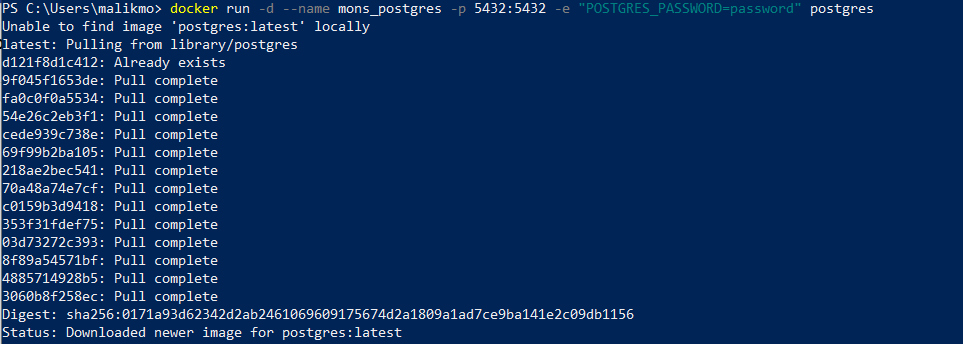
Docker: My ‘How-to’ guide

Docker and Postgres:

Create postgres database in docker container

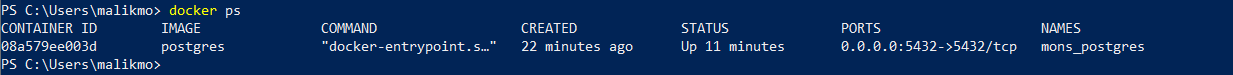
docker run -d --name mons\_postgres -p 5432:5432 -e "POSTGRES\_PASSWORD=password" postgres

If image does not exist then run will pull the image first.

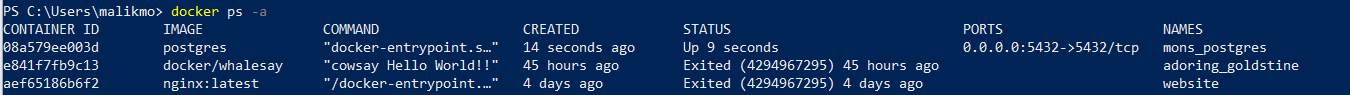


The problem with this approach is if you ever need to rebuild the container for some reason, like a new version of Postgres is released, your data will be lost.

List running processes



List all processes



Stop process



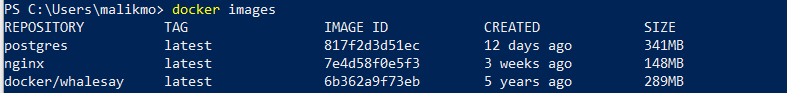
Start process



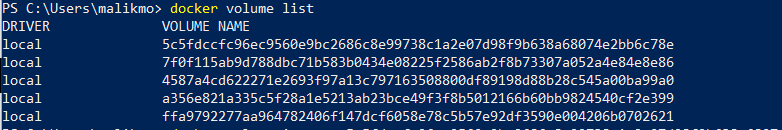
Volumes

Docker volumes are the preferred way of handling persistent data created by and used by Docker containers.

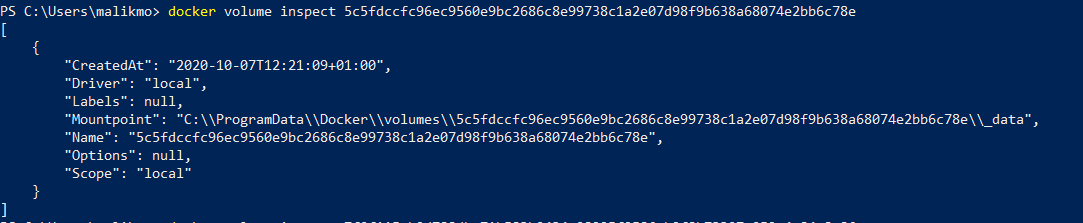
List all images



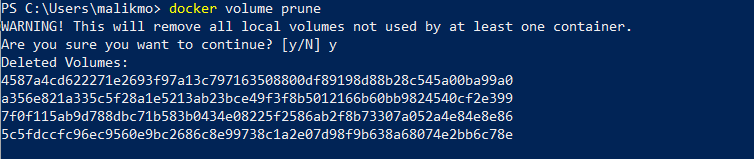
List volumes



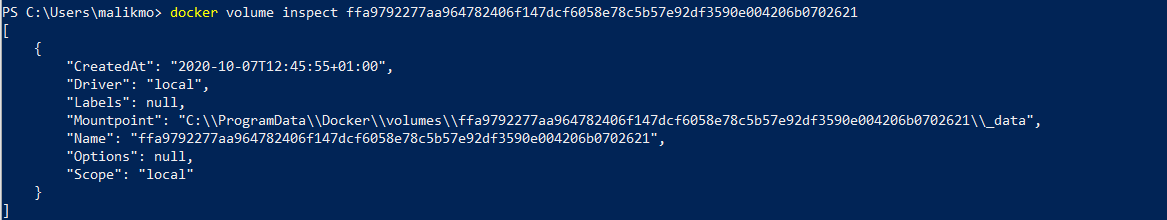
Inspect volume



Remove all volumes



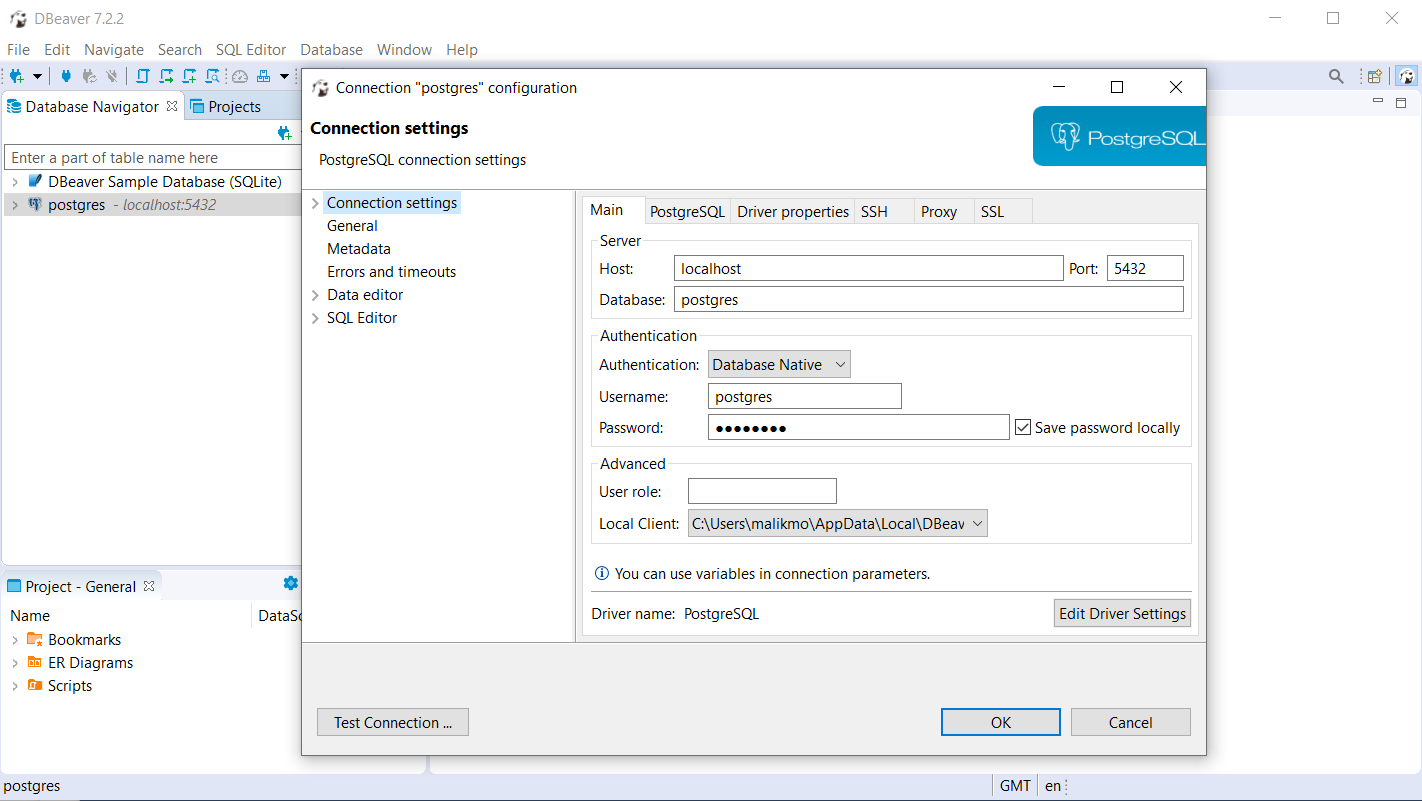


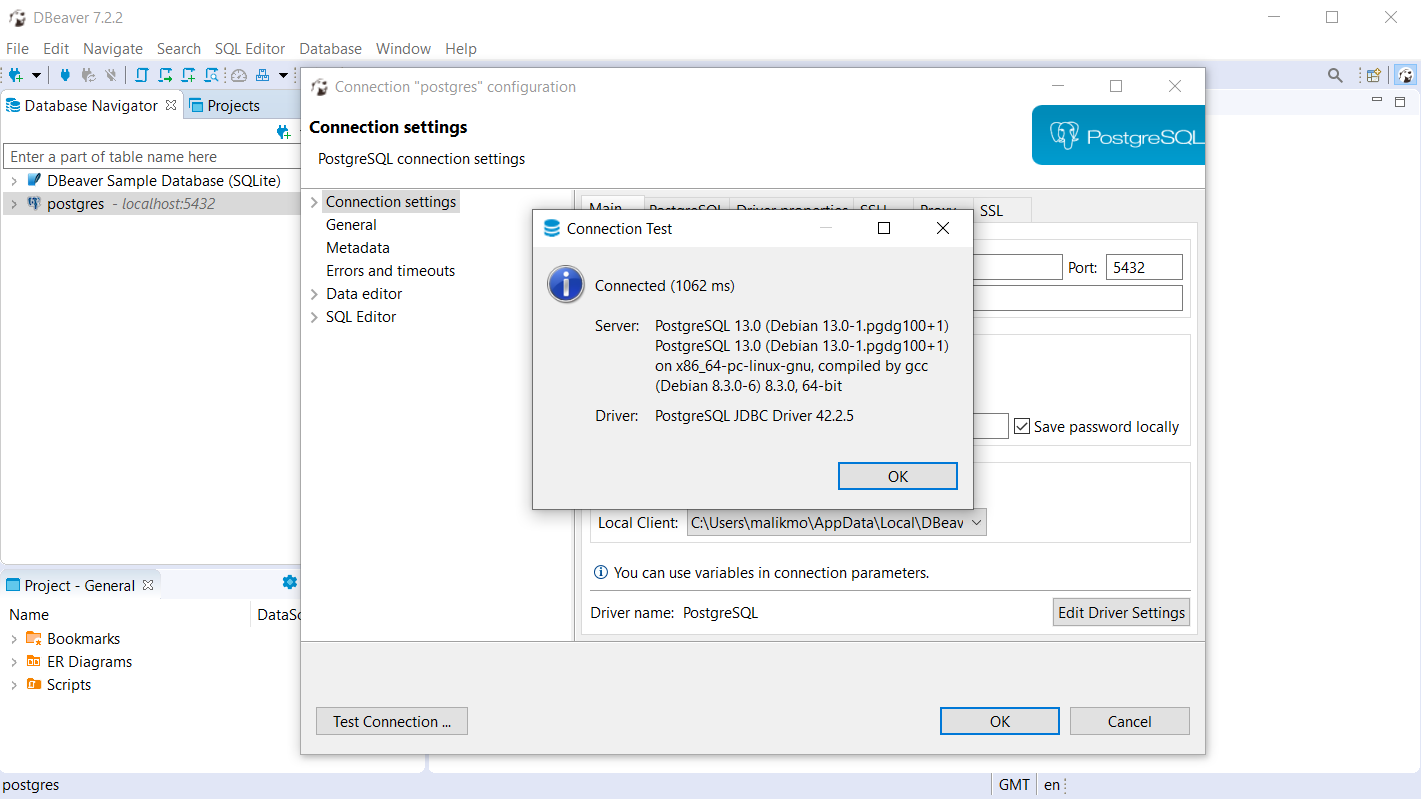


DB Tool: DBeaver

Download Community Edition from: <https://dbeaver.io/download/>

After install set up database connection





Storing Persistent Data

docker run -p 5432:5432 -d \

-e POSTGRES\_PASSWORD=postgres \

-e POSTGRES\_USER=postgres \

-e POSTGRES\_DB=dvdrental \

-v C:\temp\docker\pgdata:/var/lib/postgresql/data \

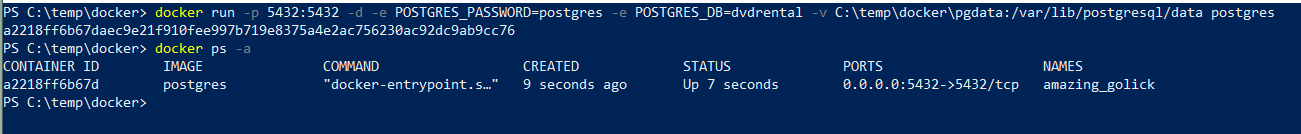
postgres

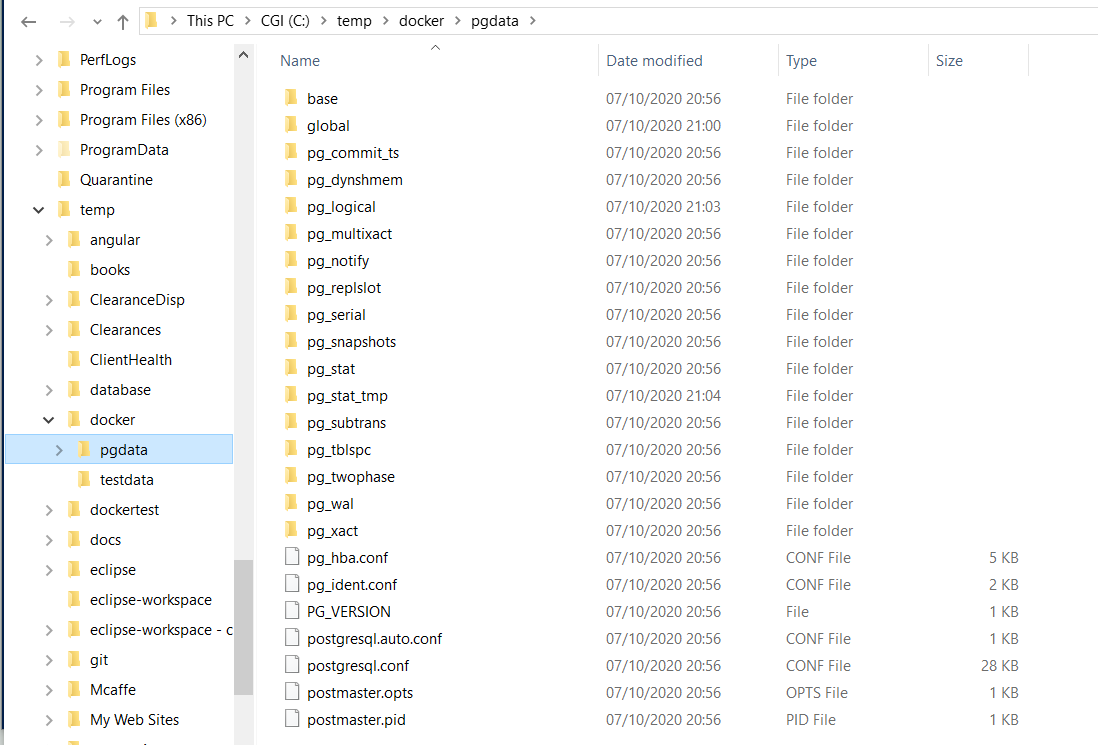
OR

docker run -p 5432:5432 -d -e POSTGRES\_PASSWORD=postgres -e POSTGRES\_DB=dvdrental -v C:\temp\docker\pgdata:/var/lib/postgresql/data postgres

psql dvdrental –h localhost –U postgres

docker exec –it <id> psql –U postgres dvdrental





Connect nodejs to postgres container

Create a temp dir

CD into temp dir

Ensure nodejs is installed: node –version

Install pg module: npm install --save pg

create app.js:

const { Client } = require('pg')

const client = new Client({

user: 'postgres',

host: 'localhost',

database: 'dvdrental',

password: 'postgres',

port: 5432

})

client.connect()

client.query('SELECT \* FROM staff', (err, res) => {

console.log(err,res)

client.end()

})

Ensure postgres container is running: docker ps

Test the app: node app.js

[Local Development Set-Up of PostgreSQL with Docker](https://towardsdatascience.com/local-development-set-up-of-postgresql-with-docker-c022632f13ea)

Start Postgres instance

Create a folder in a known location for your persistent data  
$ mkdir ${HOME}/postgres-data/

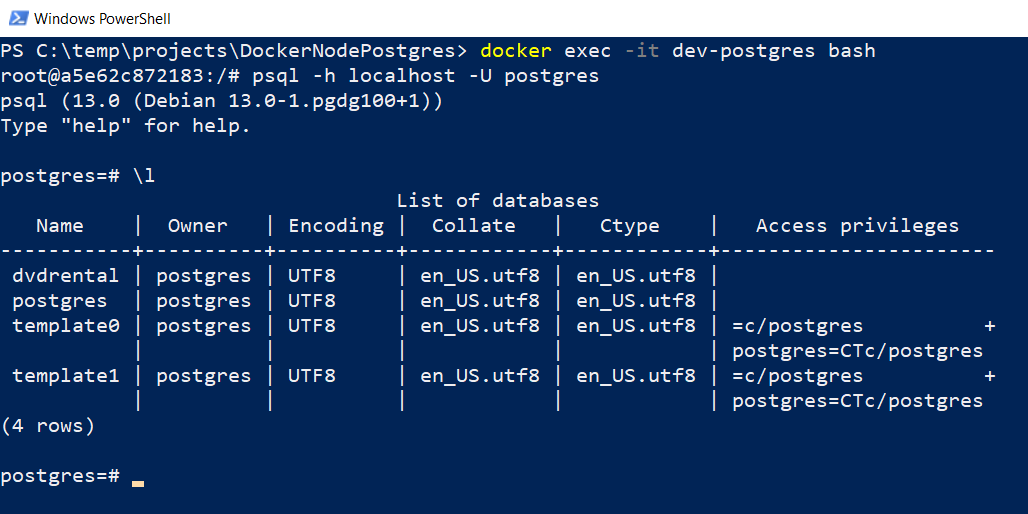
Set up postgres container:

docker run -p 5432:5432 -d -e POSTGRES\_PASSWORD=postgres -e POSTGRES\_DB=dvdrental -v C:\temp\docker\pgdata:/var/lib/postgresql/data --name dev-postgres postgres

Check container is running: docker ps

Container is now running, enter container from command line:

docker exec -it dev-postgres bash



Start pgAdmin instance

Docker pull dpage/pgadmin4

docker run -p 80:81 -e 'PGADMIN\_DEFAULT\_EMAIL=user@domain.local' -e 'PGADMIN\_DEFAULT\_PASSWORD=password' --name dev-pgadmin -d dpage/pgadmin4

The parameters that we are passing to the docker run command are:

* -p 80:80: This parameter tells docker to map the port 80 in the container to port 80 in your computer (Docker host)
* -e 'PGADMIN\_DEFAULT\_EMAIL: Environment variable for default user’s email, you will use this to log in the portal afterwards
* -e 'PGADMIN\_DEFAULT\_PASSWORD': Environment variable for default user’s password
* -d: This parameters tells docker to start the container in detached mode
* dpage/pgadmin4: This parameter tells docker to use the image that we have previously downloaded

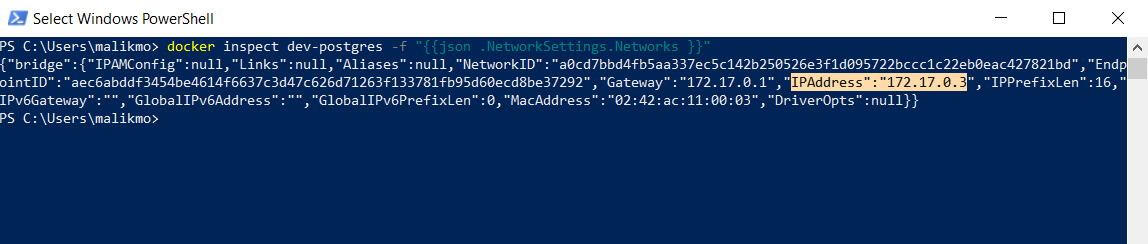
Accessing the PostgreSQL from the pgAdmin tool

We haven’t defined any network for these containers so they should be running on the default one, and if you try to access the database or the web portal through their ports, connecting via ‘localhost’ or ‘127.0.0.1’ would work just fine; but if you try connecting from one container to the other, you might encounter some connectivity issues.

We will need to look for the IP address of the PostgreSQL container on our host, you can run this command for it:

$ docker inspect dev-postgres -f "{{json .NetworkSettings.Networks }}"

docker inspect return low-level information of Docker objects, in this case, the ‘**dev-postgres**’ instance’s IP Adress. The -f parameter is to format the output as a JSON given a Go template. The output should look like this:



Copy the IPAddress value into the clipboard, which is 172.17.0.3 in my case, you will need to define the connection in the pgAdmin tool.

The next step is to go to your web browser and type [http://localhost:80](http://localhost/).



Once you are in the portal, you will need to add a new server by clicking on the “**Add New Server**” and adding the right information on the pop-up window, make sure you add the IPAdress that you copied previously in the *Host name/address* under the Connection tab.

Terms:

Dockerfile

A [Dockerfile](https://docs.docker.com/engine/reference/builder/) is a file with instructions for how Docker should build your image.

The Dockerfile refers to a base image that is used to build the initial image layer. Popular official base images include [python](https://hub.docker.com/_/python/), [ubuntu](https://hub.docker.com/_/ubuntu), and [alpine](https://hub.docker.com/_/alpine).

Additional layers can then be stacked on top of the base image layers, according to the instructions in the Dockerfile.

Finally, a thin, writable layer is stacked on top of the other layers according to the Dockerfile code.

A Dozen Dockerfile Instructions

FROM — specifies the base (parent) image.  
LABEL —provides metadata. Good place to include maintainer info.  
ENV — sets a persistent environment variable.  
RUN —runs a command and creates an image layer. Used to install packages into containers.  
COPY — copies files and directories to the container.  
ADD — copies files and directories to the container. Can upack local .tar files.  
CMD — provides a command and arguments for an executing container. Parameters can be overridden. There can be only one CMD.  
WORKDIR — sets the working directory for the instructions that follow.  
ARG — defines a variable to pass to Docker at build-time.  
ENTRYPOINT — provides command and arguments for an executing container. Arguments persist.   
EXPOSE — exposes a port.  
VOLUME — creates a directory mount point to access and store persistent data.

Docker Container

A Docker image plus the command ‘docker run image\_name’ creates and starts a container from an image.

Here’s the one line explanation to help you keep these dozen terms straight.

Basics

*Platform* — the software that makes Docker containers possible  
*Engine* — client-server app (CE or Enterprise)  
*Client* — handles Docker CLI so you can communicate with the Daemon  
*Daemon*— Docker server that manages key things  
*Volumes* — persistent data storage  
*Registry* — remote image storage  
*Docker Hub*— default and largest Docker Registry  
*Repository* — collection of Docker images, e.g. Alpine

Scaling

*Networking* — connect containers together  
*Compose* — time saver for multi-container apps  
*Swarm* — orchestrates container deployment  
*Services* — containers in production

Because we’re keeping with food metaphors, and everyone loves a baker’s dozen, we have one more related term for you: *Kubernetes*.

[**Kubernetes**](https://kubernetes.io/)automates deployment, scaling, and management of containerized applications. It is the clear winner in the container orchestration market. Instead of Docker Swarm, use Kubernetes to scale up projects with multiple Docker containers. Kubernetes is not an official part of Docker; it is more like Docker’s BFF.

[Use Docker to Create a Node Development Environment](https://auth0.com/blog/use-docker-to-create-a-node-development-environment/)

* Run ‘npm install yarn -g’ to install yarn globally

Creating a nodejs with express app with docker

* [How to Build a Node.js Application with Docker](https://www.digitalocean.com/community/tutorials/how-to-build-a-node-js-application-with-docker)

[How to Connect Docker Containers](https://www.linode.com/docs/applications/containers/docker-container-communication/)

