# PostgreSQL & pgAdmin Installation Guide

Guide for downloading and installing PostgreSQL and pgAdmin using Docker  
for the [**"PostgreSQL Course @ SoftUni"**](https://softuni.bg/modules/137/python-db)

# 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**](https://hub.docker.com/).

# 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**](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: Graphical user interface, text, application, email

Description automatically generated

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: Graphical user interface, text, application, email

Description automatically generated

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"):Table

Description automatically generated with medium confidence

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**: Graphical user interface, text, application, email

Description automatically generated

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**: Graphical user interface, text, application

Description automatically generated

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**:A picture containing application

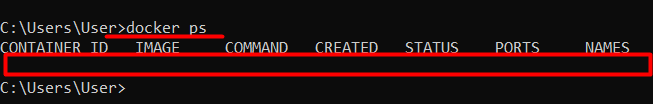
Description automatically generated

When the label becomes **green**, it means that **Docker is ready** **and running**:Graphical user interface, text, application, website

Description automatically generated

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

Description automatically generated

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**:

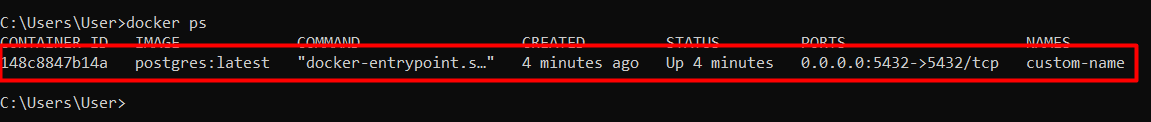
So, let's write a command to **create a new postgres container**:

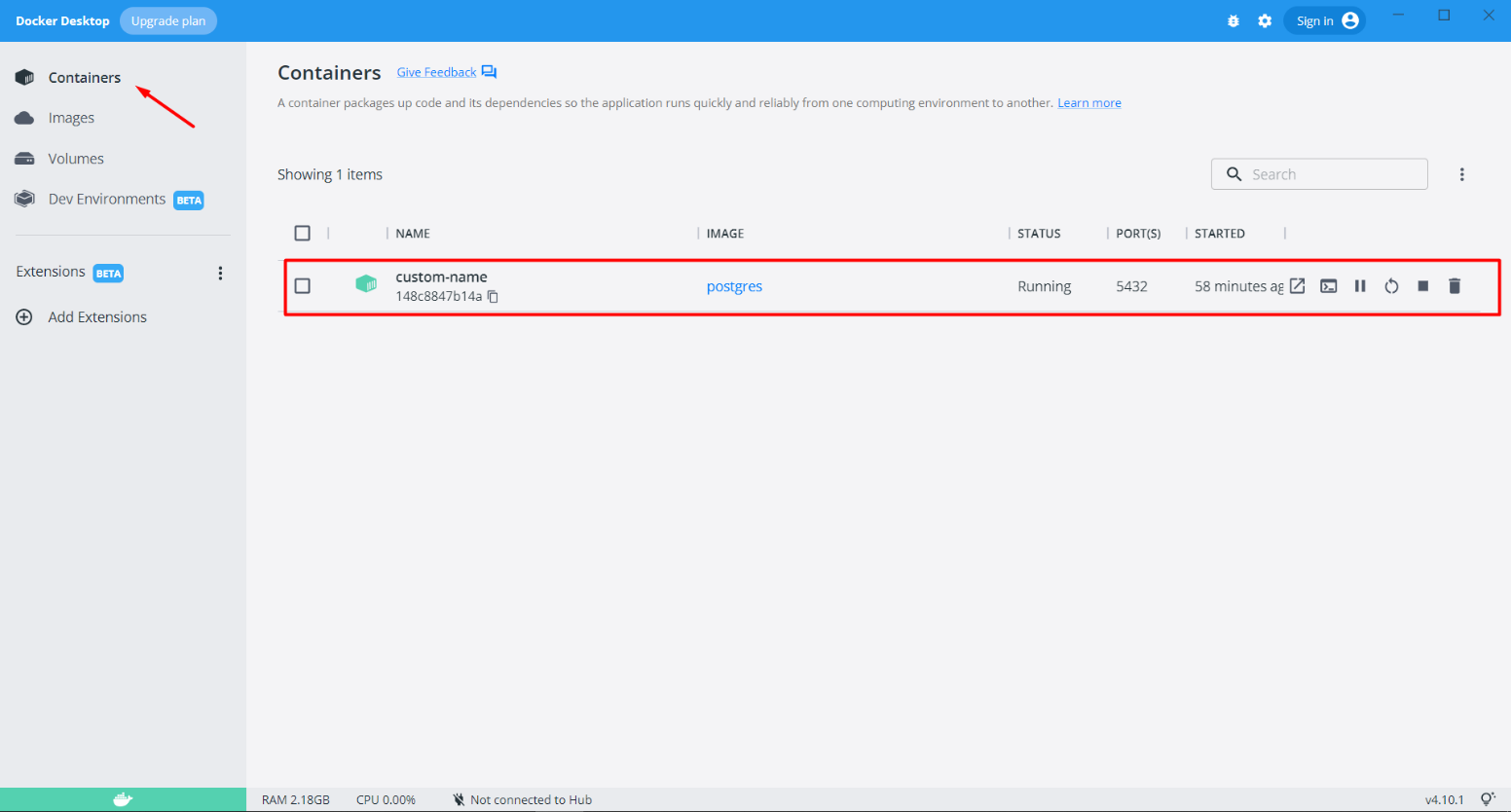
**docker run -p 5432:5432 -e POSTGRES\_USER=postgres-user -e POSTGRES\_PASSWORD=password -d -v my-postgres-data:/var/lib/postgresql/data --name custom-name postgres:latest**

* **"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 еnvironment 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/***](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: Graphical user interface

Description automatically generated with medium confidence

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:****

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

# Running pgAdmin container with Docker

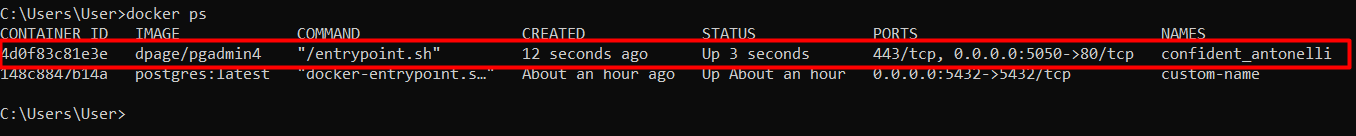
It is not enough to run PostgreSQL because we do not have direct access to a database. We need an interface to work with it. The most popular one is pgAdmin. Likewise above, we will write a similar command for creating and running a pgAdmin container:

**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**

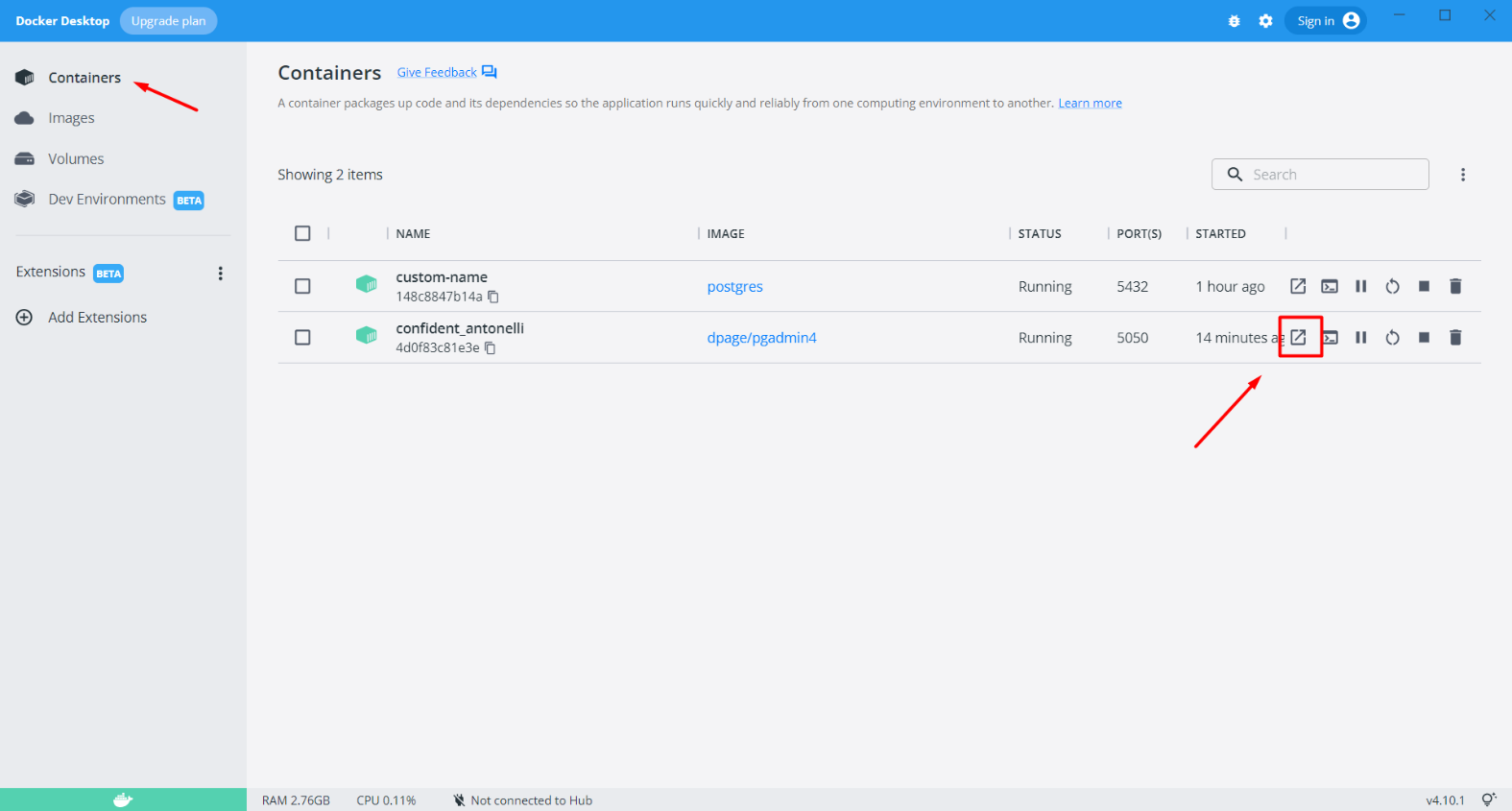
This time the option **"--name"** will not be used. Note that pgAdmin environment variables are different from the PostgreSQL ones. More information about them can be found at DockerHub pgAdmin image: [**https://hub.docker.com/r/dpage/pgadmin4**](https://hub.docker.com/r/dpage/pgadmin4).

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**: Text

Description automatically generated

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

# Connecting pgAdmin to PostgreSQL

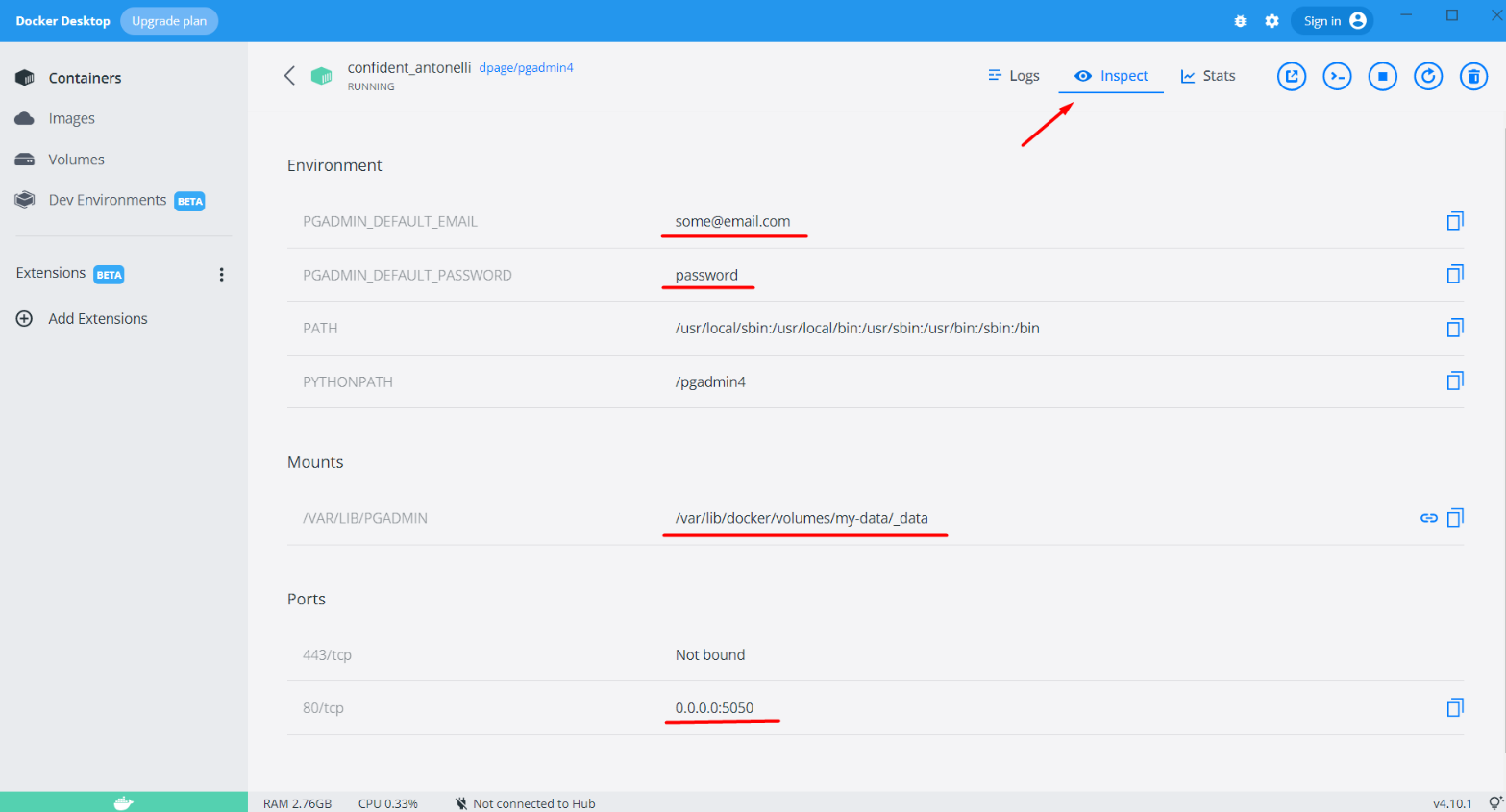
To open pgAdmin, you can use your browser and write down the URL of the port the container uses. A way to open it interactively is to **use Docker Desktop** - first, click on the **Containers** tab to access all containers, and then **click on the icon "Open with browser"** on the **created pgAdmin container**: 

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**: A screenshot of a computer

Description automatically generated with medium confidence

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:Graphical user interface, text, application, email

Description automatically generated

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:

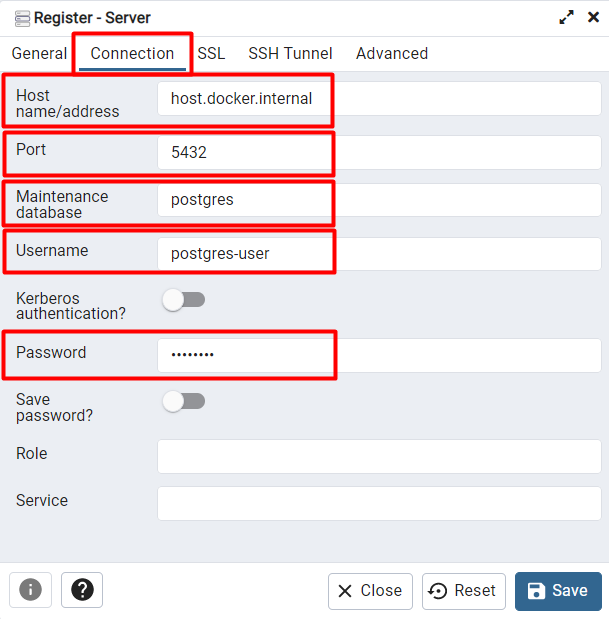
When you first log in to pgAdmin, you will not have any connections to the database, and you will need to create one. Click on the icon **"Add New Server"**:Graphical user interface, application, Word

Description automatically generated

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

Graphical user interface, application

Description automatically generated

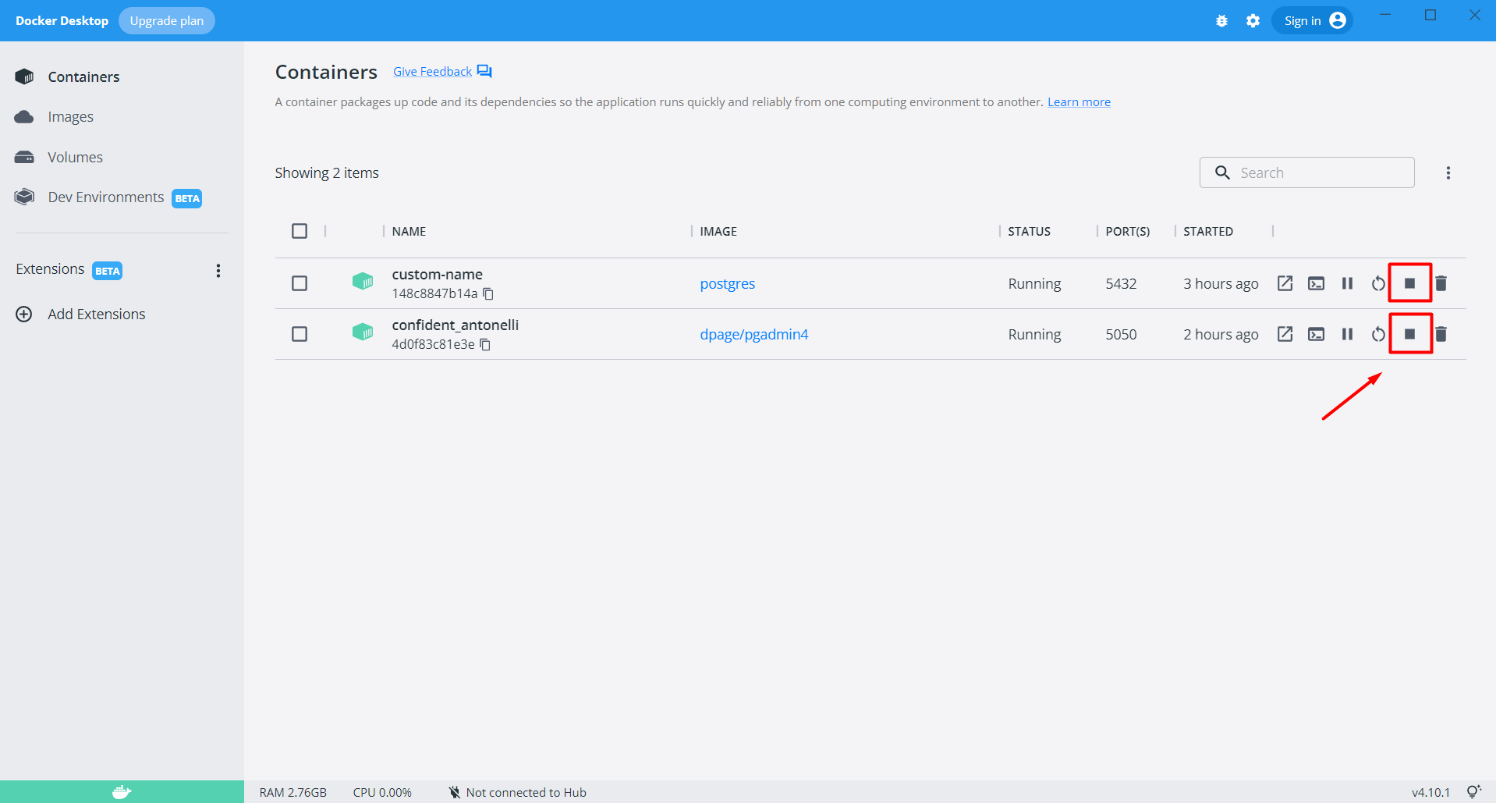
Open the Connection Tab and write down the host name (always **"host.docker.internal"**), port of the container (in this example is **"5432"**), the maintenance database is **"postgres"**, your **username** and **password** which you chose when running the PostgreSQL container (in this example "postgres-user"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

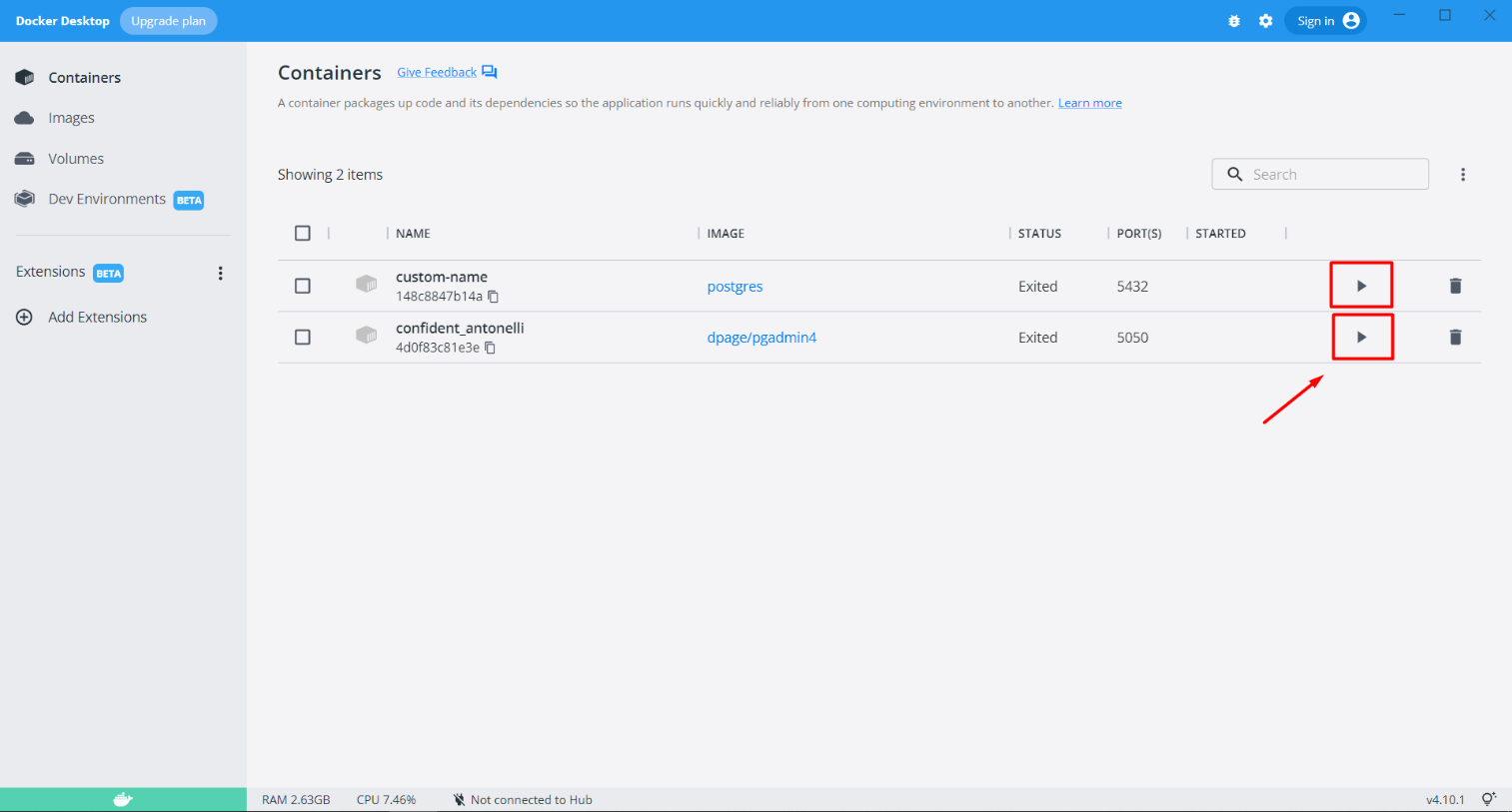
Check **if the server is correctly loaded** on the left side of the screen:Graphical user interface, application

Description automatically generated

# 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:

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**:

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**: