build (i.e. someone who committed code after the previous build was checked out by Bamboo).

Administrators can configure a plan's notifications to be sent to the build's committer(s).

custom capability

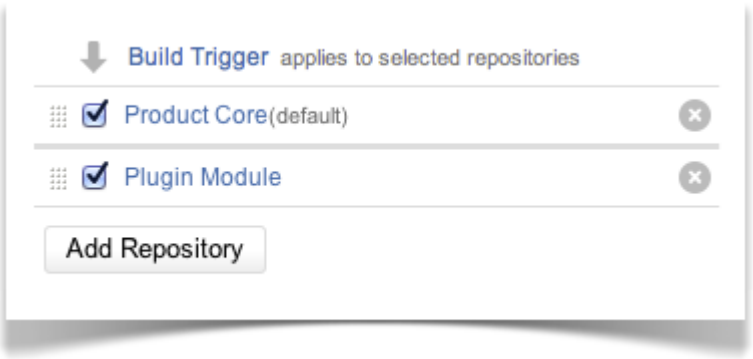
Custom capabilities can be used to control which jobs will be built by a particular agent, since agent capabilities are required to match job requirements. For example, if the builds for a particular job should only run in a Windows environment, you could create a custom capability '`operating.system=WindowsXP`' for the appropriate agent(s), and specify it as a requirement for this job.

- To create a new custom capability in your Bamboo system, see Defining a new custom capability.
- To specify a job's requirement for a custom capability, see Configuring a job's requirements.

default repository

The first repository in the list of plan repositories is the Plan's Default Repository. The default repository will be automatically checked out by any new job created.

Repository specific Plan Variables, such as *repository.revision.number,* will point to the default repository of a Plan. To address a specific repository, you must add the name of the repository to the end of the variable as follows: *repository.revision.number.product_core.*



elastic agent

An *elastic agent* is an agent that runs in the Amazon Elastic Compute Cloud (EC2). An elastic agent process runs in an elastic instance of an elastic image. An elastic agent inherits its capabilities from the elastic image that it was created from.

elastic Bamboo

Elastic Bamboo allows you to use computing resources from the Amazon Elastic Compute Cloud (EC2) to run builds. Elastic Bamboo uses a remote agent AMI (Amazon Machine Image) to create instances of remote agents in the Amazon EC2.

elastic block store

The Amazon Elastic Block Store (EBS) provides 'EBS volumes' which can attach to EC2 instances. EBS volumes (and the 'EBS snapshots' created from these volumes) provide persistent storage for your elastic instances.

If you have relatively static resources required for building your Bamboo jobs (such as, source code checkouts and Maven repository artifacts), you can add these to an EBS volume. From this volume, you can create an EBS snapshot, which effectively records the 'state' of an EBS volume at a given point in time.

elastic image

An *elastic image* is an Amazon Machine Image (AMI) that is stored in one of Amazon data centers for use with the Elastic Bamboo feature. An elastic image is used to create elastic instances, which in turn create elastic agents. Conceptually, an elastic image is equivalent to an operating system running on a computer's boot hard drive and elastic instances would be the software that runs on this operation system.

Each elastic image registered with the Amazon Web Services (AWS) has its own unique identifier, known as an **AMI ID**.

You can associate multiple elastic images with a Bamboo server. One default shared image is maintained by Atlassian in AWS, and is available to all Elastic Bamboo users.

You can also create your own custom elastic images.

elastic instance

An *elastic instance* is a running instance of an elastic image. One elastic instance is created whenever an elastic image is started. Hence, starting one elastic image multiple times, results in the creation of multiple elastic instances. Each time an elastic instance is created, one elastic agent is created on that instance.

Conceptually, an elastic instance can be thought of as a computer. The elastic agent's processes are run on this computer and the elastic image is the boot hard drive. Unlike computers, however, elastic instances are temporary and stateless. When an elastic instance is shut down:

- Any changes that an elastic instance makes to the boot hard drive (e.g. agent log file) will not persist
- Any customizations to the instance itself will also be lost.

The Amazon Elastic Block Store can provide persistent storage for your elastic instances.

executable

An *executable* is an external program that Bamboo uses during the build process. Generally, executables compile source code to generate compiled executable files (referred to as artifacts in Bamboo). Ant, Maven, MS Build or PHPUnit are just some examples of executables that can be used as part of your build process.

New executables can be defined as capabilities in Bamboo. Once an executable has been defined in Bamboo, it can be configured as part of a task.

See Defining a new executable capability.

favorites

Each Bamboo user can nominate their *favorite* plans — that is, the plans they work with the most.

Each user's favorites are displayed on the 'My' page of the Dashboard. Bamboo administrators can also configure each plan to send build result notifications to users who have nominated the plan as one of their favorites (these users are known as the plan's 'watchers').

global permission

A *global permission* is the ability to perform a particular operation in relation to Bamboo as a whole.See Granting global permissions to users or groups.

See also *plan permission*.

job

A Bamboo *job* is a single build unit within a plan. One or more jobs can be organized into one or more stages. The jobs in a stage can all be run at the same time, if enough Bamboo agents are available. A job is made up of one or more tasks.

A job:

- Processes a series of one or more tasks that are run **sequentially** on the **same** agent.
- Controls the order in which tasks are performed.
- Collects the requirements of individual tasks in the job, so that these requirements can be matched with agent capabilities.
- Defines the artifacts that the build will produce.
- Can only use artifacts produced in a previous stage.
- Specifies any labels with which the build result or build artifacts will be tagged.

Each new plan created in Bamboo contains at least one job known as the Default job.

Projects and plans can only be configured by Bamboo administrators (see Creating a plan).

label

A *label* is a convenient way to tag and group build results that are logically related to each other. Labels can also be used to define RSS feeds.

Labels can be applied to build results automatically, by specifying the label(s) in a plan (note that only Bamboo administrators can do this). Labels can also be applied to build results manually by Bamboo users.

local agent

See agent.

parent

A *parent* is a plan which triggers another plan to build whenever it completes a build.See Setting up plan build dependencies.

permission

See *plan permission* and *global permission*.

plan

A *plan* defines everything about your continuous integration build process in Bamboo.

A plan:

- Has a single stage, by default, but can be used to group jobs into multiple stages.
- Processes a series of one or more stages that are run **sequentially** using the **same** repository.
- Specifies the default repository.
- Specifies how the build is triggered, and the triggering dependencies between the plan and other plans in the project.
- Specifies notifications of build results.
- Specifies who has permission to view and configure the plan and its jobs.
- Provides for the definition of plan variables.

Every plan belongs to a project.

Projects and plans can only be configured by Bamboo administrators (see Creating a plan).

plan permission

A *plan permission* is the ability to perform a particular operation on a plan and its jobs. For each plan, different permissions can be granted to particular groups and/or users.

See Configuring a plan's permissions and Granting plan permissions in bulk.

See also *global permission*.

projects in Bamboo

A *project* is a collection of *plans*. Projects enable you to easily group and identify plans which are logically related to each other. They are especially useful when generating *reports* across multiple plans. For example, you can control access to your projects easily by using project-level permissions.

A project:

- Has none, one, or more, plans.
- Provides reporting (using the wallboard, for example) across all plans in the project.
- Provides links to other applications.
- Allows setting up permissions for all the plans it contains

Projects are created from the dashboard (**Create** > **Create project**), or the Create plan screen. Select New Project from the project dropdown when creating a new plan.

If you're using repository-stored specs in Bamboo, you can define which repositories may have access to your projects.

queue

See *build queue*.

reason

A build's *reason* is the way in which the build was triggered.

Triggering in Bamboo allows plan builds to be started automatically. Bamboo has the following trigger methods:

- **Polling the repository for changes** — Bamboo polls the source repository for changes, either periodically or according to a schedule. This ensures that a plan build only runs when code has changed in the plan's source repository.
- **Repository triggers the build when changes are committed** — Requires that your source repository is configured to fire an event to Bamboo. This has the advantage of placing minimal load on your Bamboo server.
- **Cron-based scheduling** — Builds are run according to a schedule, regardless of whether any code changes have occurred. This can allow a team to structure the day according to a predictable schedule.
- **Single daily build** — The build is run at a specified time every day.

For more information, see Triggering builds.

remote agent

See agent.

See also the Bamboo remote agent installation guide.

remote agent supervisor

A *remote agent supervisor* is an application that is installed alongside a Bamboo remote agent, by default. The remote agent supervisor is an implementation of the Java Service Wrapper.

The remote agent supervisor monitors remote agents on the machine that it is installed on. If any remote agent crashes, the remote agent supervisor will automatically attempt to restart it. If communications are lost with the Bamboo server, the remote agent will shut itself down and wait for the remote agent supervisor to restart it.

The remote agent supervisor will run on the following operating systems:

- **Linux:**
  - x86
  - x86_64
  - IA64
- **Mac OSX:**
  - all architectures
- **Solaris:**
  - x86
  - x86_64
  - IA64 (running in 32 bit mode)
  - SPARC (both 32 bit and 64 bit)
- **Windows:**
  - 32 bit
  - 64 bit

requirement

A *requirement* is specified in a job or a task. A requirement specifies a capability that an agent must have for it to build that job or task. A job inherits all of the requirements specified in its tasks.

Together, capabilities and requirements control which agents can execute builds for particular jobs. Each job can only be built by agents whose capabilities match the job's requirements.

See Configuring a job's requirements for more information.

shared capability

Shared capabilities are inherited by all applicable agents, that is, (shared) local server capabilities are inherited by all local agents, and shared remote capabilities are inherited by all remote agents. Note, however, that the value of a shared capability will be overridden by the value of an *agent-specific capability* of the same name (if one exists).

See:

Agents and capabilities and

Configuring capabilities.

stage

Stages group (or map) jobs to individual steps within a plan's build process. For example, you may have an overall build process plan that comprises a compilation step, followed by several test steps, followed by a deployment step. You can create separate Bamboo stages to represent each of these steps.

A stage:

- By default has a single job but can be used to group multiple jobs.
- Processes its jobs in **parallel**, on **multiple** agents (where available).
- Must successfully complete all its jobs before the next stage in the plan can be processed.
- May produce artifacts that can be made available for use by a subsequent stage.

Each new plan created in Bamboo contains at least one stage (for the default job) and is known as the Default stage. Stages can only be configured by Bamboo administrators.

Stock images

This page describes the latest available stock elastic images, included packages, and capabilities. Elastic Bamboo uses these images by default. In your list of elastic image configurations , an image will have "(stock image)" appended to its name.

Atlassian currently maintains the following public default elastic images:

Operating system	AWS availability
Ubuntu	All regions
Windows Server 2022	All regions

Stock Elastic Images are not guaranteed to be backwards compatible. You shouldn't rely on installed capabilities or image configuration as they can change with each version. We recommend building your own Elastic Image instead.

- Beginning with Bamboo 9.4, all stock images and elastic agents use Java 17 as the system Java. In older releases of Bamboo, the stock images and elastic agents still run on Java 11.
- While the default packages and capabilities listed below may change with each major release of Bamboo, older default images will still be available for use.
- For more information about how to get a list of stock images available for your Bamboo version, see Viewing the list of Bamboo stock images.

On this page:

Ubuntu stock image

The Ubuntu stock image is built with:

- Ubuntu 20.04 LTS
- the Bamboo elastic agent

The images contain the following default packages and capabilities:

Capability /package	Path/value
Builders	
Maven 3.9.4	`/opt/mvn-3.9`
JDKs	
JDK 8	`/opt/jdk-8`
JDK 11	`/opt/jdk-11`
JDK 17	`/opt/jdk-17`

Other	
AWS CLI	
Docker	`/usr/bin/docker` The latest version supplied with your distribution of Linux might be different for Amazon Linux and Ubuntu.
Git 3.4.0	`/usr/bin/git`

Windows stock image

The Windows stock image is built with:

- Windows Server 2022 64-bit with all updates applied
- the Bamboo elastic agent

The Windows stock image contains the following default packages and capabilities:

Capability /package	**Path/value**
Builders	
Maven 3.9.4	`C:\opt\mvn-3.9`
JDKs	
JDK 8	`C:\opt\jdk-8`
JDK 11	`C:\opt\jdk-11`
JDK 17	`C:\opt\jdk-17`
Other	
Git for Windows 2.42.0.2	`C:\opt\git\bin\git.exe`

task

A task:

- Is a small discrete unit of work, such as source code checkout, executing a Maven goal, running a script, or parsing test results.
- Is run sequentially within a job on a Bamboo working directory.

Tasks may make use of an executable if required. Tasks are configured within the scope of a job. A job can be configured to execute a number of tasks, on the same working directory. For example, before executing a Maven goal, the user could substitute specific files within the working directory, substitute version numbers, check out source repositories, or execute a script.

Final tasks for a job are always executed, even if previous tasks in the job failed.

triggering

Triggering in Bamboo allows plan builds to be started automatically. Bamboo has the following trigger methods:

- **Polling the repository for changes** — Bamboo polls the source repository for changes, either periodically or according to a schedule. This ensures that a plan build only runs when code has changed in the plan's source repository.
- **Repository triggers the build when changes are committed** — Requires that your source repository is configured to fire an event to Bamboo. This has the advantage of placing minimal load on your Bamboo server.
- **Cron-based scheduling** — Builds are run according to a schedule, regardless of whether any code changes have occurred. This can allow a team to structure the day according to a predictable schedule.
- **Single daily build** — The build is run at a specified time every day.

For more information, see Triggering builds.

watcher

A plan's *watchers* are the Bamboo users who have marked this plan as one of their favorites. Administrators can configure a plan's notifications to be sent to the plan's watchers.

Contributing to the Bamboo documentation

Would you like to share your Bamboo hints, tips and techniques with us and with other Bamboo users? We welcome your contributions.

Blogging your technical tips and guides

Have you written a blog post describing a specific configuration of Bamboo or a neat trick that you have discovered? Let us know, and we will link to your blog from our documentation.

Contributing documentation in other languages

Have you written a guide to Bamboo in a language other than English, or translated one of our guides? Let us know, and we will link to your guide from our documentation.

On this page:

Related pages:

- Author Guidelines
- Atlassian Contributor License Agreement

Updating the documentation Itself

Have you found a mistake in the documentation, or do you have a small addition that would be so easy to add yourself rather than asking us to do it? You can update the documentation page directly

Getting permission to update the documentation

Please submit the **Atlassian Contributor License Agreement.**

Our style guide

Please read our short guidelines for authors.

How we manage community updates

Here is a quick guide to how we manage community contributions to our documentation and the copyright that applies to the documentation:

- **Monitoring by technical writers.** The Atlassian technical writers monitor the updates to the documentation spaces, using RSS feeds and watching the spaces. If someone makes an update that needs some attention from us, we will make the necessary changes.
- **Wiki permissions.** We use wiki permissions to determine who can edit the documentation spaces. We ask people to sign the Atlassian Contributor License Agreement (ACLA) and submit it to us. That allows us to verify that the applicant is a real person. Then we give them permission to update the documentation.
- **Copyright.** The Atlassian documentation is published under a Creative Commons CC BY license. Specifically, we use a Creative Commons Attribution 2.5 Australia License. This means that anyone can copy, distribute and adapt our documentation provided they acknowledge the source of the documentation. The CC BY license is shown in the footer of every page, so that anyone who contributes to our documentation knows that their contribution falls under the same copyright.

How to Prevent Password Auto-completion in Bamboo

Platform notice: Server and Data Center only. This article only applies to Atlassian products on the server and data center platforms.

The content on this page includes steps to customize or extend Atlassian software (adding/changing CSS rules, HTML, JavaScript, etc.). Per the Atlassian Support Offerings, support does not include customizations made to Atlassian products. Be aware that this material is provided for your information only and using it is done so at your risk.

If you have any questions about this or any customization, please ask the community at Atlassian Answers or consider working with an Atlassian Solution Partner.

Purpose

Some organizations have security requirements that require the AutoComplete attribute to be set to "off" for Usernames and Passwords in form-based authentication. Bamboo does not prevent autocompletion by default. and this can be done by modifying the login form.

Solution

With a few modifications, the autocomplete=off attribute can be set for both the username and password fields. You will need to modify user login template to set autocomplete to **off.**

1. `Edit <BAMBVOO-INSTALLATION>atlassian-bamboo\template\simple\text.ftl`
2. Find the line

   ```
   <input type="text"[#rt/]
   ```

   And modify it to

   ```
   <input  autocomplete="off" type="text"[#rt/]
   ```

3. `Edit <BAMBVOO-INSTALLATION>atlassian-bamboo\template\simple\password.ftl`
4. Find the line

   ```
   <input type="password"<#rt/>
   ```

   And modify it to

   ```
   <input autocomplete="off" type="password"<#rt/>
   ```

5. Save the files, and restart Bamboo Server

Exporting existing plan configuration to Bamboo YAML Specs

To ease migration of your Bamboo plans to configuration as code, we have prepared the export feature.

Note: This will not export historical data for the plans.

In order to export an existing build plan to Bamboo Specs source code:

1. If you don't have Bamboo Specs project, create one.
2. Go to your build plan and select **Actions > Configure plan.**
3. On the plan configuration page, select **Actions > View plan as YAML Specs.**
4. Copy generated YAML code to your code editor by putting it into the bamboo-specs/bamboo.yaml file.

To export an existing deployment plan to Bamboo Specs source code:

1. If you don't have Bamboo Specs project, create one.
2. Go to your deployment project and select **... > Edit project.**
3. On the deployment project configuration page, select **... > View project as YAML Specs.**
4. Copy generated YAML code to your code editor by putting it into the bamboo-specs/bamboo.yaml.

The code is ready to be put into repository, with a few exceptions:

- manually created branches are not exported - you can add them manually in the UI

We advise you to cross-check the generated code with Bamboo Specs reference manual and to compare original and generated plan.

Sensitive data such as passwords or keys are exported as values encrypted using Bamboo's System-wide encryption algorithm. You will see **BAMSCRT@0@**... exported in place of the secret string. For details on generating encrypted secrets for new values, see Bamboo Specs encryption.

You will find equivalent classes for the vast majority of Bamboo concepts in the generated code, such as: plans, stages, jobs, tasks of different kinds, permissions, deployments. The export feature doesn't handle all tasks known to Bamboo. Export of such tasks should be implemented by apps vendors.

Bamboo Data Center

Data Center is our self-managed edition of Bamboo built for enterprises. It provides the deployment flexibility and administrative control you need to manage mission-critical Bamboo sites.

Server and Data Center features comparison

Want to see what's included with a Data Center license? Head to the Bamboo Server and Data Center feature comparison.

Data Center deployment options

You can deploy Bamboo Data Center in two ways:

Non-clustered (single node)

Run Bamboo Data Center on a single node, just like a Server installation. This option doesn't require any changes to your infrastructure, but it does allow you to take advantage of Data Center-only features. Quick and easy. Learn more

Clustered

Run Bamboo Data Center in a cluster with multiple nodes, and a load balancer to distribute traffic. Clustering is designed for large, or mission-critical, Bamboo instances, allowing you to provide high availability, and maintain performance as you scale. Learn more

Bamboo Server and Data Center feature comparison

If you use Bamboo, you'll have either a **Bamboo Server** or **Bamboo Data Center** license.

Your Bamboo license determines which features and infrastructure choices are available.

Feature comparison

Here's a summary of what you get with each license.

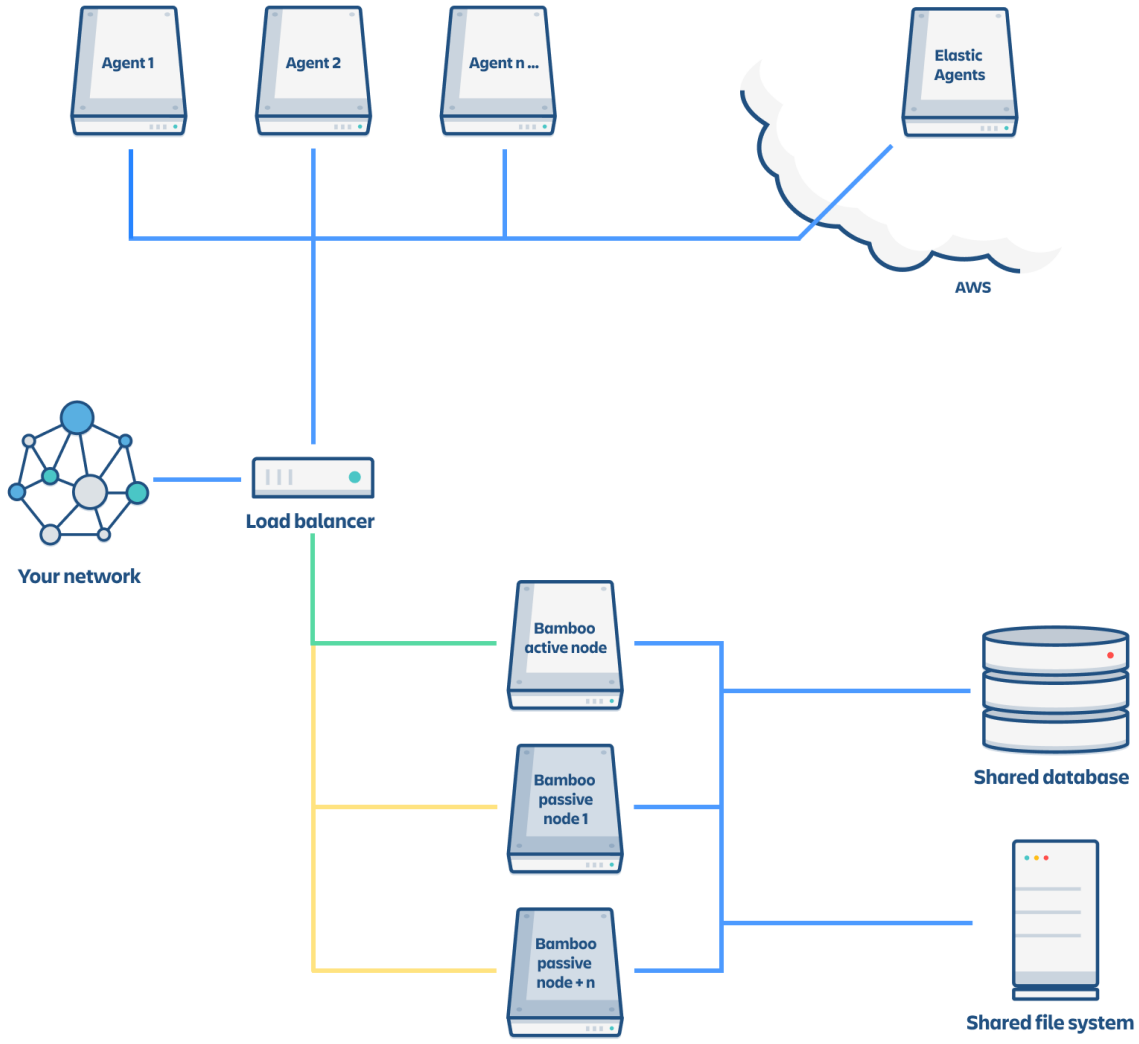
Core	Server license	Data Center license
Building and testing Create multi-stage build plans to run your automated tests.		
Jira integration Connect Jira and Bamboo to automatically link issues.		
Bitbucket integration Connect Bitbucket and Bamboo to link builds with commits and trigger builds when committing to a repository.		
Opsgenie integration Connect Opsgenie and Bamboo to inform about build statutes.		
Bamboo Specs Manage Bamboo with Specs using Java and YAML.		
IAM Role support for Elastic Agents Authenticate Elastic Bamboo with an EC2 instance profile.		
Data Center Apps approval program App vendors can check if their apps are compatible with Bamboo Data Center.		
Ephemeral agents Short-lived, single-purpose Kubernetes agents		
High availability and performance at scale		
Clustering Run Bamboo on cold standby HA.		
Seamless restart Ensures builds queue and progress consistency when Bamboo shuts down.		
Improved build resilience Be sure that your builds are delivered when Bamboo shuts down.		
Improved deployment resilience Be sure that your deployments are delivered when Bamboo shuts down.		

Artifact delivery Ensure that artifacts are delivered when Bamboo shuts down during the build.		
Rate limiting Configurable self-protection against performance drops.		
SFTP artifact handling Reduce the load and disk usage on your Bamboo server by offloading your artifacts to a dedicated storage machine.		
User management		
External user directories Store users in Active Directory, Crowd, Jira, or another LDAP directory.		
Project-level build resources Add and manage repositories and shared credentials on the project level without global admin help.		
Support for single sign-on Support for SAML 2.0, OpenID Connect, and Crowd single sign-on right out of the box.		
OAuth 2 support for external applications Grant limited or full access to your Bamboo instance to an external app on behalf of a Bamboo user.		
Deployment options		
Your own hardware Run Bamboo on your own physical servers, virtualized servers, or in a data center of your choice.		
Kubernetes support Deploy Bamboo together with other Atlassian products on a Kubernetes cluster.		

Bamboo Data Center requirements

Component overview

A Bamboo Data Center instance consists of a cluster of components, each on a dedicated machine, and connected over a high-speed LAN connection, plus the agents. This diagram depicts a typical Bamboo Data Center instance with a load balancer, three application nodes with one active node, a shared database, a shared file system, and multiple agents.



Component requirements

Each component has specific requirements, however, only the load balancer must have a publicly accessible URL. The URL of the Bamboo Data Center instance will be the URL of the load balancer, so this is the machine that you will need to assign the name of your Bamboo Server instance in the DNS. Both user and agent traffic should go through the load balancer.

The remaining machines (Bamboo cluster nodes, shared database, and the shared file system) do not need to be publicly accessible to your users.

Bamboo application nodes

Bamboo cluster nodes all run the Bamboo Data Center web application.

- Each Bamboo cluster node must be a dedicated machine.
- The machines may be physical or virtual.
- The cluster nodes must be connected to a high speed internet (that is, high bandwidth and low latency).
- The usual Bamboo Server supported platforms requirements, including those for Java and Git, apply to each cluster node. For the complete list of Bamboo Server requirements, see Supported platforms.
- The cluster nodes do not all need to be absolutely identical, but for consistent performance we recommend they should be as similar as possible.

  ⚠ If the nodes are not identical, make sure the required executables (like Git or Java) are available under the same path.

- All cluster nodes must run the same version of Bamboo Data Center.
- All cluster nodes must have synchronized clocks (for example, using NTP) and be configured with the identical timezone.

There are no limitations on the number of nodes you can run, but only one Bamboo node will be active at any time. Other nodes will be in stand by and prepared to take over if failures occur.

Load balancer

You can use the load balancer of your choice.

⚠ Load balancer is not bundled with Bamboo.

- Your load balancer should run on a dedicated machine.
- Your load balancer must have a high-speed LAN connection to the Bamboo cluster nodes (that is, high bandwidth and low latency).
- Your load balancer must support ***both*** HTTP(s) mode (for web traffic) ***and*** TCP mode (for agent traffic).
- Your load balancer must have active health checking.
- Terminating SSL (HTTPS) at your load balancer and running plain HTTP from the load balancer to Bamboo Server is highly recommended to improve performance.
- Your load balancer should support "session affinity" (also known as "sticky sessions").
- If you don't have a preference for your load balancer, we provide instructions for haproxy a popular open Source software load balancer. See Connect the new Bamboo cluster node to the load balancer.

Shared database

You must run Bamboo Data Center on an external database.

⚠ You can’t use Bamboo Server's internal H2 database with Bamboo Data Center.

- The shared database must run on a dedicated machine.
- The shared database must be available to all cluster nodes via a high-speed LAN (it must be in the same physical data center).
- All the database vendors listed in Supported platforms remain supported in Bamboo Data Center.

Shared file system

Bamboo Data Center requires a high performance shared file system such as a SAN, NAS, RAID server, or high-performance file server optimized for I/O.

- The shared file system must run on a dedicated machine.
- The file system must be available to all cluster nodes via a high-speed LAN (it must be in the same physical data center).

For more information on setting up Bamboo Data Center's shared file server, see Bamboo home migration. This section contains the requirements and recommendations for setting up NFS for Bamboo Data Center.

Agents

Bamboo Data Center elastic and remote agents have the same requirements as the Bamboo Server agents.

- They all require high-speed connection to the Bamboo Data Center cluster through the TCP Load Balancer, and to the repositories used by the Bamboo plans.
- Agents can share a dedicated machine, but **it is not recommended** to share resources with the Bamboo nodes.

Installing Bamboo Data Center

These instructions are applicable for installing Bamboo Data Center on your own hardware.

This guide covers installing for the first time, with no existing data, or migrating your Bamboo Server to a Data Center instance.

Other ways to install Bamboo Data Center:

- Kubernetes - installation on a Kubernetes cluster using our Helm charts.

Install Bamboo Data Center on a single node

If your organization doesn't need high availability or disaster recovery capabilities right now, you can install Bamboo Data Center without setting up a cluster.

Deploying Data Center on a single node will be the same as deploying a Server installation, just with a different license. Head to Bamboo installation guide to learn about all the ways in which you can install Bamboo.

Install Bamboo Data Center in a cluster

If your organization requires continuous uptime and disaster recovery, you'll want to run Bamboo Data Center in a cluster.

Before you begin

- See Clustering with Bamboo Data Center for a complete overview of hardware and infrastructure considerations.
- Make sure you environment meets requirements listed in Bamboo Data Center requirements.

> Bamboo Data Center supports only 1 active node and unlimited list of cold-standby nodes which started on-demand if active node fails to run.

Provision the shared database and filesystem nodes

Before you install the first Bamboo application node, you need to provision the shared database and shared filesystem to use with Bamboo Data Center.

1. Download Bamboo

To download Bamboo for your operating system, go to download page.

2. Create the installation directory

1. Extract the downloaded file to an install location.

   > The path to the extracted directory is referred to as the `<Bamboo installation directory>` in these instructions.

3. Provision your shared database

Set up your shared database server.

- Connecting Bamboo Server to PostgreSQL
- Connecting Bamboo Server to SQL Server
- Connecting Bamboo Server to Oracle

Ensure your database is configured to allow enough concurrent connections.

See Connecting Bamboo Server to an external database for more information, and note that clustered databases are not supported.

4. Provision your shared file system

You'll need to create a remote directory that is readable and writable by all nodes in the cluster. There are multiple ways to do this, but the simplest is to use an NFS share.

On your file server, ensure that NFS is configured with enough server processes. For example, some versions of Red Hat Enterprise Linux and CentOS have a default of 8 server processes. If you use either distribution, you may need to edit your `/etc/sysconfig/nfs` file, increase the value of `RPCNFSDCOUNT`, and restart the nfs service.

> For the file server and cluster nodes, avoid kernel and NFS version combinations that are unstable or have known NFS bugs. We recommend avoiding Linux kernel versions 3.2 to 3.8.

5. Create the local and shared home directories

Bamboo node should have access to 2 folders it uses for files storage: local and shared.
The shared folder is shared between all cluster nodes and used to keep artifacts and other build results.
The local home is used to keep node-specific files. The local home should not be accessible by other nodes.

Specify your Bamboo home directories before you run Bamboo for the first time.

Create your Bamboo home directories (without spaces in the name).

> You should not create your Bamboo home directory inside the `Bamboo installation directory` — they should be entirely separate locations. If you do put the home directory in the `Bamboo installation directory,` it will be overwritten, and lost, when Bamboo is upgraded.

6. Configure file share mount

On each cluster node, mount the shared home directory as `${BAMBOO_HOME}/shared` . Note that only th e `${BAMBOO_HOME}/shared` directory should be shared between cluster nodes. All other directories, including `${BAMBOO_HOME}` should be node-local (that is, private to each node).

> You can configure a custom location for you home shared folder. Also, when using multiple nodes, you must define a path to a shared home location. See Configuring shared home location.

Suppose your Bamboo home directory is `/var/atlassian/application-data/bamboo`, and your shared home directory is available as an NFS export called `bamboo-san:/bamboo-shared` . To configure the mount on each cluster node:

1. Add the following line to `/etc/fstab` on each cluster node.
   /etc/fstab

```
bamboo-san:/bamboo-shared /var/atlassian/application-data/bamboo/shared nfs rw,nfsvers=3,
lookupcache=pos,noatime,intr,rsize=32768,wsize=32768,_netdev 0 0
```

2. Mount the share on each node:

```
mkdir -p /var/atlassian/application-data/bamboo/shared
sudo mount -a
```

7. Synchronize system clocks

Ensure all your cluster nodes have synchronized clocks and identical timezone configuration. Here are some examples for how to do this:

```
sudo yum install ntp
sudo service ntpd start
sudo tzselect
```

```
sudo apt-get install ntp
sudo service ntp start
sudo dpkg-reconfigure tzdata
```

8. Configure Bamboo home settings

1. Open `<Bamboo installation directory>/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`.
2. Uncomment the following lines:
   - bamboo.home
   - bamboo.shared.home
3. Provide the absolute path to the `${BAMBOO_HOME}` and `${BAMBOO_SHARED_HOME}` directories.

Example:

```
bamboo.home=/var/bamboo/bamboo-home

bamboo.shared.home=bamboo-san:/bamboo-shared
```

9. Start the first cluster node installation

Start Bamboo instance and provide your Bamboo Data Center license. If you need a Bamboo Data Center license, you can purchase one that fits your needs, or, get an evaluation license. Then provide necessary DB connection settings and admin user credentials.

Install and configure your load balancer

Step 1. Configure protocols and health checks on your load balancer

Your load balancer must proxy the following protocols:

Protocol	Typical port on the load balancer	Typical port on the Bamboo cluster nodes	Notes
HTTP	80	8085	HTTP mode. Session affinity ("sticky sessions") should be enabled using the 52-character `BAMBOOSESSIONID` cookies.
HTTPS	443	8085	HTTP mode. Terminating SSL at the load balancer and running plain HTTP to the Bamboo cluster nodes is highly recommended.

TCP	54663	54663	TCP mode. For remote agents connection

Your load balancer must support session affinity ("sticky sessions") using the `BAMBOOSESSIONID` cookie. Bamboo Data Center assumes that your load balancer always directs each user's requests to the same cluster node. If it does not, users may be unexpectedly logged out or lose other information that may be stored in their HTTP session.

When choosing a load balancer, it must support the HTTP, HTTPS, and TCP protocols. Note that:

- Apache does not support TCP mode load balancing.
- HAProxy versions older than 1.5.0 do not support HTTPS.

Your load balancer must support health checks of the cluster nodes. Configure it to perform a periodic HTTP GET of `http://<bamboo>:8085/rest/api/latest/status`, where `<bamboo>` is the cluster node's name or IP address. This returns one of two HTTP status codes:

- 200 OK
- 500 Internal Server Error

If a cluster node does return 200 OK within a reasonable amount of time, the load balancer should direct traffic to it.

You should then be able to navigate to `http://<load-balancer>/`, where `<load-balancer>` is your load balancer's name or IP address. This will take you to your Bamboo Server front page.

If you don't have a particular preference or policy for load balancers, you can use HAProxy which is a popular Open Source software load balancer.

If you choose HAProxy, you must use a minimum version of 1.5.0. Earlier versions of HAProxy do not support HTTPS.

To check which version of HAProxy you use, run the following command:

```
haproxy --version
```

Here is an example `haproxy.cfg` configuration file (typically found in the location `/etc/haproxy/haproxy.cfg`. This assumes:

- Your Bamboo cluster nodes are at address 192.168.0.1 and 192.168.0.2, listening on the default ports 8085 (HTTP) and 54663 (TCP).
- Users will connect to HA Proxy on port 80 with their browser
- You have a valid SSL certificate at `/etc/cert.pem`

```
global
    log 127.0.0.1   local0
    log 127.0.0.1   local1 debug
    maxconn 4096

defaults
    log     global
    mode    http
    option  httplog
    option  dontlognull
    retries 3
    option redispatch
    maxconn 2000
    timeout connect      5000
    timeout client      50000
    timeout server      50000

frontend localnodes
    bind *:80
    mode http
#    stats enable
#    stats uri /haproxy?stats
#    stats realm Strictly\ Private
#    stats hide-version
#    stats auth admin:SECRET_PASSWORD
    default_backend nodes

frontend mainjms
    bind *:54663
    option tcplog
    mode tcp
    default_backend brokers

backend nodes
    mode http
    balance roundrobin
    option forwardfor
    http-request set-header X-Forwarded-Port %[dst_port]
    http-request add-header X-Forwarded-Proto https if { ssl_fc }
    option httpchk GET /rest/api/latest/status HTTP/1.1\r\nHost:\ 127.0.0.1
    cookie BAMBOOSESSIONID insert nocache
    server bamboo1 192.168.0.1:8085 check cookie bamboo1
#    server bamboo2 192.168.0.2:8085 check cookie bamboo2

backend brokers
    mode tcp
    option httpchk GET /rest/api/latest/status HTTP/1.1\r\nHost:\ 127.0.0.1
    server bambooTcp1 192.168.0.1:54663 check port 54663
#    server bambooTcp2 192.168.0.2:54663 check port 54663
```


Review the contents of the `haproxy.cfg` file carefully, and customize it for your environment. See http://www.haproxy.org/ for more information about installing and configuring `haproxy`.

Once you have configured the `haproxy.cfg` file, start the `haproxy` service.

```
sudo service haproxy start
```

You can also monitor the health of your cluster by navigating to HAProxy's statistics page at `http://<load-balancer-url>/haproxy?stats`. You need to uncomment stats section in the config file first.

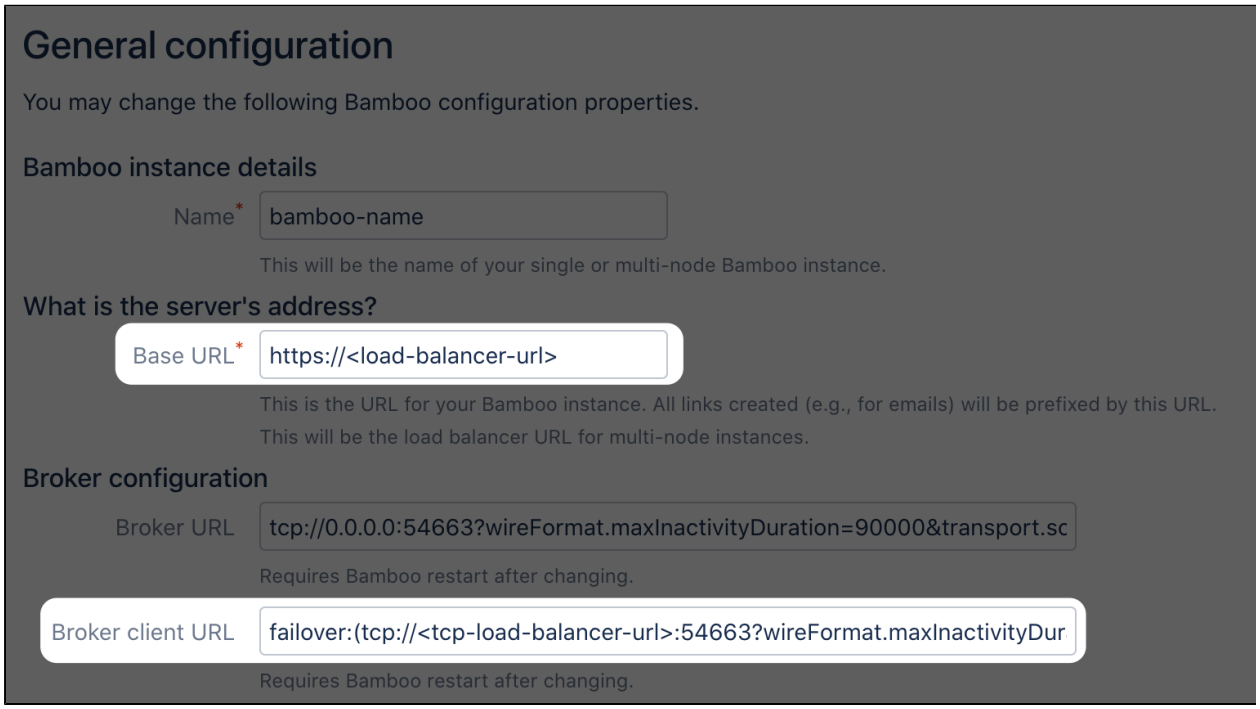
2. Configure Bamboo Server proxy settings

You must include these details into the `Connector` tag in every node of `<bamboo-installation-folder>/conf/server.xml`:

```
<Connector
    ...

    proxyName="<INSERT THE LOAD BALANCER URL>"
    proxyPort="80"
    scheme="http"
>
  ...
</Connector>
```

3. Change base-url and client-broker-url

1. In Bamboo, go to ⚙ > **System** > **General configuration.**
2. Set the Base URL and broker client URL.



4. Add a new Bamboo application node to the cluster

1. On the second node, create an empty Bamboo home directory and mount it in the same location as the Bamboo home directory on the first node.

   ⚠ Don't copy the contents of the Bamboo home directory from the first node to the second node. Bamboo will automatically populate its home directory on first startup

2. Copy the whole Bamboo installation directory from the first node to the same location on the second node.
3. Copy the `bamboo.cfg.xml` file from the root of the Bamboo local home directory on the first node to the same location on the second node.
4. Start Bamboo on the second node.
5. Check the logs to verify that the instance on the second node was able to connect to the database, create the local home folders, and wait for lock acquisition.

5. Connect the new Bamboo cluster node to the load balancer

If you are using your own hardware or software load balancer, consult your vendor's documentation on how to add the new Bamboo cluster node to the load balancer.

If you are using HAProxy, in your `haproxy.cfg` file uncomment the following lines:

```
server bamboo2 192.168.0.2:8085 check cookie bamboo2
```

```
server bambooTcp2 192.168.0.2:54663 check port 54663
```

and restart `haproxy`:

```
sudo service haproxy restart
```

Congratulations!

That's it! Bamboo Data Center is accessible under the following URL: `http://<load-balancer-url>:<port>`

Running Bamboo Data Center on a single node

You can run Bamboo Data Center on a single node, just like a Server installation. This is useful if you don't need cluster-specific benefits – such as high availability.

Benefits of running a non-clustered Data Center deployment

There are a range of reasons you may choose a single node Data Center. Some of the benefits include:

- **Keeping your existing infrastructure**
  Running on a single node means that you can upgrade from Server to Data Center without adding to your infrastructure. In most cases, moving to Data Center will be as simple as updating your license.
- **Accessing Data Center-only features**
  Your Data Center license unlocks a suite of additional features to help you easily manage enterprise-grade Bamboo instance – like SAML single sign-on and improved build resiliency

Architecture

Bamboo Data Center deployed on a single node looks just as a Server installation, and consists of:

- Bamboo Data Center server, running on a single node
- A database that Bamboo read and writes to
- Agents that execute your builds and deployments

Requirements

Non-clustered Data Center deployments follow the same minimum requirements as a Server installation. Check our Bamboo Supported platforms for more details.

App compatibility

The process for installing Marketplace apps (also known as add-ons or plugins) in Bamboo Data Center is the same as for Server. You won't have to stop Bamboo to install or update an app.

The Atlassian Marketplace indicates apps that are compatible with Bamboo Data Center. Learn more about Data Center approved apps

Ready to get started?

Deploying Data Center on a single node will be the same as deploying a Server installation, just with a different license. Head to Bamboo installation guide to learn about all the ways in which you can install Bamboo.

Moving back to Server

BAMBOO 8.0 EARLY ACCESS PROGRAM

If you no longer need Data Center, you can go back to a Server installation. This will be as easy as updating your license, but be aware that you'll immediately lose features that are exclusive to Data Center.

After moving from Data Center to Server, you will lose features exclusive to Data Center. Here's the summary of how this will affect your current Bamboo instance:

Feature	Description
SAML single sign-on Use a SAML identity provider for authentication and single-sign on.	• SAML login won't work; the users will always be redirected to a login screen. • Users who had a password on their Atlassian account before SAML single sign-on was enabled will use that to log in. • Users who joined after SAML single sign-on was enabled will need to reset their password for their Atlassian account when they log in next time.
Build resiliency Ability for the builds and deployments executed on remote agents (including EC2 agents) to continue despite Bamboo server being stopped.	• No longer available • Data stored for the purpose of supporting build resiliency is removed from the filesystem • Results returned by agents after server restart are ignored
View configuration permission Explicit permission to browse plans' and deployments' configuration without ability to edit it.	• Hidden from the permissions UI • The permission is *not* revoked from the users having it • No change to how permission are checked; actions that require the permission will still be accessible to users with the permission • The permission is implicitly granted when granting edit or clone permission • The permission is implicitly revoked when revoking *both* edit and clone permissions • Explicit granting of this permission in Bamboo Specs is ignored.

Before you begin

- This guide assumes that you're using a standalone Bamboo Data Center.
- We recommend completing this process in a staging environment, and running a set of functional tests, integration tests, and performance tests, before making these changes in production.

To move back to Bamboo Server

1. Get a license for Bamboo Server.
   You can find existing Server license at my.atlassian.com.
2. Apply the license:
   a. Go to **Administration > Licensing**.
   b. Update the license.

After updating the license, your Bamboo instance will become a Server one.

Clustering with Bamboo Data Center

Bamboo Data Center allows you to run a cluster of multiple Bamboo nodes, providing disaster recovery. We'll tell you about the benefits, and give you an overview of what you'll need to run Bamboo in a clustered environment.

Bamboo Data Center supports only 1 active node and unlimited list of cold-standby nodes which started on-demand if active node fails to run.

Benefits of clustering

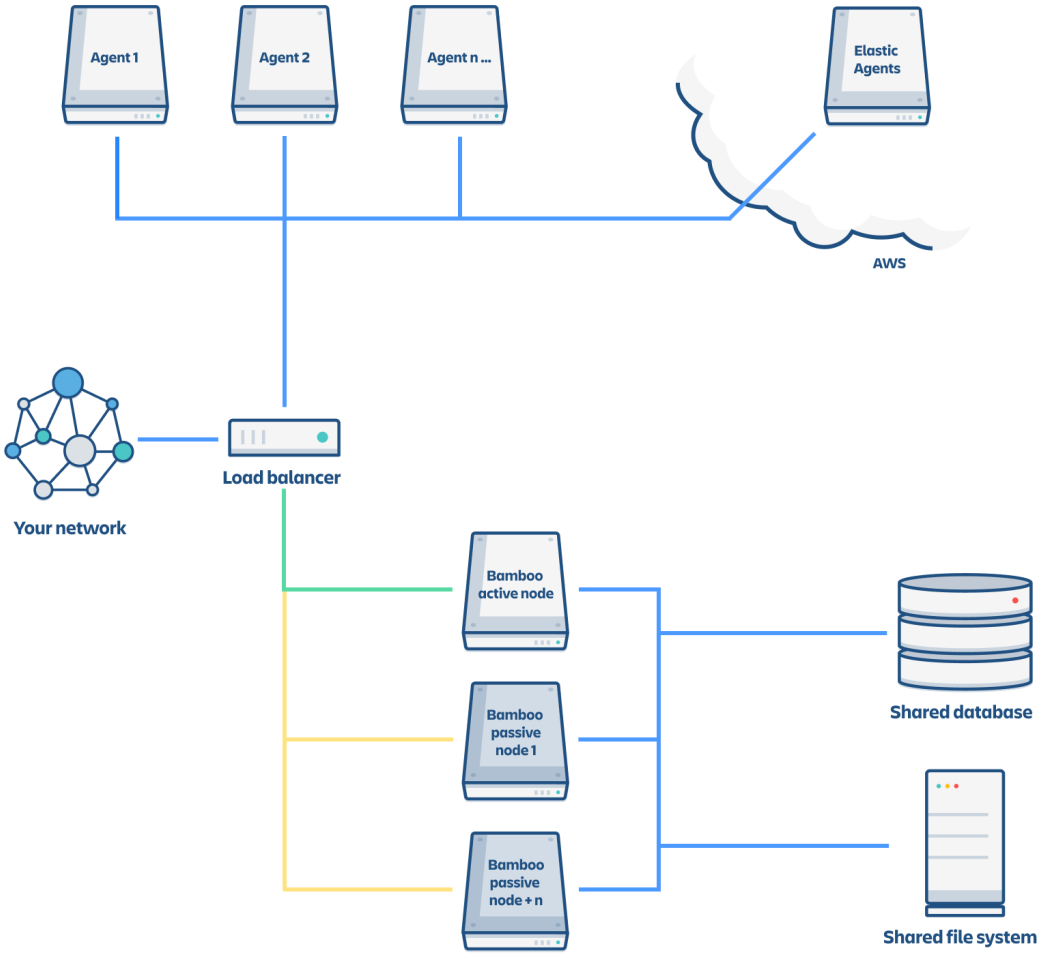
Clustering is designed for enterprises with large or mission-critical Data Center deployments that require continuous uptime and disaster recovery.

Here are some of the benefits:

- **High availability and failover**

  If one node in your cluster goes down, the others take on the load, ensuring your users have shortly interrupted access to Bamboo.

Architecture



A Bamboo Data Center cluster consists of:

- Multiple identical application nodes running Bamboo Data Center.
- A load balancer to distribute traffic to all of your application nodes.
- A shared file system that stores build results, build logs, artefacts, and other shared files.
- A database that all nodes read and write to.

Only one application node is active and process requests. Other nodes are in cold-stanby mode and ready to start if active node fails to run. A user will access the same Bamboo node for all requests until their session times out, they log out, or a node is removed from the cluster.

Infrastructure and requirements

The choice of hardware and infrastructure is up to you. Below are some areas to think about when planning your hardware and infrastructure requirements.

You should not run additional applications (other than core operating system services) on the same servers as Bamboo. Running Bamboo, Bitbucket, and Jira on a dedicated Atlassian software server works well for small installations but is discouraged when running at scale.

Bamboo Data Center can run successfully on virtual machines.

Each node does not need to be identical, but for consistent performance we recommend they are as close as possible. All cluster nodes must:

- be a dedicated machine, physical or virtual
- be located in the same data center, or region (for AWS and Azure)
- be connected in a high speed LAN (that is, high bandwidth and low latency)
- have the same OS, Java and application server version. See Supported platforms.
- have the same memory configuration (both the JVM and the physical memory) (recommended)
- be configured with the same time zone (and keep the current time synchronized). Using ntpd or a similar service is a good way to ensure this

You must ensure the clocks on your nodes don't diverge, as it can result in a range of problems with your cluster.
Your Data Center license does not restrict the number of nodes in your cluster. The right number of nodes depends on the size of your Bamboo instance, and the size of your nodes.
You should ensure your intended database is listed in the current Supported platforms. The load on an average cluster solution is higher than on a standalone installation, so it is crucial to use a supported database.
Bamboo Data Center requires a high performance shared file system such as a SAN, NAS, RAID server, or high-performance file server optimized for I/O e.g. NFS.

- The shared file system must run on a dedicated machine.
- The file system must be available to all cluster nodes via a high-speed LAN (it must be in the same physical data center).

You can use the load balancer of your choice. Bamboo Data Center does ***not*** bundle a load balancer.

- Your load balancer should run on a dedicated machine.
- Your load balancer must have a high-speed LAN connection to the Bamboo cluster nodes (that is, high bandwidth and low latency).
- Your load balancer must support ***both*** HTTP mode (for web traffic) ***and*** TCP mode (for remote agents ActiveMQ traffic).
- Terminating SSL (HTTPS) at your load balancer and running plain HTTP from the load balancer to Bamboo Server is highly recommended for performance.
- Your load balancer should support "session affinity" (also known as "sticky sessions").
- If you don't have a preference for your load balancer, we provide instructions for haproxy, a popular Open Source software load balancer.

Many load balancers require a URL to constantly check the health of their backends in order to automatically remove them from the pool. It's important to use a stable and fast URL for this, but lightweight enough to not consume unnecessary resources. The following URL returns Bamboo's status and can be used for this purpose.

URL	Expected content	Expected HTTP status
http://<bamboourl>/rest/api/latest/status	{"state":"RUNNING"}	200 OK

HTTP status code	Response entity	Description
200	{"state":"RUNNING"}	Running normally
500	{"state":"ERROR"}	An error state
503	{"state":"STARTING"}	Application is starting
503	{"state":"STOPPING"}	Application is stopping
200	{"state":"FIRST_RUN"}	Application is running for the first time and has not yet been configured

404		Application failed to start up in an unexpected way (the web application failed to deploy)

Use separate network adapters for communication between servers. Cluster nodes should have a separate physical network (i.e. separate NICs) for inter-server communication. This is the best way to get the cluster to run fast and reliably. Performance problems are likely to occur if you connect cluster nodes via a network that has lots of other data streaming through it.

App compatibility

The process for installing Marketplace apps (also known as add-ons) in a Bamboo cluster is the same as for a standalone installation. You will not need to stop the cluster, or bring down any nodes to install or update an app.

The Atlassian Marketplace indicates apps that are compatible with Bamboo Data Center. Learn more about Data Center approved apps

Bamboo home migration

As part of the cold stand-by capabilities of Bamboo Data Center, major changes were made to the home folder. The new organization is a requirement for using multiple nodes with Bamboo Data Center, but this upgrade will affect every type of Bamboo installation, including Server and single-node Data Center setups.

This upgrade splits the home folder into two parts: one containing data which will be exclusive for each instance, and another folder that contains the files that will be shared among every node. We will refer to these folders as **local home** and **shared home** respectively.

Who does it affect?

This upgrade task is applied to every instance of Bamboo that will be upgraded to Bamboo 8.0 and doesn't contain custom paths for folders that are by default placed inside Bamboo's home folder or if you execute the migration manually, before upgrading your instance.

Bamboo will not move your data if your instance is configured with custom paths. Keep in mind that multiple nodes will require access to these folders when using cold standby.

DO I USE CUSTOM PATHS?

You can find this information at **Bamboo Administration** > **System Information** > **Bamboo paths**. The path of the directories listed must all be part of the Bamboo home.

What will be changed?

Certain folders and their data will be moved to the `/shared` folder, inside Bamboo home.

If you are upgrading to a Data Center license and plan to use the cold stand-by setup, you need to make sure the shared folder will be available over the network for every node. Keeping a network-shared folder inside the Bamboo home might be sub-optimal and you might want to place it somewhere outside. You can do that by configuring a different shared home location. See Configuring shared home location.

We also took the opportunity to make minor adjustments to simplify the names of folders.

Lucene indexes were moved to the shared home folder and that impacts the velocity of reindexing in Bamboo. We deprecated Lucene in Bamboo 8.0.

Follow the list of folders and files that are being migrated. Keep in mind that the default location for `<bamboo-shared-home>` is inside the local home `<bamboo-home>/shared`.

Folder	Bamboo 7.2 path		Bamboo 8.0 path
Artifacts	<bamboo-home>/artifacts		<bamboo-shared-home>/artifacts
Attachments	<bamboo-home>/attachments		<bamboo-shared-home>/attachments
Backups	<bamboo-home>/backups		<bamboo-shared-home>/backups
Export	<bamboo-home>/export		<bamboo-shared-home>/export
Exports	<bamboo-home>/exports		<bamboo-shared-home>/exports
Index	<bamboo-home>/index		<bamboo-shared-home>/index
JMS store	<bamboo-home>/jms-store		<bamboo-shared-home>/jms-store
Plugins	<bamboo-home>/plugins		<bamboo-shared-home>/plugins
Server state	<bamboo-home>/serverState		<bamboo-shared-home>/serverState
Templates	<bamboo-home>/templates		<bamboo-shared-home>/templates
Repositories cache	<bamboo-home>/xml-data/build-dir		<bamboo-home>/build-dir

Build results	<bamboo-home>/xml-data/builds		<bamboo-shared-home>/builds
Configuration	<bamboo-home>/xml-data/configuration		<bamboo-shared-home>/configuration

How do I know if the migration completed?

Once the migration is completed, you will see the following message in the application logs:

```
 2021-01-01 00:00:00 INFO [Thread] The migration of folders to the shared home has been completed.
```

If there is an error during the execution of this upgrade task, please follow the instructions provided or if you need help, contact the Atlassian support.

Configuring shared home location

BAMBOO 8.0 EARLY ACCESS PROGRAM

When using Bamboo with multiple nodes, you must define a path to the shared home location.

To define a path to shared home:

1. Use one of the following methods:

- JVM Property:

```
Dbamboo.shared.home=<path>
```

- Environment Variable:

```
export BAMBOO_SHARED_HOME=<path>
```

- Application property in `bamboo-init.properties`:

    1. Go to `atlassian-bamboo/WEB-INF/classes/.`
    2. In the `bamboo-init.properties` file, set the following property:

    ```
    bamboo.shared.home=<path>
    ```

If you don't define a custom shared home location, Bamboo will use the following path: `{bamboo.home}/shared`

Home directory folder list

Here's the complete list of content inside the home directory in Bamboo 7.2, and what is their status after upgrade to 8.0. Remember that this table is listing the default location of the `shared` folder. You can also configure a custom location of your shared home folder. See Configuring shared home location.

Name	Description	Path in Bamboo 7.2 relative to Bamboo home	Status in Bamboo 8.0
Analytics	Used by Analytics plugin to buffer analytics events which are sent periodically.	`/analytics-logs`	*Unchanged*
Artifacts	Build artifacts.	`/artifacts`	Moved to shared home
Attachments	Contains the favicon and logo image for the custom look and feel.	`/attachments`	Moved to shared home
Backups	XML backup of the database.	`/backups`	Moved to shared home
Caches	Instance cache folder.	`/caches`	*Unchanged*
Database	Used by evaluation-only H2 DB.	`/database`	Moved to shared home
Export	ATST plugin support zip folder.	`/export`	Moved to shared home
Exports	DB backup to be exported by the node.	`/exports`	Moved to shared home
Index	Used by the deprecated Lucene index.	`/index`	Moved to shared home
JMS Store	Used by KahaDB. (ActiveMQ persistence)	`/jms-store`	Moved to shared home
Logs	Instance logs.	`/logs`	*Unchanged*
Plugins	Plugin installation folder.	`/plugins`	Moved to shared home
Server state	Used by the seamless restarts feature in Bamboo DC.	`/serverState`	Moved to shared home
Temporary	Temporary files used by the script tasks, Docker runner, build warnings task, etc.	`/temp`	*Unchanged*
Templates	Used by Freemarker to allow customization of Bamboo UI.	`/templates`	Moved to shared home
Build directory	Working directory for repository caches and the deprecated local agents builds.	`/xml-data/build-dir`	Moved to `<bamboo-home>/local-working-dir`
Build results	Contains the job builds results and their downloadable data, like logs.	`/xml-data/builds`	Moved to `<bamboo-shared-home>/builds`

Configuration	Contains the cypher, broker key storage for SSL connection to agents, serialization whitelist, crowd.properties and administration.xml file, which stores the instances settings also available through UI.	`/xml-data/configuration`	Moved to `<bamboo-shared-home>/configuration`

Set up a Bamboo Data Center cold standby

Bamboo Data Center allows you to run a cluster of Bamboo nodes in a cold standby configuration, providing higher availability. This guides walks you through the process of configuring a Data Center cluster on your infrastructure.

Not sure if a cold standby is right for you? Check out Running Bamboo Data Center in a cluster for a detailed overview.

Before you begin

Things you should know about when setting up your Data Center:

See our supported platforms for information on the database, Java, and operating systems you'll be able to use. These requirements are the same for Server and Data Center deployments.
You can see a component diagram of a typical Bamboo Data Center instance, and read about detailed requirements of each component on the Bamboo Data Center requirements page

A Bamboo Data Center instance consists of a cluster of components, each running on a dedicated machine:

- A **cluster of Bamboo application nodes** all running the same version of Bamboo Data Center web application. These can be virtual or physical machines, have synchronized clocks (for example, using NTP) and be configured with the identical timezone.
- A **load balancer** that supports *both* HTTP mode (for web traffic) *and* TCP mode (for ActiveMQ traffic), and support session affinity ("sticky sessions"). If the load balancer doesn't support both modes, then configure two separate load balancers for each traffic type, e.g. Amazon ALB and Amazon ELB.
- A **supported external database**, shared and available to all all cluster nodes.
- A **shared file system** that is physically located in the same data center, available to all clusters nodes, and accessible by NFS as a single mount point.

  > You'll need to create a remote directory that is readable and writable by all nodes in the cluster. There are multiple ways to do this, but the simplest is to use an NFS share, which we use as an example.

In this guide we'll use the following terminology:

- **Installation directory**: The directory where you installed Bamboo.
- **Local home directory**: The home or data directory stored locally on each cluster node (if Bamboo is not running in a cluster, this is simply known as the home directory).
- **Shared home directory**: The directory you created that is accessible to all nodes in the cluster, preferably via the same path. If you are not running in a cluster, this directory will be present inside the home directory.

To set up and configure your cluster

We recommend completing this process in a staging environment, and testing your cold standby installation, before moving to production.

1. Install Bamboo Data Center on the first application node

First, you'll need make a fresh installation of Bamboo Data Center in one node following the Bamboo installation instructions.

2. Provision the shared database and filesystem

Once you've installed the first Bamboo application node, you now need to provision the share database and shared filesystem to use with Bamboo Data Center.

Step 1. Provision your shared database

Set up your shared database server.

- Connecting Bamboo Server to PostgreSQL
- Connecting Bamboo Server to SQL Server
- Connecting Bamboo Server to Oracle

Ensure your database is configured to allow enough concurrent connections. For example, in PostgreSQL the default limit is usually 100 connections. If you use PostgreSQL, you may need to edit your `postgresql.conf` file, to increase the value of `max_connections`, and restart Postgres.

Note that, while cold standby nodes are mostly idle, they periodically connect to the database in order to update their status and, possibly, decide to take over the role of the primary node.

See Connecting Bamboo Server to an external database for more information, and note that clustered databases are not supported.

Step 2. Provision your shared file system

A properly resourced and configured NFS server can perform well even under very heavy load. We've created some recommendations for setting up and configuring your file server for optimal performance.

Bamboo Data Center requires a high performance shared file system, such as a storage area network (SAN), network-attached storage (NAS), RAID server, or high-performance file server optimized for input/output.

The file system must:

- run on a dedicated machine; avoid hosting other services on your NFS server
- be in the same physical data center
- be available to all Bamboo nodes via a high-speed LAN (such as 10GB ethernet or Fibre Channel)

You need to create a user account named **bamboo** on the shared file system server. This user account should have read/write permissions to the shared subdirectory of the Bamboo shared home directory.

To ensure this:

- set **bamboo** to own all files and folders in the shared subdirectory
- create **bamboo** with the user umask 0027
- assign the same UID to bamboo on all NFS Server and Bamboo cluster nodes

On your file server, ensure that NFS is configured with enough server processes. For example, some versions of Red Hat Enterprise Linux and CentOS have a default of 8 server processes. If you use either distribution, you may need to edit your **/etc/sysconfig/nfs** file, increase the value of `RPCNFSDCOUNT`, and restart the NFS service.

For the file server and cluster nodes, avoid kernel and NFS version combinations that are unstable or have known NFS bugs. We recommend avoiding Linux kernel versions 3.2 to 3.8.
When you provision your application cluster nodes later, we recommend using the following NFS mount options:

```
rw,nfsvers=3,lookupcache=pos,noatime,intr,rsize=32768,wsize=32768,_netdev
```

If your Bamboo server is running a version older than 8.0, you'll need to upgrade to 8.0 first. This will rearrange Bamboo server home, creating the shared subdirectory.

Once on 8.0, the shared subdirectory of the Bamboo Server home directory will contain all the configuration data, build and deployment result files, among other important files. The migration will consist of moving this folder to the Bamboo Data Center's NFS file system and alter the path of the shared home folder inside the ${BAMBOO_INSTALLATION_FOLDER}/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties

```
#bamboo-init.properties
bamboo.home=/your/local/home/path
bamboo.shared.home=/the/mounted/shared/filesystem/path+shared
```

Remember to back up the home folder before moving it to the new place.

The remaining of the subdirectories (analytics-logs, caches, export, local-working-dir,lib, logs, plugins, and temp), inside the now local home path, contain only caches, workspaces of the local agents and temporary files. You don't need to restore them.

Provision application cluster nodes

Provision cluster node infrastructure. You can automate this using a configuration management tool such as Chef, Puppet, or Vagrant, and/or by spinning up identical virtual machine snapshots.

Step 1. Configure file share mounts

On each cluster node, mount the shared home directory as any path. We recommend using the same filesystem path on all the nodes for the sake of simplicity. Configure that path as bamboo shared home, by exporting `${BAMBOO_SHARED_HOME}` or setting `bamboo.shared.home` in the **bamboo-init.properties** file. Alternatively, if you can skip the last step by mounting the shared directory as `${BAMBOO_HOME}/shared`, which is the default location of shared bamboo home in case it is not defined explicitly. Note that ***only*** the shared directory should be shared between cluster nodes. All other directories, including `${BAMBOO_HOME}`, should be node-local (that is, private to each node).

Example

For example, suppose your Bamboo home directory is `/var/atlassian/application-data/bamboo`, and your shared home directory is available as an NFS export called **bamboo-san:/bamboo-shared**. To configure the mount on each cluster node:

1. Add the following line to `/etc/fstab` on each cluster node.
   /etc/fstab

   ```
   bamboo-san:/bamboo-shared /var/atlassian/application-data/bamboo/shared nfs rw,nfsvers=3,
   lookupcache=pos,noatime,intr,rsize=32768,wsize=32768,_netdev 0 0
   ```

2. Mount the share on each node. Issue:

   ```
   mkdir -p /var/atlassian/application-data/bamboo/shared
   sudo mount -a
   ```

Step 2. Synchronize system clocks

Ensure all your cluster nodes have synchronized clocks and identical timezone configuration. Here are some examples for how to do this:

```
sudo yum install ntp
sudo service ntpd start
sudo tzselect
```

```
sudo apt-get install ntp
sudo service ntp start
sudo dpkg-reconfigure tzdata
```

Step 3. Install Bamboo Data Center on each node

On each cluster node, perform the same steps from Install Bamboo Data Center on the first application node but without making migrations or running a fresh installation. Before starting the new nodes in the latter steps, you need to make sure that the installation has the correct paths, and that the **bamboo.cfg.xml** from the initial node is copied inside the local home folder of this node.

If you automate the installation or clone the new cluster node, make sure that the `cluster-node.properties` file **doesn't exist** on the new cluster node. Bamboo uses the `cluster-node.properties` file to identify the nodes. Any duplicates of that file will cause multiple active nodes to start in parallel, leading to data corruption.

Step 4. Start the first cluster node

If you haven't configured the shared home folder path yet, edit **${BAMBOO_INSTALLATION_FOLDER}/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties** by altering the `bamboo.shared.home` property. Remember that every node will need access to this folder over the NFS.

```
#bamboo-init.properties
bamboo.home=/your/local/home/path
bamboo.shared.home=/the/mounted/shared/filesystem/path+shared
```

Install and configure your load balancer

Step 1. Configure protocols and health checks on your load balancer

Your load balancer must proxy three protocols:

Protocol	Typical port on the load balancer	Default port on the Bamboo cluster nodes	Notes
HTTP	80	8085	HTTP mode. Session affinity ("sticky sessions") should be enabled.
HTTPS	443	8085	HTTP mode. Terminating SSL at the load balancer and running plain HTTP to the Bamboo cluster nodes is highly recommended.
TCP	54663	54663	TCP mode. Agents (ActiveMQ) use TCP to connect to Bamboo nodes.

Your load balancer must support session affinity ("sticky sessions"). Bamboo Data Center assumes that your load balancer always directs each user's requests to the same cluster node.

When choosing a load balancer, it must support the HTTP, HTTPS, and TCP protocols. Note that:

- Apache doesn't support TCP mode load balancing.
- HAProxy versions older than 1.5.0 do not support HTTPS.

If your load balancer supports health checks of the cluster nodes, configure it to perform a periodic HTTP GET to http://<bamboo>:8085/rest/api/latest/status, where <bamboo> is the cluster node's name or IP address.

If a cluster node does not return 200 OK within a reasonable amount of time, the load balancer should not direct any traffic to it.

You should then be able to navigate to `http://<load-balancer>/`, where `<load-balancer>` is your load balancer's name or IP address. This should take you to your Bamboo front page.

Improving instance stability with rate limiting

When automated integrations or scripts send requests to Bamboo in huge bursts, it can affect Bamboo's stability, leading to drops in performance or even downtime. With rate limiting, you can control how many external REST API requests automations and users can make and how often they can make them, making sure that your Bamboo instance remains stable.

Rate limiting is available for **Bamboo Data Center** only.

Skip to

How rate limiting works

Here's some details about how rate limiting works in Bamboo.

Rate limiting targets only external REST API requests, which means that requests made within Bamboo aren't limited in any way. When users move around the Bamboo user interface, viewing projects, transitioning issues, and completing other actions, they won't be affected by rate limiting, as we're seeing this as a regular user experience that shouldn't be limited.

Let's use an example to better illustrate this:

- When a user views an issue in Bamboo, a number of requests are sent in the background — these requests ask Bambo for comments, assignees, attachments, etc. Since this traffic is internal to Bamboo, it won't be limited.
- When the same user opens up the terminal on their laptop and sends a request to get information about an issue, it will be rate limited because it's made outside of Bamboo.

Authentication mechanisms

To give you more details on how we recognize which requests should be limited, we're targeting external HTTP requests with these authentication mechanisms:

- Basic auth
- OAuth
- JSESSIONID cookie

Out of the many available techniques for enforcing rate limits, we've chosen to use **token bucket**, which gives users a balance of tokens that can be exchanged for requests. Here's a summary of how it works:

- Users are given tokens that are exchanged for requests. One token equals one request.
- Users get new tokens at a constant rate so they can keep making new requests. This is their *Requests allowed*, and can be, for example, 10 every 1 minute.
- Tokens are added to a user's personal bucket until it's full. This is their *Max requests* and allows them to adjust the usage of tokens to their own frequency, for example 20 every 2 minutes instead of 10 every 1 minute, as specified in their usual rate.
- When a user tries to send more requests than the number of tokens they have, only requests that can draw tokens from the bucket will be successful. The remaining ones will end in a 429 error message (too many requests). The user can retry those requests once they get new tokens.

Bamboo works best when used with our other products like Bitbucket, or Jira. Technically, products like these are external to Bamboo, so they should be limited. In this case, however, we're treating them as belonging to the same user experience and don't want to enforce any limits for requests coming from or to these products.

The way it is now:

- **Server:** Not limited in any way.
- **Cloud:** There's a known issue that applies rate limits to requests coming from/to cloud products. We're working hard to disable rate limits for cloud products and should make that happen soon. For now, you can use a workaround. For more info, see Removing rate limits for Atlassian cloud products.

The general assumption is that Marketplace apps are installed on a Bamboo instance, make internal requests from within Bamboo, and shouldn't be limited. But, as always, **it depends on how an app works**.

- **Internal:** If an app in fact works internally, enhancing the user experience, it won't be limited. An example of such app would be a special banner that's displayed on a Scrum board. Let's say this banner checks all issues that were done and shows this sprint's winner — a user who's completed the most issues in this sprint. Traffic like that would be internal, not limited.
- **External:** Apps whose requests are external to Bamboo are limited. Let's say we have an app that displays a wallboard on TV. It asks Bamboo for details about boards, issues, assignees, etc. and then reshuffles and displays them in its own way as the earlier mentioned wallboard. An app like that sends external requests and behaves just like a user sending requests over a terminal.

It really depends on the app, but we're assuming most of them shouldn't be limited.
Rate limiting is available for Data Center, so you most likely have a cluster of nodes behind a load balancer. You should know that each of your users will have a separate limit on each node (rate limits are applied per node, not per cluster).

In other words, if they have used their *Requests allowed* on one node and were rate limited, they could theoretically send requests again if they started a new session on a different node. Switching between the nodes isn't something users can do, but keep in mind that this can happen.

Whatever limit you've chosen (e.g. 100 requests every 1 hour), the same limit will apply to each node, you don't have to set it separately. This means that each user's ability to send requests will still be limited, and Bamboo will remain stable regardless of which node their requests are routed to.
Setting the right limit depends on many factors, so we can't give you a simple answer. We have some suggestions, though.

Finding the right limit

The first step is to understand the size of traffic that your instance receives. You can do this by parsing the access log and finding a user than made the most REST requests over a day. Since UI traffic is not rate limited, this number will be higher than what you need as your rate limit. Now, that's a base number — you need to modify it further based on the following questions:

1. Can you afford to interrupt your users' work? If your users' integrations are mission-critical, consider upgrading your hardware instead. The more critical the integrations, the higher the limit should be — consider multiplying the number you found by two or three.
2. Is your instance already experiencing problems due to the amount of REST traffic? If yes, then choose a limit that's close to the base number you found on a day when the instance didn't struggle. And if you're not experiencing significant problems, consider adding an extra 50% to the base number — this shouldn't interrupt your users and you still keep some capacity.

In general, the limit you choose should keep your instance safe, not control individual users. Rate limiting is more about protecting Bamboo from integrations and scripts going haywire, rather than stoping users from getting their work done.

How to turn on rate limiting

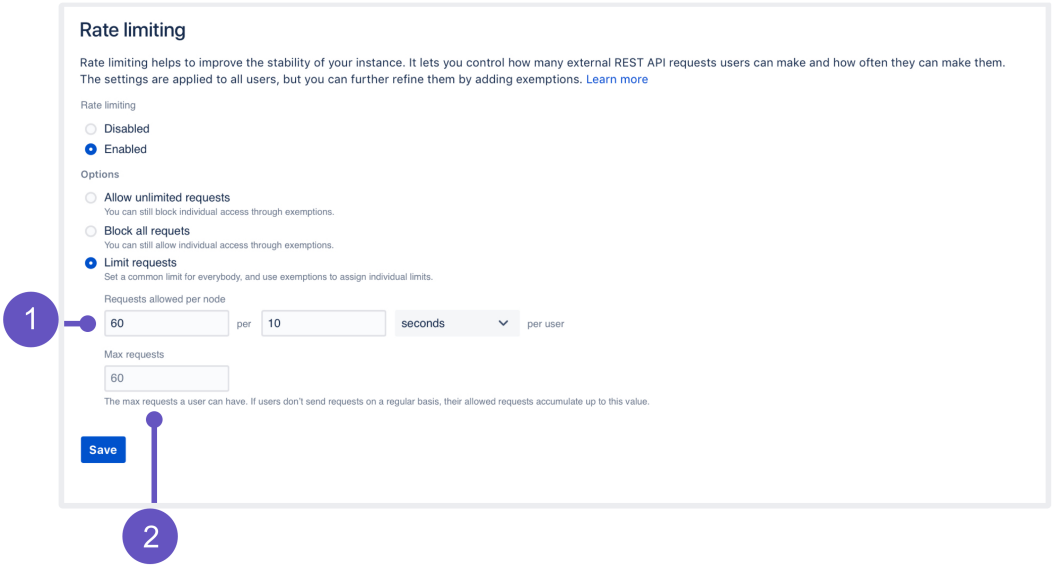
> You need to be a Bamboo Administrator to turn on rate limiting.

To turn on rate limiting:

1. Go to **Administration > System > Rate limiting**.
2. Change the status to **Enabled**.
3. Select one of the options: **Allow unlimited requests**, **Block all requests**, or **Limit requests**. The first and second are all about allowlisting and blocklisting. For the last option, you'll need to enter actual limits. You can read more about them below.
4. Click **Save**.
5. Make sure to add exemptions for users who really need those extra requests, especially if you've chosen allowlisting or blocklisting. See Adding exemptions.

Limiting request — what it's all about

As much as allowlisting and blocklisting shouldn't require additional explanation, you'll probably be using the **Limit requests** option quite often, either as a global setting or in exemptions.



Let's have a closer look at this option and how it works:

1. **Requests allowed:** Every user is allowed a certain amount of requests in a chosen time interval. It can be 10 requests every second, 100 requests every hour, or any other configuration you choose.
2. **Max requests (advanced):** Allowed requests, if not sent frequently, can be accumulated up to a set maximum per user. This option allows users to make requests at a different frequency than their usual rate (for example, 20 every 2 minutes instead of 10 every 1 minute, as specified in their rate), or accumulate more requests over time and send them in a single burst, if that's what they need. Too advanced? Just make it equal to *Requests allowed*, and forget about this field — nothing more will be accumulated.

Examples

Requests allowed: 10/hour | Max requests: 100

- One of the developers is sending requests on a regular basis, 10 per hour, throughout the day. If they try sending 20 requests in a single burst, only 10 of them will be successful. They could retry the remaining 10 in the next hour when they're allowed new requests.
- Another developer hasn't sent any requests for the past 10 hours, so their allowed requests kept accumulating until they reached 100, which is the max requests they can have. They can now send a burst of 100 requests and all of them will be successful. Once they used up all available requests, they have to wait for another hour, and they'll only get the allowed 10 requests.
- If this same developer sent only 50 out of their 100 requests, they could send another 50 right away, or start accumulating again in the next hour.

Requests allowed: 1/second | Max requests: 60

- A developer can choose to send 1 request every second or 60 requests every minute (at any frequency).

- Since they can use the available 60 requests at any frequency, they can also send all of them at once or in very short intervals. In such a case, they would be exceeding their usual rate of 1 request per second.

Finding the right limit

Setting the right limit depends on many factors, so we can't give you a simple answer. We have some suggestions, though.

Finding the right limit

The first step is to understand the size of traffic that your instance receives. You can do this by parsing the access log and finding a user than made the most REST requests over a day. Since UI traffic is not rate limited, this number will be higher than what you need as your rate limit. Now, that's a base number — you need to modify it further based on the following questions:

1. Can you afford to interrupt your users' work? If your users' integrations are mission-critical, consider upgrading your hardware instead. The more critical the integrations, the higher the limit should be — consider multiplying the number you found by two or three.
2. Is your instance already experiencing problems due to the amount of REST traffic? If yes, then choose a limit that's close to the base number you found on a day when the instance didn't struggle. And if you're not experiencing significant problems, consider adding an extra 50% to the base number — this shouldn't interrupt your users and you still keep some capacity.

In general, the limit you choose should keep your instance safe, not control individual users. Rate limiting is more about protecting Bamboo from integrations and scripts going haywire, rather than stoping users from getting their work done.

Adding exemptions

Exemptions are, well, special limits for users who really need to make more requests than others. Any exemptions you choose will take precedence over global settings.

> After adding or editing an exemption, you'll see the changes right away, but it takes up to 1 minute to apply the new settings to a user.

To add an exemption:

1. Open the **Exemptions** tab.
2. Click **Add exemption**.
3. Find the user and choose their new settings.
    - You can't choose groups, but you can select multiple users.
    - The options available here are just the same as in global settings: Allow unlimited requests, Block all requests, or assign custom limit.
4. Click **Save**.

If you want to edit an exemption later, just click **Edit** next to a user's name in the **Exemptions** tab.

Recommended: Add an exemption for anonymous access

Bamboo sees all anonymous traffic as made by one user: **Anonymous**. If your rate limits are not too high, it might happen that a single user drains the limit assigned to anonymous. It's a good idea to add an exemption for this account with a higher limit, and then observe whether you need to increase it further.

Identifying users who have been rate limited

When a user is rate limited, they'll know immediately as they'll receive an HTTP 429 error message (too many requests). You can identify users that have been rate limited by opening the *List of limited accounts* tab on the rate limiting settings page. The list shows all users from the whole cluster.

When a user is rate limited, it takes up to 5 minutes to show it in the table.

Unusual accounts

You'll recognize the users shown on the list by their name. It might happen, though, that the list will show some unusual accounts, so here's what they mean:

- **Unknown:** That's a user that has been deleted in Bamboo. They shouldn't appear on the list for more than 24 hours (as they can't be rate limited anymore), but you might see them in the list of exemptions. Just delete any settings for them, they don't need rate limiting anymore.
- **Anonymous:** This entry gathers all requests that weren't made from an authenticated account. Since one user can easily use the limit for anonymous access, it might be a good idea to add an exemption for anonymous traffic and give it a higher limit.

Adding limited requests to the log file

You can also view information about rate limited users and requests in the Bamboo log file. This is useful if you want to get more details about the URLs that requests targeted or originated from.

To add limited requests to the log file:

1. Go to **Administration > System > Logging and profiling**.
2. Click **Configure logging level for another package**.
3. Set the package name to:

```
com.atlassian.ratelimiting.internal.requesthandler.logging
```

4. Set the logging level to *DEBUG*, and click **Add**.
5. Every rate limited requests will now appear in the Bamboo log file.

Getting rate limited — user's perspective

When users make **authenticated** requests, they'll see rate limiting headers in the response. These headers are added to every response, not just when you're rate limited.

Header	Description
`X-RateLimit-Limit`	The max. number of requests (tokens) you can ever have. New tokens won't be added to your bucket after reaching this limit. Your admin configures this as *Max requests*.
`X-RateLimit-Remaining`	The remaining number of tokens. This is what you have and can use right now.
`X-RateLimit-Interval-Seconds`	The time interval in seconds. You get a batch of new tokens every such time interval.
`X-RateLimit-FillRate`	The number of tokens you get every time interval. Your admin configures this as *Requests allowed*.
`retry-after`	How long you need to wait until you get new tokens.

When you're rate limited and your request doesn't go through, you'll see the HTTP 429 error message (too many requests). You can use these headers to adjust scripts and automations to your limits, making them send requests at a reasonable frequency.

Allowlisting URLs and external applications

Allowlisting URLs and resources

We've also added a way to allowlist whole URLs and resources on your Bamboo instance. This should be used as quick fix for something that gets rate limited, but shouldn't.

For example, a Marketplace app added some new API to Bamboo. The app itself is used from the UI, so it shouldn't be limited, but it might happen that Bamboo sees this traffic as external and applies the rate limit. In this case, you could disable the app or increase the rate limit, but this brings additional complications.

To work around issues like that, you can allowlist the whole resource added by the app so it works without any limits.

To add URLs the allowlist, configure `-Dcom.atlassian.ratelimiting.whitelisted-url-patterns` as a system property with the URLs in a comma-separated list. For example:

```
-Dcom.atlassian.ratelimiting.whitelisted-url-patterns=/**/rest/applinks/**,/**/rest/capabilities,/**/rest/someapi
```

For more info on setting system properties, see Configuring your system properties.

For more info on how to create URL patterns, see AntPathMatcher: URL patterns.

Allowlisting external applications

You can also allowlist consumer keys, which lets you remove rate limits for external applications integrated through AppLinks.

Find your app's consumer key

To find your app's consumer key:

1. Go to **> Overview**.
2. In the left menu, scroll down to **Manage apps** and select **Application links**.
3. Open **Incoming authentication** and copy the consumer key.

Add the consumer key to the allowlist

To add consumer keys to the allowlist, configure `-Dcom.atlassian.ratelimiting.whitelisted-oauth-consumers` as a system property with the consumer keys in a comma-separated list.

After entering the consumer key, the traffic coming from the related application will no longer be limited.

Adjusting your code for rate limiting

We've created a set of strategies you can apply in your code (scripts, integrations, apps) so it works with rate limits, whatever they are. For more info, see Adjusting your code for rate limiting.

Adjusting your code for rate limiting

Whether it's a script, integration, or app you're using — if it's making external REST API requests, it will be affected by rate limiting. Until now, you could send an unlimited number of REST API requests to retrieve data from Bamboo, so we're guessing you haven't put any restrictions on your code. When admins enable rate limiting in Bamboo, there's a chance your requests will get limited eventually, so we want to help you prepare for that.

Before you begin

To better understand the strategies we've described here, it's good to have some some basic knowledge about rate limiting in Bamboo. When in doubt, head to Improving instance stability with rate limiting and have a look at the first paragraph.

Quick reference

- **Success:** When your request is successful, you'll get a `2xx` code.
- **Error:** When your request fails, you'll get a `4xx` code. If you're rate limited, it will be `429` (too many requests).

The following HTTP headers are added to every **authenticated** request affected by rate limiting:

Header	Description
`X-RateLimit-Limit`	The max number of requests (tokens) you can have. New tokens won't be added to your bucket after reaching this limit.
`X-RateLimit-Remaining`	The remaining number of tokens. This value is as accurate as it can be at the time of making a request, but it might not always be correct.
`X-RateLimit-Interval-Seconds`	The time interval in seconds. You get a batch of new tokens every time interval.
`X-RateLimit-FillRate`	The number of tokens you get every time interval.
`retry-after`	How long you need to wait until you get new tokens. If you still have tokens left, it shows 0; this means you can make more requests right away.

Strategies

We've created a set of strategies you can apply to your code so that it works with rate limits. From very specific to more universal, these reference strategies will give you a base, which you can further refine to make an implementation that works best for you.

1. Exponential backoff

This strategy is the most universal and the least complex to implement. It's not expecting HTTP headers or any information specific to a rate limiting system, so the same code will work for the whole Atlassian suite, and most likely non-Atlassian products, too. The essence of using it is observing whether you're already limited (wait and retry, until requests go through again) or not (just keep sending requests until you're limited).

⊕ Universal, works with any rate limiting system.

⊕ Doesn't require too much knowledge about limits or a rate limiting system.

⊖ High impact on a Bamboo instance because of concurrency. We're assuming most active users will send requests whenever they're available. This window will be similar for all users, making spikes in Bamboo performance. The same applies to threads — most will either be busy at the same time or idle.

⊖ Unpredictable. If you need to make a few critical requests, you can't be sure all of them will be successful.

Summary of this strategy

Here's the high-level overview of how to adjust your code:

1. **Active:** Make requests until you encounter a `429`. Keep concurrency to a minimum to know exactly when you reached your rate limit.
2. **Timeout:** After you received a `429`, start the timeout. Set it to 1 second for starters. It's a good idea to wait longer than your chosen timeout — up to 50%.
3. **Retry:** After the timeout has passed, make requests again:
    a. Success: If you get a `2xx` message, go back to *step 1* and make more requests.
    b. Limited: If you get a `429` message, go back to *step 2* and double the initial timeout. You can stop once you reach a certain threshold, like 20 minutes, if that's enough to make your requests work.

With this strategy, you'll deplete tokens as quickly as possible, and then make subsequent requests to actively monitor the rate limiting status on the server side. It guarantees you'll get a `429` if your rate is above the limits.

2. Specific timed backoff

This strategy is a bit more specific, as it's using the `retry-after` header. We're considering this header an industry standard and plan to use it across the Atlassian suite, so you can still be sure the same code will work for Bitbucket and Bamboo, Server and Data Center, etc. This strategy makes sure that you will not be limited, because you'll know exactly how long you need to wait before you're allowed to make new requests.

⊕ Universal, works with any rate limiting system within the Atlassian suite (and other products using `retry-after`) — Bitbucket and Bamboo, Server and Data Center, etc.

⊕ Doesn't require too much knowledge about limits or a rate limiting system.

⊖ High impact on a Bamboo instance because of concurrency. We're assuming most active users will send requests whenever they're available. This window will be similar for all users, making spikes in Bamboo performance. The same applies to threads — most will either be busy at the same time or idle.

Summary of this strategy

Here's the high-level overview of how to adjust your code:

1. **Active:** Make requests and observe the `retry-after` response header, which shows the number of seconds you need to wait to get new tokens. Keep concurrency level to minimum to know exactly when the rate limit kicks in.
    a. Success: If the header says 0, you can make more requests right away.
    b. Limited: If the header has a number greater than 0, for example 5, you need to wait that number of seconds.
2. **Timeout:** If the header is anything above 0, start the timeout with the number of seconds specified in the header. Consider increasing the timeout by a random fraction, up to 20%.
3. **Retry:** After the timeout specified in the header has passed, go back to *step 1* and make more requests.

With this strategy, you'll deplete tokens as quickly as possible, and then pause until you get new tokens. You should never hit a 429 if your code is the only agent depleting tokens and sends requests synchronously.

3. Rate adjustment

This strategy is very specific and expects particular response headers, so it's most likely to work for Bamboo Data Center only. When making requests, you'll observe headers returned by the server (number of tokens, fill rate, time interval) and adjust your code specifically to the number of tokens you have and can use.

⊕ It can have the least performance impact on a Bamboo instance, if used optimally.

⊕ Highly recommended, especially for integrations that require high-volume traffic.

➕ Safe, as you can easily predict that all requests that must go through will in fact go through. It also allows for a great deal of customization.

➖ Very specific, depends on specific headers and rate limiting system.

Summary of this strategy

Here's the high-level overview of how to adjust your code:

1. **Active:** Make requests and observe all response headers.
2. **Adjust:** With every request, recalculate the rate based on the following headers:
    - `x-ratelimit-interval-seconds`: The time interval in seconds. You get a batch of new tokens every time interval.
    - `x-ratelimit-fillrate`: The number of tokens you get every time interval.
    - `retry-after`: The number of seconds you need to wait for new tokens. Make sure that your rate assumes waiting longer than this value.
3. **Retry:** If you encounter a 429, which shouldn't happen if you used the headers correctly, you need to further adjust your code so it doesn't happen again. You can use the `retry-after` header to make sure that you only make requests when the tokens are available.

Customizing your code

Depending on your needs, this strategy helps you to:

By following the headers, you should know how many tokens you have, when you will get the new ones, and in what number. The most useful headers here are `x-ratelimit-interval-seconds` and `x-ratelimit-fillrate`, which show the number of tokens available every time interval. They help you choose the perfect frequency of making your requests.
You can wait to perform complex operations until you're sure you have enough tokens to make all the consecutive requests you need to make. This allows you to reduce the risk of leaving the system in an inconsistent state, for example when your task requires 4 requests, but it turns out you can only make 2. The most useful headers are `x-ratelimit-remaining` and `x-ratelimit-interval-seconds`, which show how many tokens you have right now and how long you need to wait for the new ones.
With all the information returned by the headers, you can create more strategies that work best for you, or mix the ones we've described here. For example:

- If you're making requests once a day, you can focus on the max requests you can accumulate (`x-ratelimit-limit`), or lean towards the remaining number of tokens if a particular action in Bamboo triggers your app to make requests (`x-ratelimit-remaining`).
- If your script needs to work both for Bamboo Data Center and some other application, use all headers for Bamboo and focus on the universal `retry-after` or request codes if the app detects different software.

Build resiliency in Bamboo Data Center

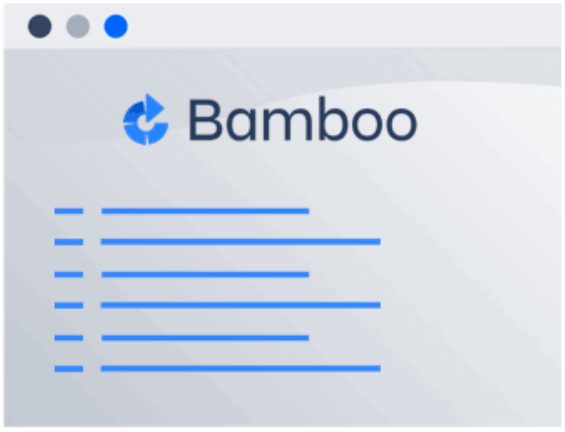
BAMBOO DATA CENTER

In Bamboo versions earlier than 8.0, when the server's work got interrupted or if a server went down for more than 5 minutes, Bamboo builds would fail due to lack of connection of the building agent with the server. Bamboo agents were designed to die when they couldn't connect to a server for longer than 5 minutes.

With Bamboo Data Center, the agent will continue its work and finish building even if the connection with the server is lost. Once the agent's building work is done, it tries to connect to the server. If the server is already online, the agent will send build results, logs, and artifact to the server, and pick up the next tasks from the server. If the server is still down, the agent will try to reconnect with the server after some time.

If agent is started with the agent wrapper, by default, the agent tries to connect to the server 1440 times, or until it's successful. You can change this value by going to `$BAMBOO_AGENT_HOME/conf/wrapper.conf` and modifying the `wrapper.max_failed_invocations` value.

If agent it not started with the agent wrapper, it will try to transmit results 10 times with 5-minute intervals, and then terminate if not successful. However, if manually restarted, the agent will go into the 'retry' loop again provided the result is not removed from the disk.



If the transmission problems are caused by the network failure, the effective timeout is considerably shorter as in such case the server recognizes that the agent is offline and terminates the build on its end. This behaviour is configured by *heartbeat timeouts*. For more information, see Changing the remote agent heartbeat interval.

It is important to understand that this improved build resiliency to server failures will work only if the build process can be finished. Bamboo will not be able to finish the build if:

- a child process is failing or stopped
- an agents process is stopped while the build is running
- a resource required for build process is unavailable (this includes resources provided by the Bamboo server, like REST endpoints and artifacts from other builds)
- a build is failing because of intermittent infrastructure problems

Build resiliency with elastic agents

Same logic applies to agents started at EC2 environment. To achieve it Bamboo agent is started using the Tanuki wrapper, which is also used by the remote agent. The wrapper allows to restart Bamboo agent when Java process is interrupted by connection timeout error.

If you’re using elastic images provided by Bamboo 8.0 (or based on them), elastic agents use the agent wrapper and can fully benefit from improved build resiliency. Old images are still functional but will work with the ‘short’ timeout only.

After server restart, elastic agents that use the agent wrapper are able to fully resume their operation. Agents without wrapper are allowed to return the result they worked on but then they will terminate.

Disabling elastic tunnel is no longer prerequisite for seamless restarts/improved build resiliency.

Bamboo DC local agents

So far Bamboo had two types of agents: **local agents** and **remote agents**. Remote agents had three deployment strategies:

- remote agents on the local machine
- remote agents on the machine other than Bamboo server
- remote agents on AWS also known as elastic agents

⚠ Local agents are no longer supported in Bamboo Data Center. Because of that we decided to standardize the naming and from now on we will refer to remote agents as simply **agents**.

Remote but local?

Keep in mind that you still can run agents on your local machine.

The suggestion for users of Local Agents is to install agents into the local machine. Be aware that this solution is recommended only for single-node instances. If you are running multiple nodes, you are not expected to have agents running in the same machine, as the agents will be fallible to the same issues that will affect a node. Furthermore, for agents to work with the resilience features of Bamboo Data Center, they need to be able to connect to multiple nodes. With multiple nodes, there are no clear guarantees which node will be working as the active and which nodes will be in the stand by mode. Agents need to be connected through a load balancer in order to they keep normal functioning even during a node failure.

Migrating Bamboo 7.X.X to Bamboo 8.X.X

Overview

The recommended paths for upgrading Bamboo to a new version differ depending on whether you want to move to a new server or not:

Upgrading Bamboo locally	Upgrading Bamboo with a move to a new server
Perform the steps as described on this page. Make sure that your new Bamboo instance is not installed in the same directory as the original Bamboo instance.	1. Clone your Bamboo instance into the new location. See Cloning a Bamboo instance. 2. Follow the upgrade steps on the cloned Bamboo instance as described below on this page. The cloned instance on the new server is referred to as original Bamboo instance.

In both scenarios, the new Bamboo instance uses the home directory and the database of the original Bamboo instance.

We recommend that you test the Bamboo upgrade on a QA server before deploying to production.

If you are a Bamboo plugin developer, see our Bamboo API Changes by Version guide, which outlines changes in Bamboo that may affect Bamboo plugins compiled for earlier versions of Bamboo.

Before you begin

- Read upgrades notes specific to your version of Bamboo. See Bamboo upgrade notes.
- Read End of support announcements for Bamboo.
- Check whether the system where you are going to install the new Bamboo instance meets Bamboo platform requirements. See Supported platforms.
- Only import data to an instance running the same version.

The installation path is referred to as `<bamboo-install>` and points to the directory into which you extracted the Bamboo package. It is different from `<bamboo-home>`, which points to the directory where Bamboo data is stored.

1. Export and back up the existing Bamboo data

a. Export the Bamboo database

There are two database backup scenarios, depending on whether you are using an embedded or external database.

Embedded HSQL database	External database
Create an export ZIP file for the original Bamboo instance. For more information, see Exporting data for backup. The export may take a long time to complete and may require a large amount of disk space, depending on the number of builds and tests in your system. HSQL is not recommended for production Bamboo instances.	Use native database tools to create a backup. For more information about external databases, see Connecting Bamboo to an external database.

b. Upgrade License in Bamboo (only when upgrading to DATA CENTER)

1. Go to **Administration > License Details**.
2. Enter your Bamboo Data Center license key.

c. Stop Bamboo

Stop the original Bamboo instance.

If you have Bamboo running as a Windows service, uninstall the service by using the `UninstallService.bat` executable that came with your Bamboo instance.

d. Back up the Bamboo configuration

When the original Bamboo instance is shut down, back up your `<bamboo-home>` directory, which contains the `builds` and `configuration` directories. You can compress it into a ZIP file.

2. Download and install a new Bamboo instance

To upgrade Bamboo, you must install a new Bamboo instance in a `<bamboo-install>` directory that is different from the `<bamboo-install>` directory of the original Bamboo instance .

This upgrade scenario uses the home directory and the external database of the original Bamboo instance.

- Make sure that the original Bamboo instance is not running before you start the new installation.
- To prevent data loss during updates or reinstallation, the `<bamboo-home>` directory must be different from the `<bamboo-install>` directory.

Follow these guidelines to install a new Bamboo instance:

MacOS

1. The Mac installer deletes the previous version of Bamboo.
2. Follow the Mac OS X install instructions.

Linux

1. Delete your old `<bamboo-install>` directory to remove any legacy files.
2. Follow the Linux install instructions.

Windows

1. The Windows installer deletes the previous version of Bamboo.
2. Follow the Windows install instructions.
3. Configure Bamboo to run as a service on Windows, using the `service.bat` executable.

3. Configure the new Bamboo instance

a. Set the home directory for the new Bamboo instance

Set the `<home-directory>` to use the `<home-directory>` of the original Bamboo instance:

1. Go to the new Bamboo instance `<bamboo-install>` directory. It is the directory where you installed Bamboo.
2. Open `atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`

   NOTE: For Bamboo 5.1 and earlier, the file path is:
   `<bamboo-install>/webapp/WEB-INF/classes/bamboo-init.properties`
3. Set the `bamboo.home` variable to use the `<bamboo-home>` path of the original Bamboo instance.

b. Update any installed apps

If you installed any apps in addition to the pre-installed system apps:

- Check if all apps are compatible with the new version of Bamboo.
- Update any apps that are out-of-date.
- Disable any apps that are incompatible with the new version of Bamboo.

c. Automatic update of remote agents

For Bamboo 3.2 and later, remote agents are updated automatically. Remote agents automatically detect when a new version is available and downloads new classes from the server. For more information, see Bamboo remote agent installation guide.

> If you're upgrading Bamboo **from a version earlier than 8.0 to 8.0 or later**, and you are also upgrading the **Java version** on your Remote Agents **from 8 to 11**, you need to either update wrapper configuration manually or download a new Remote Agent JAR from your upgraded Bamboo and reinstall the wrapper.

d. Migrate your existing Bamboo configurations over to your new Bamboo installation

If you have modified properties in configuration files of your existing Bamboo installation, make the same modifications in your new Bamboo installation. However, because the properties in the configuration files may have changed between versions, you cannot simply copy the configuration files from your existing installation and replace the equivalent files in the new installation.

For each file you have modified in your existing Bamboo installation, you need to **manually edit each equivalent file in your new Bamboo installation and re-apply your modifications**.

The table below lists the most commonly modified files and their locations within your Bamboo Installation Directory:

File	Location in Bamboo installation	Description
setenv.bat (Windows) or setenv.sh (Linux)	`bin`	Configuring your system properties
seraph-config.xml	`atlassian-bamboo/WEB-INF/classes`	Modified if you had integrated Bamboo with Crowd
server.xml	conf	Modified in the following situations: • If you had previously changed Bamboo's root context path. • If you had previously secured Bamboo with Tomcat using SSL or proxy server.

e. Check database access permission

Before you start the new Bamboo instance, make sure that it has the *write* access to the database, which is required to complete the upgrade tasks.

After that process the instance is now in 8.X.X version, if you don’t want Bamboo in cluster then proceed to “5. Start Bamboo”.

Bamboo configured in that way will have all Data Center features when you use Data Center licence in **Administration License Details**

4. Install Bamboo Data Center in a cluster

See Install Bamboo Data Center

5. Start Bamboo

Start Bamboo

Once you have installed Bamboo and set the bamboo.home property, start the new Bamboo instance. The upgrade runs automatically.

You can check whether the upgrade was successful in the `atlassian-bamboo.log` file.

Upgrading Bamboo may require reindexing.

Depending on the number of existing builds and tests, the reindexing process may take a significant amount of time, during which Bamboo will not be available.

Version-specific upgrade notes

The version-specific notes provide additional information to the main upgrade documentation. We recommend reading the version-specific notes for the original and new Bamboo instance versions. See Version-specific upgrade notes

Troubleshooting

If you followed the documentation and you still have problems with the upgrade process:

- Check the How to Upgrade/Migrate Bamboo article in the Bamboo Knowledge Base.
- Check other Knowledge Base articles.
- Ask questions at https://answers.atlassian.com.
- You can also create a support ticket. To help us address the issue, attach the `atlassian-bamboo.log` file to the ticket.

Elastic agent supervisor

Elastic agents use the agent supervisor for monitoring the agent process. If an elastic agent crashes, the supervisor will automatically attempt to restart it, and the agent will try to reconnect to a Bamboo node.

The elastic agent supervisor is an implementation of the Java Service Wrapper, the same as the remote agent supervisor. For more information about the supervisor, see remote agent supervisor.

All Java Service Wrapper files are downloaded together with the elastic agent bootstrap files from an S3 bucket provided by Atlassian during an elastic instance startup. The elastic agent installer automatically installs the Java Service Wrapper on the elastic instance.

> If you use a custom elastic image that was created before Bamboo 8.0, you must recreate it with the new version of the `atlassian-bamboo-agent-elastic-installer.jar` on it in order to use the Elastic agent supervisor.

Configuring wrapper properties

The elastic agent process is executed by the Java Service Wrapper installed on the elastic instance and is controlled by various settings defined in the `wrapper.conf` file. The file is created by the elastic agent startup script.

Wrapper configuration properties can't be modified once the agent is running. However, you can override them by passing wrapper properties to the elastic agent Installer. You can override any of the Java Service Wrapper properties.

Elastic agent startup scripts provided in the atlassian-bamboo-elastic-image use a dedicated `WRAPPER_PROPERTIES` variable that can carry one or multiple properties and is passed to the elastic agent startup command as follows:

Unix, Linux

```
java -Dbamboo.agent.syncOnly ${WRAPPER_PROPERTIES:-} -jar $bambooAgentBin/*installer*.jar
```

Windows

```
"%JAVA_HOME%/bin/java" %WRAPPER_PROPERTIES% -jar C:\opt\bamboo-elastic-agent\bin\atlassian-bamboo-agent-elastic-installer.jar
```

You can set the `WRAPPER_PROPERTIES` variable in an elastic image startup script. For example, to increase the initial and maximum Java heap size, use the following scripts:

Unix, Linux

```
sudo su -c "echo \"export WRAPPER_PROPERTIES='\"-Dwrapper.java.maxmemory=2048\" \"-Dwrapper.java.initmemory=512\"'\" >> /etc/profile.d/bamboo.sh"
```

Windows

```
powershell.exe -Command "[System.Environment]::SetEnvironmentVariable(\"WRAPPER_PROPERTIES\", '\"-Dwrapper.java.maxmemory=2048\" \"-Dwrapper.java.initmemory=512\"', 'Machine')"
```

Values containing spaces can cause problems and the best way to avoid them is to use a `\" key=value\"` syntax (in quotes escaped with backslashes). For example, to set the maximum amount of memory for the JVM to 2048 MB, use:

```
\"-Dwrapper.java.maxmemory=2048\"
```

Running Bamboo Data Center on a Kubernetes cluster

If you're running self-managed environments and looking to adopt modern infrastructure, you can deploy your Atlassian Data Center products on Kubernetes clusters. By leveraging Kubernetes, you can drive greater agility amongst your teams and enjoy a simplified administrative experience at scale without compromising your organization's regulatory requirements.

What is Kubernetes?

Kubernetes (K8s) is a high-availability rapid deployment and container orchestration framework that allows you to easily manage and automate your deployments in one place. Because it makes heavy use of containers, your applications stay up-to-date with no downtime and always have safe and reliable access to the dependencies they require. Learn more on the official Kubernetes website

How does it work?

Kubernetes automates the management of containerized applications. It provides a centralized control plane to manage containers and the underlying infrastructure, automate scaling, rollouts and rollbacks, and more. The platform abstracts away the underlying infrastructure and provides a unified way of managing containers and applications, making it easier for developers to build, deploy, and run applications at scale.

Why Kubernetes?

Kubernetes is a powerful platform that comes with a number of benefits, including:

- Improved agility
- Simplified administration
- Deployment automation
- Automated operations for containers
- Security enhancements
- Accelerated upgrades and rollbacks
- Better scalability and resiliency

On top of that, the ability to manage your infrastructure as code by using simple YAML files helps you reduce unnecessary resource consumption.

How does Atlassian integrate with Kubernetes?

Manage Kubernetes with Helm charts

To help you deploy our products, we've created Data Center Helm charts—customizable templates that can be configured to meet the unique needs of your business. You can even choose how to run them: either on your own hardware or on a cloud provider's infrastructure. This allows you to stay in control of your data and meet your compliance needs while still using a more modern infrastructure. Helm charts have their own lifecycle, so updates contain certain features and are upgraded automatically.

Helm charts provide the essential building blocks needed to deploy Atlassian Data Center products (Jira, Confluence, Bitbucket, Bamboo, and Crowd) in Kubernetes clusters and give you the capability to integrate with your operation and automation tools. Learn more about Helm charts

Use Docker images for improved agility

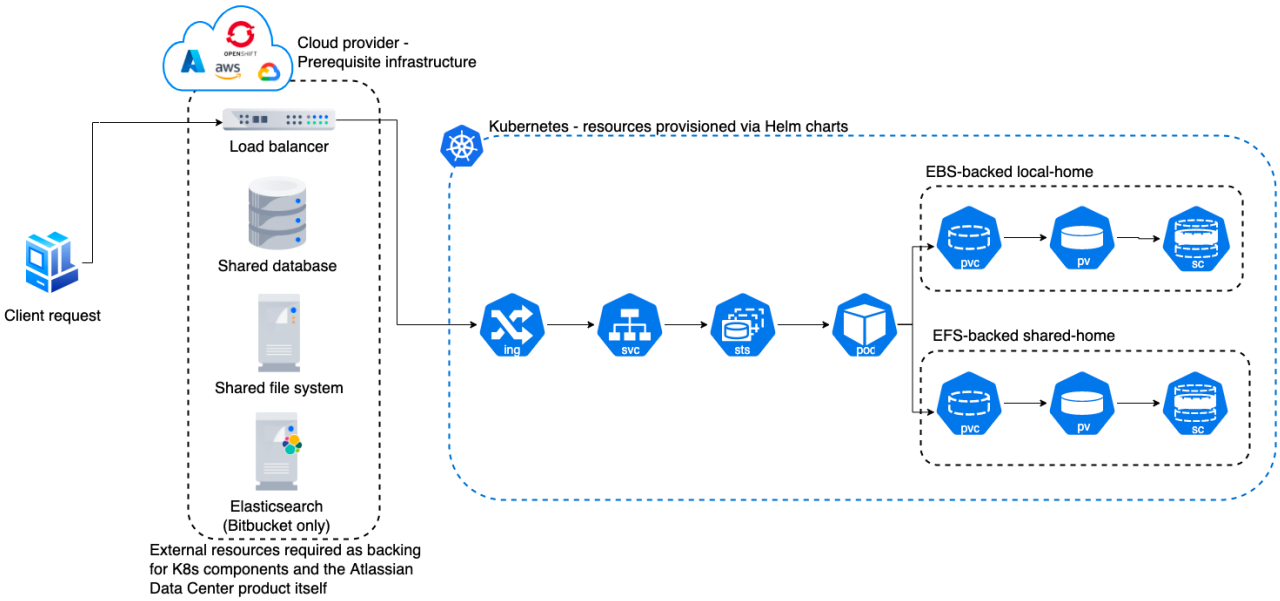
To speed up development, you can take advantage of Data Center's hardened Docker container images. Using our Docker container images as part of your Data Center deployment allows you to cut significant time by streamlining and automating workflows.

After defining your required configuration once, you can instantly deploy exact replicas of your environment from the command line at every stage of your deployment lifecycle, giving you the agility needed to keep valuable work moving forward, and the flexibility to accommodate your organization's evolving development strategy over time.

Learn Kubernetes deployment architecture

The Kubernetes cluster can be a managed environment, such as Amazon EKS, Azure Kubernetes Service, Google Kubernetes Engine, or a custom on-premise system. We strongly recommend you set up user management, central logging storage, a backup strategy, and monitoring just as you would for a Data Center installation running on your own hardware.

Here's an architectural overview of what you'll get when deploying your Data Center application on a Kubernetes cluster using the Helm charts:



The following Kubernetes entities are required for product deployment:

- **Ingress and Ingress controller** (ing)—the Ingress defines the rules for traffic routing, which indicate where a request will go in the Kubernetes cluster. The Ingress controller is the component responsible for fulfilling those rules.
- **Service** (svc)—provides a single address for a set of pods to enable load-balancing between application nodes.
- **Pod**—a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
- **StatefulSets** (sts)—manages the deployment and scaling of a set of pods requiring persistent state.
- **PersistentVolume** (pv)—a "physical" volume on the host machine that stores your persistent data.
- **PersistentVolumeClaim** (pvc)—reserves the Persistent Volume (PV) to be used by a pod or potentially multiple pods.
- **StorageClass** (sc)—provides a way for administrators to describe the "classes" of storage they offer.

Install your Data Center application on a Kubernetes cluster

To install and operate your Data Center application on a Kubernetes cluster using our Helm charts:

1. Follow the requirements and set up your environment according to the Prerequisites guide.
2. Perform the installation steps described in the Installation guide.
3. Learn how to upgrade applications, scale your cluster, and update resources using the Operation guide.

Upgrading Bamboo Data Center

This guide applies to Bamboo Data Center running in a cluster. Upgrading Bamboo to the next version will incur a downtime.

> If you're using Bamboo Data Center on a single node, you can follow the upgrade process for Bamboo Server.

Plan your upgrade

1. Check upgrade requirements

Make sure your environment meets the minimum requirements to run the desired version of Bamboo Data Center. To ensure your instance works correctly after the upgrade, see the latest End of support announcements

You can update any version of Bamboo Data Center to the latest version, as there is no required upgrade path.

Learn more about migrating to Bamboo Data Center

If you're planning to migrate from Bamboo Server to Bamboo Data Center, remember to:

- Check the infrastructure and hardware requirements described in Clustering with Bamboo Data Center.
- Get familiar with Migrating Bamboo 7.X.X to Bamboo 8.X.X and read about Bamboo home migration.

> You can upgrade to Bamboo Server 8 and newer without migrating to Bamboo Data Center 8.

2. Complete pre-upgrade checks

1. See the Bamboo release notes and Version specific upgrade notes for the version you plan to upgrade to (and any in between).
2. Go to **Administration** > **Troubleshooting and support tools**, then review the Log analyzer for any issues that may need to be resolved.
3. Check the compatibility of your apps with the version of Bamboo Data Center you plan to upgrade to.
    a. Go to **Administration** > **Manage apps** > **Bamboo update check**.
    b. Select the version you plan to upgrade to, then select **Check**.

    If your users rely on particular apps, you may want to wait until the apps are compatible with the desired version of Bamboo Data Center.
4. Make sure that all important builds are finished before the upgrade. A new Bamboo Data Center version might not be able to process old build results or complete the old builds successfully.

3. Upgrade in a test environment

Before you upgrade Bamboo Data Center to the next major or minor version, perform the upgrade in a test environment first:

1. Create a staging copy of your current production environment.
2. Follow the upgrade steps below.
3. If possible, test any unsupported apps, customizations, and proxy configuration.

Upgrade Bamboo Data Center

> The installation path is referred to as `<bamboo-install>` and points to the directory into which you extracted the Bamboo Data Center package. It's different from `<bamboo-home>` and `<bamboo-shared-home>`, which both point to the directories where data is stored.

1. Stop the cluster

Before you upgrade, you need to stop all the nodes in the cluster. Start with all your non-primary nodes, so they won't try to take over after you stop the primary node.

If you run Bamboo as a Windows service, uninstall the service by using the `UninstallService.bat` executable from Bamboo binaries.

Configure your load balancer to redirect the traffic away from Bamboo until the upgrade is complete on all nodes. Your router settings shouldn't allow agents and user traffic until all nodes are updated and confirmed operational.

2. Back up your data

Export your database

Use native database tools to create a backup. For more information about external databases, see Connect Bamboo to an external database.

Back up your configuration

Create a backup for <bamboo-home> directories from each node as well as the `<bamboo-shared-home>` directory. At the bare minimum, you should back up:

- `bamboo.cfg.xml` and `cluster-node.properties` from `<bamboo-home>` on each node.
- all content of the `configuration` directory from `<bamboo-shared-home>`.

3. Download Bamboo Data Center

Download the appropriate file for your operating system from our software download page.

4. Upgrade the first node

A. Install a new instance

To upgrade, you need to install a new Bamboo Data Center instance in a `<bamboo-install>` directory that's different from the original instance's installation directory.

- Before you start the new installation, make sure the original instance isn't running.
- To prevent data loss during updates or reinstallation, the `<bamboo-install>` directory must be different from the `<bamboo-home>` and `<bamboo-shared-home>` directories.

B. Configure the new instance

Set home directories

Set the `<bamboo-home>` and `<bamboo-shared-home>` to use the same paths as the original Bamboo instance:

1. Go to the new instance's `<bamboo-install>` directory
2. Open `atlassian-bamboo/WEB-INF/classes/bamboo-init.properties`
3. Set the `bamboo.home` variable to use the `<bamboo-home>` path of the original instance
4. Set the `bamboo.shared.home` variable to use the `<bamboo-shared-home>` path of the original instance

Install a JDBC driver

If the JDBC driver for your database isn't bundled with Bamboo, you need to install it for the new Bamboo instance yourself. See Connect Bamboo to an external database for detailed instructions.

Migrate the existing configurations to your new Bamboo installation

If you've modified any properties in the configuration files of your existing Bamboo installation, make the same modifications in your new installation. Note that you can't simply copy the configuration files from the existing installation and replace the equivalent files in your new installation, because the properties in the configuration files may have changed between versions.

For each file you've modified in your existing Bamboo installation, you need to **manually edit each equivalent file in your new Bamboo installation and re-apply your modifications**.

The table below lists the most commonly modified files and their locations within your Bamboo installation directory.

File	Location in Bamboo installation	Description
setenv.bat (Windows) or setenv.sh (Linux)	`bin`	Configuring your system properties
seraph-config.xml	`atlassian-bamboo/WEB-INF/classes`	Modified if you had integrated Bamboo with Crowd
server.xml	`conf`	Modified in the following situations: • If you had previously changed Bamboo's root context path. • If you had previously secured Bamboo with Tomcat using SSL or proxy server.

Check database access permission

Before you start the new Bamboo instance, make sure that it has the write access to the database, which is required to complete the upgrade tasks.

C. Start Bamboo

Start the new Bamboo instance and wait for the upgrade to finish. You can check whether the upgrade was successful in the `atlassian-bamboo.log` file.

Don’t reconfigure your load balancer yet. Traffic to Bamboo should be restored only after all nodes are updated and confirmed operational.

Upgrading Bamboo may require reindexing. Depending on the number of existing builds and tests, the reindexing process may take a significant amount of time, during which Bamboo won't be available.

D. Update any installed apps

If you installed any apps in addition to the pre-installed system apps:

- Check if all apps are compatible with the new version of Bamboo.
- Update any apps that are out of date.
- Disable any apps that are incompatible with the new version of Bamboo.

5. Install Bamboo on remaining nodes

The next step is replicating your upgraded Bamboo directories to other nodes in the cluster.

1. Stop Bamboo on the first node.
2. Install the new Bamboo version on the next node the same way you did on the first node.
    a. Make sure to use <bamboo-home> and <bamboo-shared-home> specific to the node.
    b. You don’t need to update installed apps again.
    c. You can copy any modified configuration files from the first node.
3. Start Bamboo and confirm that everything works as expected.

4. Stop Bamboo on this node before continuing with the next node.

Repeat this process for each remaining node.

6. Start Bamboo

Once all nodes have been upgraded, you can start Bamboo Data Center on each node, one at a time. Then, restore your load balancer configuration to direct agents and user traffic to Bamboo.

Automatic update of remote agents

Most of the time, remote agents are updated automatically. Remote agents detect when a new version is available and download new classes from the server.

For more information, see Bamboo remote agent installation guide.

Troubleshooting

If you followed the documentation and still experience problems with the upgrade process:

- Check the How to upgrade/migrate Bamboo article in the Bamboo Knowledge Base.
- Check other Knowledge Base articles.
- Ask questions at https://answers.atlassian.com.
- Create a support ticket. To help us address the issue, attach the `atlassian-bamboo.log` file to the ticket.