PostgreSQL & pgAdmin Installation Guide

Guide for downloading and installing PostgreSQL and pgAdmin using Docker
for the "Python Web Basics Course @ SoftUni"

Introduction to Docker

Docker is an open platform for developing, shipping and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

What is a Docker Image?

An image is a read-only template (file) that acts as a set of instructions to create a Docker container.

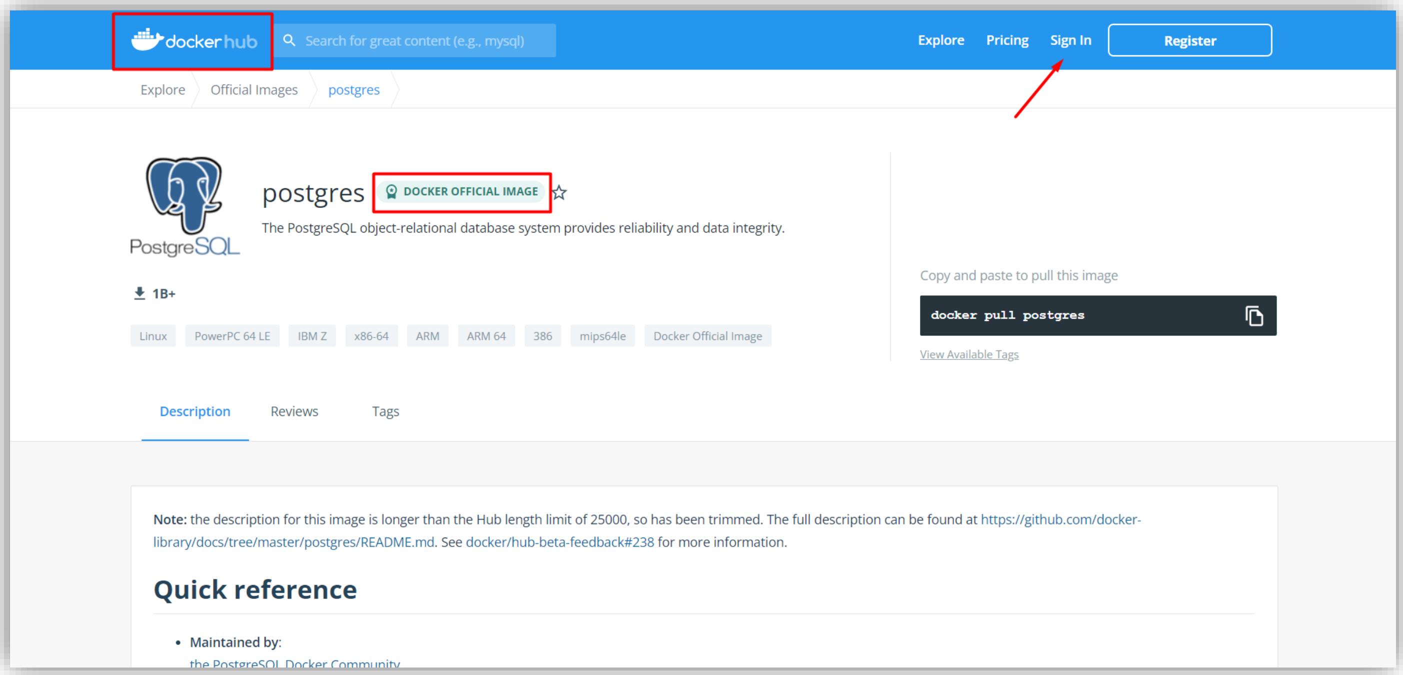
What is a Docker Container?

A container is a runnable instance of an image. It is a way to package applications with everything they need inside of the package, including the dependencies and all the configurations necessary. In other words, a container is portable, and everything in it is packaged in one isolated environment.

All containers are stored in a container repository, and all public containers can be found at DockerHub.

Running PostgreSQL container with Docker

First, let's first look at the Docker PostgreSQL image on DockerHub. Here is a link to that image in DockerHub: https://hub.docker.com/_/postgres. As you see, it is an official image, and because it is open-source and free to use, you do **NOT need to sign in** to use it:



The PostgreSQL object-relational database system provides reliability and data integrity.

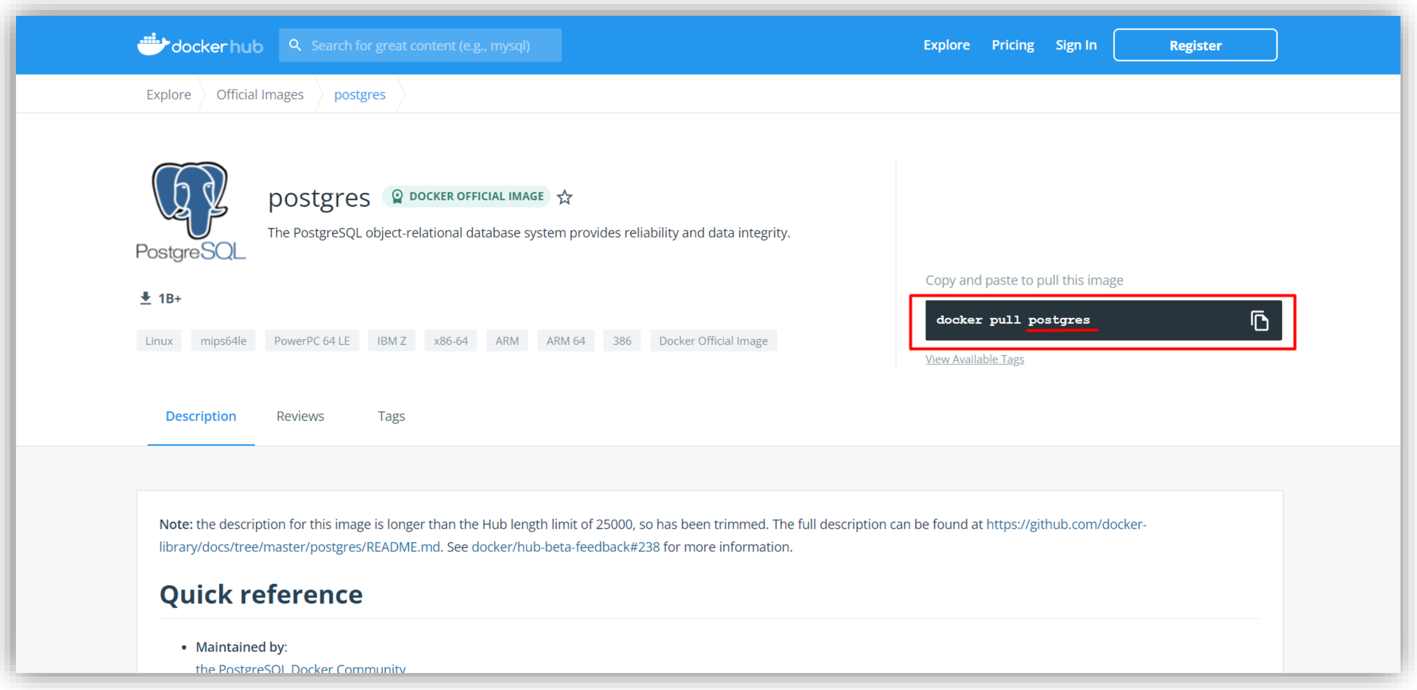
```
docker pull postgres
```

Note: the description for this image is longer than the Hub length limit of 25000, so has been trimmed. The full description can be found at https://github.com/docker-library/docs/tree/master/postgres/README.md. See docker/hub-beta-feedback#238 for more information.

Quick reference

- Maintained by:
  the PostgreSQL Docker Community

On the right side of the screen can be found a generated command to pull (download) the image. We can see that the **PostgreSQL image is named "postgres"** and that is one that we should use when creating a container:



postgres

The PostgreSQL object-relational database system provides reliability and data integrity.

```
docker pull postgres
```

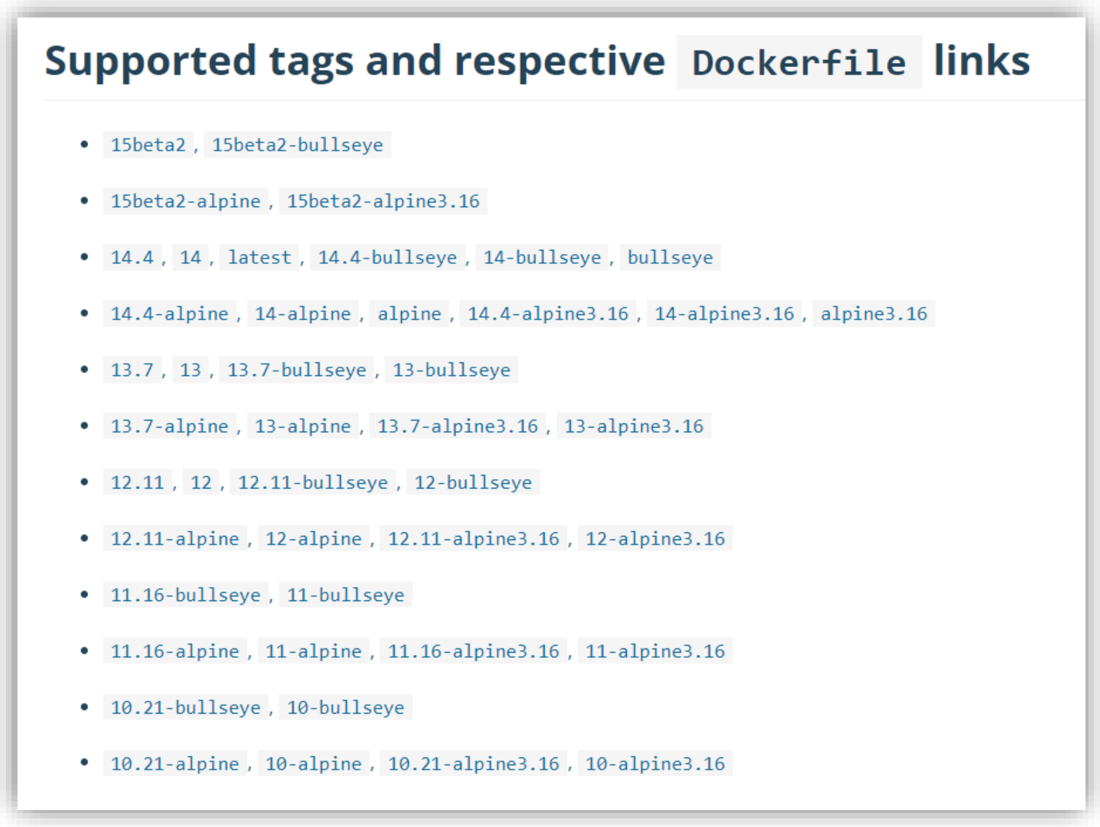
Note: the description for this image is longer than the Hub length limit of 25000, so has been trimmed. The full description can be found at https://github.com/docker-library/docs/tree/master/postgres/README.md. See docker/hub-beta-feedback#238 for more information.

Quick reference

- Maintained by:
  the PostgreSQL Docker Community

When you scroll down, you can see **all the versions of PostgreSQL,** which are available on DockerHub. If you don't want to use the latest version, you could choose to specify it when pulling an image using the syntax

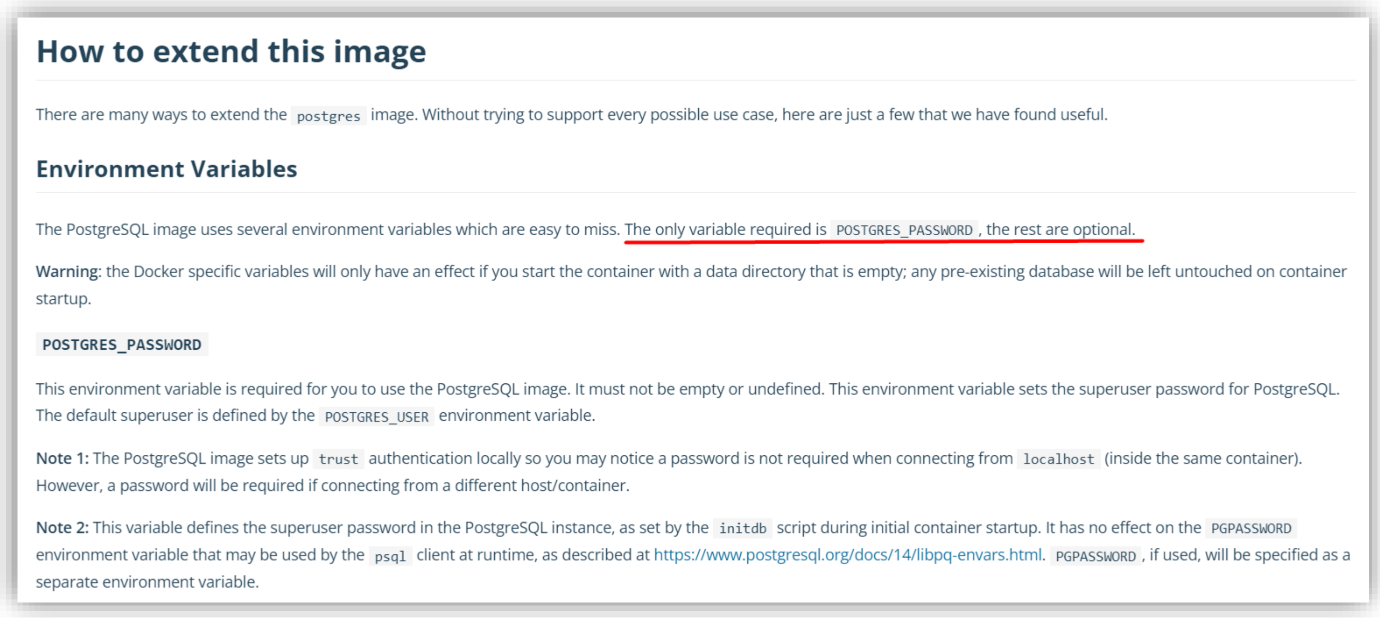
"postgres:{version}" (e.g., "postgres:12.11"):



Supported tags and respective `Dockerfile` links

- `15beta2`, `15beta2-bullseye`
- `15beta2-alpine`, `15beta2-alpine3.16`
- `14.4`, `14`, `latest`, `14.4-bullseye`, `14-bullseye`, `bullseye`
- `14.4-alpine`, `14-alpine`, `alpine`, `14.4-alpine3.16`, `14-alpine3.16`, `alpine3.16`
- `13.7`, `13`, `13.7-bullseye`, `13-bullseye`
- `13.7-alpine`, `13-alpine`, `13.7-alpine3.16`, `13-alpine3.16`
- `12.11`, `12`, `12.11-bullseye`, `12-bullseye`
- `12.11-alpine`, `12-alpine`, `12.11-alpine3.16`, `12-alpine3.16`
- `11.16-bullseye`, `11-bullseye`
- `11.16-alpine`, `11-alpine`, `11.16-alpine3.16`, `11-alpine3.16`
- `10.21-bullseye`, `10-bullseye`
- `10.21-alpine`, `10-alpine`, `10.21-alpine3.16`, `10-alpine3.16`

Below, in the section "How to extend this image", you can see its **"Environment Variables"**. In Docker, variables are used to specify different attributes in a container. PostgreSQL has one **required** variable - **POSTGRES_PASSWORD**:



How to extend this image

There are many ways to extend the `postgres` image. Without trying to support every possible use case, here are just a few that we have found useful.

Environment Variables

The PostgreSQL image uses several environment variables which are easy to miss. The only variable required is `POSTGRES_PASSWORD`, the rest are optional.

Warning: the Docker specific variables will only have an effect if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup.

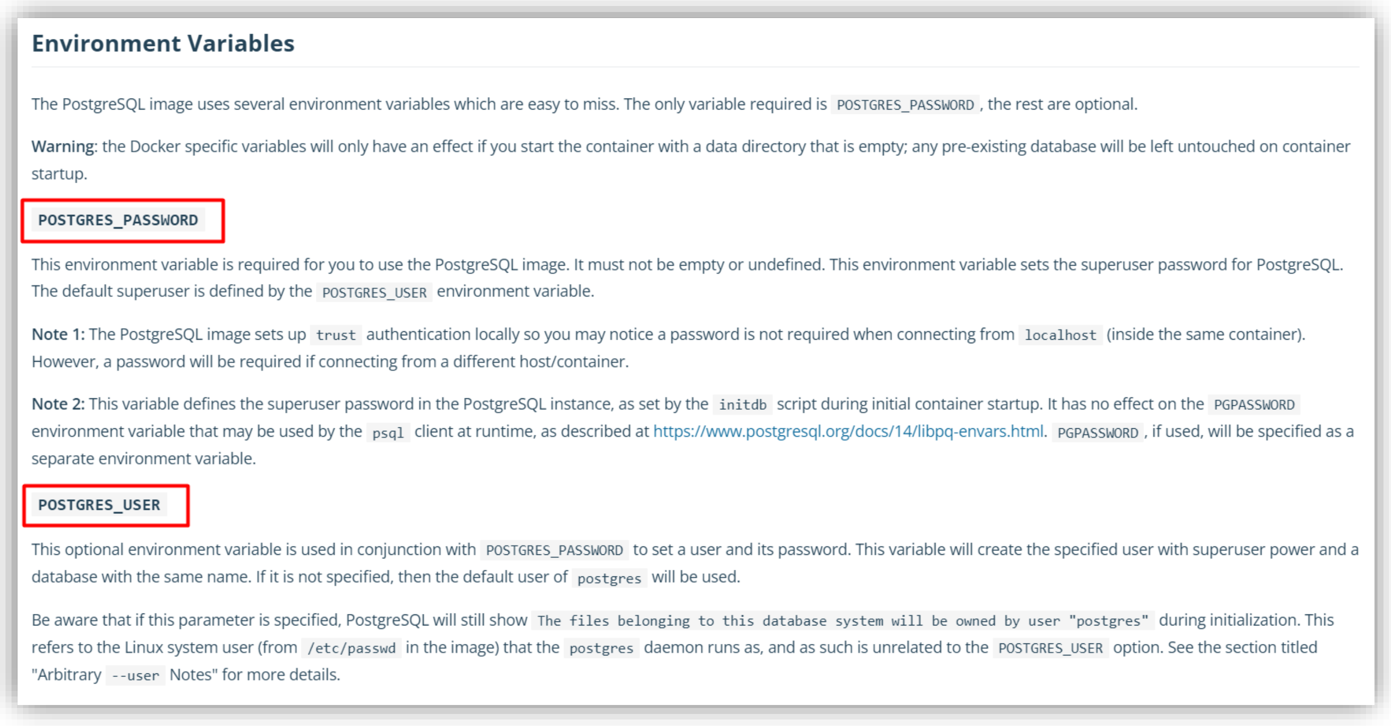
`POSTGRES_PASSWORD`

This environment variable is required for you to use the PostgreSQL image. It must not be empty or undefined. This environment variable sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable.

Note 1: The PostgreSQL image sets up `trust` authentication locally so you may notice a password is not required when connecting from `localhost` (inside the same container). However, a password will be required if connecting from a different host/container.

Note 2: This variable defines the superuser password in the PostgreSQL instance, as set by the `initdb` script during initial container startup. It has no effect on the `PGPASSWORD` environment variable that may be used by the `psql` client at runtime, as described at https://www.postgresql.org/docs/14/libpq-envars.html. `PGPASSWORD`, if used, will be specified as a separate environment variable.

To create a PostgreSQL container, we will set a **"POSTGRES_PASSWORD"** and a **"POSTGRES_USER"**. They are the **password** and the **username** you will need to **connect to the database server**:



Environment Variables

The PostgreSQL image uses several environment variables which are easy to miss. The only variable required is `POSTGRES_PASSWORD`, the rest are optional.

Warning: the Docker specific variables will only have an effect if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup.

`POSTGRES_PASSWORD`

This environment variable is required for you to use the PostgreSQL image. It must not be empty or undefined. This environment variable sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable.

Note 1: The PostgreSQL image sets up `trust` authentication locally so you may notice a password is not required when connecting from `localhost` (inside the same container). However, a password will be required if connecting from a different host/container.

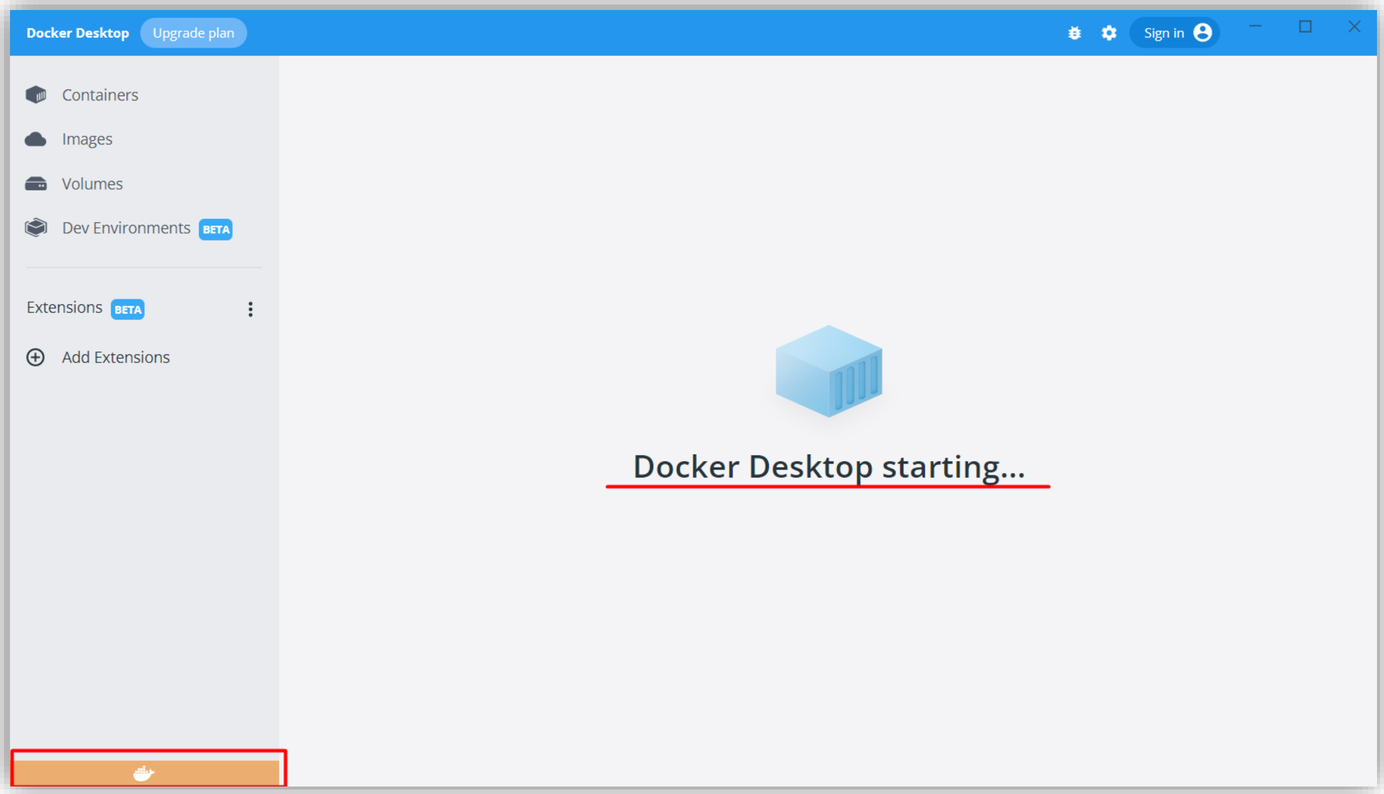
Note 2: This variable defines the superuser password in the PostgreSQL instance, as set by the `initdb` script during initial container startup. It has no effect on the `PGPASSWORD` environment variable that may be used by the `psql` client at runtime, as described at https://www.postgresql.org/docs/14/libpq-envars.html. `PGPASSWORD`, if used, will be specified as a separate environment variable.

`POSTGRES_USER`

This optional environment variable is used in conjunction with `POSTGRES_PASSWORD` to set a user and its password. This variable will create the specified user with superuser power and a database with the same name. If it is not specified, then the default user of `postgres` will be used.

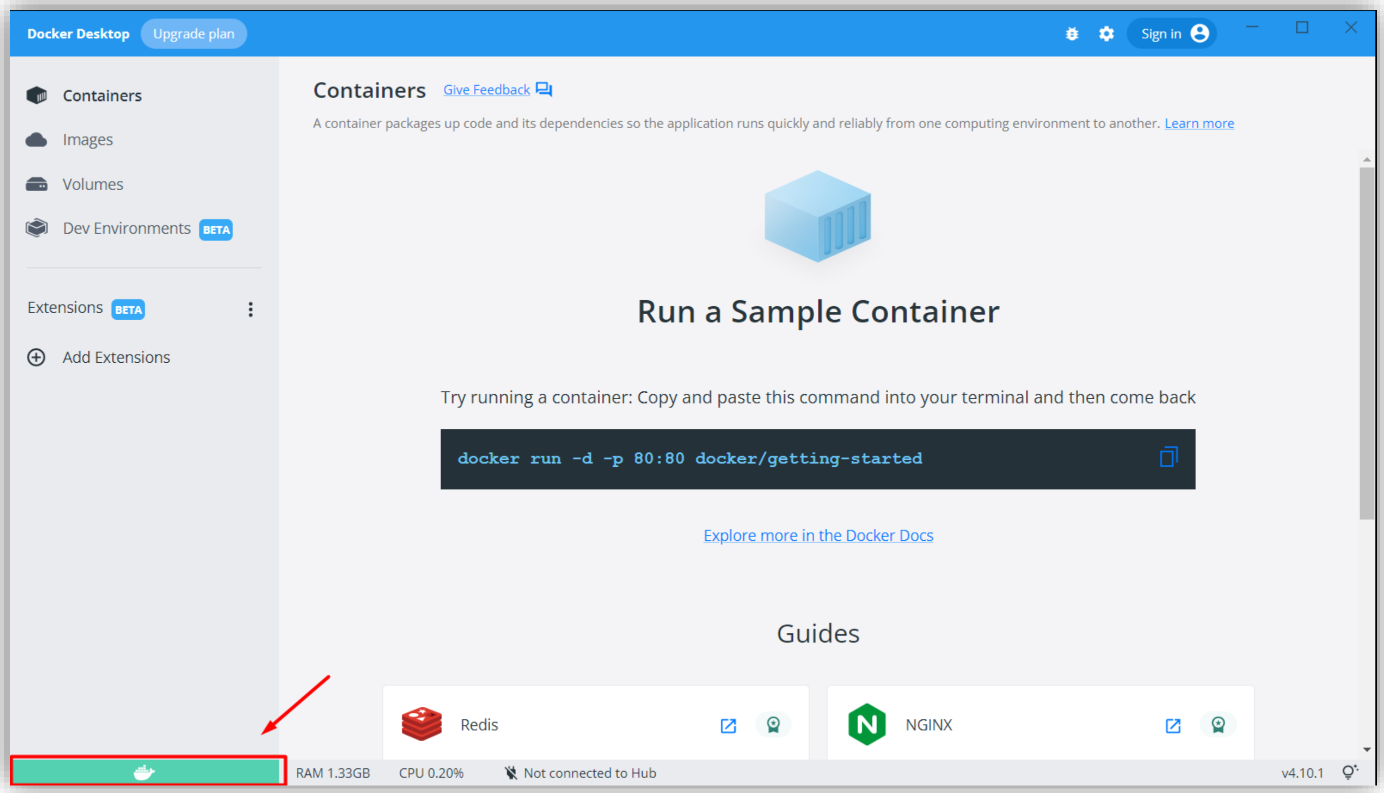
Be aware that if this parameter is specified, PostgreSQL will still show `The files belonging to this database system will be owned by user "postgres"` during initialization. This refers to the Linux system user (from `/etc/passwd` in the image) that the `postgres` daemon runs as, and as such is unrelated to the `POSTGRES_USER` option. See the section titled "Arbitrary `--user` Notes" for more details.

Let's go back to Docker to create our first container. To work with Docker, both **the terminal** on your local machine and **Docker Desktop** will be needed. First, **open Docker Desktop**. When you see an **orange** label on the left side of the screen, it means that **Docker is starting**:



Docker Desktop starting...

When the label becomes **green**, it means that **Docker is ready and running**:



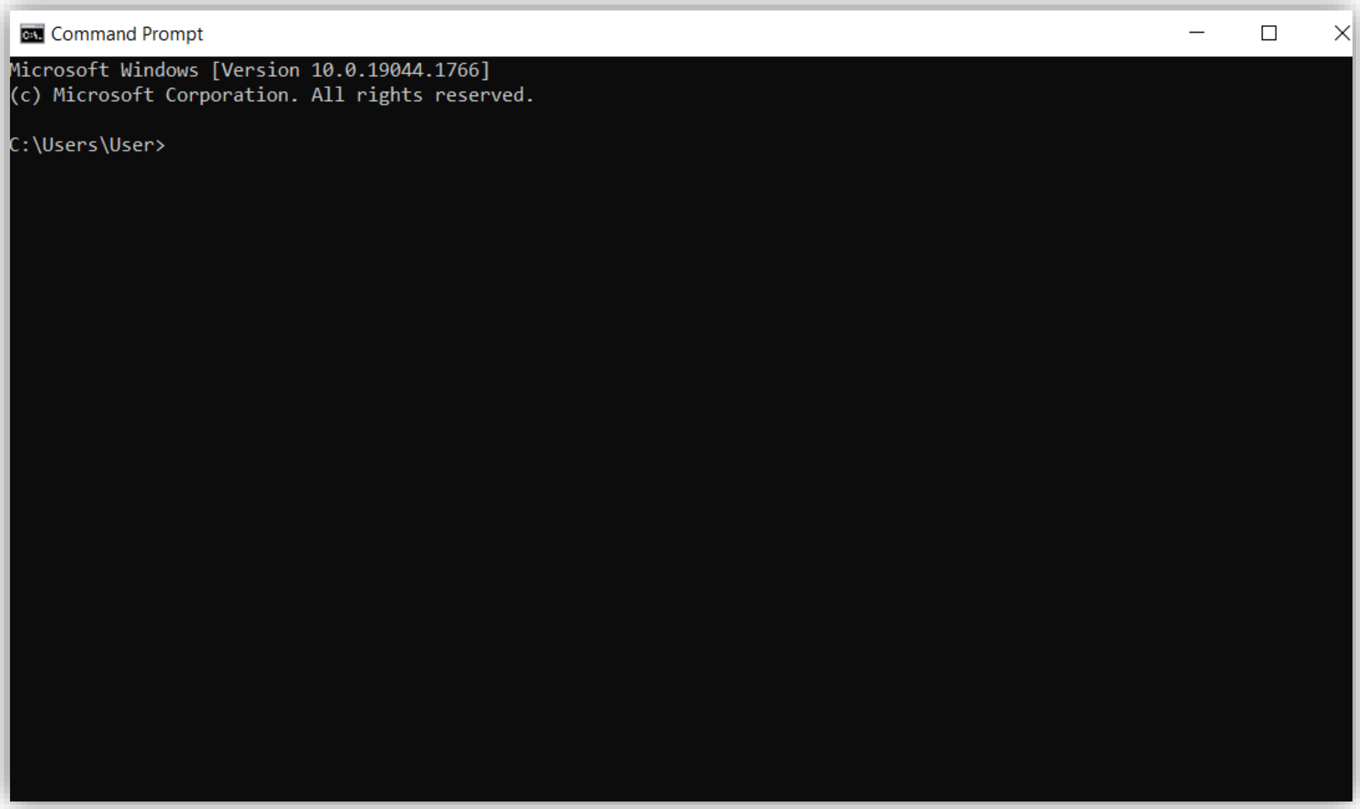
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Run a Sample Container

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

Next, open the terminal. For this guide, it will be used Windows Command Prompt, but the commands are the **same for each OS**:

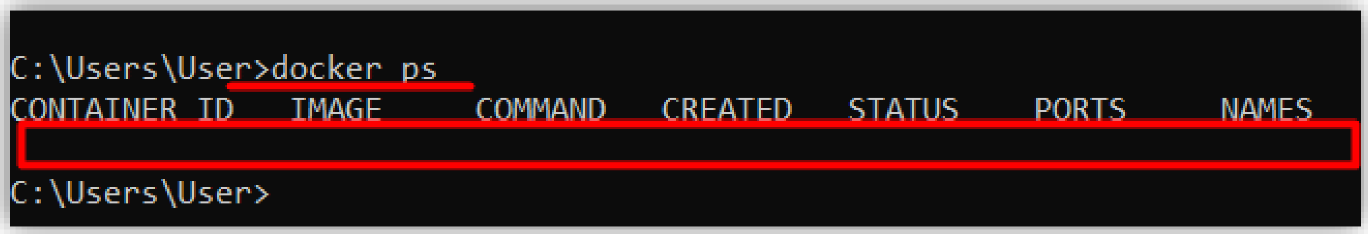


Command Prompt

```
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>
```

Each docker command starts with **"docker"**. For example, if you want to **see all running containers** write **"docker ps"**. As no containers have been created yet, the **table is empty**:



```
C:\Users\User>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\User>
```

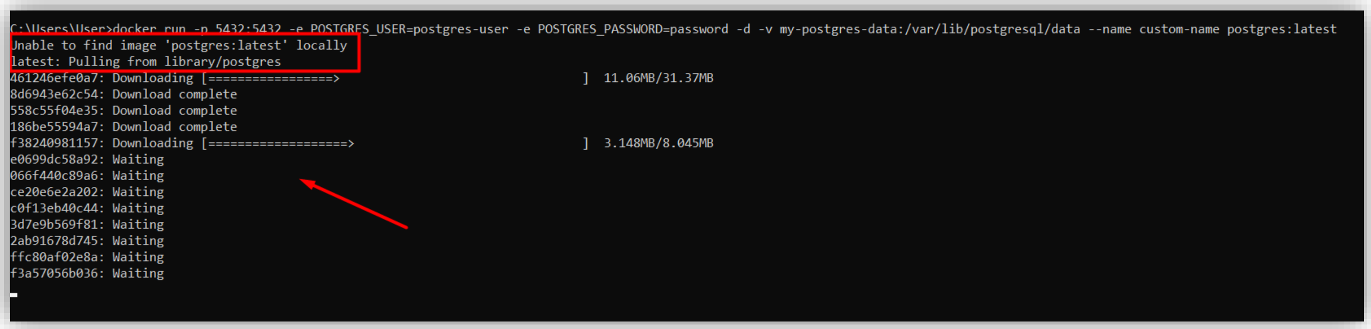
So, let's write a command to **create a new postgres container**:

docker run -p 5432:5432 -e POSTGRES_USER=postgres-user -e POSTGRES_PASSWORD=password -d -v my-postgres-data:/var/lib/postgresql/data --name custom-name postgres:latest

- **"run"** creates a new container. You can have as many containers using one image as you like.
- **"-p"** specifies the port the container will be using. Note: two containers cannot use the same port. In this case, we will map port 5432 in the container to port 5432 on the Docker host. The Docker host port is always the same. For PostgreSQL - it is 5432, and for pgAdmin - it is 80.
- **"-e"** shows that the following attribute will be an environment variable. They are always passed as key-value pairs. The username chosen is "postgres-user" and the password chosen is "password". You could write a username and a password of your choice.

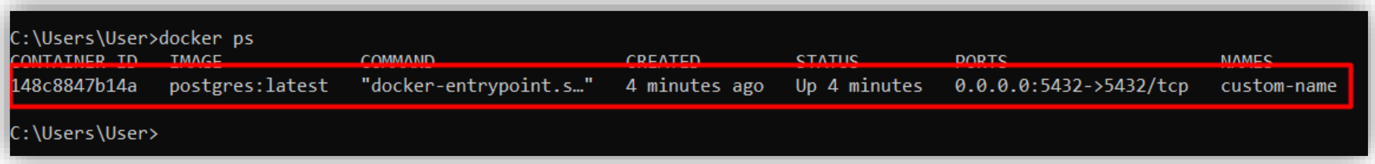
- **"-d"** starts a container in detached mode.
- **"-v"** creates a volume for that container. *(A volume is a file or folder stored on the local machine where all the files when you are using a container are stored. These files are always deleted each time a container is stopped. If you do not want that data to be deleted and you want to use it in the future, you should explicitly create a volume that will contain the path on the local machine where all data should be stored. More about the volumes: https://docs.docker.com/storage/).* In this case, we will use "named" volume. First, we named the volume where the data will be stored (e.g., "my-postgres-data"). Next, we wrote the path where the file or directory is mounted in the container. For each container, there is a specific path. For **PostgreSQL** - it is **"/var/lib/postgresql/data"** and for **pgAdmin** - it is **"/var/lib/pgadmin"**.
- **"--name"** option assigns a name to the container. Otherwise, docker generates a random string name. You could choose not to name the container. In this example, it is called "custom-name".
- At the end of the command, write the image you want to use, e.g., **"postgres:latest"**.

When you run a container, Docker first tries to find its image on your local machine. If it does not find it, it **automatically pulls (downloads) the image from the DockerHub**. Then, it **creates a container** and **starts it**. You can see the process below:



```
C:\Users\User>docker run -p 5432:5432 -e POSTGRES_USER=postgres-user -e POSTGRES_PASSWORD=password -d -v my-postgres-data:/var/lib/postgresql/data --name custom-name postgres:latest
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
461246efe0a7: Downloading [=================>                                 ]  11.06MB/31.37MB
8d6943e62c54: Download complete
558c55f04e35: Download complete
186be55594a7: Download complete
f38240981157: Downloading [===================>                               ]  3.148MB/8.045MB
e0699dc58a92: Waiting
066f440c89a6: Waiting
ce20e6e2a202: Waiting
c0f13eb40c44: Waiting
3d7e9b569f81: Waiting
2ab91678d745: Waiting
ffc80af02e8a: Waiting
f3a57056b036: Waiting
```

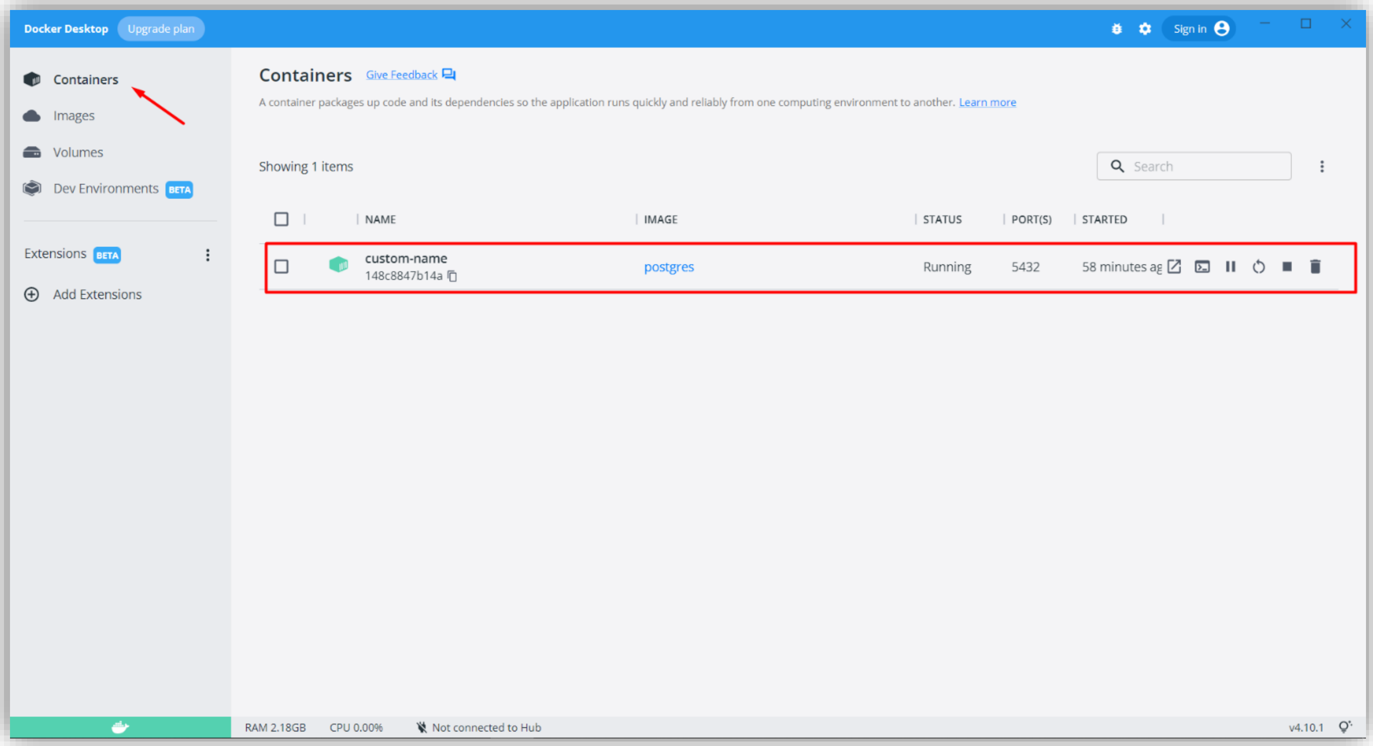
To check if the creation of the container is successful, write again the command **"docker ps"**. In the list of containers, there should be the one we created. Let's look at the information about the container: the container **ID is "148c8847b14a"**, the name is **"custom-name"**, the image is **"postgres:latest"**, the port it is using on the local machine is **"5432"**, mapped to port **"5432"** on the Docker host:



```
C:\Users\User>docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS                    NAMES
148c8847b14a   postgres:latest   "docker-entrypoint.s…"   4 minutes ago    Up 4 minutes    0.0.0.0:5432->5432/tcp   custom-name

C:\Users\User>
```

When you open the Docker Desktop **Containers** tab, you can see it there too:



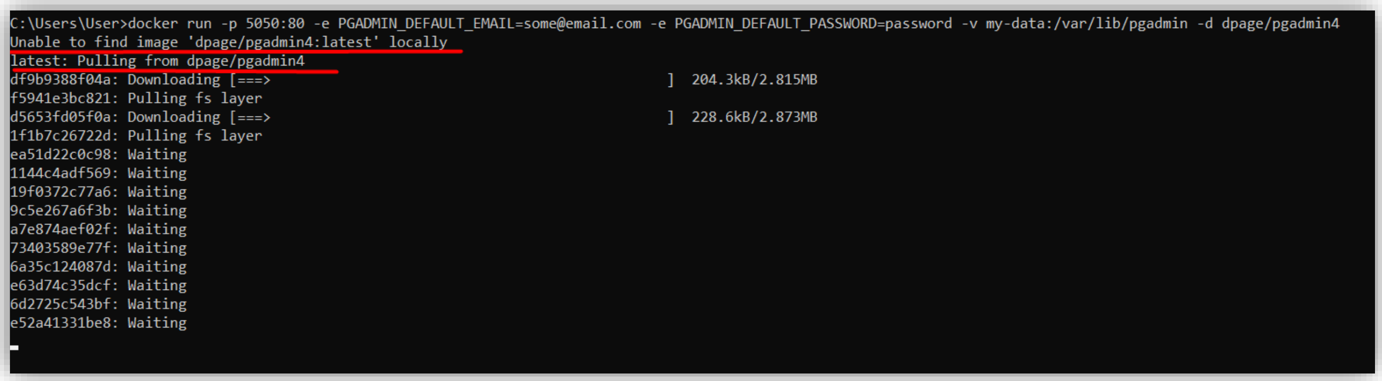
Running pgAdmin container with Docker

It is not enough to run PostgreSQL because we do not have direct access to a database. We need an interface to work with it. The most popular one is pgAdmin. Likewise above, we will write a similar command for creating and running a pgAdmin container:

```
docker run -p 5050:80 -e PGADMIN_DEFAULT_EMAIL=some@email.com -e
PGADMIN_DEFAULT_PASSWORD=password -v my-data:/var/lib/pgadmin -d dpage/pgadmin4
```

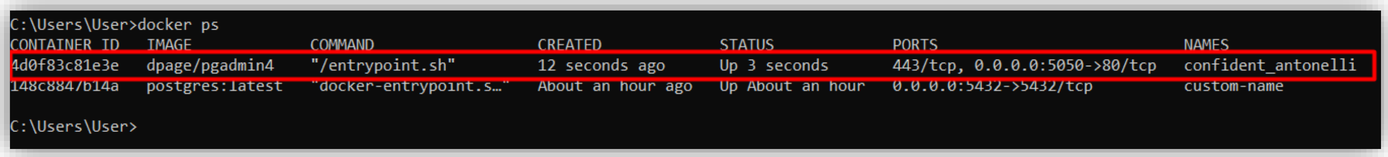
This time the option **"--name"** will not be used. Note that pgAdmin environment variables are different from the PostgreSQL ones. More information about them can be found at DockerHub pgAdmin image: https://hub.docker.com/r/dpage/pgadmin4.

When you run the command, you will see that **Docker could not find the image** because we hadn't pulled it before we ran the container. So, **Docker automatically starts pulling** it from DockerHub, and then **creates a container**:



```
C:\Users\User>docker run -p 5050:80 -e PGADMIN_DEFAULT_EMAIL=some@email.com -e PGADMIN_DEFAULT_PASSWORD=password -v my-data:/var/lib/pgadmin -d dpage/pgadmin4
Unable to find image 'dpage/pgadmin4:latest' locally
latest: Pulling from dpage/pgadmin4
df9b9388f04a: Downloading [===>                                               ]  204.3kB/2.815MB
f5941e3bc821: Pulling fs layer
d5653fd05f0a: Downloading [===>                                               ]  228.6kB/2.873MB
1f1b7c26722d: Pulling fs layer
ea51d22c0c98: Waiting
1144c4adf569: Waiting
19f0372c77a6: Waiting
9c5e267a6f3b: Waiting
a7e874aef02f: Waiting
73403589e77f: Waiting
6a35c124087d: Waiting
e63d74c35dcf: Waiting
6d2725c543bf: Waiting
e52a41331be8: Waiting
```

Write down again the **"`docker ps`"** command to ensure everything works correctly:



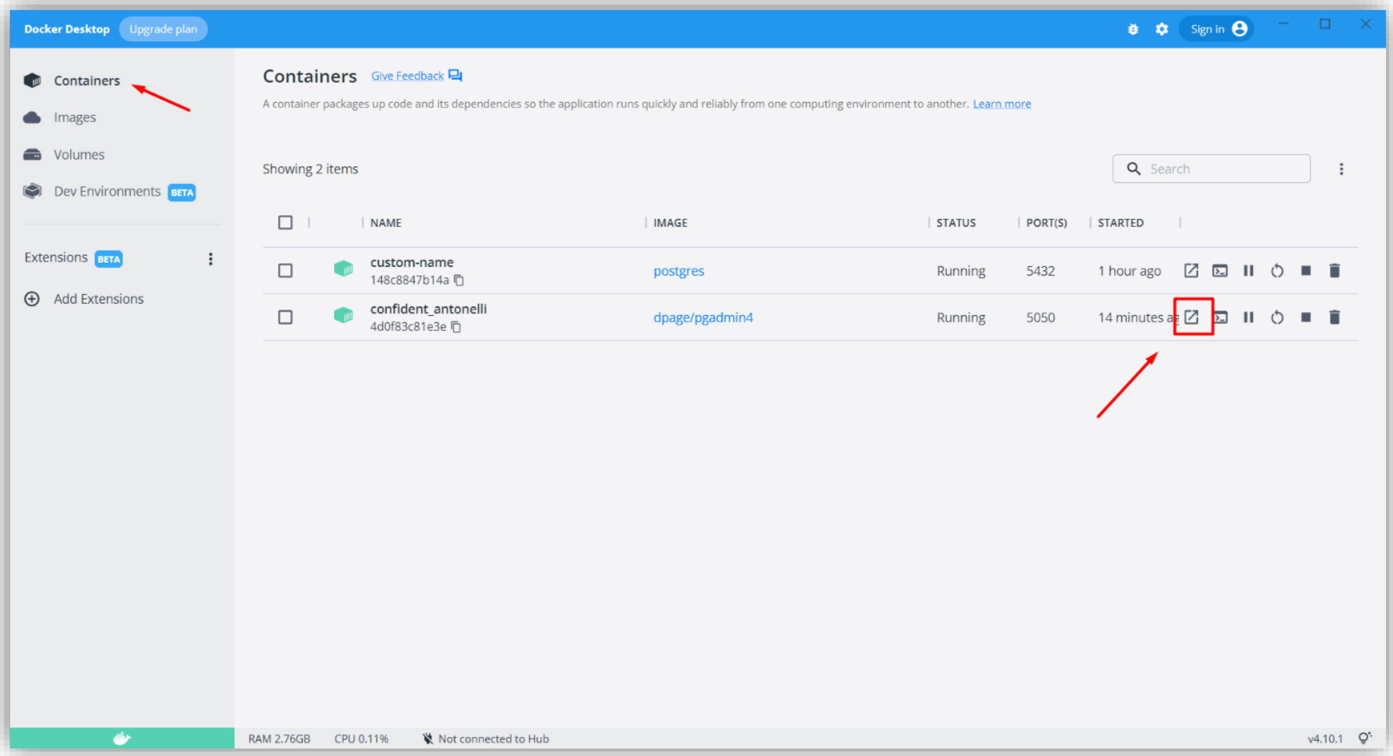
```
C:\Users\User>docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED            STATUS            PORTS                             NAMES
4d0f83c81e3e   dpage/pgadmin4    "/entrypoint.sh"         12 seconds ago     Up 3 seconds      443/tcp, 0.0.0.0:5050->80/tcp    confident_antonelli
148c8847b14a   postgres:latest   "docker-entrypoint.s…"   About an hour ago  Up About an hour  0.0.0.0:5432->5432/tcp           custom-name

C:\Users\User>
```

Connecting pgAdmin to PostgreSQL

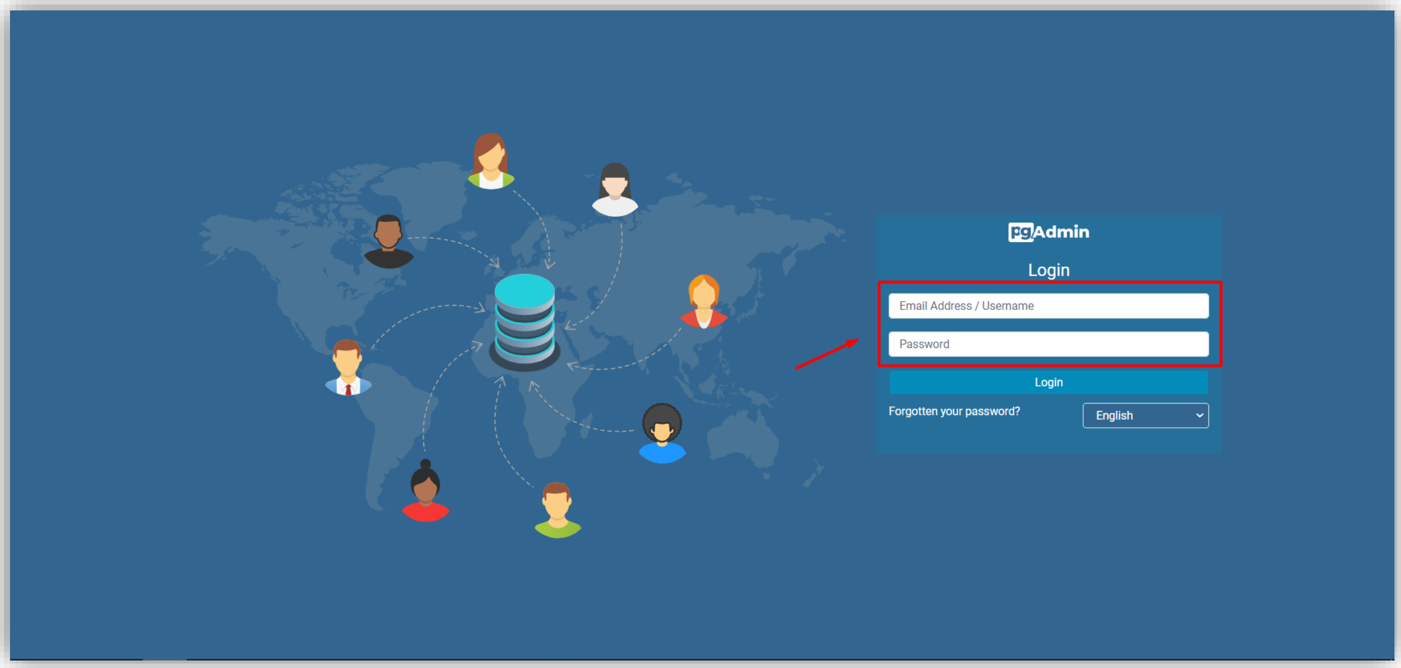
To open pgAdmin, you can use your browser and write down the URL of the port the container uses. A way to open it interactively is to **use Docker Desktop** - first, click on the **Containers** tab to access all containers, and then **click on the**

icon "Open with browser" on the **created pgAdmin container**:

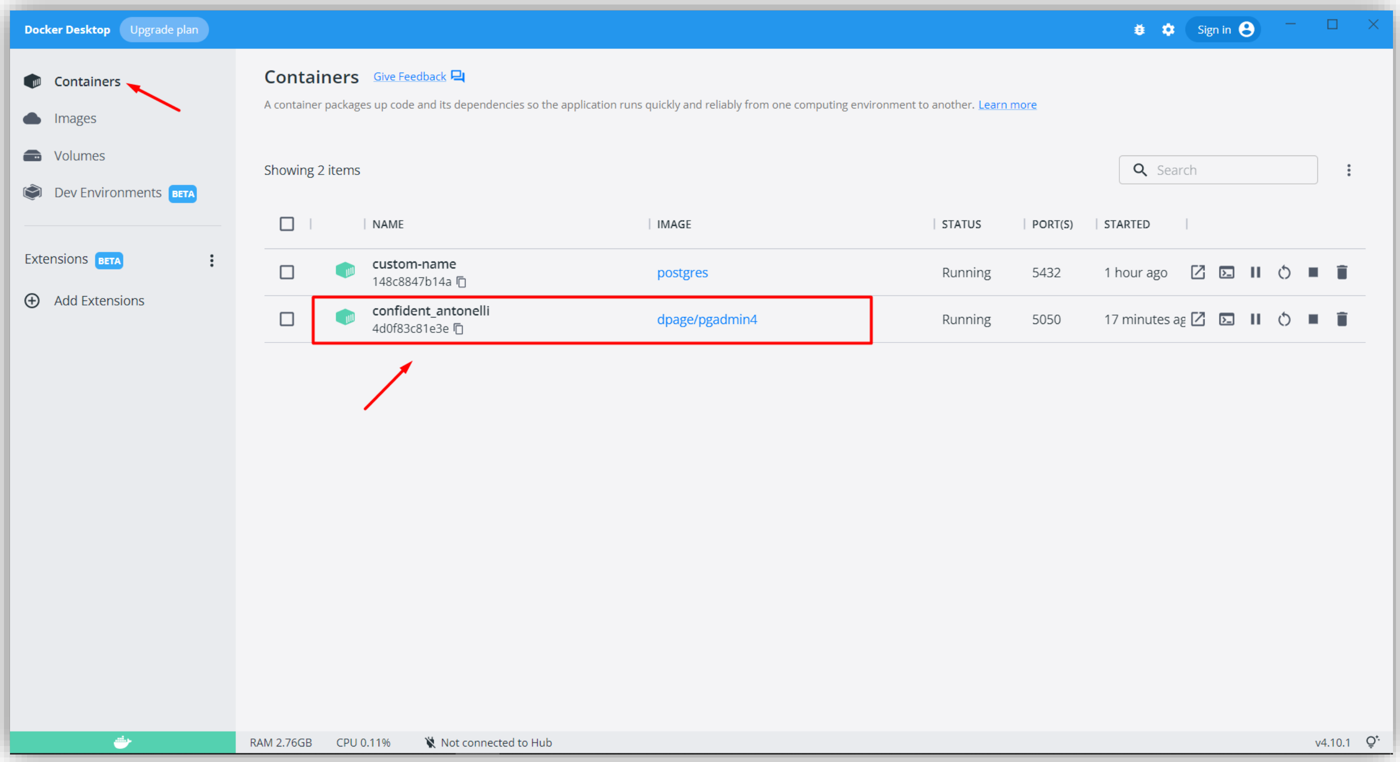


	NAME	IMAGE	STATUS	PORT(S)	STARTED
☐	custom-name 148c8847b14a	postgres	Running	5432	1 hour ago
☐	confident_antonelli 4d0f83c81e3e	dpage/pgadmin4	Running	5050	14 minutes ag

Keep in mind that pgAdmin needs a couple of minutes to load, so the first time you open the page, it could raise a "This page isn't working" error. In this case, wait a minute and reload the URL. When the page loads, **fill out the form and log in**:



If you **forgot the name and the password**, you can always return to Docker Desktop and **inspect the container**. To do that, click on the **Containers** tab and then **click on the container's row** to open the interface of the container:



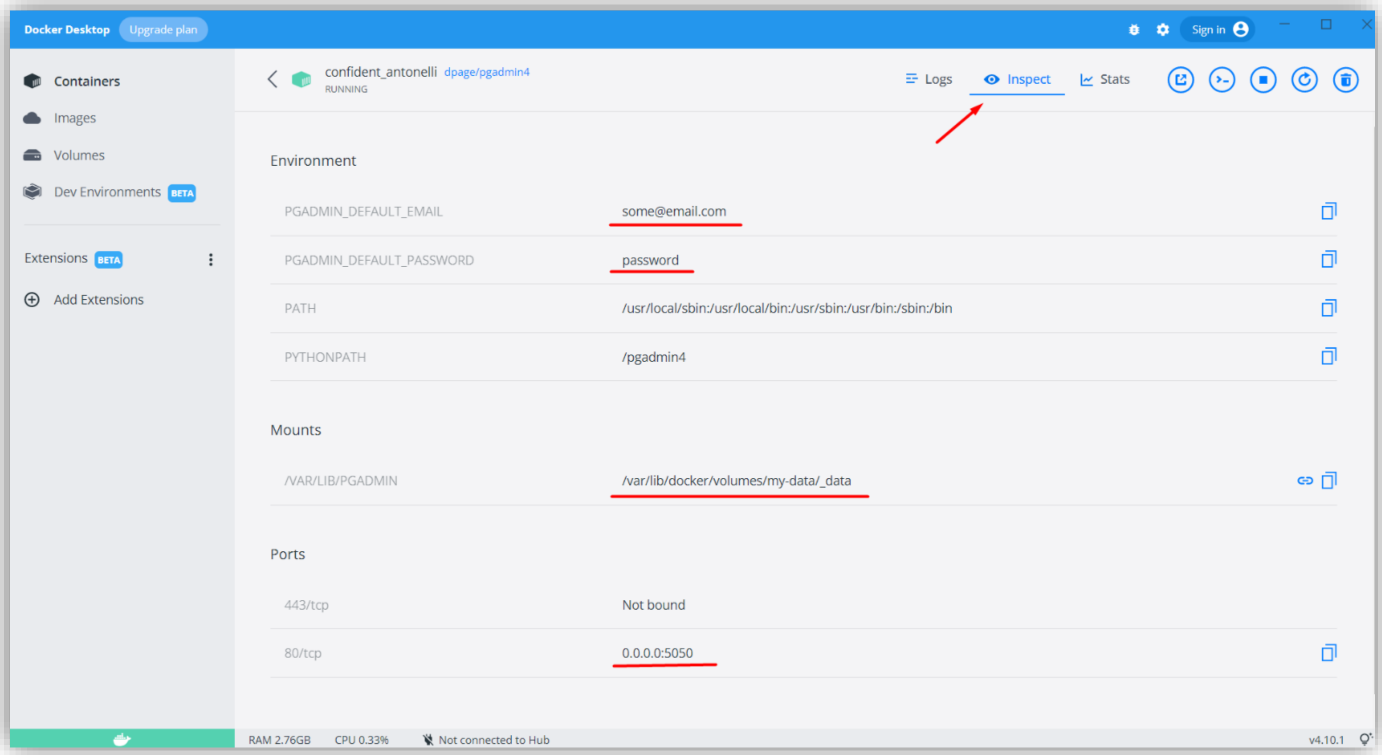
Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
custom-name 148c8847b14a	postgres	Running	5432	1 hour ago
confident_antonelli 4d0f83c81e3e	dpage/pgadmin4	Running	5050	17 minutes ag

Click on **"Inspect"** to see the **main information** of the container. There you can find the username and the password, the path to the pgAdmin volumes, and the used port:



Environment

PGADMIN_DEFAULT_EMAIL	some@email.com
PGADMIN_DEFAULT_PASSWORD	password
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PYTHONPATH	/pgadmin4

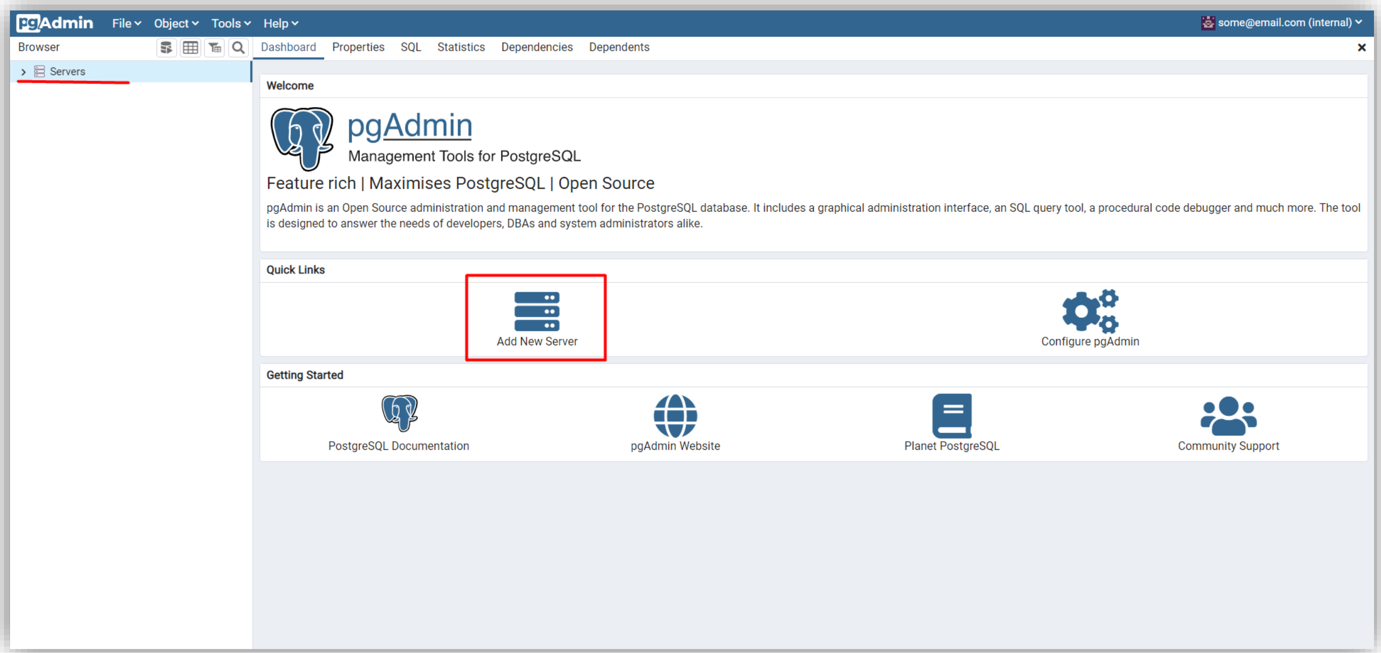
Mounts

/VAR/LIB/PGADMIN	/var/lib/docker/volumes/my-data/_data

Ports

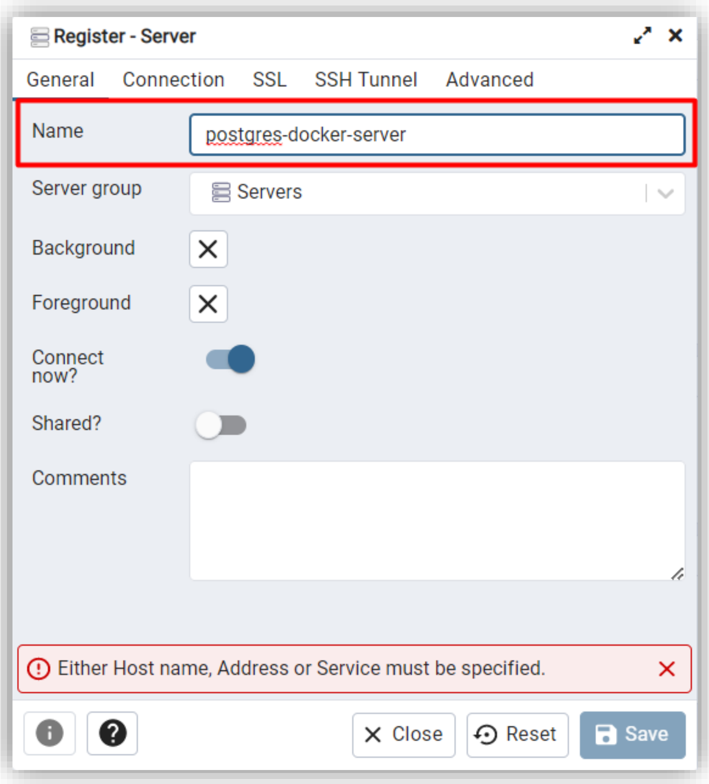
443/tcp	Not bound
80/tcp	0.0.0.0:5050

When you first log in to pgAdmin, you will not have any connections to the database, and you will need to create one. Click on the icon **"Add New Server"**:



pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.

Write a **server name of your choice**:



Name: postgres-docker-server

Server group: Servers

Background

Foreground

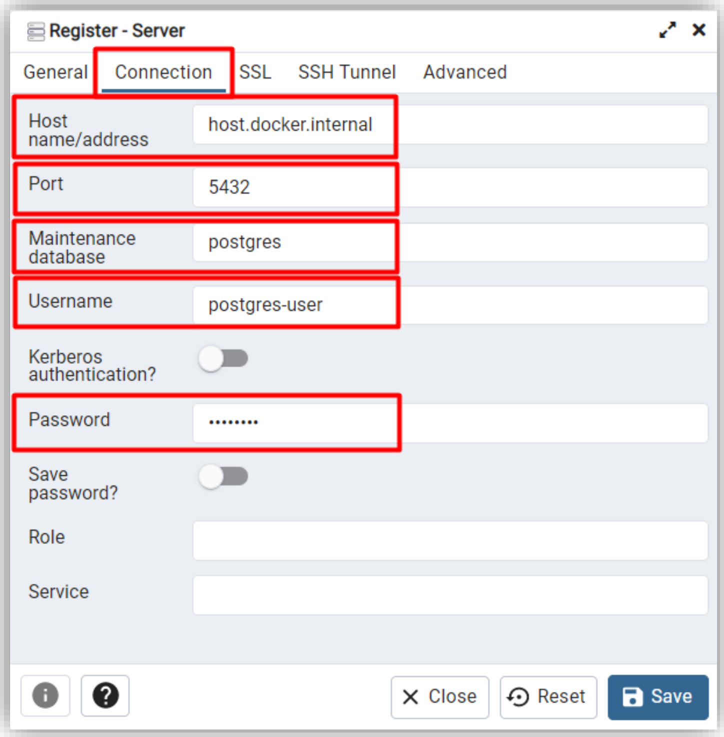
Connect now?

Shared?

Comments

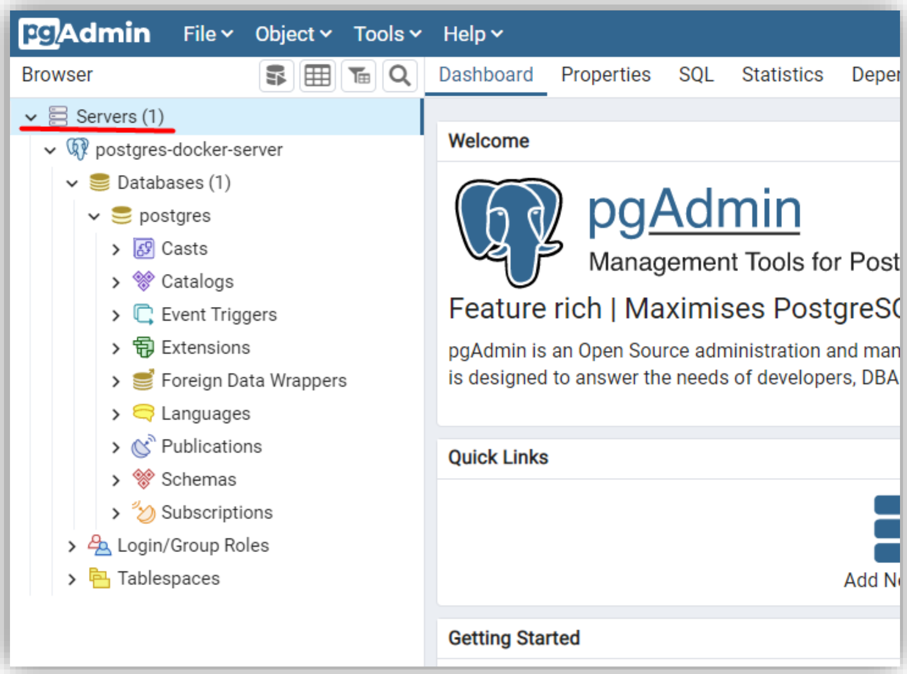
Either Host name, Address or Service must be specified.

Open the Connection Tab and write down the host name (always **"host.docker.internal"**), port of the container (in this example is **"5432"**), the maintenance database is **"postgres"**, your **username** and **password** which you chose when running the PostgreSQL container (in this example "postgres-user" and "password"). Click **"Save"** to save the server:



Host name/address	host.docker.internal
Port	5432
Maintenance database	postgres
Username	postgres-user
Kerberos authentication?	
Password	••••••••
Save password?	
Role	
Service	

Check **if the server is correctly loaded** on the left side of the screen:

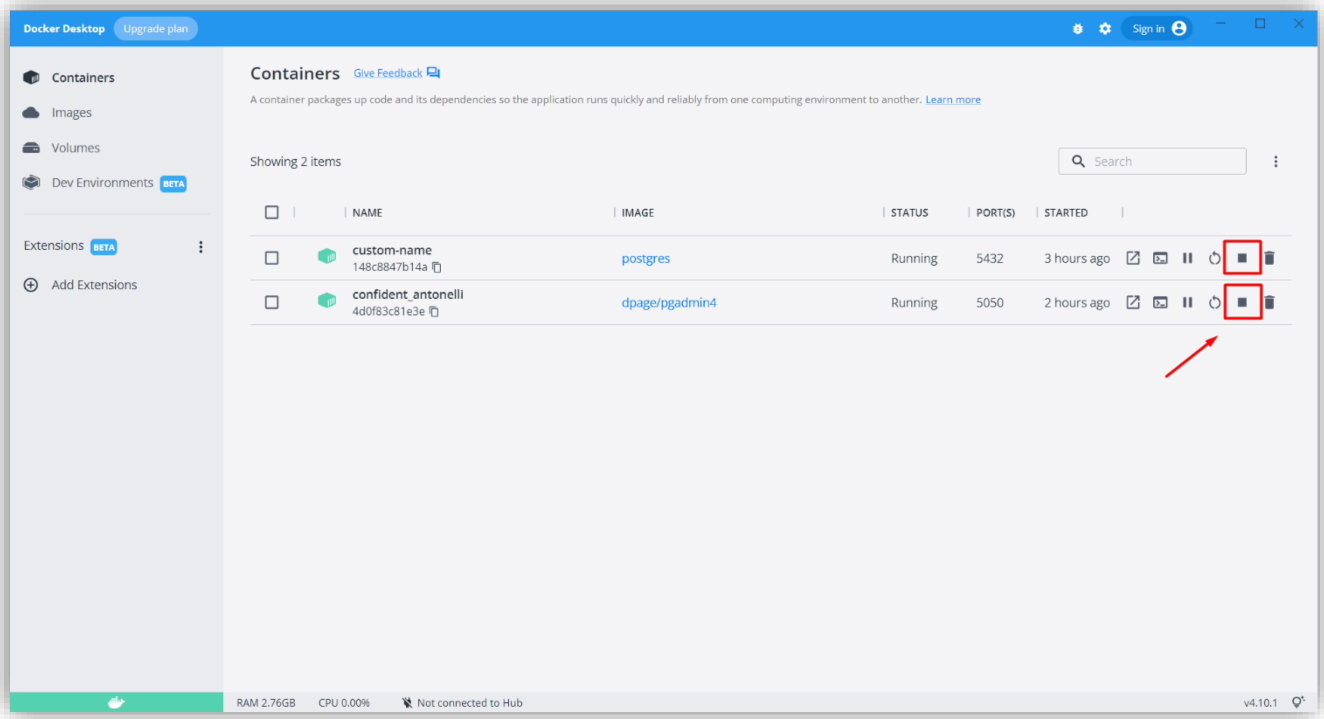


pgAdmin is an Open Source administration and man
is designed to answer the needs of developers, DBA

Working with Docker Desktop

You should **NOT run a new container** whenever you want to work with PostgreSQL and pgAdmin.

First, when you finish working with the containers, you can write the command **"docker stop {container_id/name}"** in the terminal (e.g., "docker stop custom-name"), or you can **stop** it **using Docker Desktop** by **clicking on the button "Stop"** for the specific container you want to stop:



Containers

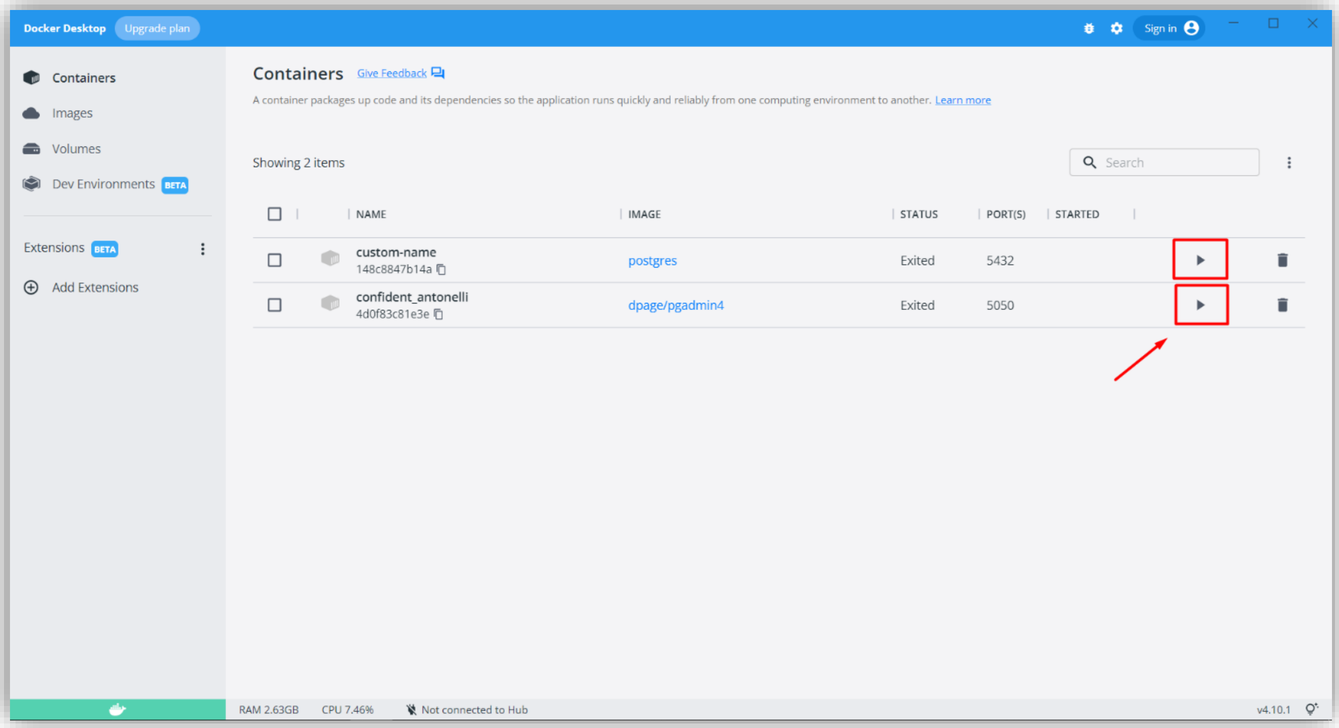
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
custom-name 148c8847b14a	postgres	Running	5432	3 hours ago
confident_antonelli 4d0f83c81e3e	dpage/pgadmin4	Running	5050	2 hours ago

When the container is stopped, and you want to **continue working with it**, you can write the terminal command **"docker start {container_id/name}"** (e.g., "docker start custom-name"), or you can open Docker Desktop Containers

Tab and **click on the "Start" button**:



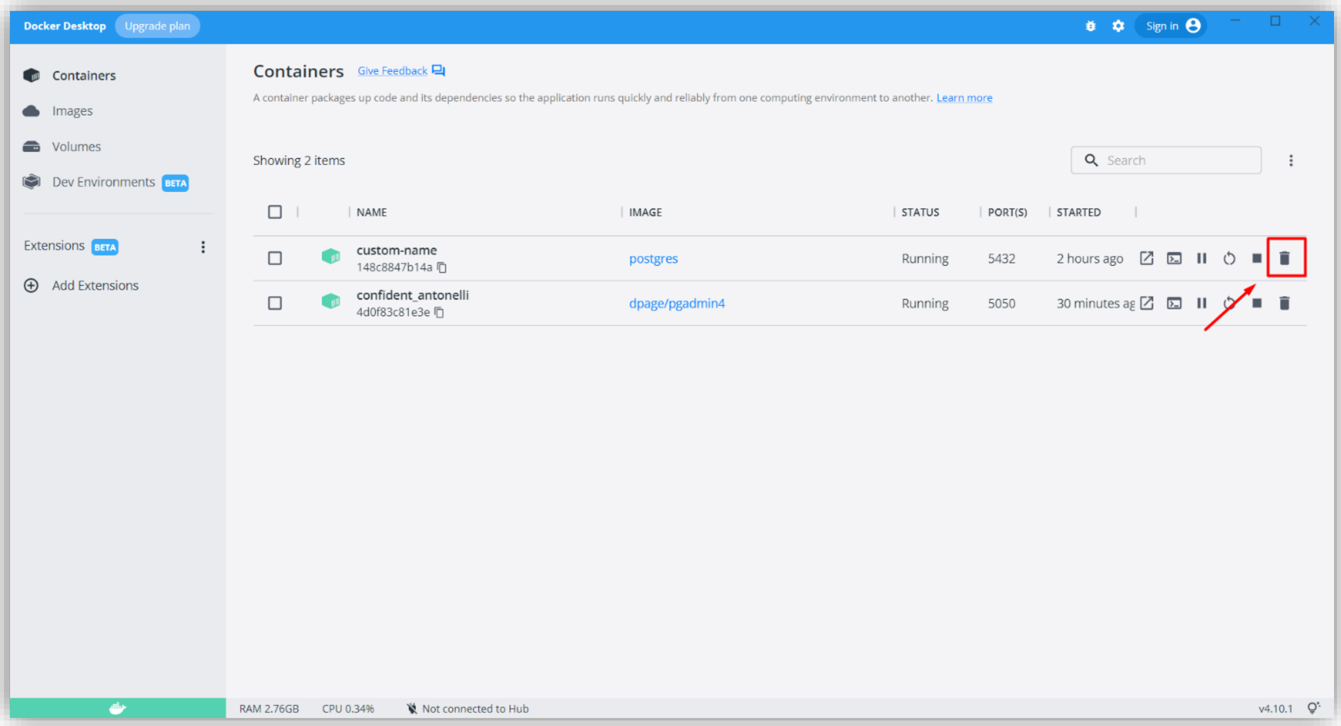
Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
custom-name 148c8847b14a	postgres	Exited	5432	
confident_antonelli 4d0f83c81e3e	dpage/pgadmin4	Exited	5050	

Suppose you do not need a container anymore - you will not use it. In that case, you can remove it using the terminal command **"docker rm {container_id/name}"** (e.g. "docker rm custom-name"), or you can **remove** it from the **Docker Desktop interface by clicking on the "Delete" button**:



Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
custom-name 148c8847b14a	postgres	Running	5432	2 hours ago
confident_antonelli 4d0f83c81e3e	dpage/pgadmin4	Running	5050	30 minutes ag