<https://docs.docker.com/engine/examples/postgresql_service/>

Dockerize PostgreSQL

Note: - If you don’t like sudo then see [*Giving non-root access*](https://docs.docker.com/engine/installation/binaries/#giving-non-root-access)

Installing PostgreSQL on Docker

Assuming there is no Docker image that suits your needs on the [Docker Hub](http://hub.docker.com/), you can create one yourself.

Start by creating a new Dockerfile:

Note: This PostgreSQL setup is for development-only purposes. Refer to the PostgreSQL documentation to fine-tune these settings so that it is suitably secure.

*#*

*# example Dockerfile for https://docs.docker.com/examples/postgresql\_service/*

*#*

**FROM** ubuntu

**MAINTAINER** SvenDowideit@docker.com

*# Add the PostgreSQL PGP key to verify their Debian packages.*

*# It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc*

**RUN** apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8

*# Add PostgreSQL's repository. It contains the most recent stable release*

*# of PostgreSQL, ``9.3``.*

**RUN** echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" > /etc/apt/sources.list.d/pgdg.list

*# Install ``python-software-properties``, ``software-properties-common`` and PostgreSQL 9.3*

*# There are some warnings (in red) that show up during the build. You can hide*

*# them by prefixing each apt-get statement with DEBIAN\_FRONTEND=noninteractive*

**RUN** apt-get update && apt-get install -y python-software-properties software-properties-common postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3

*# Note: The official Debian and Ubuntu images automatically ``apt-get clean``*

*# after each ``apt-get``*

*# Run the rest of the commands as the ``postgres`` user created by the ``postgres-9.3`` package when it was ``apt-get installed``*

**USER** postgres

*# Create a PostgreSQL role named ``docker`` with ``docker`` as the password and*

*# then create a database `docker` owned by the ``docker`` role.*

*# Note: here we use ``&&\`` to run commands one after the other - the ``\``*

*# allows the RUN command to span multiple lines.*

**RUN** /etc/init.d/postgresql start &&\

psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\

createdb -O docker docker

*# Adjust PostgreSQL configuration so that remote connections to the*

*# database are possible.*

**RUN** echo "host all all 0.0.0.0/0 md5" >> /etc/postgresql/9.3/main/pg\_hba.conf

*# And add ``listen\_addresses`` to ``/etc/postgresql/9.3/main/postgresql.conf``*

**RUN** echo "listen\_addresses='\*'" >> /etc/postgresql/9.3/main/postgresql.conf

*# Expose the PostgreSQL port*

**EXPOSE** 5432

*# Add VOLUMEs to allow backup of config, logs and databases*

**VOLUME** ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

*# Set the default command to run when starting the container*

**CMD** ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c", "config\_file=/etc/postgresql/9.3/main/postgresql.conf"]

Build an image from the Dockerfile assign it a name.

$ docker build -t eg\_postgresql .

Run the PostgreSQL server container (in the foreground):

$ docker **run** --rm -P --name pg\_test eg\_postgresql

There are 2 ways to connect to the PostgreSQL server. We can use [*Link Containers*](https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/), or we can access it from our host (or the network).

Note: The --rm removes the container and its image when the container exits successfully.

Using container linking

Containers can be linked to another container’s ports directly using -link remote\_name:local\_alias in the client’s docker run. This will set a number of environment variables that can then be used to connect:

$ docker run --rm -t -i --link pg\_test:pg eg\_postgresql bash

postgres@7ef98b1b7243:/$ psql -h $PG\_PORT\_5432\_TCP\_ADDR -p $PG\_PORT\_5432\_TCP\_PORT -d docker -U docker --password

Connecting from your host system

Assuming you have the postgresql-client installed, you can use the host-mapped port to test as well. You need to use docker ps to find out what local host port the container is mapped to first:

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

5e24362f27f6 eg\_postgresql:latest /usr/lib/postgresql/ About an hour ago Up About an hour 0.0.0.0:49153->5432/tcp pg\_test

$ psql -h localhost -p 49153 -d docker -U docker --password

Testing the database

Once you have authenticated and have a docker =# prompt, you can create a table and populate it.

psql (9.3.1)

Type "help" **for** help.

$ docker=*# CREATE TABLE cities (*

docker(*# name varchar(80),*

docker(*# location point*

docker(*# );*

CREATE TABLE

$ docker=*# INSERT INTO cities VALUES ('San Francisco', '(-194.0, 53.0)');*

INSERT 0 1

$ docker=*# select \* from cities;*

name | location

---------------+-----------

San Francisco | (-194,53)

(1 row)

Using the container volumes

You can use the defined volumes to inspect the PostgreSQL log files and to backup your configuration and data:

$ docker **run** --rm --volumes-**from** pg\_test -t -i busybox sh

/ *# ls*

bin etc lib linuxrc mnt proc **run** sys usr

dev home lib64 media opt root sbin tmp var

/ *# ls /etc/postgresql/9.3/main/*

environment pg\_hba.conf postgresql.conf

pg\_ctl.conf pg\_ident.conf start.conf

/tmp *# ls /var/log*

ldconfig postgresql

## <https://crondev.com/persistent-postgresql-inside-docker/>

## [Persistent PostgreSQL inside Docker](https://crondev.com/persistent-postgresql-inside-docker/)

[*Alen Komljen*](https://crondev.com/author/komljen/)*December 28, 2015*

Most recommended way to persist data inside docker is to create data only container. However to simplify things it is also possible just to mount a directory from the host and to use that location as persistent storage. Also, this way it is easy enough to dockerize existing Postgres installations.

For existing postgres installations here are some preparation steps (Ubuntu 14.04 with postgres 9.3):

Stop postgres if running:

|  |  |
| --- | --- |
| 1 | sudo stop postgresql |

Copy data directory to new location, for testing purposes:

|  |  |
| --- | --- |
| 1 | sudo cp -r /var/lib/postgresql /var/lib/postgresql-docker |

Here is the Dockerfile I used:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | FROM komljen/ubuntu    ENV PG\_VERSION 9.3  ENV USER docker  ENV PASS SiHRDZ3Tt13uVVyH0ZST  ENV PGDATA /var/lib/postgresql/$PG\_VERSION/main  ENV PGRUN /var/run/postgresql    RUN \    echo "deb http://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main" \         > /etc/apt/sources.list.d/pgdg.list && \    curl -sL https://www.postgresql.org/media/keys/ACCC4CF8.asc \         | apt-key add - && \    apt-get update && \    apt-get -y install \            postgresql-${PG\_VERSION} \            postgresql-contrib-${PG\_VERSION} && \    rm -rf /var/lib/apt/lists/\*    RUN \    echo "host all all 0.0.0.0/0 md5" >> /etc/postgresql/${PG\_VERSION}/main/pg\_hba.conf && \    echo "listen\_addresses='\*'" >> /etc/postgresql/${PG\_VERSION}/main/postgresql.conf    COPY start.sh start.sh    EXPOSE 5432  RUN rm /usr/sbin/policy-rc.d  CMD ["/start.sh"] |

And a start bash script:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #!/usr/bin/env bash  #===============================================================================  CONF="/etc/postgresql/${PG\_VERSION}/main/postgresql.conf"  POSTGRES="/usr/lib/postgresql/${PG\_VERSION}/bin/postgres"  #-------------------------------------------------------------------------------  chown -R postgres /var/lib/postgresql  mkdir -p "${PGRUN}/${PG\_VERSION}-main.pg\_stat\_tmp"  chown -R postgres $PGRUN  su postgres -c "/usr/lib/postgresql/${PG\_VERSION}/bin/initdb"    echo "Creating superuser: ${USER}"  su postgres -c "${POSTGRES} --single -D ${PGDATA} -c config\_file=${CONF}" <<EOF  CREATE USER $USER WITH SUPERUSER PASSWORD '$PASS';  EOF  #-------------------------------------------------------------------------------  exec su postgres -c "${POSTGRES} -D ${PGDATA} -c config\_file=${CONF}"  #=============================================================================== |

This postgres container will create superuser which can be used to create other users and databases. Also it will try to initiate new database each time, but in case the data is already present it will fail, which is expected.

Now just build and run persistent postgres docker container:

|  |  |
| --- | --- |
| 1  2 | sudo docker build -t cron/postgres .  sudo docker run -d --name test -v /var/lib/postgresql-docker:/var/lib/postgresql -v /var/run/postgresql-docker:/var/run/postgresql cron/postgres |

Hope you enjoyed reading this blog post. Please leave a comment if encounter on any issues or to get some additional answers.

**[](http://twitter.com/share?url=https://crondev.com/persistent-postgresql-inside-docker/&text=Persistent+PostgreSQL+inside+Docker+)**0**[](http://www.facebook.com/sharer.php?u=https://crondev.com/persistent-postgresql-inside-docker/)**66**[](http://www.linkedin.com/shareArticle?mini=true&url=https://crondev.com/persistent-postgresql-inside-docker/)**3**[](https://plus.google.com/share?url=https://crondev.com/persistent-postgresql-inside-docker/)**16

[](http://www.techbar.me/)

Follow me

### [Alen Komljen](http://www.techbar.me/)

I'm a DevOps/Cloud engineer with experience that spans a broad portfolio of skills, including cloud computing, software deployment, process automation, shell scripting and configuration management, as well as Agile development and Scrum. This allowed me to excel in solving challenges in cloud computing, and the entire IT infrastructure along with my deep interest in OpenStack, Ceph, Docker and the open-source community.

[DevOps](https://crondev.com/category/devops/)  [docker](https://crondev.com/tag/docker/), [postgresql](https://crondev.com/tag/postgresql/)