Lab 2 Report

Rong Guo

C00290890

Catalog

[Use the Docker CLI 3](#_Toc193255621)

[Use Docker Compose 4](#_Toc193255622)

[Run multiple instance of the Postgres database 5](#_Toc193255623)

[Run Postgres container in a controlled network 6](#_Toc193255624)

[Manage the resources 6](#_Toc193255625)

[Override the default CMD and ENTRYPOINT in Docker Compose 6](#_Toc193255626)

[Override the default CMD and ENTRYPOINT with docker run 8](#_Toc193255627)

[Use volumes 8](#_Toc193255628)

[View volume contents 10](#_Toc193255629)

[Remove volumes 10](#_Toc193255630)

[Run a container 11](#_Toc193255631)

[Use a bind mount 12](#_Toc193255632)

[Access the file on the Docker Desktop Dashboard 13](#_Toc193255633)

[Stop your container 14](#_Toc193255634)

[Set up 15](#_Toc193255635)

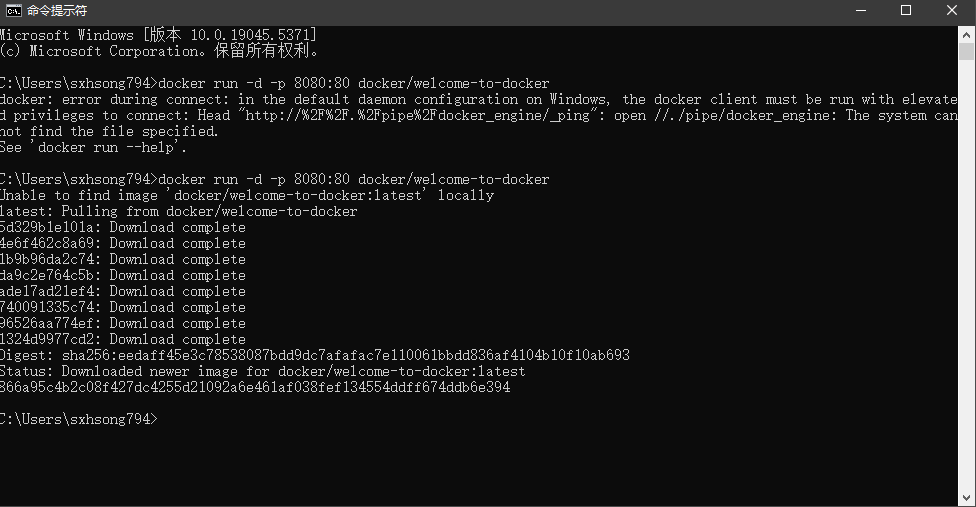
[Build the images 15](#_Toc193255636)

[Run the containers 16](#_Toc193255637)

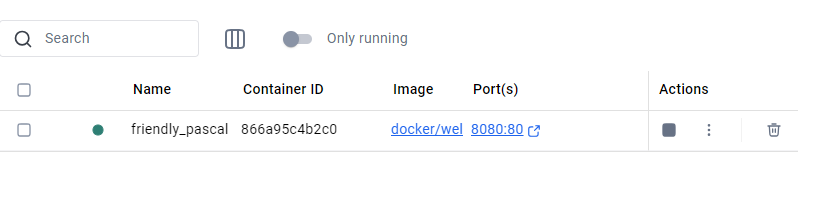
[Simplify the deployment using Docker Compose 17](#_Toc193255638)

Use the Docker CLI

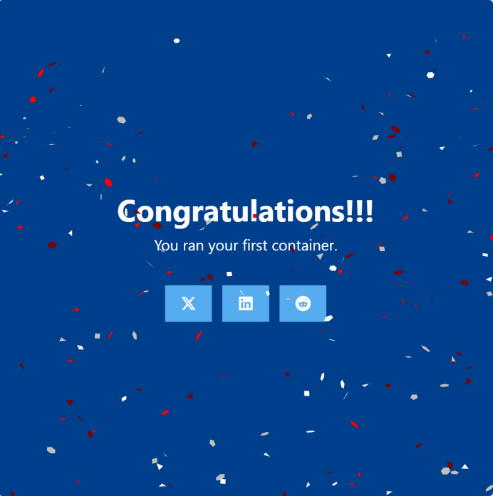
1. In a terminal, run the following command to start a new container



1. Verify the published port by going to the Containers view of the Docker Desktop Dashboard.

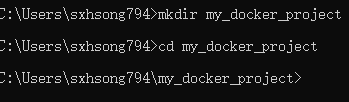


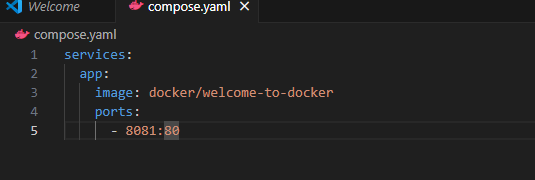
1. Open the website by either selecting the link in the Port(s) column of your container or visiting http://localhost:8080 in your browser.



Use Docker Compose

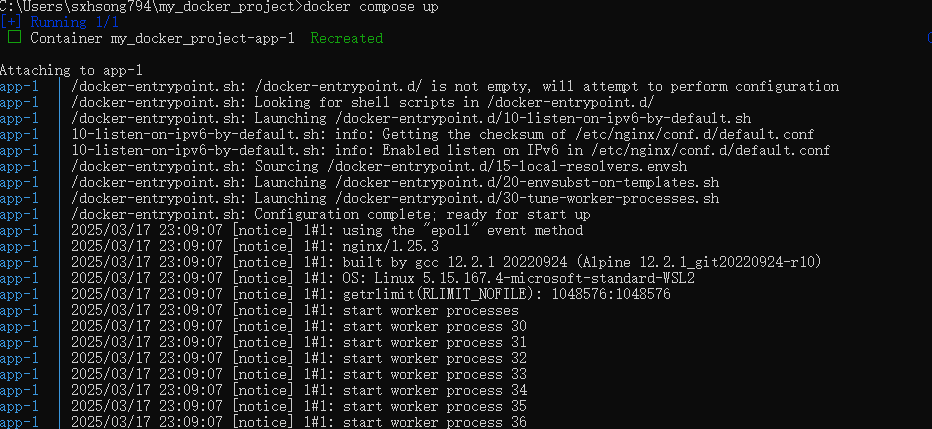
1. Create a new directory and inside that directory, create a compose.yaml



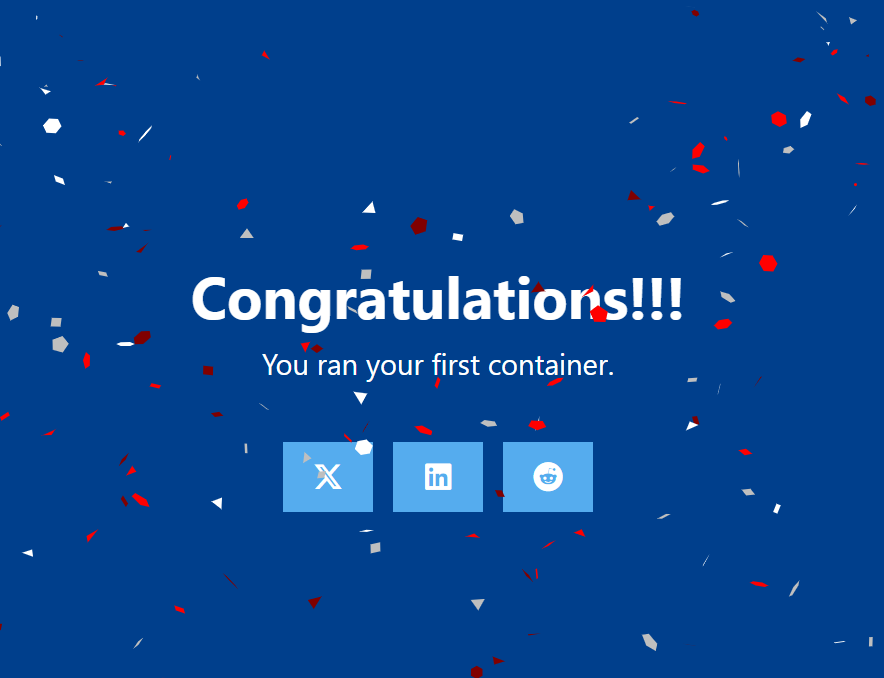


2. Open a terminal and navigate to the directory you created in the previous step.

3. Use the docker compose up command to start the application.

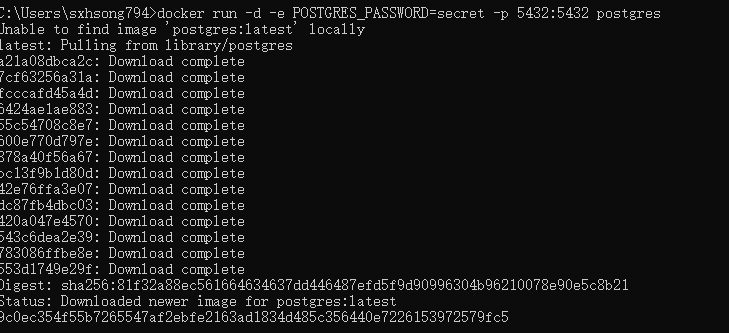


1. Open your browser to http://localhost:8081.



Run multiple instance of the Postgres database

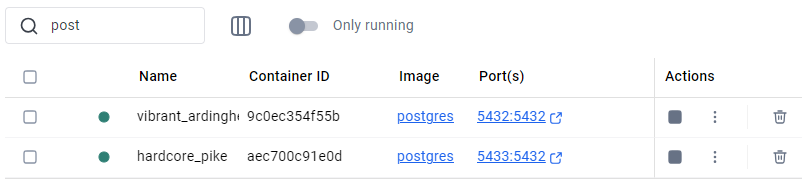
1. Start a container using the Postgres image. This will start the Postgres database in the background, listening on the standard container port 5432 and mapped to port 5432 on the host machine.



1. Start a second Postgres container mapped to a different port. This will start another Postgres container in the background, listening on the standard Postgres port 5432 in the container, but mapped to port 5433 on the host machine. I override the host port just to ensure that this new container doesn't conflict with the existing running container.



1. Verify that both containers are running by going to the Containers view in the Docker Desktop Dashboard.

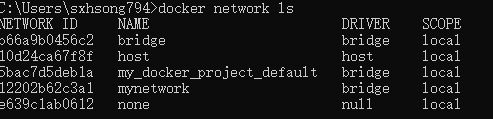


Run Postgres container in a controlled network

1. Create a new custom network



1. Verify the network. This command lists all networks, including the newly created "mynetwork".



1. Connect Postgres to the custom network. This will start Postgres container in the background, mapped to the host port 5434 and attached to the mynetwork network.



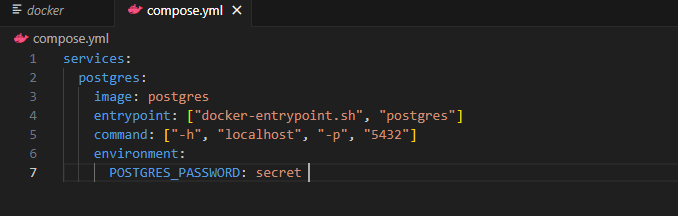
Manage the resources

1. This is where the docker run command shines again. It offers flags like --memory and --cpus to restrict how much CPU and memory a container can use.

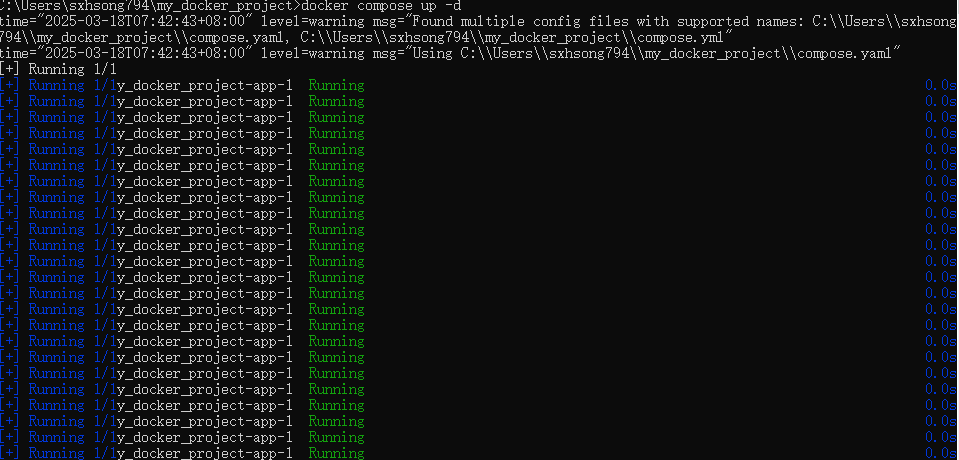


Override the default CMD and ENTRYPOINT in Docker Compose

1. Create a compose.yml file

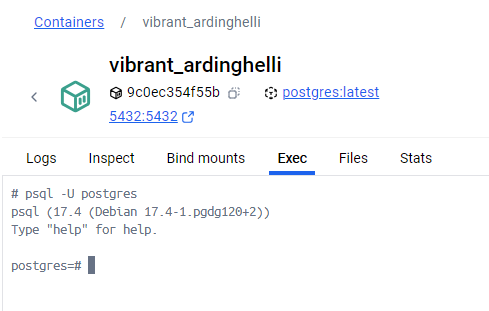


1. Bring up the service



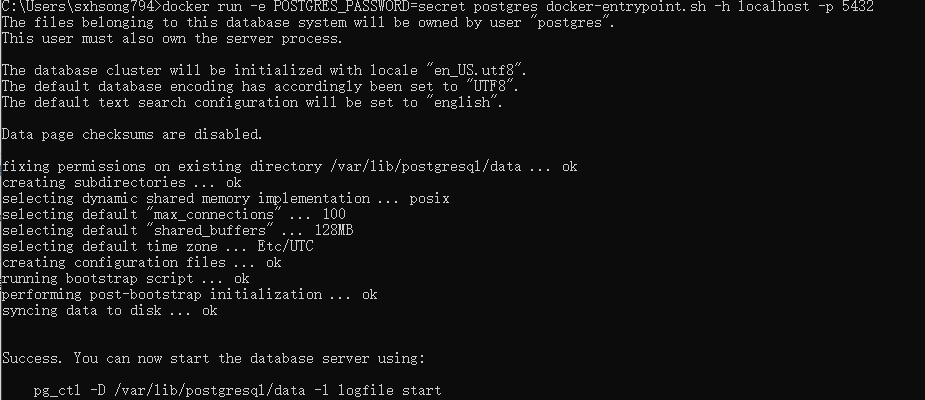
1. Verify the authentication with Docker Desktop Dashboard.

Open the Docker Desktop Dashboard, select the Postgres container and select Exec to enter into the container shell.



Override the default CMD and ENTRYPOINT with docker run

1. This command runs a Postgres container, sets an environment variable for password authentication, overrides the default startup commands and configures hostname and port mapping.

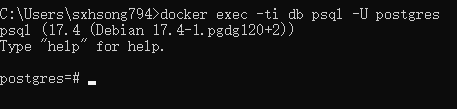


Use volumes

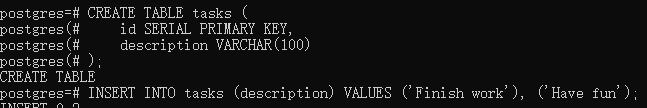
1. Start a container using the Postgres image. This will start the database in the background, configure it with a password, and attach a volume to the directory PostgreSQL will persist the database files.



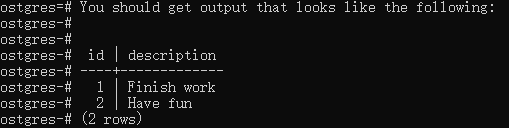
1. Connect to the database



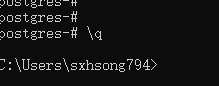
1. In the PostgreSQL command line



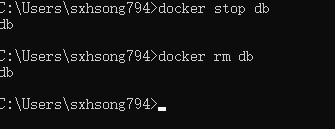
1. Verify the data is in the database



1. Exit out of the PostgreSQL shell



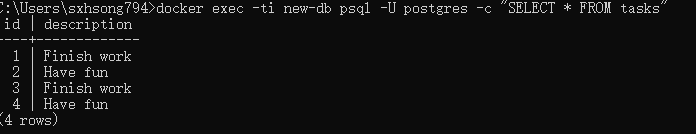
1. Stop and remove the database container. Remember that, even though the container has been deleted, the data is persisted in the postgres\_data volume



1. Start a new container by running the following command, attaching the same volume with the persisted data. It have noticed that the POSTGRES\_PASSWORD environment variable has been omitted. That’s because that variable is only used when bootstrapping a new database

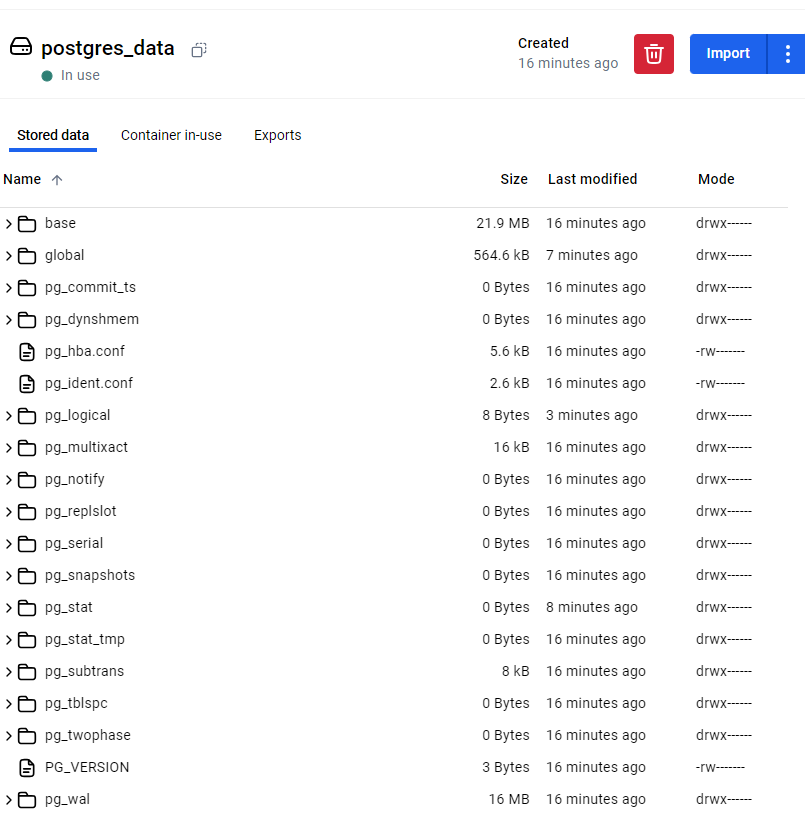


1. Verify the database still has the records



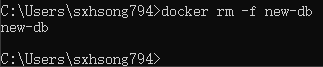
View volume contents

1. The Docker Desktop Dashboard provides the ability to view the contents of any volume, as well as the ability to export, import, and clone volumes.

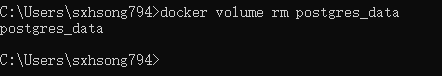


Remove volumes

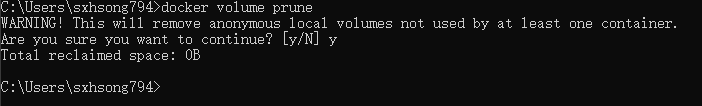
1. Run docker rm -f new-db



1. Select the **Delete Volume** option on a volume in the Docker Desktop Dashboard.

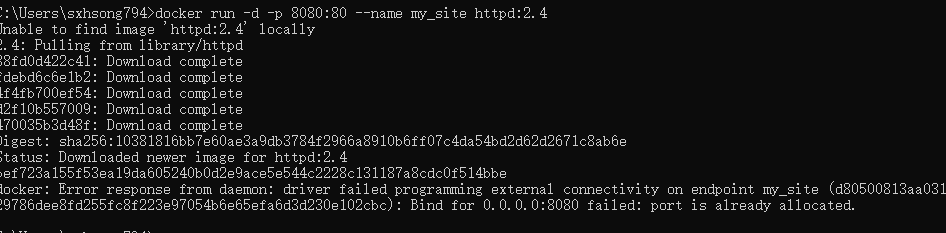


1. Use the docker volume prune command to remove all unused volumes

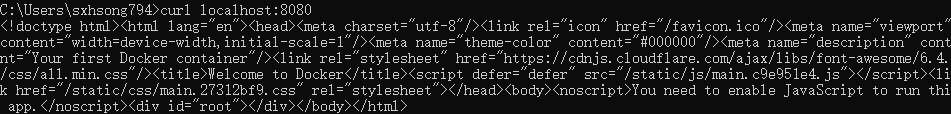


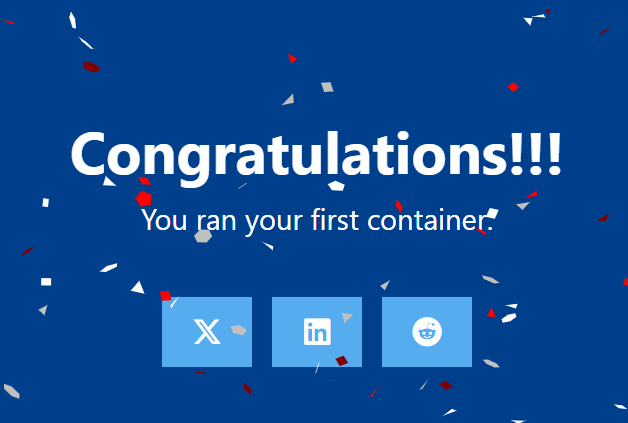
Run a container

1. Start a container using the httpd image. This will start the httpd service in the background, and publish the webpage to port 8080 on the host.



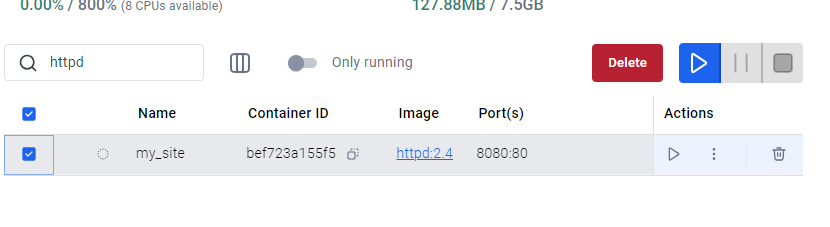
1. Open the browser and access http://localhost:8080



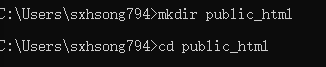


Use a bind mount

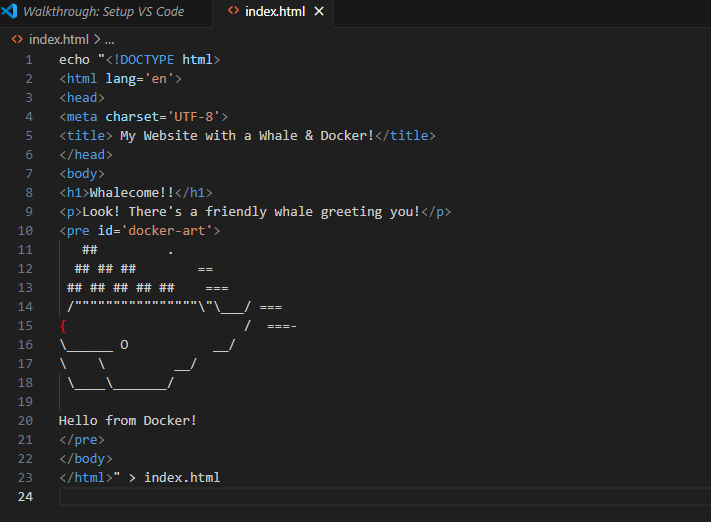
1. Delete the existing container by using the Docker Desktop Dashboard



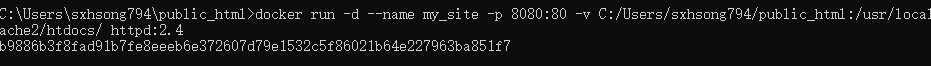
1. Create a new directory called public\_html on your host system.



1. Navigate into the newly created directory public\_html and create a file called index.html with the following content. This is a basic HTML document that creates a simple webpage that welcomes you with a friendly whale.

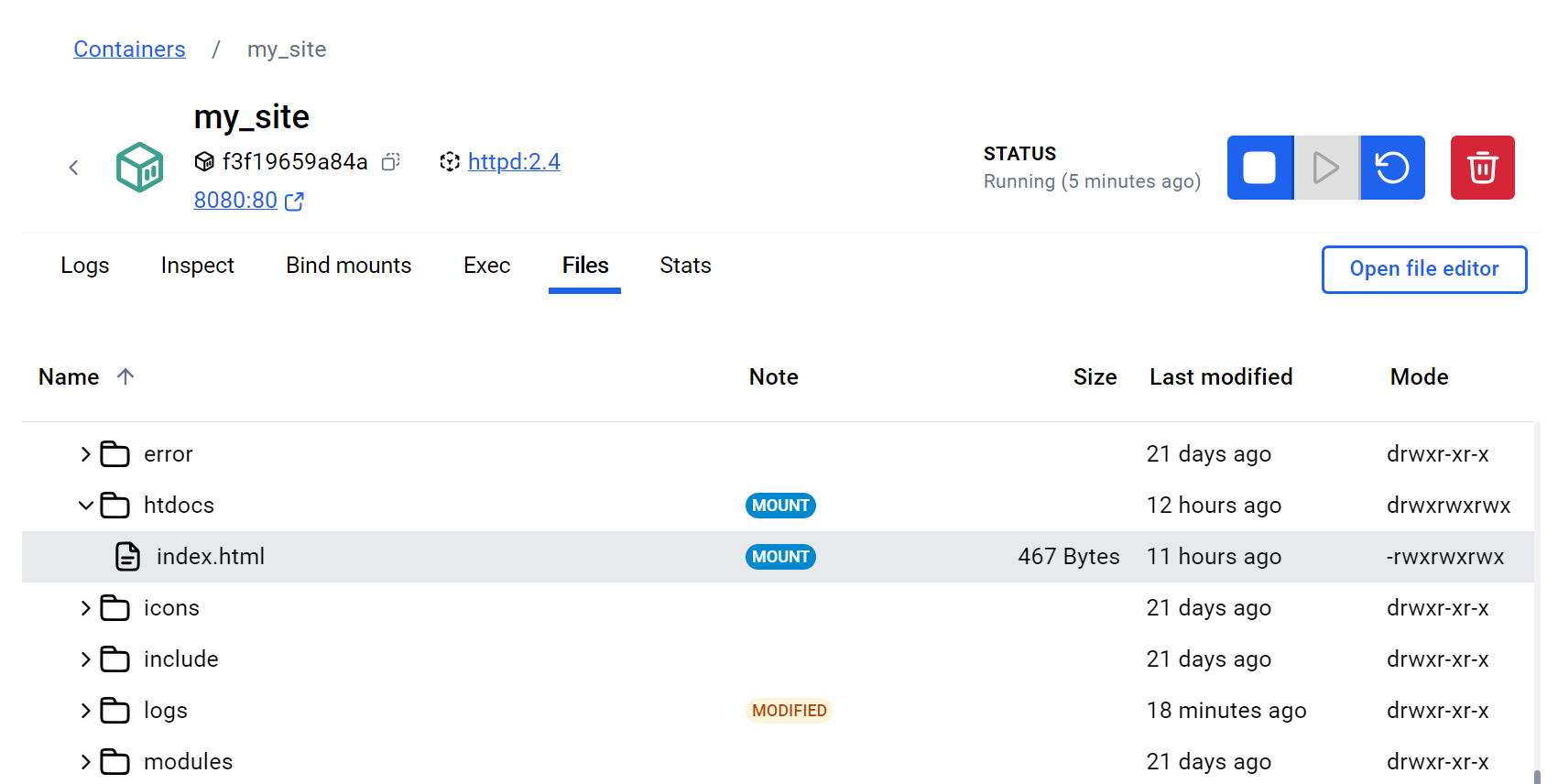


1. It's time to run the container. The --mount and -v examples produce the same result. You can't run them both unless you remove the my\_site container after running the first one

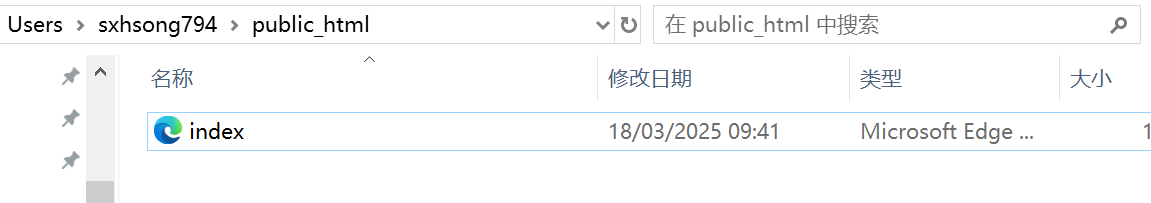


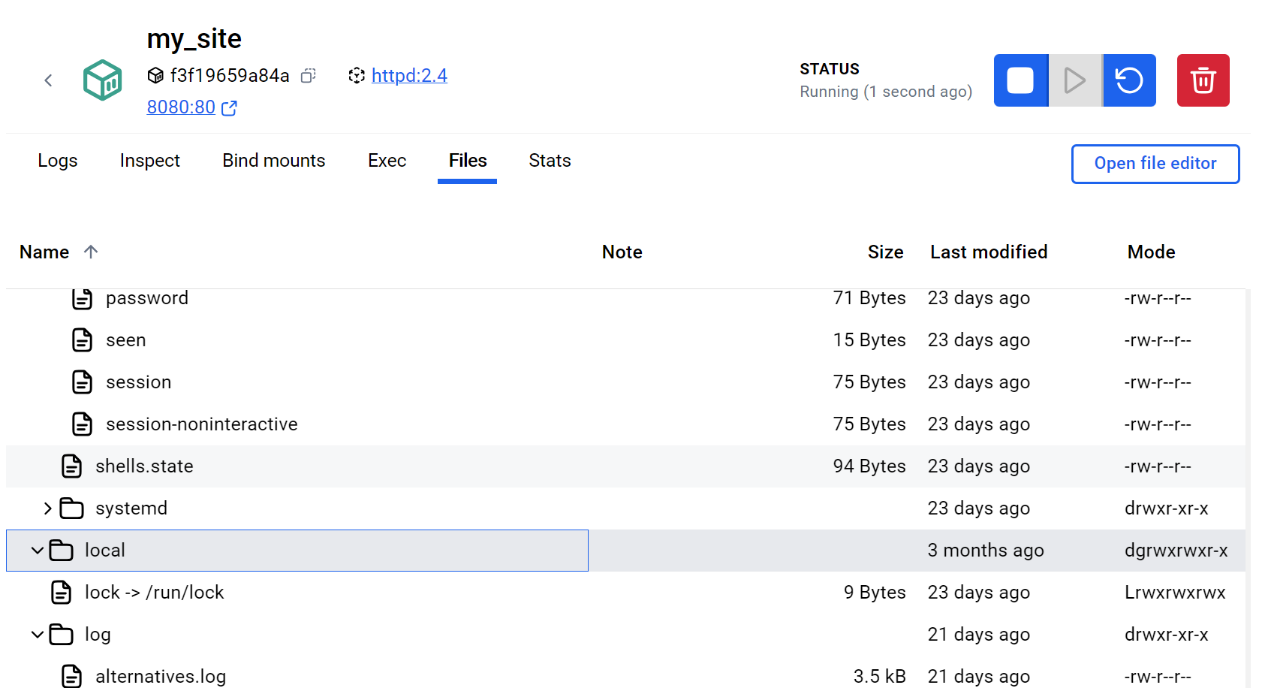
Access the file on the Docker Desktop Dashboard

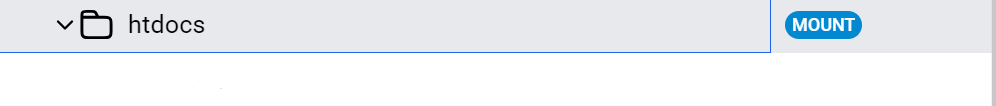
1. View the mounted files inside a container by selecting the container's Files tab and then selecting a file inside the /usr/local/apache2/htdocs/ directory. Then, select Open file editor.



1. Delete the file on the host and verify the file is also deleted in the container. You will find that the files no longer exist under Files in the Docker Desktop Dashboard.

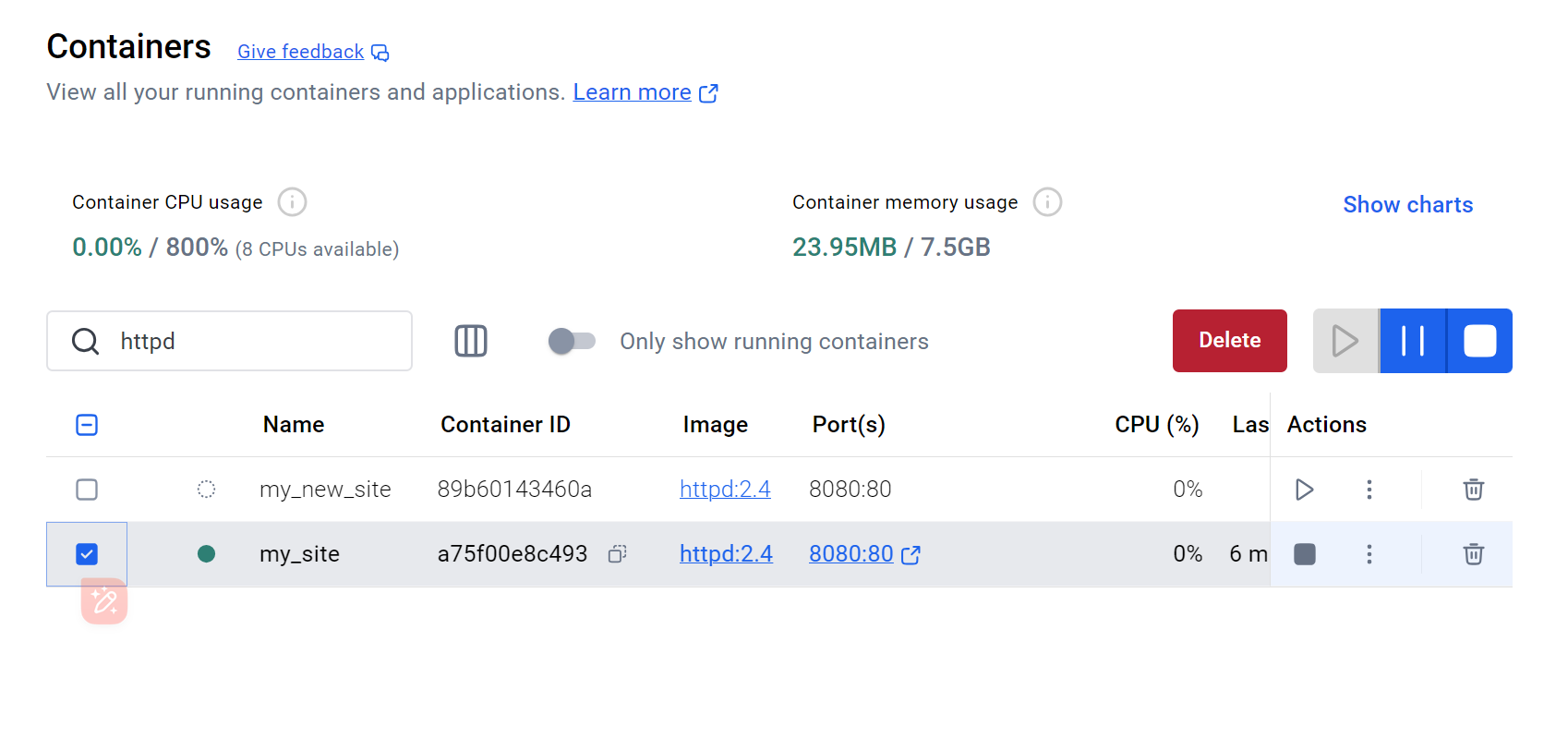






Stop your container

1. Go to the Containers view in the Docker Desktop Dashboard.
2. Locate the container I'd like to stop.
3. Select the Delete action in the Actions column.

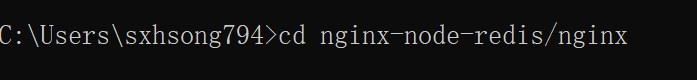


Set up

1. Get the sample application. If you have Git, you can clone the repository for the sample application. Otherwise, you can download the sample application. Choose one of the following options.

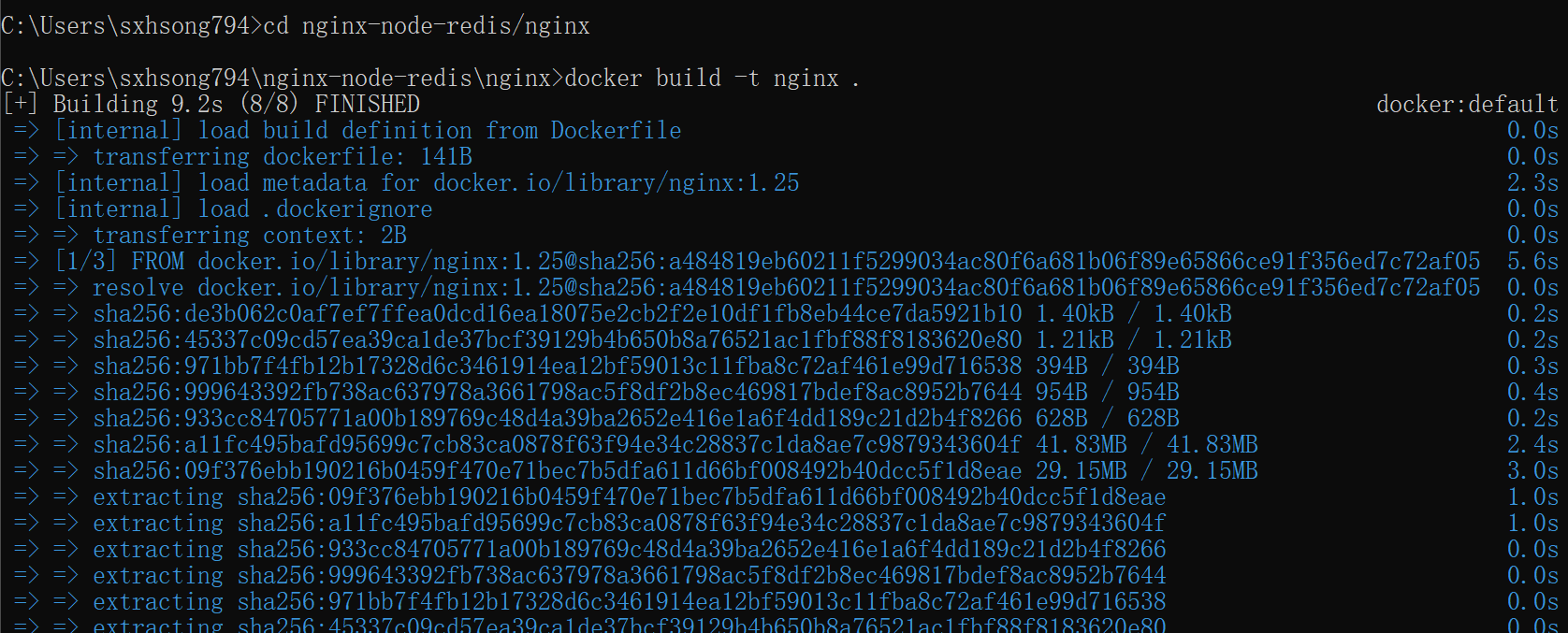


1. Navigate into the nginx-node-redis directory

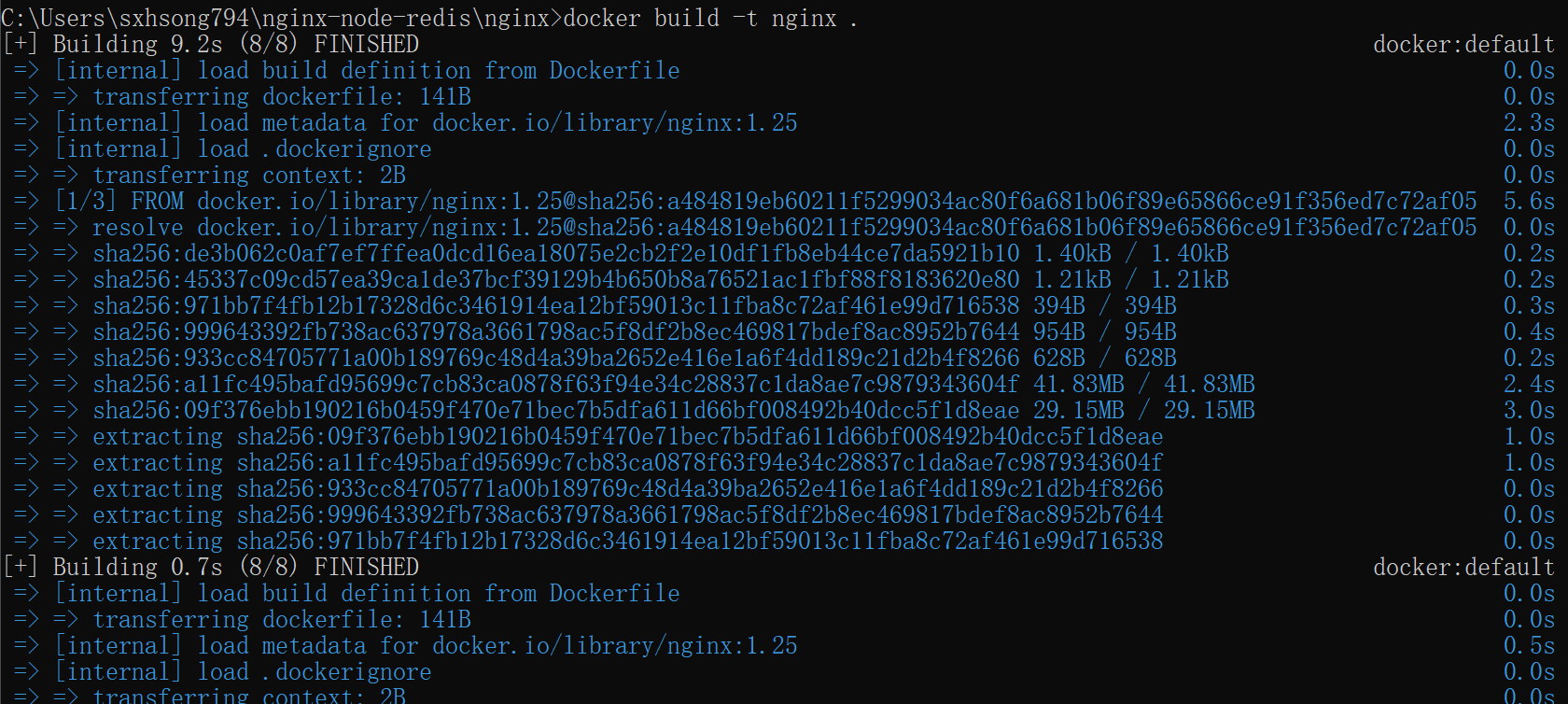


Build the images

1. Navigate into the nginx directory to build the image

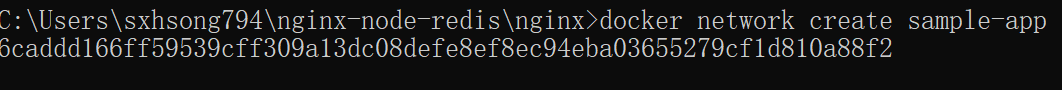


1. Navigate into the web directory and run the following command to build the first web image

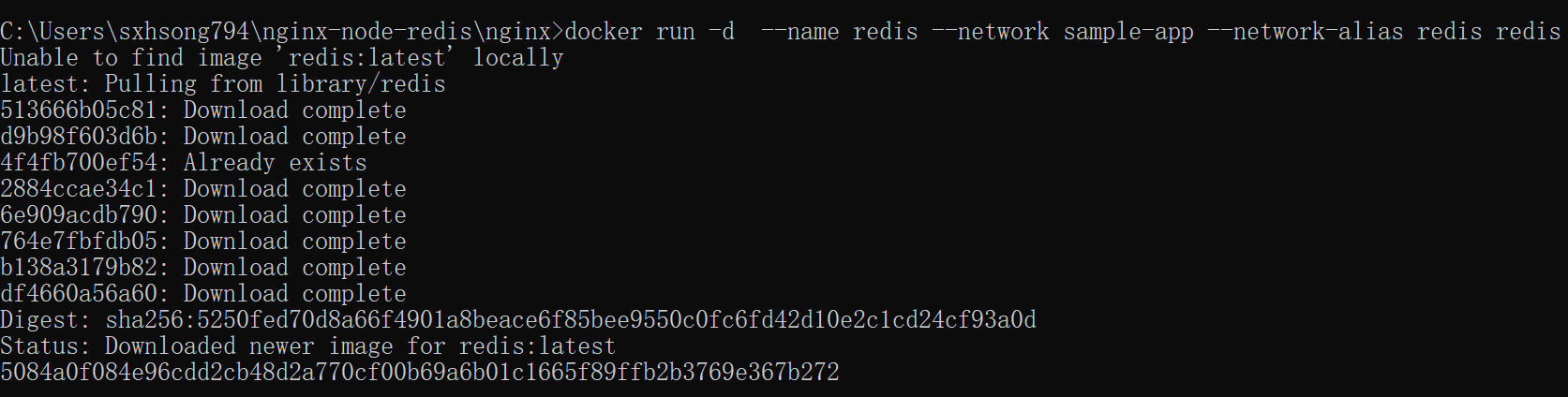


Run the containers

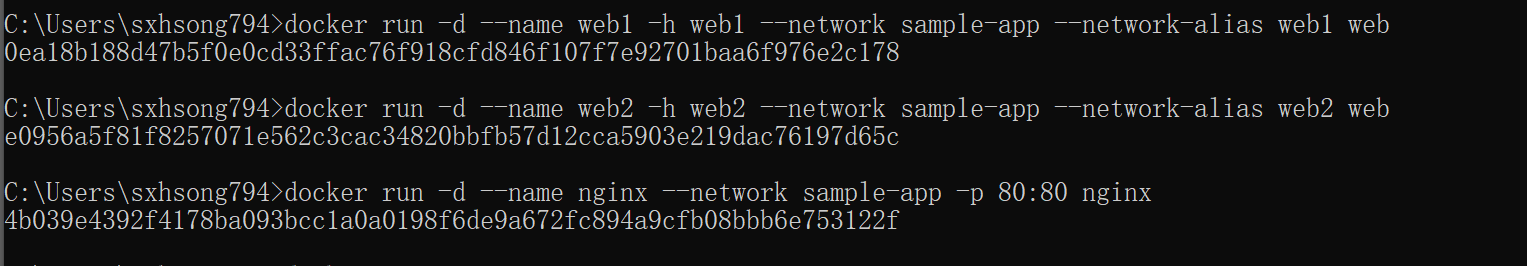
1. Before you can run a multi-container application, you need to create a network for them all to communicate through.



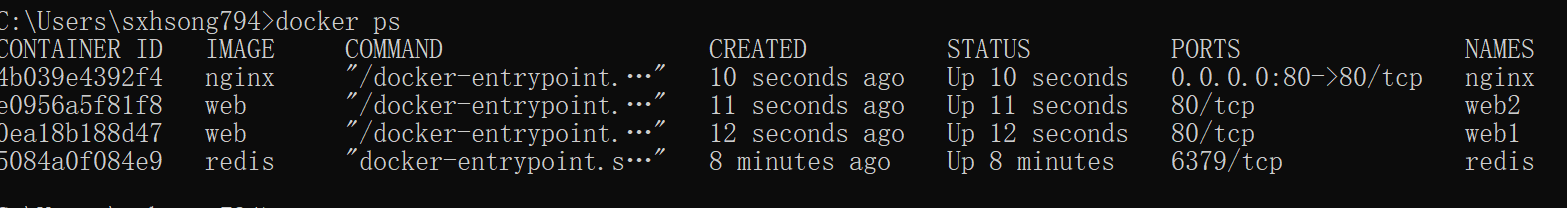
1. Start the Redis container by running the following command, which will attach it to the previously created network and create a network alias (useful for DNS lookups)



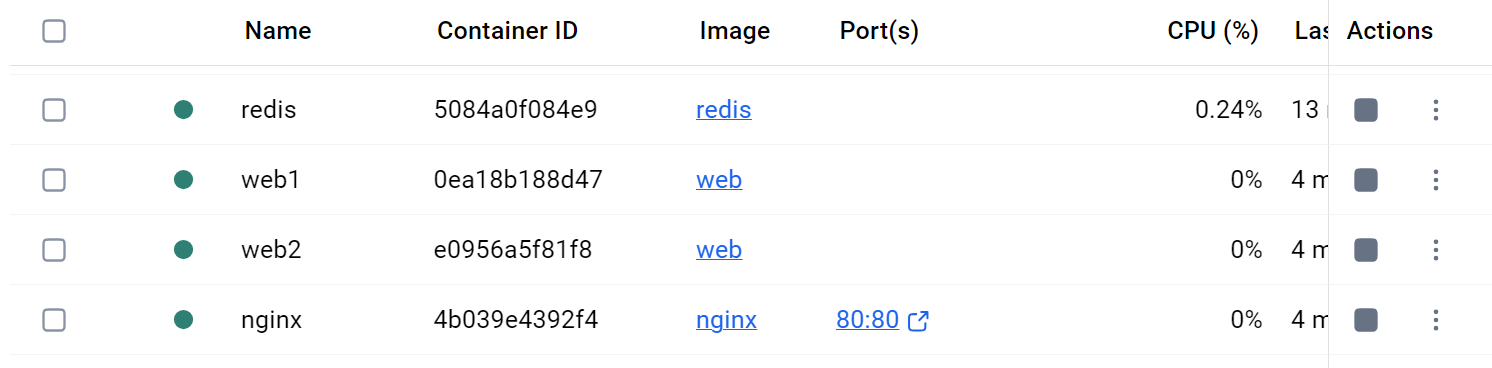
1. Start the first web container, second web container and the Nginx container



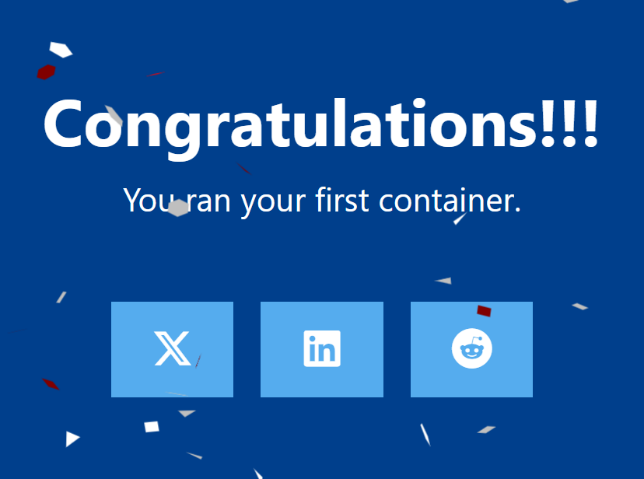
1. Verify the containers are up



1. See the containers and dive deeper into their configuration.



1. With everything up and running, you can open http://localhost:8080 in your browser to see the site.



Simplify the deployment using Docker Compose

1. Use the docker compose up

