Install and Start PostgreSQL on Alpine Linux

luppeng Containers, Database 6 Minutes

I found that installing and starting PostgreSQL on Alpine Linux to be a not so straight forward task. So here's a short post on how I did it.

Caveats:

- I am using the official Alpine Linux package repository
- These steps were performed within a vanilla Alpine Linux LXC container (but I try to avoid going too much into LXC here)
- I am using a custom data directory located at `/var/lib/postgresql/data`

Step 1: Determine PostgreSQL / Alpine Linux Version

Each version of Alpine Linux uses a different official package repository, with each having a specific of PostgreSQL. For example, Alpine Linux v3.10's package repository only has PostgreSQL 11.6 available. However, depending on your version of Alpine Linux and PostgreSQL required, you can explicitly specify the repository + PostgreSQL version to pull from.

You can check the available versions of PostgreSQL at the Alpine Linux package repository.

For simplicity, I will use PostgreSQL 11.6 that is within the Alpine Linux v3.10 package repository.

Step 2: Update Package Repository Index and Install the PostgreSQL Package

```
~ # apk update
fetch http://dl-cdn.alpinelinux.org/alpine/v3.10/main/x86_64/APKINDEX.tar.gz
fetch
http://dl-cdn.alpinelinux.org/alpine/v3.10/community/x86_64/APKINDEX.tar.gz
v3.10.4-3-g0ed412d466 [http://dl-cdn.alpinelinux.org/alpine/v3.10/main]
v3.10.4-7-g6ee081a8fb [http://dl-cdn.alpinelinux.org/alpine/v3.10/community]
OK: 10343 distinct packages available

~ # apk add postgresql
(1/12) Installing ncurses-terminfo-base (6.1_p20190518-r0)
(2/12) Installing ncurses-terminfo (6.1_p20190518-r0)
(3/12) Installing ncurses-libs (6.1_p20190518-r0)
(4/12) Installing libedit (20190324.3.1-r0)
(5/12) Installing db (5.3.28-r1)
(6/12) Installing libsasl (2.1.27-r4)
(7/12) Installing libldap (2.4.48-r0)
(8/12) Installing libpq (11.6-r0)
(9/12) Installing postgresql-client (11.6-r0)
(10/12) Installing tzdata (2019c-r0)
(11/12) Installing libxml2 (2.9.9-r3)
(12/12) Installing postgresql (11.6-r0)
Executing busybox-1.30.1-r3.trigger
OK: 37 MiB in 31 packages
```

Note: It is possible to combine these two steps into one, using the second command with the `--no-cache` flag. This approach will save ~ 2MB of space.

Step 3: Create Required Folders for PostgreSQL

At this point, we need to set up two folders for PostgreSQL:

Step 3.1: Folder for PostgreSQL Socket

PostgreSQL requires a directory path for create and store its `.s.PGSQL.5432` socket. By default, this would be in the `/run/postgresql` directory. However, my experience is that when running Alpine Linux in an LXC container, changes in this folder does not persist between container restarts (same container, just stop/start).

If having to manually recreate the folder and start the daemon after a reboot is fine with you, then do the following:

```
~ # mkdir /run/postgresql
~ # chown postgres:postgres /run/postgresql/
```

However, if you intend to have the PostgreSQL daemon auto start after the OS boots, then skip this sub-section for now.

Step 3.2: Data folder for the database system

PostgreSQL requires a folder to store its the database files, so lets create one:

```
### Change to postgres user, and navigate to its home directory
~ # su postgres -
/root $ cd
~ $ pwd
/var/lib/postgresql

### Create the data directory, and make it less permissive
~ $ mkdir /var/lib/postgresql/data
~ $ chmod 0700 /var/lib/postgresql/data
```

Step 4: Create a New Database Cluster with initdb

PostgreSQL's *initdb* command helps us to setup the database instance and create a new PostgreSQL database cluster. We will use this to, as the name suggests, initialize the database:

```
~ $ initdb -D /var/lib/postgresql/data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locales
COLLATE: C
CTYPE: C.UTF-8
MESSAGES: C
MONETARY: C
NUMERIC: C
TIME: C
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default timezone ... UTC
```

```
selecting dynamic shared memory implementation ... sysv
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... sh: locale: not found
2020-01-31 18:17:25.809 UTC [327] WARNING: no usable system locales were found
ok
syncing data to disk ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success.
```

Step 5: Allow Remote Connections via Password Authentication (Optional)

Depending on your needs, you may want to allow remote connections to this PostgreSQL database instance:

```
~ $ echo "host all all 0.0.0.0/0 md5" >> /var/lib/postgresql/data/pg_hba.conf
~ $ echo "listen_addresses='*'" >> /var/lib/postgresql/data/postgresql.conf
```

(Credit)

Step 6: Changing the UNIX Socket Directories Value (Optional)

Do you intend to have the PostgreSQL daemon auto start after your Alpine Linux LXC container boots up? If yes, do this step. Otherwise, skip.

The PostgreSQL daemon stores its UNIX socket based on the values defined in `unix_socket_directories` of the `postgresql.conf` file. The default value is `'/run/postgresql,/tmp'`.

As mentioned above, my experience is that when running Alpine Linux in an LXC container, changes in the `/run` folder does not persist between container restarts (same container, just stop/start). Upon restarting the container, the PostgreSQL daemon will not be able to start as it cannot find the `/run/postgres` folder to store its socket.

To solve this, simply remove it from the `unix_socket_directories` setting:

```
$  vi /var/lib/postgresql/data/postgresql.conf
...
unix_socket_directories = '/tmp'
...
```

Step 7: Start the PostgreSQL Database Server with pg_ctl

```
~ $ pg_ctl start -D /var/lib/postgresql/data
waiting for server to start....2020-01-31 18:21:59.904 UTC [340] LOG: listening
on IPv4 address "0.0.0.0", port 5432
2020-01-31 18:21:59.904 UTC [340] LOG: listening on IPv6 address "::", port 5432
2020-01-31 18:21:59.908 UTC [340] LOG: listening on Unix socket
"/run/postgresql/.s.PGSQL.5432"
2020-01-31 18:21:59.918 UTC [340] LOG: listening on Unix socket
"/tmp/.s.PGSQL.5432"
2020-01-31 18:21:59.946 UTC [341] LOG: database system was shut down at 2020-01-
31 18:17:26 UTC
```

```
2020-01-31 18:21:59.963 UTC [340] LOG: database system is ready to accept
connections
done
server started
```

Step 8: Test the PostgreSQL Server

Connect to our PostgreSQL server and create a database within it to ensure that it is working as expected:

```
~ $ psql
psql (11.6)
Type "help" for help.

postgres=# \c
You are now connected to database "postgres" as user "postgres".

postgres=# SELECT datname FROM pg_database;
  datname
-----------
 postgres
 template1
 template0
(3 rows)

postgres=# create database demo;
CREATE DATABASE

postgres=# SELECT datname FROM pg_database;
  datname
-----------
 postgres
 demo
 template1
 template0
(4 rows)
```

Step 9: Enable PostgreSQL Daemon to Auto Start After System Boot

In this step, we would want to enable the PostgreSQL daemon to auto start when Alpine Linux boots up. As Alpine Linux uses OpenRC for its init system, we will add a simple custom shell script that simply starts the PostgreSQL daemon as the *postgres* user.

I created a *start* script in the `/etc/local.d/` folder with executable permissions:

```
(Change back to root user)
# touch /etc/local.d/postgres-custom.start
# chmod +x /etc/local.d/postgres-custom.start
# vi /etc/local.d/postgres-custom.start
```

And added this simple command within the script:

```
#!/bin/sh
su postgres -c 'pg_ctl start -D /var/lib/postgresql/data'
```

Save the script. Add the `local` service to the `default` runlevel, and reload the state:

```
~ # rc-update add local default
 * service local added to runlevel default
~ # openrc
```

```
 * Caching service dependencies ...          [ ok ]
 * Starting local ...                        [ !! ]
```

It is expected to experience some warnings when reloading the state as the PostgreSQL daemon is already running from our previous steps.

Lastly, we can test that the auto start it works by:

- Restarting the server (or in my case, the LXC container)
- SSH into the Alpine Linux instance
- Run `psql` as the *postgres* user to connect to the local PostgreSQL database
    - `~ # su postgres -c psql`
- Observe that the new database we created in Step 7 is still around
    - ```
      psql (11.6)
      Type "help" for help.

      postgres=# SELECT datname FROM pg_database;
        datname
      -----------
       postgres
       demo       <-- the database we created earlier is still here!
       template1
       template0
      (4 rows)
      ```

Summary

This post covers a simple “up and running” setup – there are a lot more other features (like logging) that can be configured.

Although I have been using PostgreSQL for most of my posts now, I am always learning something new about it (e.g. the UNIX socket directory). I’m not sure if the lack of persistence in the `/run` folder is just a characteristic of Alpine Linux in LXC Containers, or Alpine Linux in general.

Hope this post has been useful for you

https://luppeng.wordpress.com/2020/02/28/install-and-start-postgresql-on-alpine-linux/
      ```