

Setup PostgreSQL on Windows with Docker

 medium.com/@elanderson/setup-postgresql-on-windows-with-docker-15388766586f

Eric Anderson

February 18, 2018

Over the weekend I finally got the chance to start reading [A Curious Moon](#) by [Rob Conery](#) which is a book on learning [PostgreSQL](#) by following the fictional Dee Yan as she is thrown into database administrator role at an aerospace startup.

I have a lot of experience using Microsoft's SQL Server, but up until now, I haven't touched PostgreSQL. For personal projects SQL Server's cost and be prohibitive and the release of Rob's book added up to a good time to give PostgreSQL a try.

Install Directly or not?

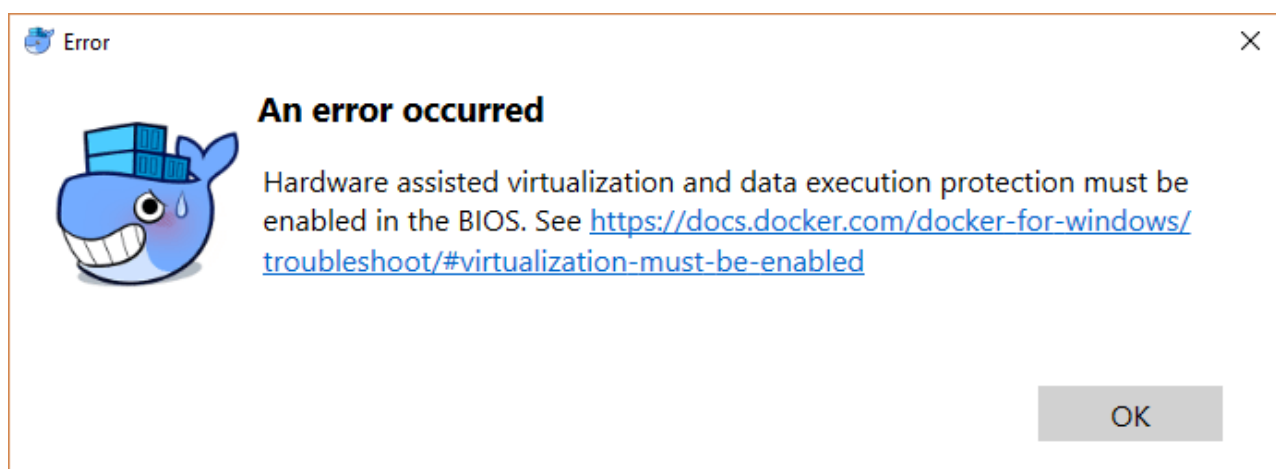
On the download section of the official Postgres site, there is an option to download an installer. This is the route I was going to at first, but in Rob's book, he suggests using a VM for Postgres installation on Windows. This kicked off a lot of searching on my part and didn't find a good definitive answer on why that is or isn't the way to do.

In the end, I decided to try and run the Postgres process using Docker instead installing directly on Windows or dealing with a full VM.

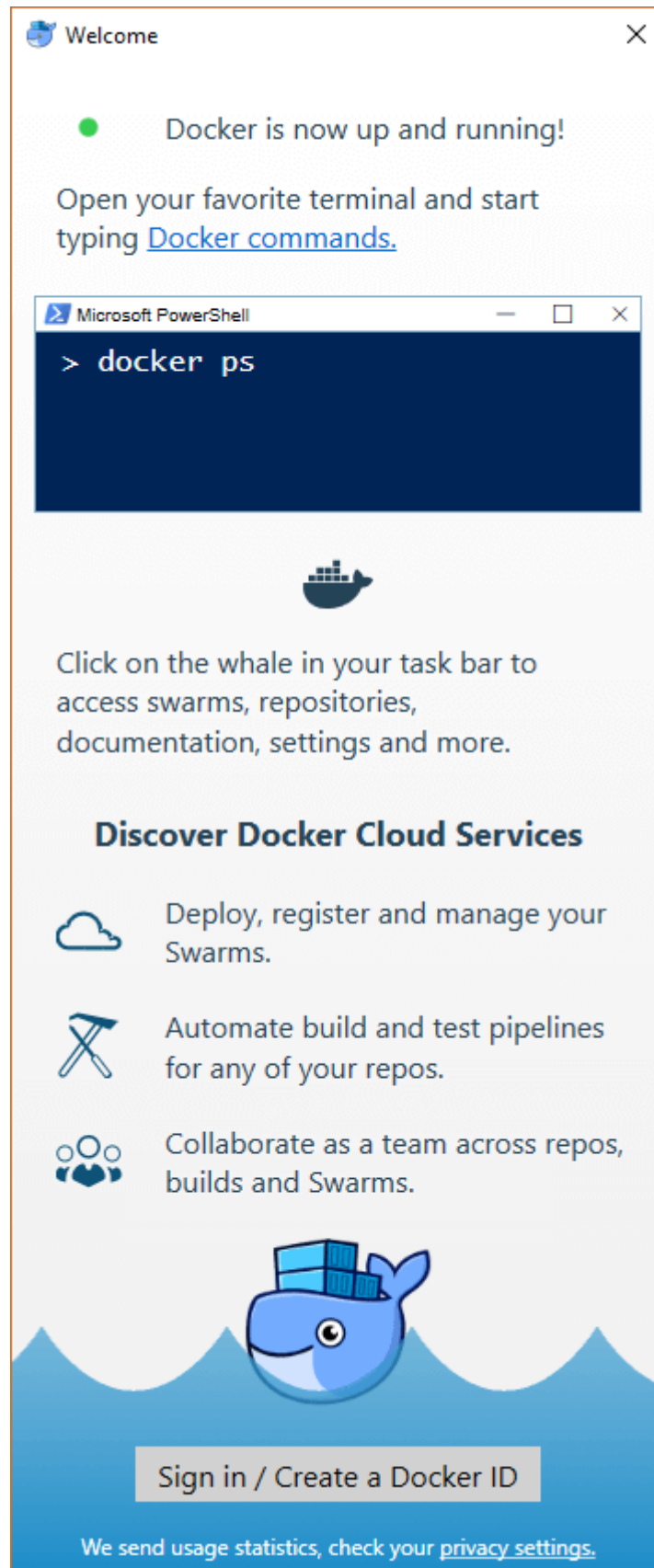
Installing Docker

Head to [this](#) link and click the Get Docker link to download the installer. After the install is complete you will have to log out and back in. When I logged back in I got a message about Hyper-V not being enabled.

After logging back in I then got the following message about hardware-assisted virtualization not being enabled.



After tweaking my BIOS settings and logging back in I was greeted by the Docker welcome screen.



Open a command prompt and run the following command.

```
docker run hello-world
```

You should output that starts with the following if your installation is working.

Hello from Docker!

This message shows that your installation appears to be working correctly.

What about Postgres?

Getting up and going with a container running Postgres was pretty simple and could be done with the following command which will create a container and expose the port used by Postgres so it can be accessed from the host.

```
docker run -p 5432:5432 --name yourContainerName -e POSTGRES_PASSWORD=yourPassword -d postgres
```

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. Thankfully I found [this](#) blog post which shows how to use a secondary container for the data leaving the Postgres container able to be destroyed and recreated as needed. The following is the command I used to create my data container.

```
docker create -v /var/lib/postgresql/data --name PostgresData alpine
```

The above creates a container named PostgresData based on the Alpine image. It is important that the -v parameter matches the path that Postgres expects.

Now that we have a container that will keep our data safe let's create the actual Postgres container with the following command.

```
docker run -p 5432:5432 --name yourContainerName -e POSTGRES_PASSWORD=yourPassword -d --volumes-from PostgresData postgres
```

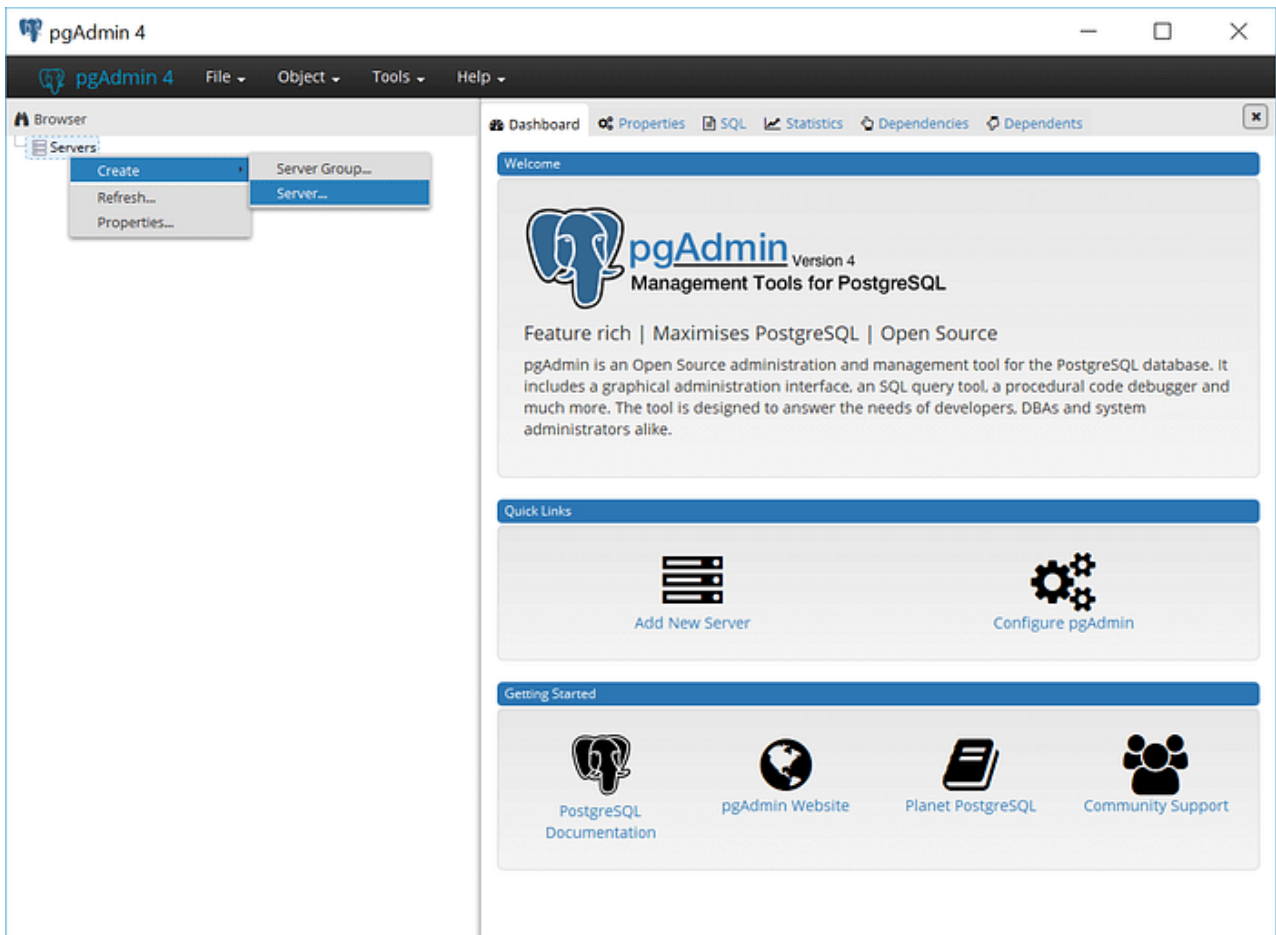
The only difference from the first example run command is the addition of `--volumes-from PostgresData` which tells the container to use the PostgresData container.

If you run the `docker ps -a` command it will show you all your containers.

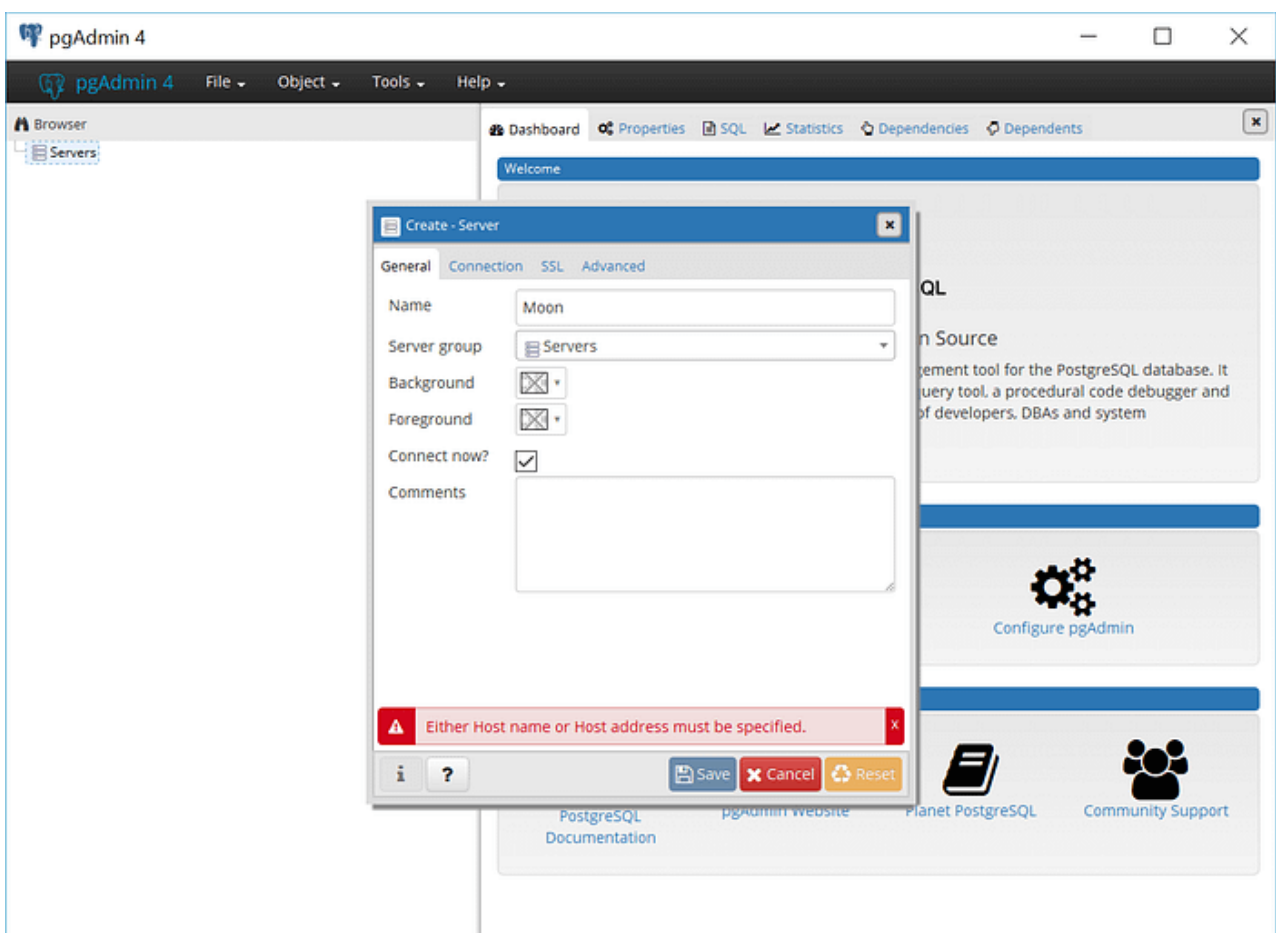
As you can see in my example I have two containers only one of which is actually running. Make sure you don't remove the data container just because it will never show as running.

Connect to Postgres

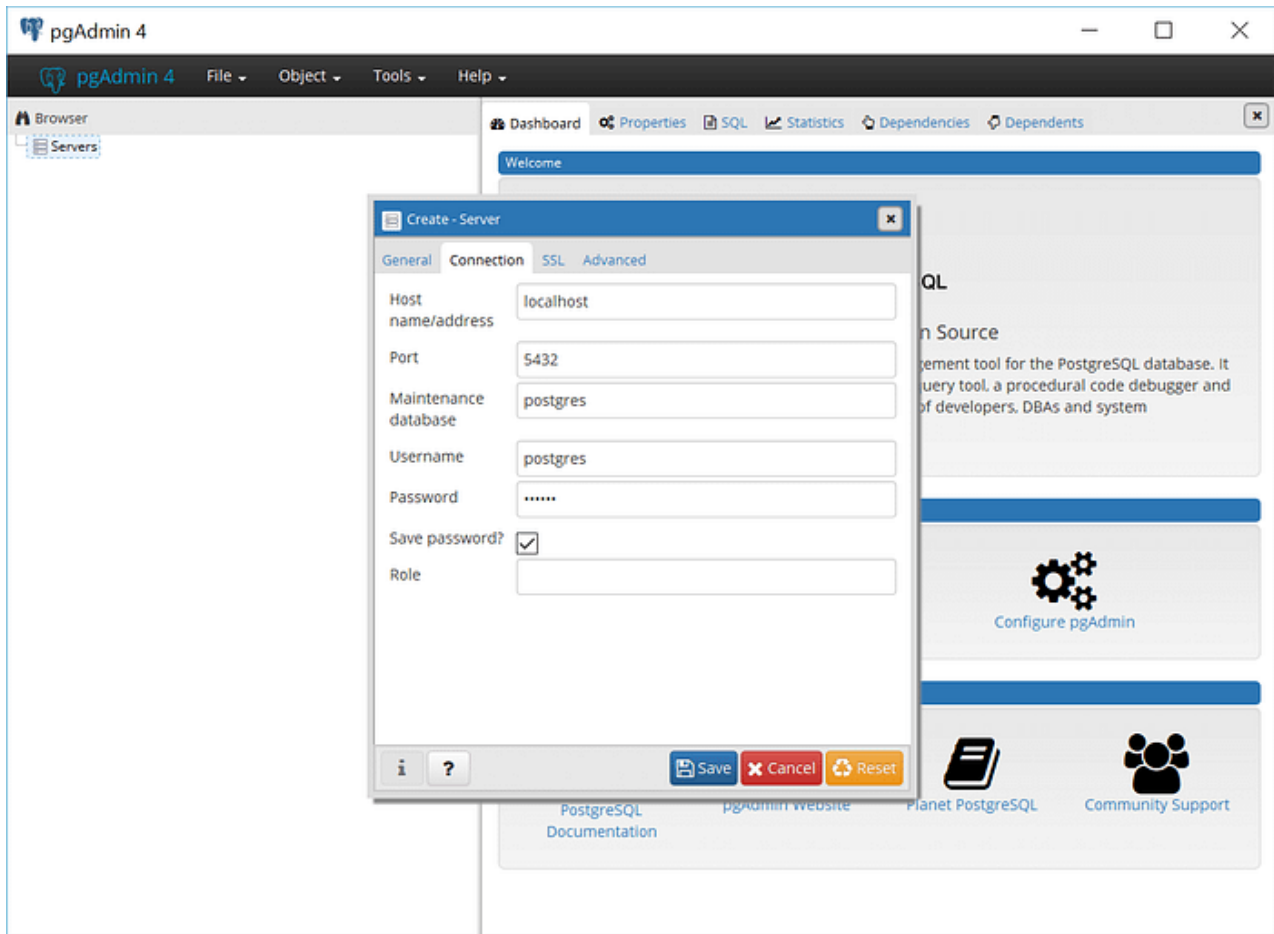
To verify all was working I downloaded pgAdmin from [here](#). Run the installer and then open the application. Right-click on Server and click Create > Server.



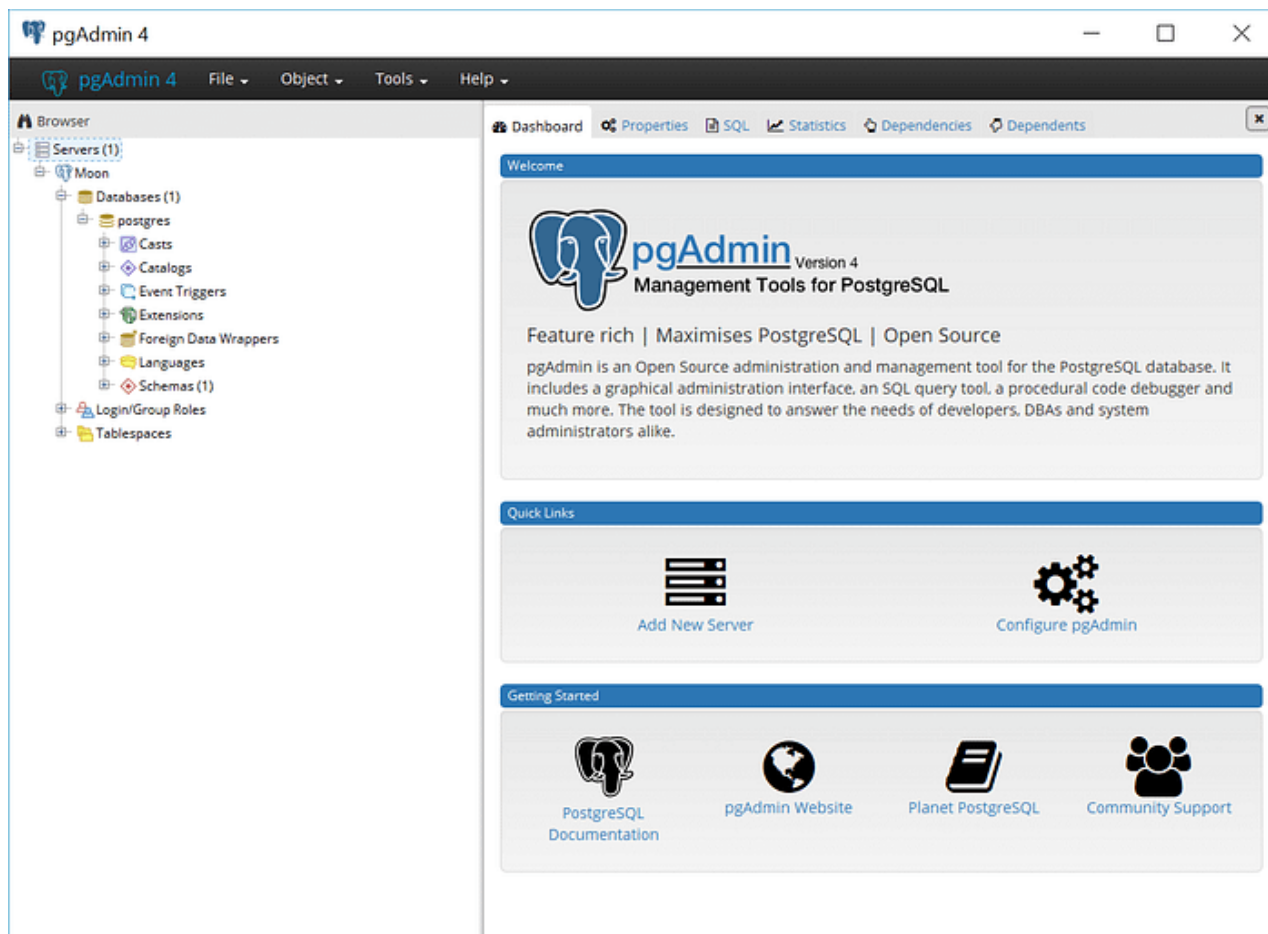
On the Create Server dialog enter a Name for your server and then switch over to the Connection tab.



On the Connection tab for Host use localhost and in the Password field use the password you used for POSTGRES_PASSWORD on the docker run command.



Click Save to close the dialog and connect to the server. The following is an example screenshot of what you will see showing the available databases on the server, which is just the default database in this case.



Wrapping Up

Make sure to check out the official docs [here](#) for more information as needed.

Other than the storage portion getting Postgres up and running in Docker was pretty simple. I hope like me this will give you a good jumping off point to learn more about both Docker and Postgres.

If anyone has any alternate ways to deal with persistent storage please leave a comment.
Originally published at [Eric L. Anderson](#).