link:

<https://www.optimadata.nl/blogs/1/n8dyr5-how-to-run-postgres-on-docker-part-1>

**How to run Postgres on Docker part 1**

Craig Healey 12-9-2019 14:57

Categorieën: Blog, PostgreSQL



As a multi-platform DBA, I spend a lot of time learning about the latest features available on a variety of open-source and proprietary databases. I usually use VirtualBox images to create my sandbox databases, but they can take up quite a bit a disk space and resources. Especially if I’m playing around with things like database clustering solutions, when I need multiple databases running together. So the other day, I thought about using Docker containers.

Docker is named after the containers that you see on ships. It ensures standardization, such as in the transport sector, by choosing a single form of transport. Docker is an open source framework that makes it possible to package an application and/or database in a lightweight, portable container. This makes installing an application on a server as easy as installing a mobile app on your tablet or smartphone.

All the required components are built into a container. By using containers, resources can be isolated, access to services restricted and processes assigned, so that you can indicate exactly how much of, for example, CPU, memory or I / O load can be used by a container.

In this blog I explain how you can create a PostgreSQL Docker container in 7 steps.

**How to start?**

There are dozens of databases available on Docker’s Hub (a cross between GitHub and Google Play Store), Oracle, SQL Server, Postgres, MariaDB, MySQL, Mongo and more. And you can get them in a variety of versions, from the latest beta to legacy versions. Plus, containers are part of the whole [Microservices Architecture](https://www.intertech.com/Blog/top-devops-trends-in-2019/) that DevOps keep going on about, and inevitably lead to talking about [Kubernetes](https://dzone.com/articles/6-devops-trends-to-watch-in-2019).

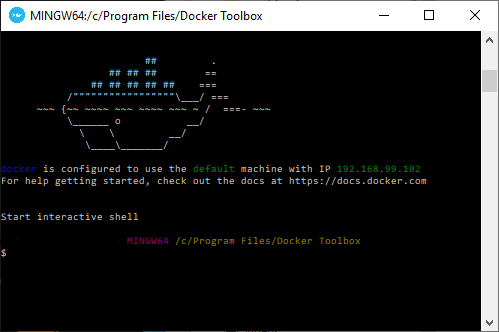
Who doesn’t want to learn a whole different technology before they can get down to playing with a brand-new database feature? OK, maybe not the whole Docker, Docker Compose, Kubernetes stack. But Docker itself is relatively straightforward. Let me demonstrate by spinning up a few PostgreSQL containers on a windows laptop.

**STEP 1: Download Docker**

First you want to download Docker. You have a choice here of the latest Docker Desktop for Windows, which requires you to have a free Docker Hub account and disables VirtualBox in order to run, or the older Docker Toolbox. I don’t want to mess with my existing VirtualBox images, so let’s go with Docker Toolbox.

Download the exe from <https://github.com/docker/toolbox/releases> and run it. The default installation includes an old version of VirtualBox, as well as Git for Windows. If you have a new version of VirtualBox, skip it. On my system the installer detects Git but not VirtualBox. When it finishes you can view the 2 shortcuts it creates – Docker Quickstart Terminal (Command Line) and Kitematic (GUI).

**STEP 2: Run the Docker Quickstart Terminal**

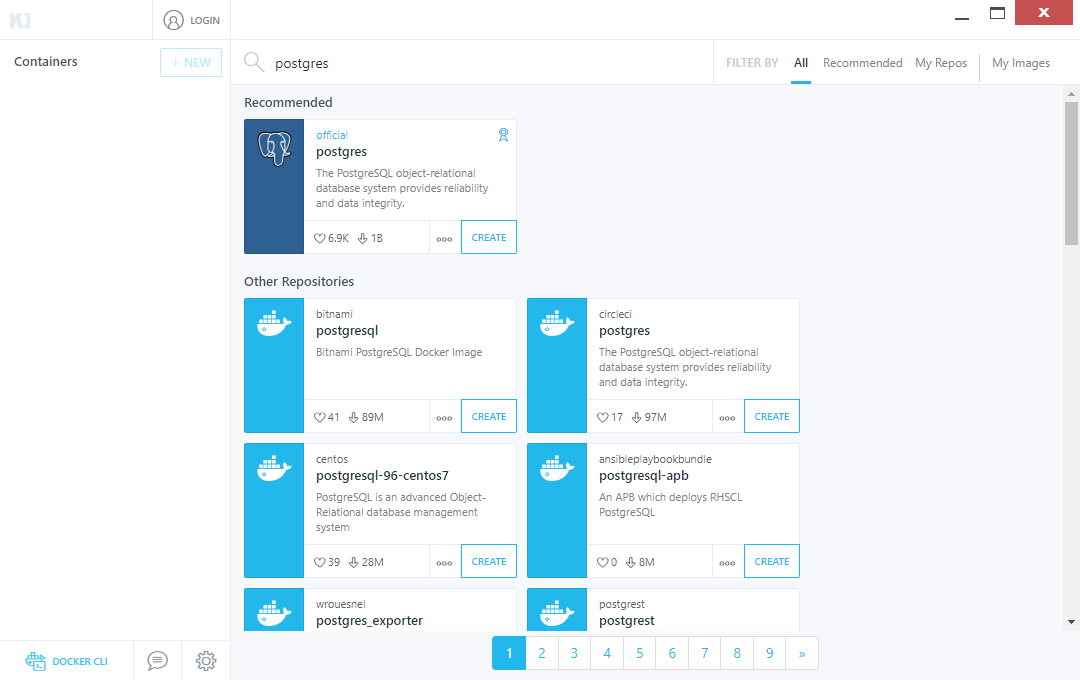
If you have VirtualBox open you’ll notice that it has created a new machine called default. On Windows, at least in the toolbox version, everything is run inside this VirtualBox machine.  After a while the initial setup finishes and you should see an ASCII image of a [whale](https://forums.docker.com/t/where-does-the-idea-of-a-whale-on-the-logo-come-from/10874), a note telling you about the default machine and its IP address, and an interactive shell. Note the IP address, it will be useful later.

**STEP 3: My first container**

So now you have Docker running, you can also run the GUI. Click on Kitematic, and if it says, “We couldn’t find a native setup…”, choose “use VirtualBox”. Then skip the “Connect to Docker Hub” screen and you’re presented with various recommended Docker images. You can search all the images available on Docker Hub. There are over a dozen categories, such as Operating Systems, DevOps Tools, Databases and Programming Languages. They even have media servers and twitter clients.

Both the Terminal and Kitematic check to see if there is a default VirtualBox machine available, and if there isn’t, they create one. So, you can delete the VirtualBox machine and start over. But be careful, because all of your container information is on that machine. Not a problem if you’re just messing around, but if you’ve spent time building numerous containers you might want to have a snapshot of your machine.

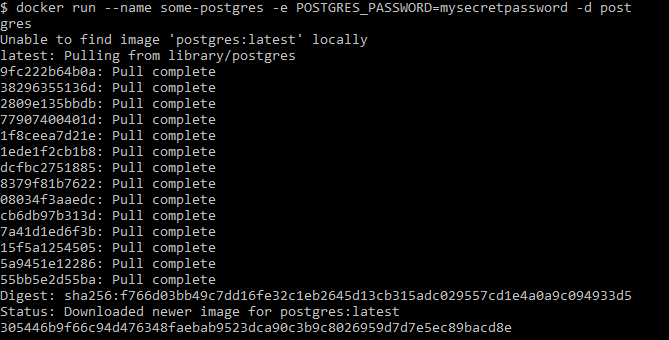
**STEP 4: Select PostgreSQL image**



In the search box at the top of Kitematic, type postgres. You’ll see lots of images available. We are going to use the first one, which is the official postgres image. Click on the three dots to the left of “create” and select “view on Docker Hub”. This opens a browser window. If you scroll down to the “How to use this image” section you’ll see

docker run --name some-postgres -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

Type (or copy and paste) this into the Terminal window, being careful to use two minus signs before “name”, then hit return. It will say “unable to find image ‘postgres:latest’ locally”, and it will go on to download and extract numerous files. When it’s finished it should return a sha256 line, a Status line and a hash.



**STEP 5: Test your container**

Congratulations, you’ve created your first container!

However, in its current state, it’s pretty useless. We need to be able to access the container in order to run SQL commands from psql. But first, let’s check it’s working. Type

docker ps

This will return a container ID (the first 12 characters from the hash), the image name (in this case, postgres), command, created, status, ports and the name of the container (some-postgres).

C:\Users\adriaan\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D3A673A.tmp

**STEP 6: Connect via EXEC**

So, now we need to actually connect to the container, in order to do some work. Type the following

docker exec -it some-postgres bash

C:\Users\adriaan\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\83706358.tmp

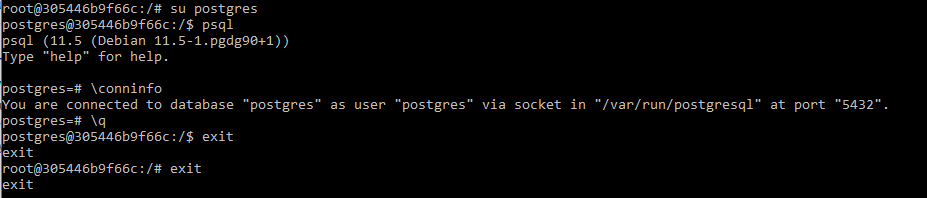
Now we have root access to the container. Notice the container ID in the command prompt. To access postgres you need to change to user ‘postgres’ and then run psql. To exit psql, type \q

su postgres

psql

\conninfo

\q



Type exit once more, and you leave the interactive session you were in and return to the Docker Terminal. To check that the container is still running, and you have indeed only exited an interactive session, type

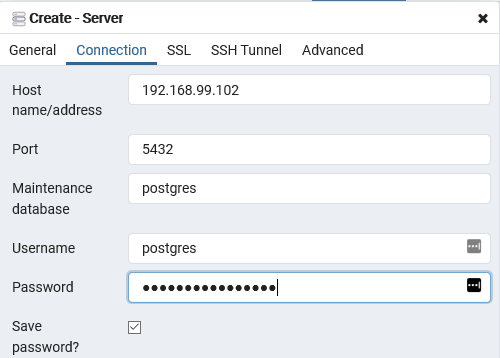
docker ps

**STEP 7: GUI PgAdmin**

You’ll see that the status of your some-postgres container is still up.

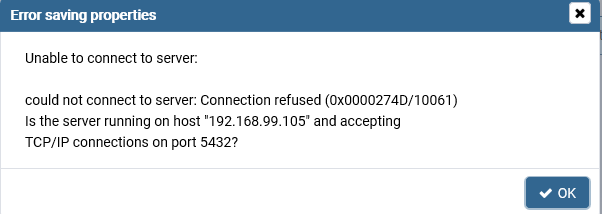
That’s the postgreSQL command line, but what about using a GUI tool such as pgAdmin? If you don’t have a copy, you can download it from <https://www.pgadmin.org/>

Run it and create a new server. Give it a name in the general tab and click on the “Connection” tab.

Fill in the Host name/address with the default machine’s IP address from when you first ran Terminal. Note that this means your container is connecting through the VirtualBox machine, rather than directly. Fill in the password as mysecretpassword (which is what it was set to in the command line when you created the container).

Select save password and click save.

And it doesn’t work!

There is one flag missing from the container creation command, and it’s quite important for getting pgAdmin to work. It’s the port command, and it maps a container’s internal and external port. But first, we need to clean up the first container.

rm

If you try to re-run the original command we used to create the container it will give you an error saying the container name is still in use. So, to remove the container, first stop the container

docker stop some-postgres

Then use

docker rm some-postgres

You can also use the -rm flag when you create a container. That way, when you finish with it, it isn’t saved. If you want to get rid of everything, you can use

docker system prune

Now, re-run the original docker run command, but this time with -p 5432:5432

docker run --name some-postgres -p 5432:5432 -e

POSTGRES\_PASSWORD=mysecretpassword -d postgres

Type

docker ps

C:\Users\adriaan\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\BFDCA93E.tmpto check it’s running. You’ll notice that this time, as well as a different container ID, the ports are slightly different.

Now, when you create a new server in pgAdmin, it should find it without a problem.

**CONGRATULATIONS!**

If you have taken all these steps then you have a complete server environment with PostgreSQL, packed in a container that you can use anywhere and with anything.

In part two, I’ll explain some of the common Docker commands, and look at typical actions in Docker such as creating images, networks and volumes.