Dockerizing a Larger PostgreSQL Installation
What could possibly go wrong?

Mladen Marinović
mladen.marinovic@atos.net
@mladenmarinovic
About me

- Working at Atos for the last 6 months
  - Payment authorization
  - Integration with complex system
  - Mostly java and PHP
  - Part time PG admin
- Python projects for relaxing
- Currently working with Docker and Ansible to automate system installations, updates and monitoring.

mladen.marinovic@atos.net
@mladenmarinovic
Disclaimer

► The examples presented are simplified to demonstrate the usage of Docker and/or Ansible
► The design is the one that worked for us (as of today cca 660GiB of fleet management data), it might not work/be compatible with your requirements
# Index

- The problem
- Why migrate to Docker?
- Design choices
- Storage
- Building the images – with problems
- Loading data
- Data replication
- Backups – with problems
- Monitoring running instances
- Operational problems
- Missing functionalities
The problem

► The old system with PG 9.1 needed to be updated
  – Old database software (9.1 09/2011, EOL 09/2016)
  – Old hardware (Slow HDD/RAID, low on RAM)
  – Missing features
    • Replication - failover
    • Continuous backup with retention policies
    • PITR

► New and better hardware for serving an increasing amount of clients
  – Implement all the missing features
  – And more 😊
Containers are like light virtual machines.
https://www.docker.com/what-docker
Docker (2)

- Image – a template for creating containers
- Container – a running instance of an image
- The images can be created by hand or with Dockerfiles
  - The Dockerfile is a file that describes the steps for creating an image
  - It is possible to use images from Docker Hub
FROM – Base image to start from
MAINTAINER – author/maintainer of this file
RUN – step to run
VOLUME – exposed volumes
EXPOSE – exposed ports
CMD – the main process to run
Why migrate to Docker?

► Hype... No

► Repeatability
  – We don’t want surprises on production servers
  – No surprises after OS updates / patches.
  – Dev/Test/QA/Production – all the same

► Problems
  – State – containers are usually volatile, database are usually the opposite
  – Maturity - is the technology mature enough (1.6 when this was implemented, new projects now in 1.8)

This is the Docker logo. The whale is called Moby Dock.
Design choices

➤ How many containers? 1, 2...
➤ Rule of thumb: one task -> one container
➤ So we have one container for the server process

➤ We will add additional containers later
Containers inside servers have open ports. We can map those ports to the server ports so that they can be accessible from the outside. Docker networking is a complicated topic, and you can make containers on separate machines communicate using Ipv6, but it seemed like an overkill in our situation.
Storage

▸ Where would the data be?
  – Containers are volatile
  – Data only containers? A nice concept
  – But we have SSDs and HDDs

▸ Containers can use volumes from other containers

http://container42.com/2014/11/18/data-only-container-madness/
In a data only container, the data is in the container. In out case, the data is physically located outside the data container.
The external volumes are mapped by default as root:root inside the container. If the file is created in a container (with some user that exists only in the container) the file is visible from the outside as messagebus:messagebus or something similar.

An example for the init script:

```bash
#!/bin/sh

chmod 775 /var/run/postgresql
chown postgres:postgres /var/run/postgresql

chmod 755 /etc/postgresql
chown root:root /etc/postgresql

chmod 755 /var/lib/postgresql
chown postgres:postgres /var/lib/postgresql

chmod 1775 /var/log/postgresql
chown root:postgres /var/log/postgresql
```
Docker calculates hasher for every line of the Dockerfile. If the sequence of lines is similar to some sequence that it has cached, it will use the cached version instead of reissuing the command. If you add a comment, it changes the line hash so Docker is forced to run the command.
Dockerfiles (2)

Tip 1 - error example

Step 2: RUN DEBIAN_FRONTEND=noninteractive apt-get update

--- Using cache
--- 946f6a390fa7

Step 3: RUN DEBIAN_FRONTEND=noninteractive apt-get install -y postgresql-9.4-9.4.5-0+deb8u1

--- Running in a436cefa4472
Reading package lists...
Building dependency tree...
E: Version '9.4.5-0+deb8u1' for 'postgresql-9.4' was not found
Dockerfiles (3)

marin@server:$ docker images --tree
Warning: '--tree' is deprecated, it will be removed soon. See usage.

- e9ebf606e14c Virtual Size: 188.1 MB
  - a02efed98f9f Virtual Size: 188.3 MB
    - 37eabe49e4e1 Virtual Size: 188.3 MB
      - 07f6e0c8e560 Virtual Size: 188.3 MB Tags: ubuntu:14.04.2, ubuntu:trustly
      - 53f49c28f032 Virtual Size: 188.3 MB
  - 02f7f7bdf0c4 Virtual Size: 209.3 MB
    - 233e567e29 Virtual Size: 212.1 MB
      - 2c6c60ffedf76 Virtual Size: 213.7 MB
    - 61d4b2e9046 Virtual Size: 213.7 MB
      - 3119673f Virtual Size: 255.2 MB
        - 4611a5a5a25 Virtual Size: 256 MB ---+ RUN apt-get update # 2015-06-19
          - 1c27f6f9d580 Virtual Size: 277 MB
        - 6d1962368c81 Virtual Size: 392.6 MB
          - a5921d02eddf Virtual Size: 421.6 MB
        - cc0c3d5d83 Virtual Size: 421.6 MB
          - a72d95590ee Virtual Size: 421.6 MB
        - 457d7e576c99 Virtual Size: 421.6 MB Tags: pg/base:9.4.4
          - 28153fe5d20 Virtual Size: 256 MB ---+ RUN apt-get update # 2015-10-21
            - 4584c85425d1 Virtual Size: 277 MB
          - 690485964f Virtual Size: 319.1 MB
            - 152cb2d7d365 Virtual Size: 348 MB
          - 40e5f829232d Virtual Size: 348 MB
            - 3c3dfe714 Virtual Size: 348 MB Tags: pg/base:9.4.5, pg/base:latest
            - 637737e2a991e Virtual Size: 348 MB Tags: pg/base:9.4.5, pg/base:latest

© Atos - 2015
The default locale that is present is “POSIX”. This example code is for Ubuntu. For Debian use something like this:

RUN echo "en_US.UTF-8 UTF-8" >> /etc/locale.gen
RUN locale-gen && dpkg-reconfigure --frontend noninteractive locales
RUN update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
Or you can try with gosu, https://github.com/tianon/gosu
Building the images

- Automate the image building with Ansible
- Deploy Dockerfiles on build server and create images
- Parameters can vary in the Dockerfiles
  - Created from templates with placeholders
    - `{{ variable }}`
- Problems
  - The Ansible docker module was problematic
    (inconsistencies between docker-py versions made build fail)
  - Use the docker command directly
Building the images (2)

- name: Create data container
  shell: docker run
  --name data
  --detach=true
  \{\{ data_container_volumes | join(' ') \}\}
  postgresql/data
  register: data_container_id

- name: Create server container
  shell: docker run
  --name server
  --detach=true
  --expose 5432
  --publish \{\{ pg_server_port \}\}:5432
  --volumes-from data
  postgresql/server
  register: server_container_id
Loading data

- Data from previous instance available from pg_dump dump
- Use pg_restore to load it
  - From where?
  - Use a separate container for one time and manual interventions
  - Additional software in that container
  - Utility container
- How to connect
  - Exposed port
  - Unix socket on shared base volume (works, but feels awkward)
This would probably work without the restore command because we are streaming WAL files. But, because the replication slots can’t be preinitialised, it is possible to lose some WAL file if you don’t connect soon enough.
Data replication (2)

- restore_command is problematic
  - Access data from container on other machine
  - What about another container providing a RSync server exposing the WAL archive
  - Sure, why not, more containers
## Backups

- Create daily backups
  - `pg_basebackup` every midnight
  - Compress the backup
  - Retention policy for backups
    - 7 daily, 5 weekly, 12 monthly, 5 yearly
  - Delete old backups that are obsolete

- How?
  - OK, another container running a cron job
  - Hint: use pigz for compression or something similar or it will last forever

500GiB 1 core -> 8 hours
500GiB 8 cores -> 2 hours
Backups (2)

- Where to take backups from
  - Master or slave

- From master
  - Works fine, all backup info in the backup_label file and in the backup file (e.g. 000000030000000000000000E.00000060.backup)
Why do we need all this data?
- We need to know which WAL files we need to keep for this specific backup
- We need to know the end time to know that we can’t recover to a time previous than that with this backup
Backups (4)

- From slave
  - Tricky, backup info in the backup_label file doesn’t contain end WAL and time
  - Backup file not created
  - How to get them?

- Last WAL
  - Stream all WALs, get the name of the last one (works but it is ugly)
  - Delete the WALs afterwards
Backups (5)

Deducing the name of the last WAL

- A better solution would be to deduce the WAL name from the current LSN
- pg_current_xlog_location()
  - ERROR: recovery is in progress
  - HINT: WAL control functions cannot be executed during recovery.
- pg_last_xlog_replay_location() – Works
  - 0/7000060
  - But which timeline?
  - select pg_xlogfile_name(pg_last_xlog_replay_location());
    - ERROR: recovery is in progress
    - HINT: pg_xlogfile_name() cannot be executed during recovery.
Backups (6)

Deducing Last WAL

- Can we use the output of `pg_basebackup` with `verbose` on
  NOTICE: `pg_stop_backup` complete, all required WAL
  segments have been archived
  `pg_basebackup`: base backup completed
- No info there if WALs are not fetched
- If you fetch `(--xlog-method=fetch, or stream)` the WALs
  then the log contains this data
  transaction log start point: 0/13000028 on
  timeline 2
  transaction log end point: 0/130000F0
  `pg_basebackup`: base backup completed
- But this is useless because we get the files in the backup so
  we can just look for their names
It might not be correct if the slave is intentionally delayed.
Backups (8)

▶ Hint from 2ndQuadrant (Marco Nenciarini)
- Usint the pg_controldata tool on the backup folder will get you the information for the last WAL and the time when the last change was made
Backups (9)

- Hint – your backup will fail if the slave is promoted during the backup
  - transaction log start point: 0/7000060 on timeline 1
  - pg_basebackup: starting background WAL receiver
  - pg_basebackup: could not get transaction log end position from server: ERROR: the standby was promoted during online backup
  - HINT: This means that the backup being taken is corrupt and should not be used. Try taking another online backup.
The replication slot becomes active after you use it for the first time. A better approach would be to mark the start location when the slot is created. But for this we will have to wait for 9.5.
Monitoring

- All containers should be alive
  - If the server container dies ➔ promote the slave
  - If other containers die ➔ raise warning
  - This is managed from outside – HA coordination software (corosync, ucarp, ...)

- Analysing logs
  - Another container running some custom scripts and pgBadger
When you have a lot of containers, especially the PostgreSQL server with a large number of connected clients and a moderate value for work_mem, it is easy to use a lot of RAM.

To free RAM the OOM killer will kill some process, it is likely that it will be our server container 😞

There is a parameter to disable this in Docker (--oom-kill-disable) but it seems it is not working very good.

It is better to limit the number of connections, and use a connection pooler (pgBouncer). Maybe an idea for another container on the stack 😊
The server freezing was caused by a kernel bug and strongswan. There were no more problems after the last kernel update.
Operational problems (2)

- Transparent Huge Pages (THP)
- They cause problems when the kernel decides to defragment the RAM.
- The server becomes very slow for about an hour

```
echo never >
/sys/kernel/mm/transparent_hugepage/enabled
```
Missing functionalities

- Detailed monitoring of hardware, not only PostgreSQL
- Centralised logging to another server for analysis and correlation
- pgBouncer 😊
- Enhancements in PG 9.5 that will make some things easier
QUESTIONS?
Thanks