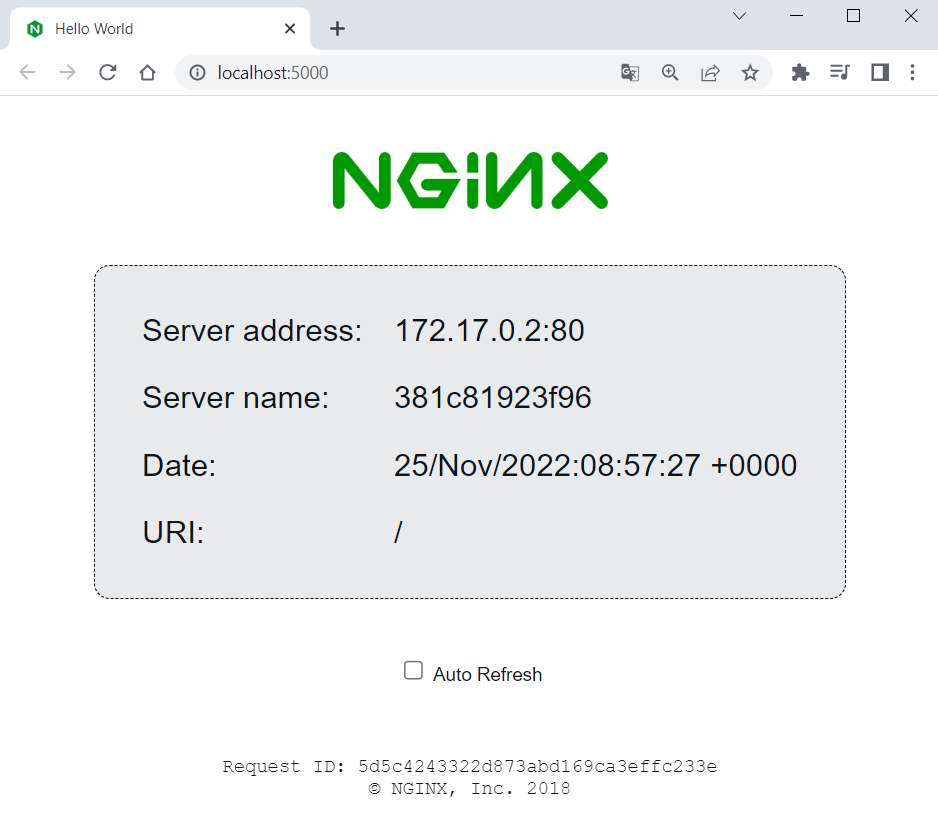
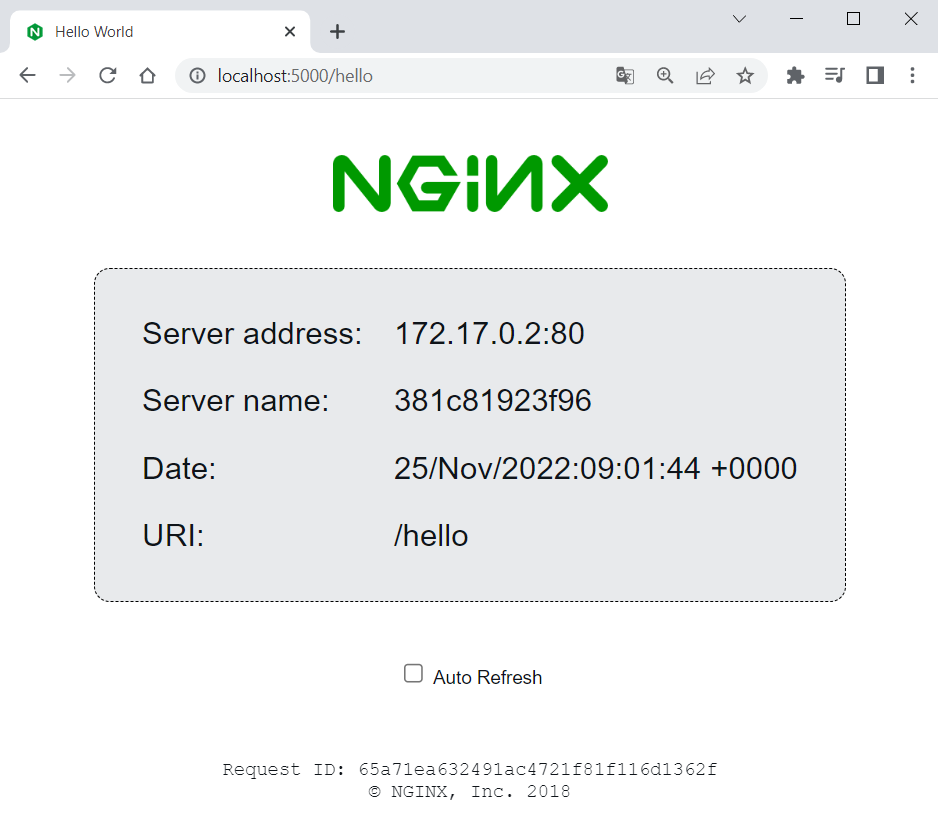
# Lab: Containers and Docker

Lab for the ["Containers and Clouds"](https://softuni.bg/trainings/4332/containers-and-cloud-september-2023) course @ SoftUni

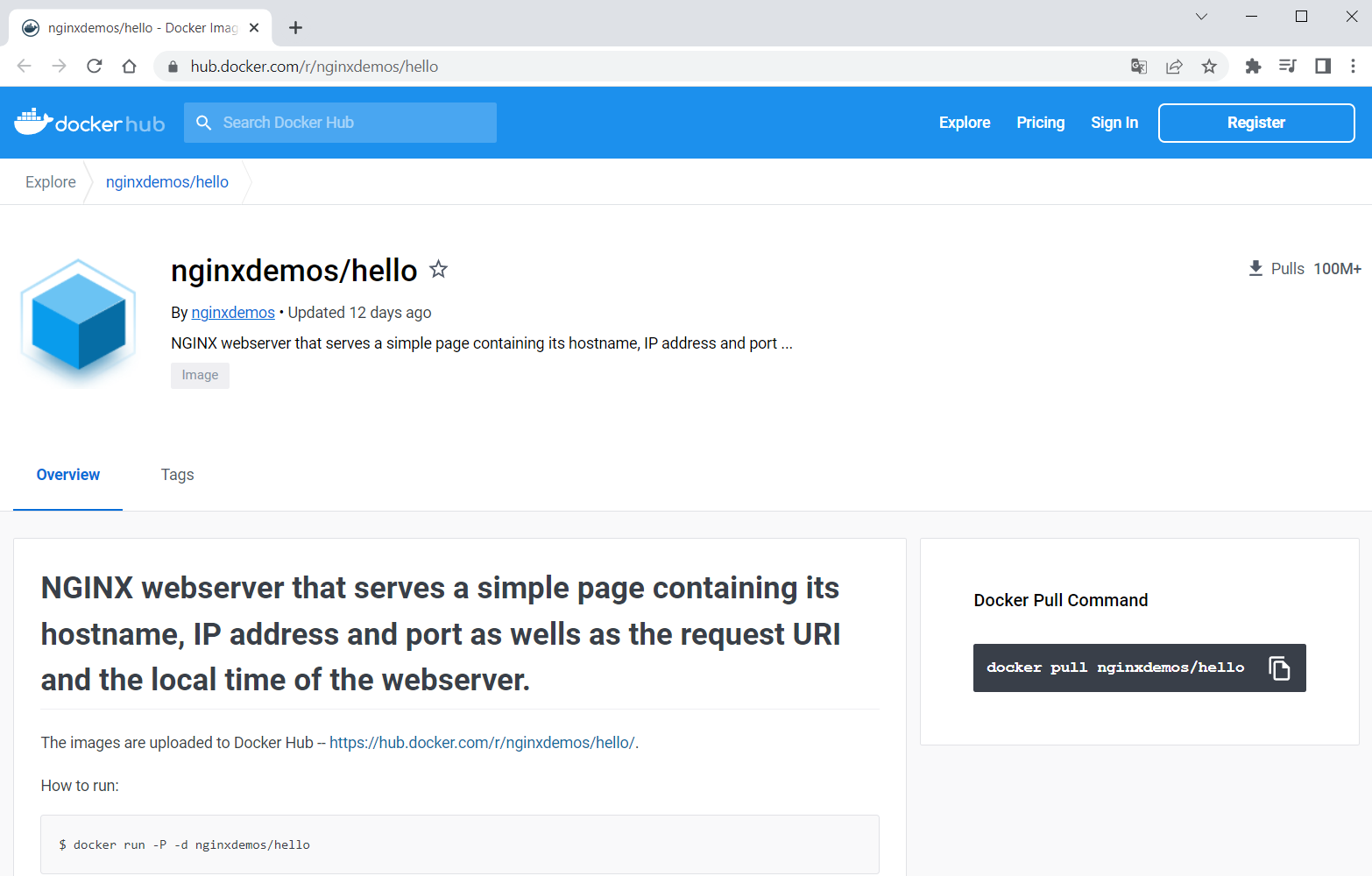
## NGINX Server Container

In this task, we will run a **simple NGINX server** in a **Docker** **container**. The **server** only returns **some server information**:

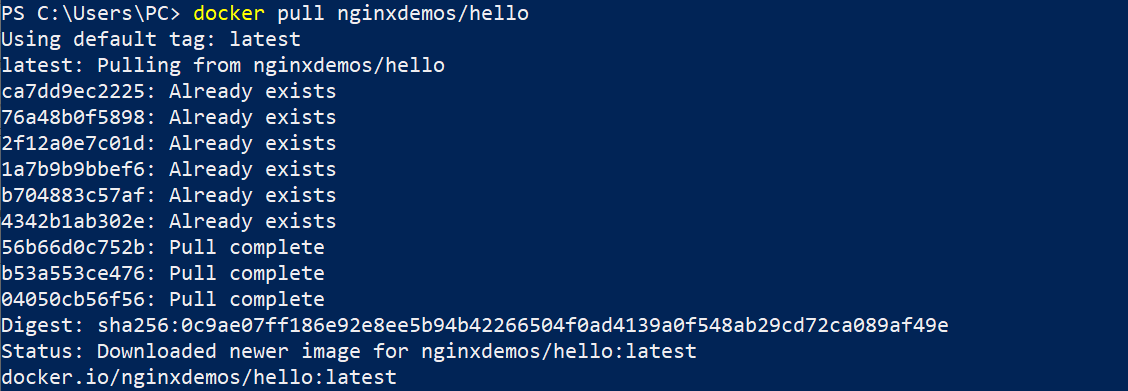
 

### Download Image

To create a **NGINX** **server** **container**, we shall first **pull the NGINX server image** from Docker Hub. You can find the image documentation on **Docker** **Hub** **here**: <https://hub.docker.com/r/nginxdemos/hello>. You can always **refer to the documentation** to get instructions on how to pull, build and run the image:

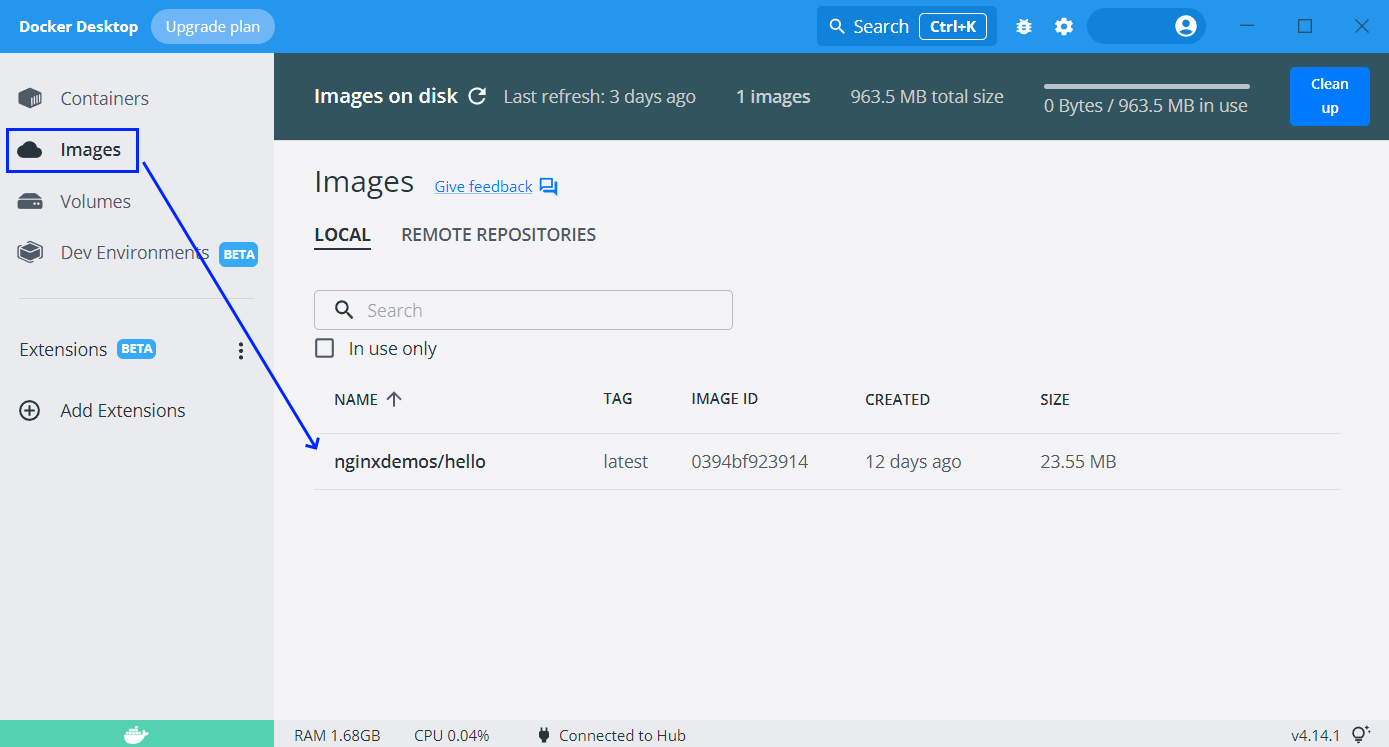


Open a **CLI**, for example, **PowerShell** or **Windows Terminal** or **Command Prompt** (**PowerShell** is recommended), and let's first **pull the Docker image** of the server. You should just use the **pull** **command** from the documentation:

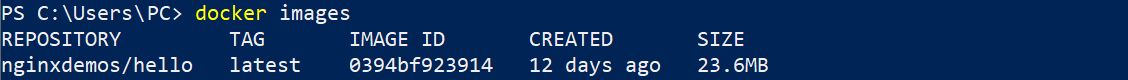


You can see that the **latest image is downloaded by default**. In addition, some of the **image layers** existed from before (from other installations), so they **were not downloaded again** – this is the advantage of Docker image layers.

You can look at your **downloaded** **images** in **Docker** **Desktop**, in the **[Images]** **tab**:

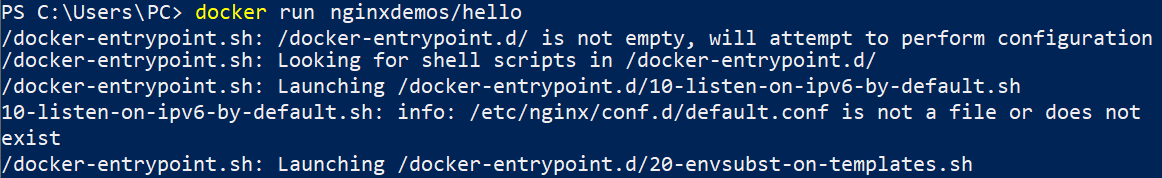


You can also use the command below to **display a list of all images** you have:

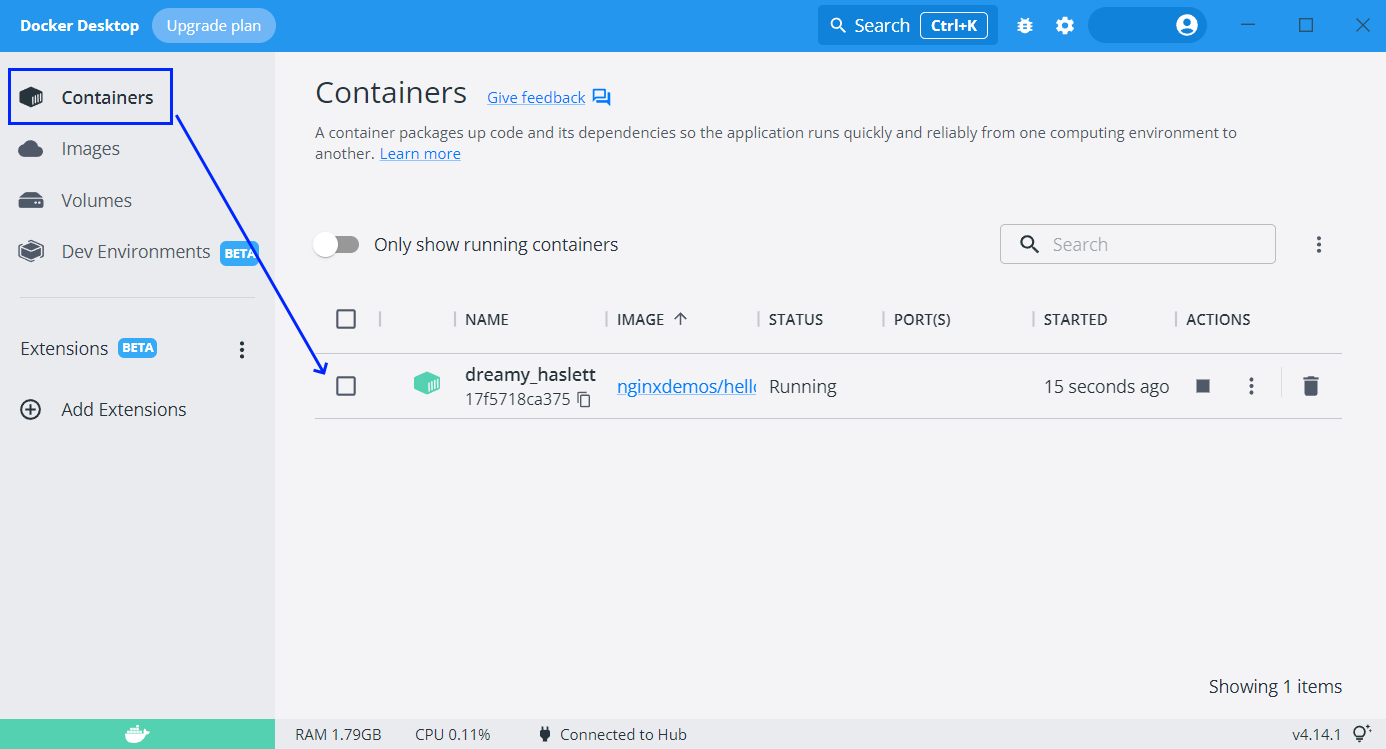


### Run a Container

Now we want to **run a container** with the **NGINX server image**, which is already downloaded from Docker Hub to our local machine. Use the **docker** **run** command and try this way:

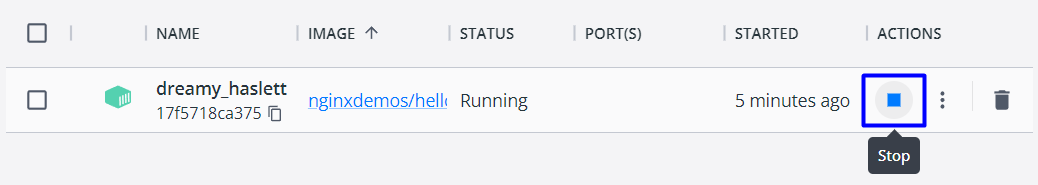
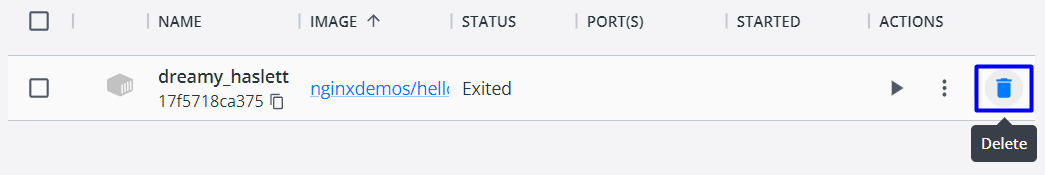


Now we have a **running container** with a **random name**, as we did not set it explicitly. We can see it on **Docker** **Desktop** **[Containers]**:

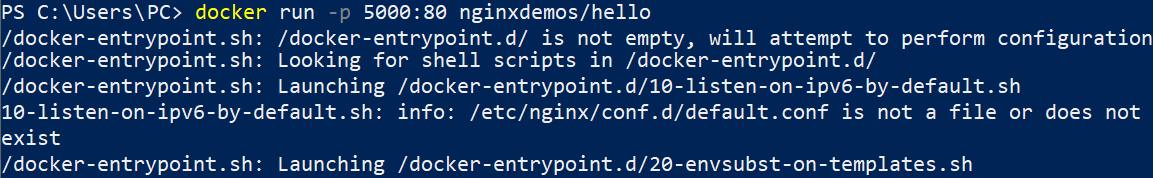


However, you can see that the "**PORT(S)**" **column** **is empty**, which means that **our container cannot be accessed** through the Internet, as it is isolated.

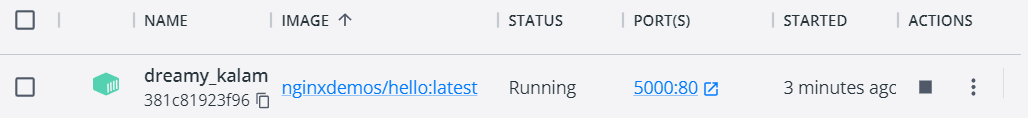
To fix this, we should **expose a port from the container**. But first, let's **stop** **and** **delete the container** we already have by **clicking on the buttons** in **Docker** **Desktop**:

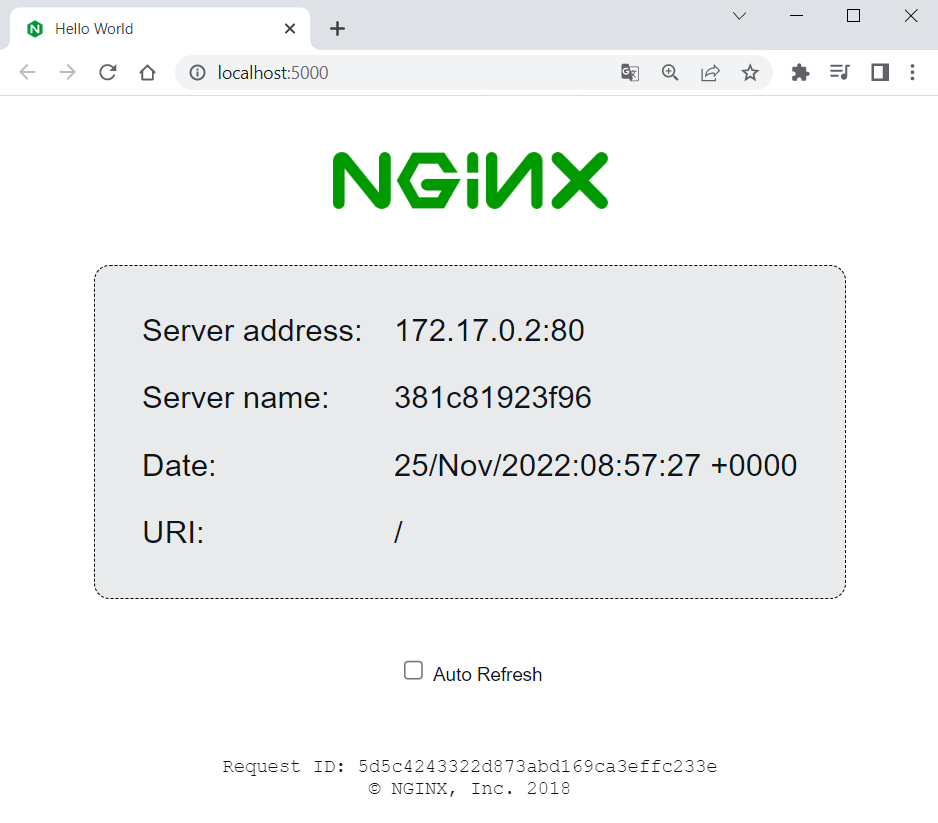
Let's **create another container** and **publish its port to the host**. This is done with the **-p** **option**. After it, we shall add the **port we want the server to be on our machine** (it can be any free port, but let's use **5000**) and the **internal** **port the server uses** – in our case **80**. Do it like this:



Now the **container is exposed**:



So, you can go to **http://localhost:5000** and **access the working server**:



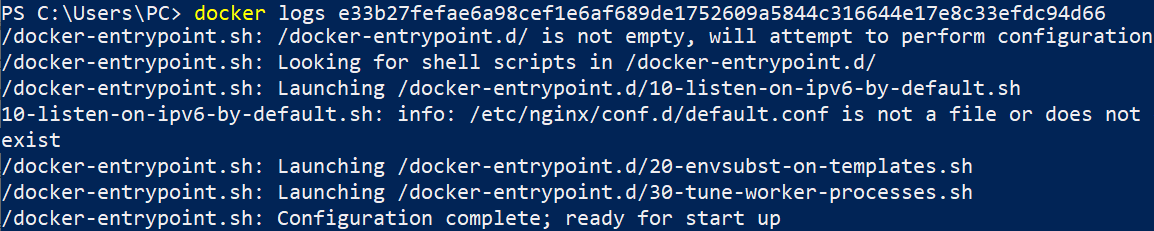
**Stop and delete the container** again and let's do one more thing.

### Run a Named Container in Detached Mode

This time, we want to **run a container** with a **name** and **in detached mode** (the container will be running in the background). To do this, use the **--name** **option** with the **container name you want**, and the **-d** **option** for **detached mode**:



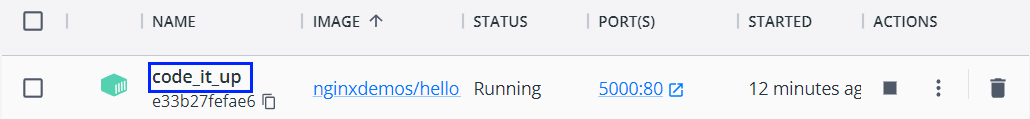
As you can see, now we have only the **container id** returned in the terminal and the **container logs are now shown** (because of the **detached mode**). However, you can **see the logs** with **docker** **logs** and the **container id or name**:

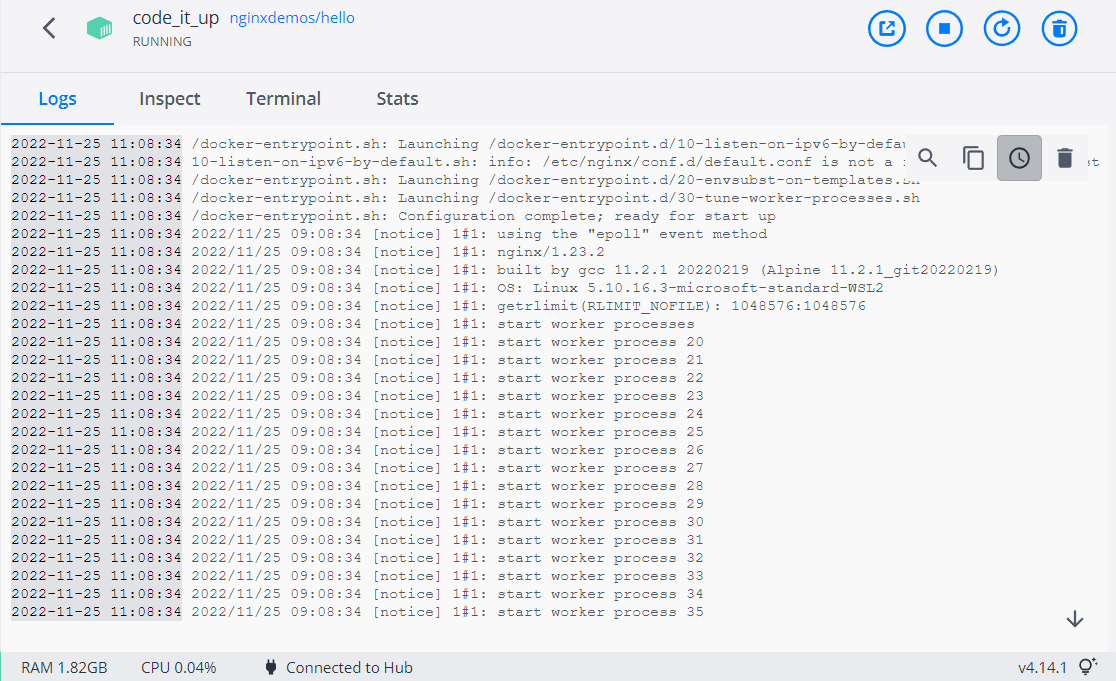


You can also use only the **first two symbols of the container's id**, not the whole one when they are unique (we have no other container with the same first symbols):



Or you can see them directly from **Docker** **Desktop** when you **click on the container's name**:

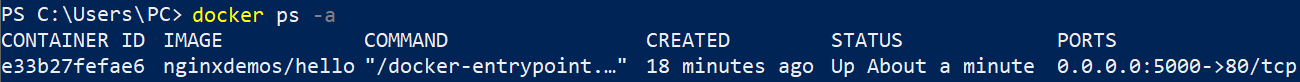




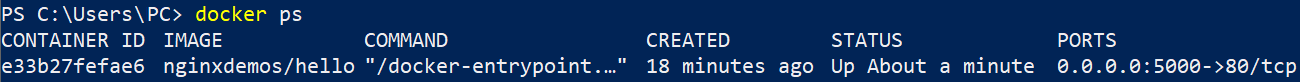
Notice that the **name of our container** is how we explicitly set it to be.

### Examine and Delete Container and Image

We can **see all containers** we have like this:



To **see** **all running containers**, use:



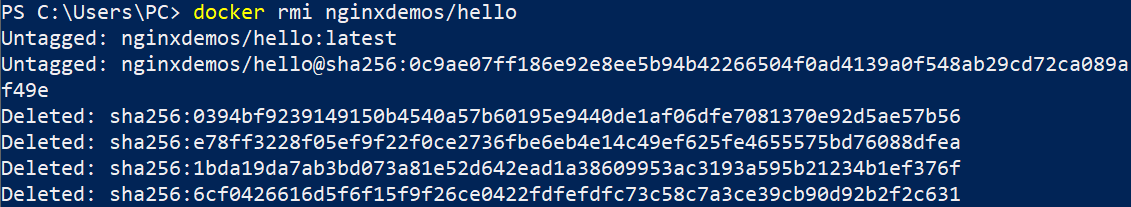
In our case, we have a **single running container** and we have the **same output** from the two commands.

Now let's use the terminal to **stop and delete our container**. Use the following commands with the **container id** or **container name**:

Both commands return the **id or the name of the container**.

Finally, we can also **delete the NGINX server image**:



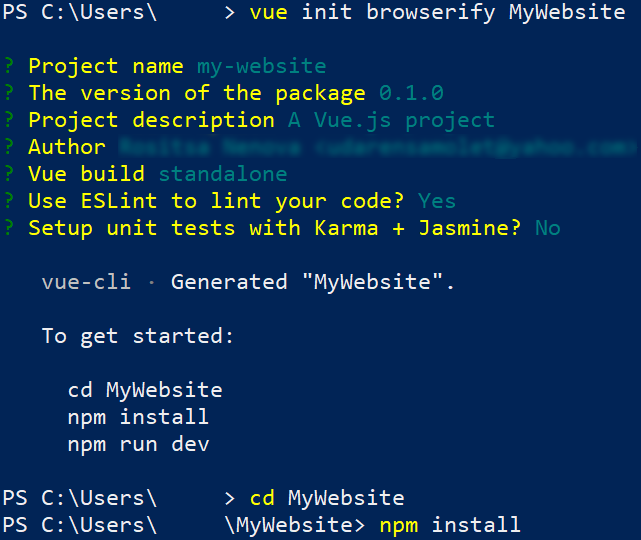
That's how we are supposed to work with **Docker images and containers** at a basic level.

## Vue.js App in Container

Now we will see how to run a **Vue.js app** in a **Docker** **container**. We will **create an app just for the demo**.

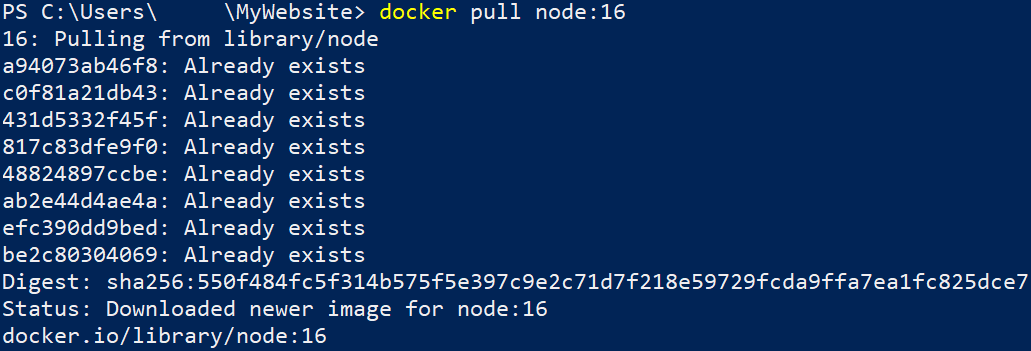
### Create and Set Up a Vue.js App

Let's use Terminal to create an app called **MyWebsite**, in a folder you choose:

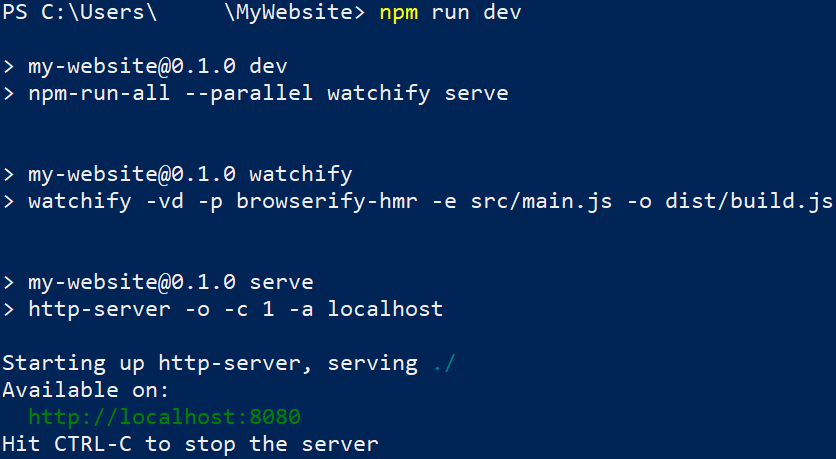


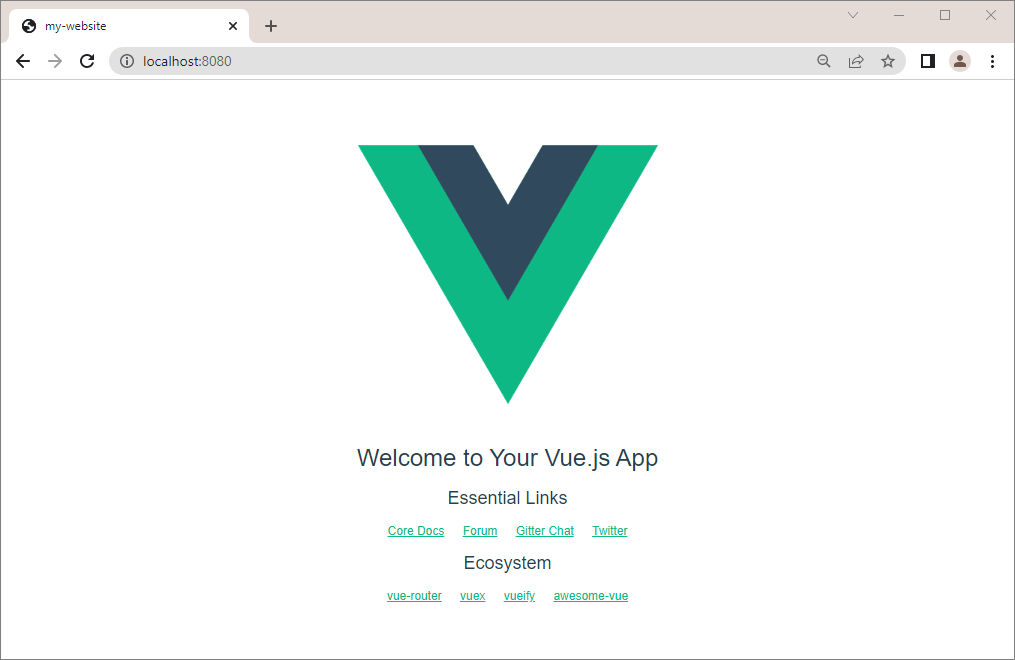
Let's now **pull the image** we will need to run the app – **NodeJs**.

(<https://hub.docker.com/_/node>). Note that the **image version** should be **the** **same** as the **app's Node version**:

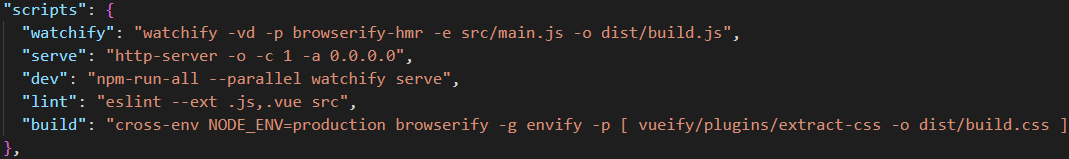


Now let's **run the application locally** in the standard way to check if **everything works as expected**:





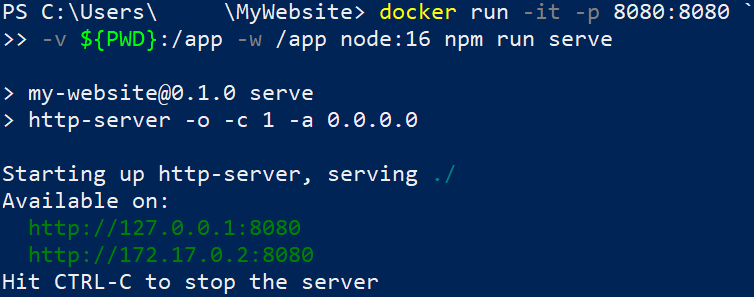
Now we should **modify the app's** **package.json** **file**, so that the **app runs on the IP we want**. **Open the file** in any editor and **change the scripts section** **settings** like this:



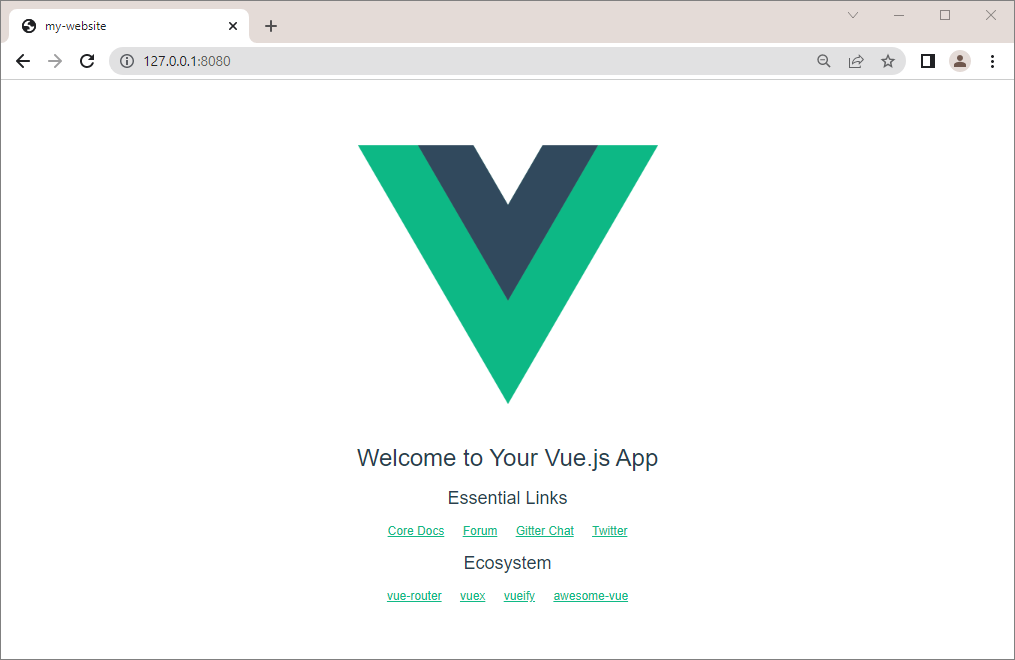
### Run the App in a Container

To run the Vue.js **app in a container**, we will need to **create a container** with an **exposed port**, a **volume** and an **interactive shell**, so that we can **run the app inside the container** with the **docker** **run** command.

To do so, execute the following command:



**Access** the app at [**http://127.0.0.1:8080**](http://127.0.0.1:8080) to validate that the app is running:



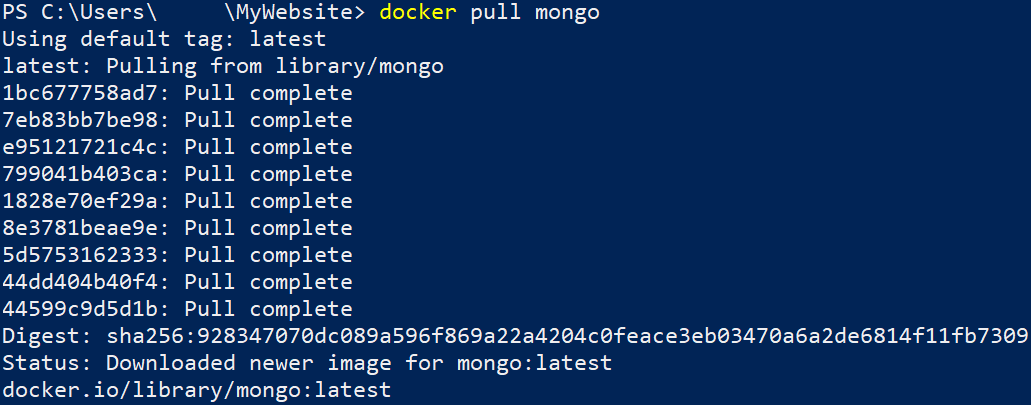
## MongoDB in Container

Our next task is to **run a container** with a **Mongo** **database** in it. To do this, we will need the **following image** from **Docker** **Hub**: <https://hub.docker.com/_/mongo>

You can look at the "**How** **to** **use** **this** **Image**" **section** to learn how to **run the database container**. However, we will also **show and explain** the process step by step.

### Create the Container

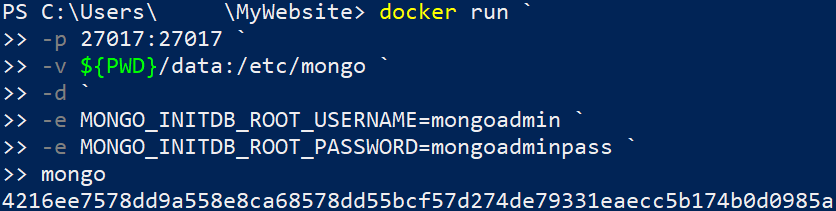
First, pull the latest MongoDB image with the **docker** **pull** **mongo** command:



You can examine the **documentation** on how to use the image: <https://hub.docker.com/_/mongo>

### Run a Database Container

Our next step is to **run** the container, using the following command:



* **docker run** → starts a new Docker container;
* **-p 27017:27017** → sets the external and internal ports to **27017**, so that we can access the MongoDB from outside the container;
* **-v ${PWD}/data:/etc/mongo**
  + ${PWD} → the host directory;
  + **/etc/mongo** → the container directory;
* **-d** → runs the container in detached mode (it will run in the background);
* **-e MONGO\_INITDB\_ROOT\_USERNAME=mongoadmin** → sets the admin username;
* **-e MONGO\_INITDB\_ROOT\_PASSWORD=mongoadminpass** → sets the admin password;
* **mongo** → specifies the image.

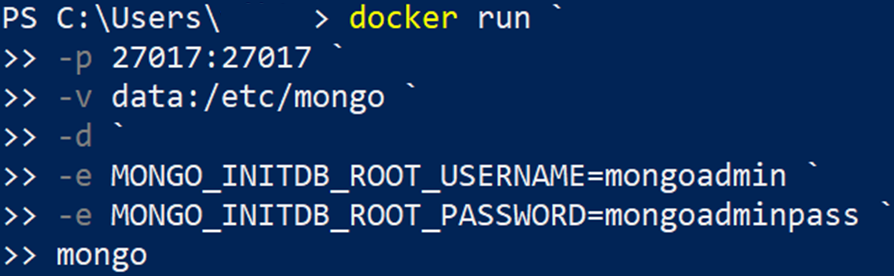
You should disable host's MongoDB Server instances or use another port!

Admin password should always follow the rules from the documentation.

When MongoDB Server container is started, other apps can log in to it and use the database.

**Run a Database Container with Volume**

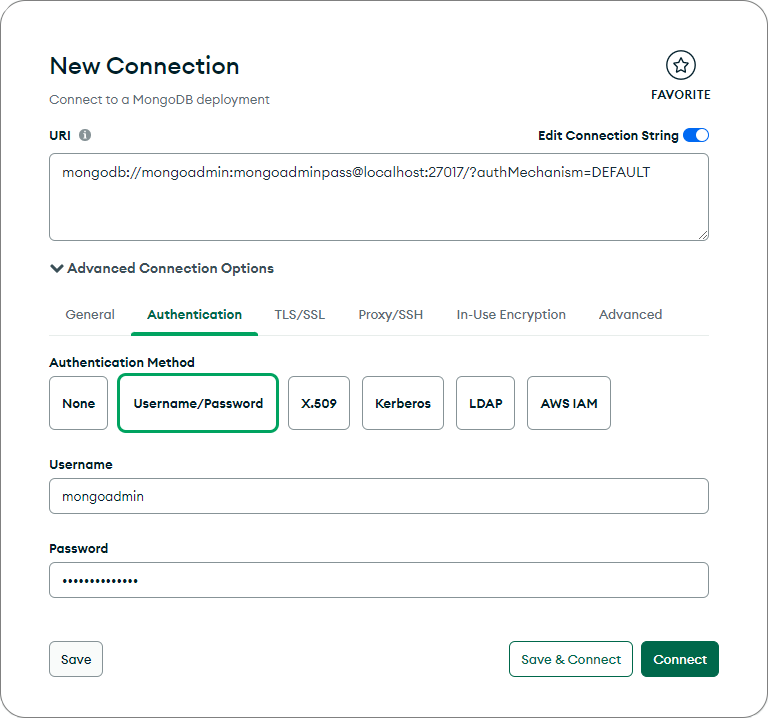
To persist data after container is stopped, **create a volume, using the following command:**

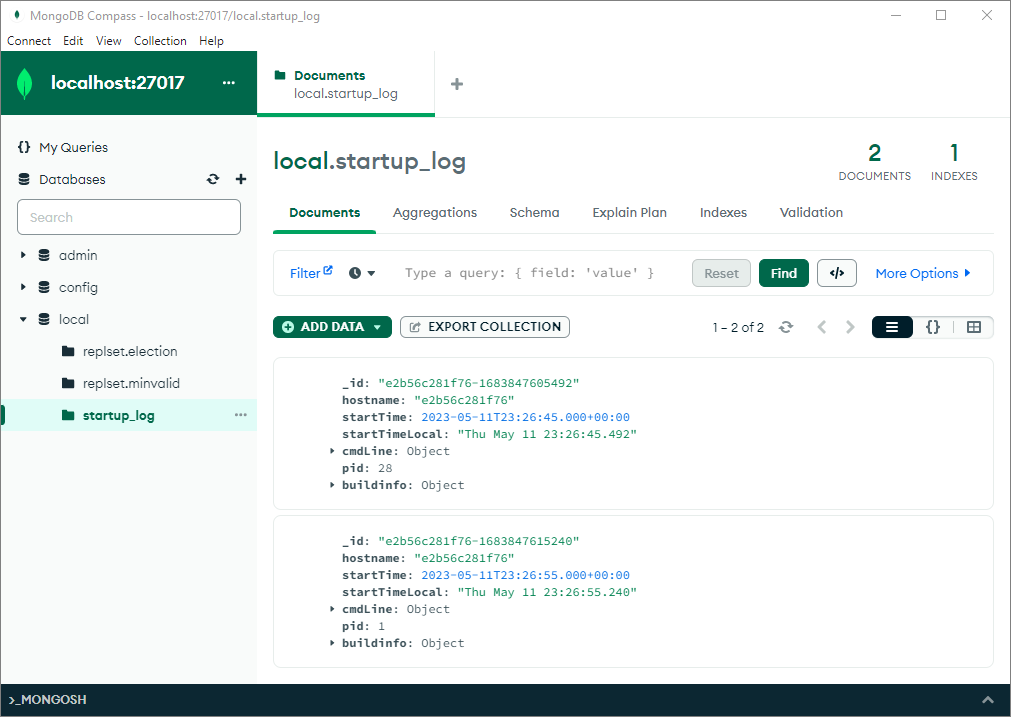


You can then easily **backup** or **restore** the **data** from the volume.

### Connect to the Database Container through MongoDB Compass

You can connect to the container database in **MongoDB Compass,** using the **username** andpassword that we created in the **previous** step**:**

****

****