**Building Docker images – PostgreSQL**

• a. Section 1.1 Creating a Custom Dockerfile for PostgreSQL

*Nano dockerfile has the below commands for building image:*

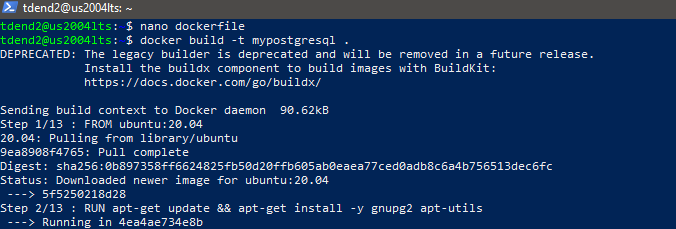
# # example Dockerfile for https://docs.docker.com/engine/examples/postgresql\_service/ # FROM ubuntu:20.04 # Add the PostgreSQL PGP key to verify their Debian packages. # It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc RUN apt-get update && apt-get install -y gnupg2 apt-utils RUN DEBIAN\_FRONTEND=noninteractive TZ="America/Chicago" apt-get -y install tzdata RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com: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/ focal-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 software-properties-common postgresql-9.3 postgresqlclient-9.3 postgresql-contrib-9.3

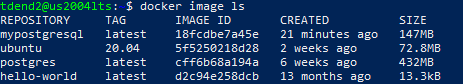
# Note: The official Debian and Ubuntu images automatically ``apt-get clean`` # after each ``apt-get`` # Run the rest of the commands as the ``postgres`` user created by the ``postgres-9.3`` package when it was ``apt-get installed`` USER postgres # Create a PostgreSQL role named ``docker`` with ``docker`` 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"]

—---------------------------------------------------------------------------------------

b. Section 1.2 Building a Custom PostgreSQL Docker Image

Created dockerfile using nano Dockerfile which opens the nano editor where we can write the instructions as required and save it and go to the next step.





—----------------------------------------------------------------------------------------------

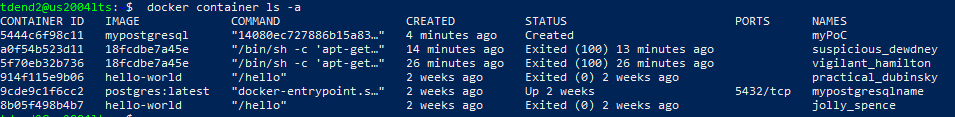
c. Section 1.3 Container creation and verification

• Connect to the DB using psql from your host system

*docker run -it -d --name myPoC mypostgresql*



Test the container:



c. connect to the DB from your local host system (your host VM)

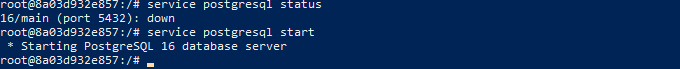
i. Note: It’s not connecting to the container using a bash shell.



It was troublesome with docker daemon when connecting to container using bash. So stopped(*docker stop myPoC*), removed(*docker rm myPoC*), recreated and started the container(docker run -it -d --name myPoC mypostgresql). Installed postgresql within the docker daemon using bash command as follows:

*apt install postgresql postgresql-client*

Then checked the status of postgresql and start it as follows:



d. created a container called myPoCHost, accessible from the Host VM.

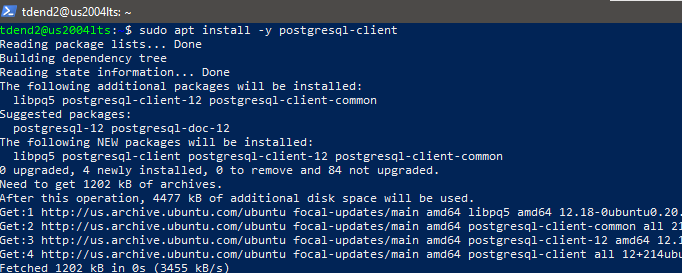
e. installed psql on your host VM.

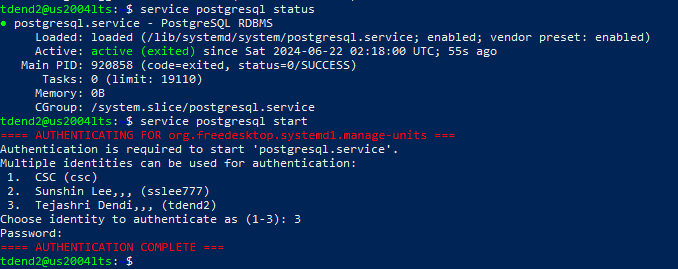
i. In short,

1. sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt

$(lsb\_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

2. sudo apt install -y postgresql-client





• Connected to the DB using psql from another container

b. Use ‘user-defined-bridge-network’

i. ‘--link’ is legacy and deprecated.https://docs.docker.com/network/links/

c. connect to the DB from another container.

d. create a container called myPoCAnother, accessible from

another host.

e. create another temporary container to test accessibility, adding an

option ‘-rm’ is recommended to delete it automatically after you exit from the

container.

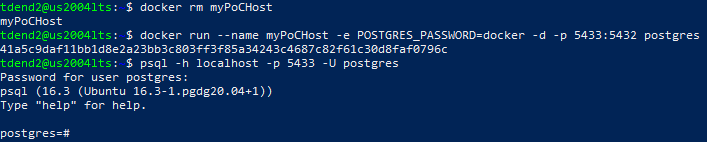
#### 

#### **2. Create and Run PostgreSQL Container**

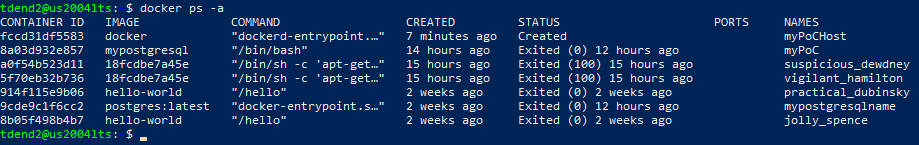
Ensure you have a running PostgreSQL container. Here's a simple example of creating a PostgreSQL container:

bash

*docker run --name myPoCHost -e POSTGRES\_PASSWORD=docker -d -p 5433:5432 postgres*



This command creates a container named *myPoCHost*, sets a password for the *postgres* user, and maps port 5432 on your host to port 5432 on the container.



As too many containers are already running on the port 5432, connection is denied to the psql server from that port. So I stopped them and removed and recreated and started the container myPocHost and then tried to connect to the psql server but the connection was refused. So created the container on other ports like 5433(myPoCHost), 5434(myPoCAnother).

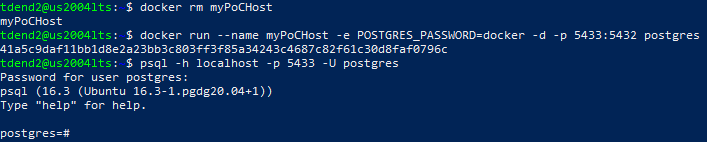
#### **3. Connect to the PostgreSQL Database from Your Host VM**

Using *psql*, connect to the PostgreSQL database running in the container:

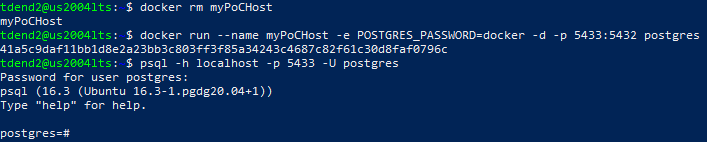
bash

*psql -h localhost -p 5433 -U postgres*

Enter the password (*docker*) when prompted.



Summary:



#### **4. Explanation**

* **Screenshot**: showing the connection from the host VM to the PostgreSQL database.
* **Explanation**: the *psql* command connects to the PostgreSQL server running in the Docker container using the -h option to specify the host (*localhost*), and *-U* to specify the user (*postgres*).

### **Default Database (*postgres*)**

* By default, when we connect without specifying a database name (*-d option), psql* will connect to the *postgres* database. This database is typically used for administrative tasks and can be used to create other databases and manage users within the PostgreSQL server.

### **Part 2: Connect to the DB Using psql from Another Container**

#### **1. Create a User-Defined Bridge Network**

Create a user-defined bridge network to allow containers to communicate with each other:

bash

docker network create *user-defined-bridge-network*



#### **2. Run PostgreSQL Container with the Network**

Run your PostgreSQL container attached to this network:

bash

*docker run --name myPoCHost --network user-defined-bridge-network -e POSTGRES\_PASSWORD=docker -d postgres*

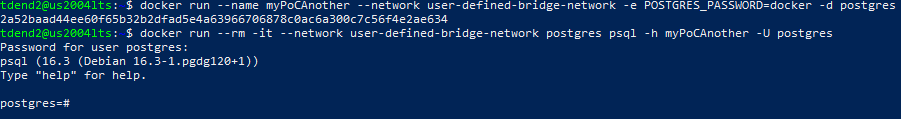


#### **3. Create and Connect from Another Container**

Ran another container and connected to the PostgreSQL container using *psql:*

*docker run --name myPoCAnother --network user-defined-bridge-network -e POSTGRES\_PASSWORD=docker -d -p 5434:5432 postgres*

Here forwarding the traffic from port 5434 to port 5432. So port 5434 is docker-proxy here.



#### **Explanation**

* **Screenshot**: showing the connection from the secondary container to the PostgreSQL database.

docker run --rm -it --network user-defined-bridge-network postgres psql -h myPoCAnother -p 5434 -U postgres

* **Explanation**: *docker run* command is used to started a new container that runs the psql client. The *--network* option connects the new container to the *user-defined-bridge-network* network, allowing it to communicate with the PostgreSQL container. The -h option specifies the host (the name of the PostgreSQL container), and -U specifies the user (*postgres*).

Summary:

Step(1)

docker network create user-defined-bridge-network

Step(2)

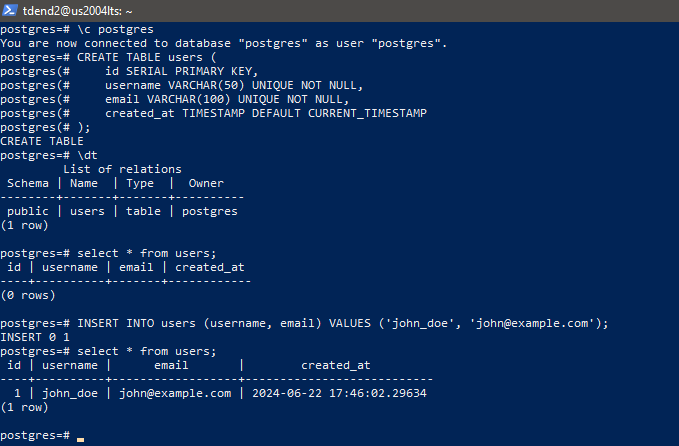
docker run --name myPoCAnother --network user-defined-bridge-network -e POSTGRES\_PASSWORD=docker -d -p 5434:5432 postgres

Step(3)

docker run --rm -it --network user-defined-bridge-network postgres bash psql -h myPoCAnother -p 5434 -U postgres

Step(4)

Connected to *postgres database* as *postgres user* as follows and performed a few operations in the database like creating a table and populating it as follows:

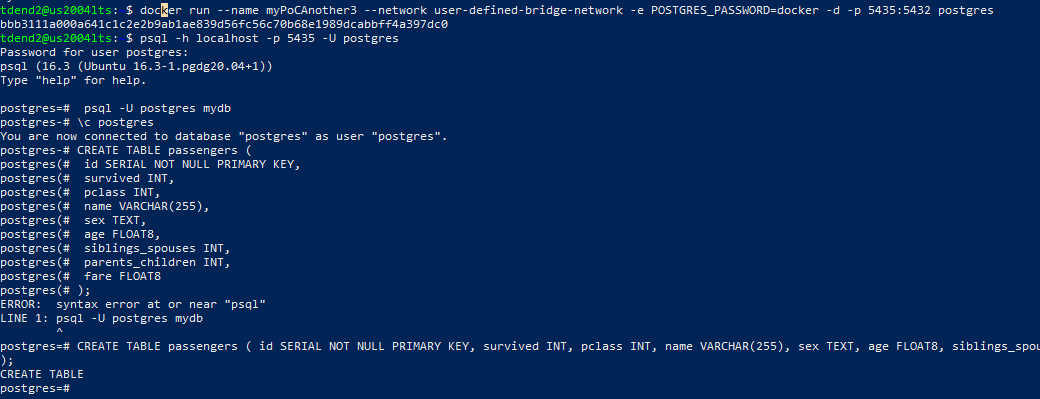


### **Default Database (*postgres*)**

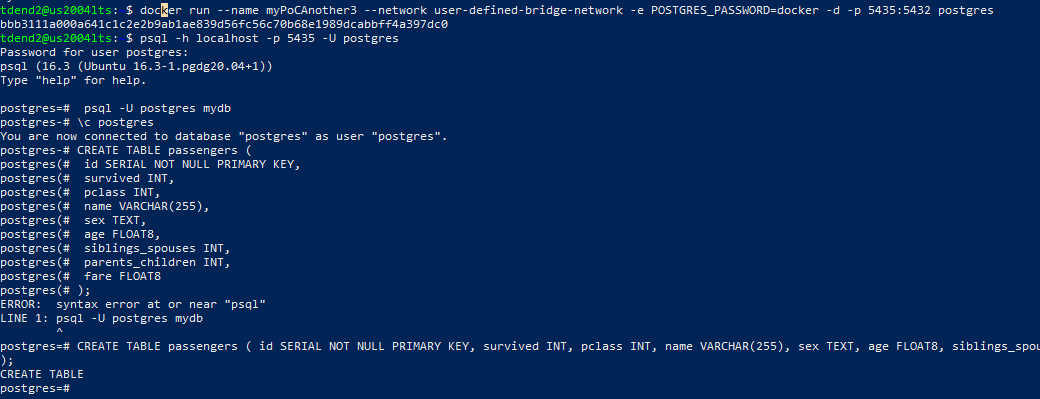
By default, when you connect without specifying a database name (-d option), *psql* will connect to the *postgres* database. This database is typically used for administrative tasks and can be used to create other databases and manage users within the PostgreSQL server.

The database here refers to the default database (*postgres*) provided by the PostgreSQL Docker image. You connect to it using the *psql* client and the appropriate credentials (*-h, -p, -U,* and password). If we intend to connect to a different database that we created within the PostgreSQL server, we would specify that database name using the -d option in *psql* command.

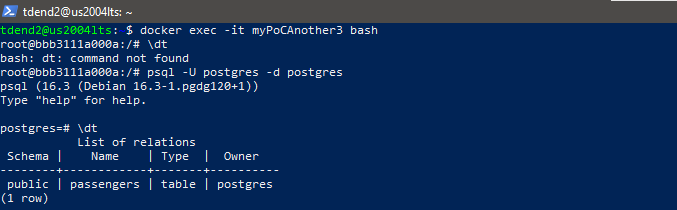
Created another container named ***mypoCAnother3*** connected to network:



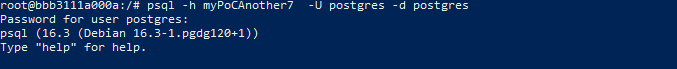
Created passenger table in the postgres database connected to psql within myPoCAnother3 container:



As mypoCAnother exited. Created another container named ***mypoCAnother7*** and accessed it from the container named ***myPoCAnother3*** as follows:Both containers are connected to common network.

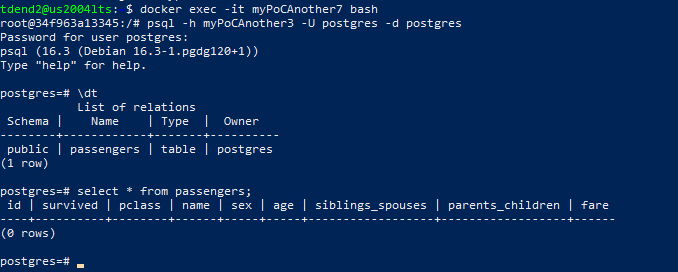


The 12 characters of container ID of myPoCAnother3 are ***bbb3111a000a***

****

**Finally, accessed postgres database from another containe**r.

Now connected to to container ***myPoCAnother7***  and then connected to another container ***myPoCAnother3***  from it and accessed passengers table avaialble in postgres database as follows:



myPocHost was not accessible from another containers as shown below:





The containers which are not created using -rm are stopped and removed as follows:

docker stop <container name>

docker rm <container name>

==========================THE END=======================