POC: PostgreSql Central Backup with pg_dump.

Author: Gerrit Seré
Date: 9 April 2021

Who are we?

The Flemish Government is divided in 12 policy areas. In the Department for the Environment (dOMG), we are a small team of 8 people and maintain some on-premise Ict infrastructure.

For our cloud environment, we use OpenNebula for the hypervisors and Ceph for the storage environment. Almost 2100 KVM servers run on this cloud environment and nearly 480 are PostgreSql servers. Both environments are very stable and this makes us very happy. Our developer teams make a lot of web applications, which are mainly written in Java. Those applications are running on Apache Tomcat and increasingly on Spring boot servers. On the database side, we chose PostgreSql for the nice GIS features. The PostgreSql servers are running on CentOS, while the web applications are running on Ubuntu. Why the difference? It is partly historical. Previously, a lot of developers chose Ubuntu for their first steps, so we chose the same operation system.

Important Warning Caution!!!

CAUTION

pg_dump - pg_restore

You can use "pg_dump" to create a dump file from a database with a lower version of PostgreSql. But the restore with "pg_restore" should be with the same PostgreSql version. In this POC we use the highest PostgreSql version for dumps and restores. It works in our environment but we won't follow this path.

A better choice is to log in to the virtual server with "ssh" and then start "pg_dump" locally. The dump file is then stored on the central backup server. Something like:

```
centralbackup$ ssh postgres@server "pg_dump -d database" >
/backup/server/date_database.dump
```

Environment

Like many organisation we have 3 environments:

- Development
- Staging

- Production

Those environments are separated into vlans. Each application runs in its own KVM server and as standard we spin up 2 instances. Our load balancer spreads the load over these 2 servers. This is in part why we have so many virtual servers. This is very flexible and spreads the risk. We use an in house solution for orchestration of our applications. Each time an application changes version, we destroy the virtual server and build it again from scratch.

A lot of applications need a database. Here we have the strategy of 1 KVM PostgreSql server only having 1 database. If an application needs 2 databases, we create a new KVM PostgreSql server. For each database we only deploy 1 KVM server. We don't have a high availability PostgreSql environment. Nearly all our PostgreSql servers have the same version, 11.x. Next year, we are upgrading them to 13.x.

So, one service runs in one KVM server. It's like a docker container but the container is the KVM server. This gives us a lot of advantages:

- Easy to maintain.
- Easy to roll out more servers if the load is high.
- Less risk if something is broken.

E.g. Let's suppose we have an application for the registration of cars on motorways. This means we have in total 3 servers and the host names of the KVM servers look like this:

- car-de-1
- car-de-2
- car-postgres-de-1

car = Name of the application.
de = Development environment.
1 = Number with makes the host name unique.

If the load is high, we deploy more KVM's servers. Naming car-de-3, car-de-4, … but this is hardly ever the case in development or staging environments.

What else do we have?

- 2 data centers.
- Inside the data center, 10Gb network.
- Zabbix for monitoring.
- Apache for reverse proxy.
- Bacula for backup.
- Subversion.
- Minio for S3 storage. Ceph can also do S3 but with minio we can backup the data more easily. We still need to find a good solution for taking backups of S3 storage.

Backup PostgreSql

Looking at the car-postgres-de-1 server, we have 2 persistent disks which are mounted on:

- /data
- /dbexport

In the '/data' folder are the current database files and in the '/dbexport' folder, the pg_dump files.

For backing up the data, we have 2 backup solutions.

- Barman
- Bacula

Barman is our central backup solution for PostgreSql but it does not appear in the story for this POC.

Every night we do a pg_dump of all our databases. The existing dump file in the '/dbexport' folder are always overwritten. When the pg_dump is finished, Bacula backups the '/dbexport' folder. This is only in the production environment. Otherwise we would have too many Bacula clients. Doing a pg_dump is also for consistency. Checking that the data is ok.

In our environments, we have a lot of databases but most of them are rather small. The biggest, however, is around 500 GB.

The problem?

When deploying a new PostgreSql KVM server, our application teams need to specify the disk size for those 2 folders. Most of the time, it is anyone's guess and most of the time the estimate is too high. Because we have so many KVM PostgreSql servers, we waste a lot of disk space. Let's drop this data into a table:

Table 1. Total diskinfo in TB

Environment	Servers	data size	data use	data avalable	%data used	dbexport size	dbexport used	dbexport avalable	%dbexport used
Development	169	4.36	1.33	2.83	32.04%	2.69	0.29	2.27	11.16%
Staging	148	4.22	1.39	2.64	34.57%	2.91	0.46	2.32	16.49%
Production	157	8.86	4.70	3.80	55.25%	3.04	0.95	1.97	32.56%

All the data is in TB. For development, we have 169 servers. The total diskspace of 169 servers is 4.36TB, 1.33TB is used and 2.83TB is availible. So, 32.04% of the diskspace is used. For the '/dbexport' folder there is only 11.16% used. This means that for development nearly 90% of the diskspace is unused. Some space is indeed wasted.

Current backup strategy

Let's go a little bit deeper into our current backup strategy. This is a very easy procedure. The "pgdump.sh" script runs by cron at a specific time.

pgdump.sh

```
#!/bin/bash

POSTGRESDIR=`find /usr -name "pgsql-*" 2>/dev/null`; POSTGRESDIR=$POSTGRESDIR/bin
PATH=$POSTGRESDIR:$PATH

if [ -d /dbexport ]; then
   for DATABASE in `psql -A -t -c "select datname from pg_database where datname not
in ('postgres', 'template0', 'template1');"`; do
      /usr/bin/pg_dump -Fc -Z0 ${DATABASE} | /usr/bin/pigz >
/dbexport/${DATABASE}.dump.gz
   done
fi
```

We get a list of our databases. Most of the time, this is just a single database. We use the pigz program for compression. pigz uses all CPU cores when doing compression.

The POC: Install the central backup server.

Instead of doing local backups, we want to use a central pg_dump backup solution. This setup is more complicated because a lot more can go wrong. The central backup server, which is also a KVM server, is pulling the data from the clients. This is a safer approach than doing push backups. The clients do not need access to the backup server.

The specification for the central backup server:

- ubuntu 20.04
- 16 GB RAM
- 16 - 32 CPUs
- a ZFS persistent disk of 600 GB

For doing a lot of backups in parallel, together with compression, we need a lot of CPU power. Let's do some tests between 16 and 32 CPUs.

Persistent zfs disk

The persistent disk is mounted on '/backup'. What is special is that we use zfs for this file system. A while ago, some developers from different operation systems started working together to make a uniform zfs. OpenZfs. In the future we should have the same zfs filesystem on different operation systems with the same or almost the same features. Installing zfs is not part of this POC but there

are many tutorials to get you up and running, ex: Linuxhint. But, here it is anyway.

```
# zpool create -o ashift=12 backup /dev/vdb1
# zfs set compression=gzip backup
# zfs set recordsize=1M backup
```

Let's have a zfs overview:

```
# zpool list
NAME     SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
backup   600G  96.3G   503G        -         -     5%    16%  1.00x    ONLINE  -

# zfs list
NAME     USED  AVAIL     REFER  MOUNTPOINT
backup  17.5M   581G      264K  /backup

# zfs get compression,recordsize backup
NAME    PROPERTY     VALUE     SOURCE
backup  compression  gzip      local
backup  recordsize   1M        local
```

When creating a zfs filesystem, zfs needs to known the sector size. Many disks lie about their sector size and tell the OS that is 512 bytes. This is ok for small disks but nowadays, disks with more then 4TB are very common and they have a bigger sector size, at least 4096 bytes. Some SSD or NVME disks have an even bigger sector size. So, for performance reasons it is important to set the sector size correctly. Check ZFS zpool with wrong ashift. How bad is this?

On the zfs file system we set 2 properties. On zfs filesystems we always set compression on. As standard this is set to lz4 but in our case a higher compression algorithm is better. To achieve better compression we set the maximum record size to 1MB.

There are many other performance tuning possibilities but those two are the most important.

PostgreSql client

On the central server backup server, we only need to install a PostgreSql client for doing our backup and restores. Most of our PostgreSql servers are some version of 11, a few are 9.6 and the oldest is version 9.3.25. Ubuntu 20.04 uses version 12 as standard. PostgreSql client version 12 works very well most of the time when doing backup and restores from previous PostgreSql versions. Even our eldest PostgreSql version 9.3.25 only gives a few warnings. Perhaps, if your backup or restores don't work, then you may need to install extra PostgreSql client versions. This could make things more complicated.

Installing the PostgreSql client and user:

```
# apt install postgresql-client-12
# chown postgres:postgres /backup
# useradd -m -s /bin/bash postgres
# su - postgres
```

pg_dump asks for a password when setting up a connection. To get rid of it, use a ".pgpass" file in the home folder of the PostgreSql user. The format of this file is:

```
host:port:db_name:user_name:password
```

For each server there is one entry. That means a lot of entries. For this POC, we use the same password everywhere. This is fine for a POC.

```
$ vi  ~/.pgpass
*:*:*:*:your_password

$ chmod 600  ~/.pgpass
$ mkdir scripts
```

Parallel

For backing up a lot of clients in parallel we need to install some special software which can handle this. Parallel will do this job perfectly.

```
# apt install parallel
```

Take some time to get used to it, watch some good videos on youtube. There are plenty of possibilities. For this POC we used:

```
parallel has a lot of options:
   --keep-order         : Sequence of output same as the order of input.
   --jobs x             : How many jobs can be started simultaneously. When x is 0,
start all jobs.
   --joblog <LOGFILE>   : For each job logs the startdate, how long the job has
runned, the command which was run.
   --colsep ':'         : The input file is a column separeted file.
   {1} {2}              : First and second field of the input file.
   :::: backup.postgres : The input file for parallel
```

Putting everything together

Now that the basic components are installed, we need to write 3 small scripts. All those scripts are in the home directory of the postgres user '/home/postgres/scripts':

- backupPostgres.sh (main script)
- getDatabaseSize.sql (get the size of the PostgreSql databases)
- dumpDatabase.sh (make a dump file of the PostgreSql database)

backupPostgres.sh

```
#!/usr/bin/bash

# 200907-1643
DATE=$(date +"%y%m%d-%H%M")

LOGFOLDER="/backup/log"
LOGFILE="${LOGFOLDER}/${DATE}.joblog"
ERRORFILE="${LOGFOLDER}/${DATE}.error"

[ ! -d "${LOGFOLDER}" ] && mkdir "${LOGFOLDER}"

# cat development.postgres (list of servers)
# car-postgres-de-1
# airpolution-postgres-de-1

cat development.postgres | parallel --jobs 0  psql -h {} -U postgres -f
./getDatabaseSize.sql 2> ${ERRORFILE} | sort  -t ':' -rnk 3,3  > backup.postgres

# cat backup.postgres (the biggest database are listed first)
# host-name:database:size in bytes: size human can read

# car-postgres-de-1:car_development:8360607:8165 kB
# airpolution-postgres-de-1:airpolution_development:8319647:8125 kB

parallel --keep-order --jobs 16 --joblog $LOGFILE --colsep ':'
/home/postgres/scripts/dumpDatabase.sh {1} {2} :::: backup.postgres
```

getDatabaseSize.sql

```
COPY (select :'HOST'
hostname,datname,pg_database_size(datname),pg_size_pretty(pg_database_size(datname))
from pg_database where datname not in ('postgres', 'template0', 'template1')) TO
STDOUT CSV DELIMITER ':';
```

dumpDatabase.sh

```
# Dump postgres

SERVER=$1
DATABASE=$2
WORKDIR="/backup"
LOGDIR="/backup/log"

# Main program

# 200907-1643
DATE=$(date +"%y%m%d-%H%M")

WORKDIR+="/${SERVER}"

[ ! -d "${WORKDIR}" ] && mkdir "${WORKDIR}"

pg_dump -h $SERVER -d $DATABASE -U postgres -Fc -Z0 >
${WORKDIR}/${DATE}_${DATABASE}.dump 2> ${LOGDIR}/${DATE}_${SERVER}_${DATABASE}
```

The main script is 'backupPostgres.sh'. Everything gets logged in '/backup/log'. In the 'development.postgres' file, we define all the PostgreSql servers which we want to back up. First we execute an sql script. The result is a 'backup.postgres' file, ordered by the size of the database in bytes and in human readable size. The intention is to start with the biggest databases first because they mostly take the most time to back up. This is not always the case because some PostgreSql databases have many indexes and the dump file is surprisingly slow.

The actual backup starts at the last line of this script, which starts the 'dumpDatabase.sh' script. The result is that for each PostgreSql server we create a folder and put the dump file into it. The dump file starts with a date followed by the name of the database. Manually change the parameter '--jobs 16' to '--jobs X' to do the tests.

The results

In total, there are 140 servers to connect to and 177 databases to back up:

- 6 databases between 90GB and 120GB
- 11 databases between 10GB and 20GB
- 8 databases between 20GB and 30GB
- Lots of small databases

Each run is started in a screen session because it takes a lot of time.

```
$ cd /home/postgres/scripts
$ date; ./backupPostgres.sh; date
```

Regulary check the status with these commands.

- top
- htop
- zpool status 2
- vmstat 2
- dstat -tam 2
- iostat -xz
- iftop -Bb
- sar -n DEV 2

'zpool status 2' shows much less throughput then the other commands. This is because zfs compress the data before writing to disk. In our case this is a factor of 3.5. When multiplying this value with the output of 'zpool status 2', the result is closer to the other commands. Check the compression rate with 'zfs get compressratio backup'.

Because this is a live system, the figures only give an indication.

First run

Table 2. First run

CPUs	parallel dumps	zfs MB Backup	compression	Total MB Backup	Time in seconds	Time in Minutes	MB/S
16	16	284672	3.5	996352	7292	121	136
16	32	284672	3.5	996352	7249	120	137
32	32	284672	3.5	996352	7084	118	140
32	64	284672	3.5	996352	8695	144	114

284672 MB is the total backup size on zfs. But this data is compressed with a factor of 3.5. On an normal filesystem like ext4 or xfs, the backup size is 996352 MB. The 'parallel dumps' heading is the '--jobs' setting of parallel. Starting twice as much jobs as CPUs is most of the time not an advantage.

More CPUs does not mean that the complete backup window is smaller. But checking 'iftop' on a 32 CPU system, network throughput regularly is between 350MB and 400MB. With 16 CPUs, most of the time between 200MB and 250MB.

Optimization?

Watching the backup at regularly times is important. After some time there was only 1 client still backing up. Taking a closer look at this client, there were 2 big databases, 109GB and 93GB and only 1 CPU. Giving this client 2 CPUs reduced the backup window a lot. Knowing how to update some OS parameters is important but knowing your environment is even more important for doing performance optimization.

Table 3. Give 1 client 2 CPUs

CPUs	parallel dumps	zfs MB Backup	compression	Total MB Backup	Time in seconds	Time in Minutes	MB/S
16	16	284672	3.5	996352	5946	99	167
16	32	284672	3.5	996352	5792	96	172
16	177	284672	3.5	996352	5926	98	168
32	32	266240	3.6	958464	5743	95	166
32	64	284672	3.5	996352	4790	79	208
32	177	284672	3.5	996352	4782	79	208

Giving one client an extra CPU reduces the time significantly, between 20 to 30 minutes. In the third and last entry of table 3, we started all dumps in parallel. In total 177 dumps were started with no significant reduction in time.

Errors/Warnings

- Check the '/backup/log' folder for errors and warnings. Every PostgreSql Server has a log or error log file.
- Using the pg_dump/pg_restore with a lower or higher version of the PostgreSql server can throw errors or warnings. Study them carefully. Sometimes we need a different PostgreSql client (pg_dump,pg_restore). That makes things more complicated.

  This is a warning when restoring a 9.3.25 PostgreSql server with a 12.5 pg_restore program.

  ```
  pg_restore: while INITIALIZING:
  pg_restore: error: could not execute query: ERROR:  unrecognized configuration
  parameter "idle_in_transaction_session_timeout"
  Command was: SET idle_in_transaction_session_timeout = 0;
  pg_restore: error: could not execute query: ERROR:  unrecognized configuration
  parameter "row_security"
  Command was: SET row_security = off;
  pg_restore: warning: errors ignored on restore: 2
  ```

- Some clients have 'pg_dump: warning: WITH OIDS is not supported anymore ...' messages. This causes a problem when upgrading to a newer version of PostgreSql, we need to change the database schema. In total there were 18 databases with this problem.
- Check the file '<data>.joblog'. This file is created by parallel. It shows some usefull information like the 'Starttime', how much data was transferred, the exit code, the command which was executed ... the 'Starttime' is in Unix timestamp as standard. Conversion can easily done with the command 'date -d@<Starttime>'

  What is interesting when there are many failed jobs, is that there are possibilities to resume jobs. Check the '--resume',' --resume-failed' and '--retry-failed' of the 'parallel' command.

Other findings

The intention was doing parallel backups but what else was discovered?

1. Some clients have a persistent disk but the PostgreSql dump file is so small that maybe there is no need to add a persistent disk.
2. After we make a dump file in production we back this file up to Bacula. This takes up one license. In total we can save 160 licenses. That could save a lot of money.
3. All dump are made from 'pg_dump' version 12.5. Why not use these dumps for testing upgrades?

Restore

Backups are important but restores are even more important. It is a good practise to regularly test your restores.

In this case login to the 'car-postgres-de-1' server. First drop the old database and create a new one.

```
[postgres@car-postgres-de-1]$dropdb car_development;
[postgres@car-postgres-de-1]$createdb car_development;
```

Do the restore:

```
pg_restore --host=car-postgres-de-1  --dbname=car_development --username=postgres --no
-privileges --no-owner 201007-1909_car_development.dump
```

After some time, the restore is finished. Check if everything is ok. It's very important!!!

Rolling out in production

What should we consider when we continue with a central backup solution:

- Maintenance scripts to clean up old PostgreSql dumps.
- We definitely need to backup the global objects (pg_dumpall -r ...)
- A lot of the daily PostgreSql dump would be the same. Certainly those from development or staging. One can think of dedupping those dumps in zfs. Most of the time this is a bad idea. A better solution is reducing duplicate files. A few solutions exist rdfind, fdupes, dupeGeru, fslint. When choosing between soft and hard links, choose hard links. It is not the first time that a soft link points to nowhere.
- On 1 December 2020, OpenZfs 2.0 was released with a better algorithm for compression. Check Zstd compression support.
- There is a lot more to tell about the commands 'pg_dump' and 'pg_dumpall'. A good starting point: How to effectively dump PostgreSQL databases

Conclusion

This was an interesting POC. The main intention was to reduce disk space when centralizing the backup. And this was a success because we can control the disk size much better. Also interesting was that we found some small problems which were easy to fix. Which backup strategy is better? Both methods have advantages and disadvantages but I have a slight preference for a central backup solution, because:

- It is easier to control the backups.
- The backup is at one place.
- Restores can be done everywhere.
- Save us a lot of Bacula licenses.