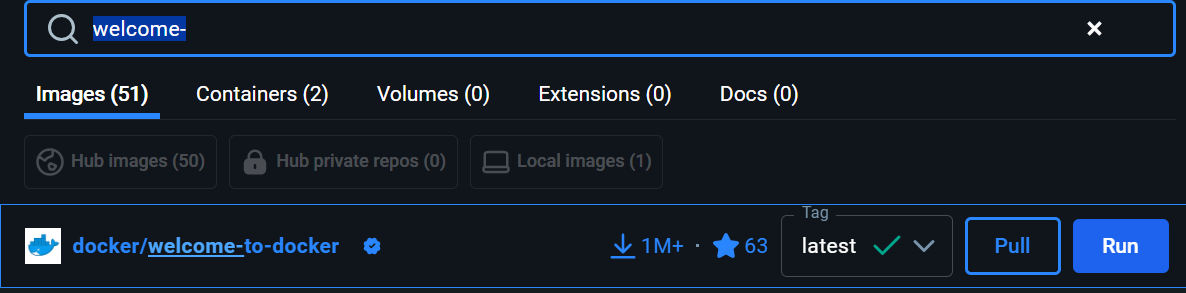
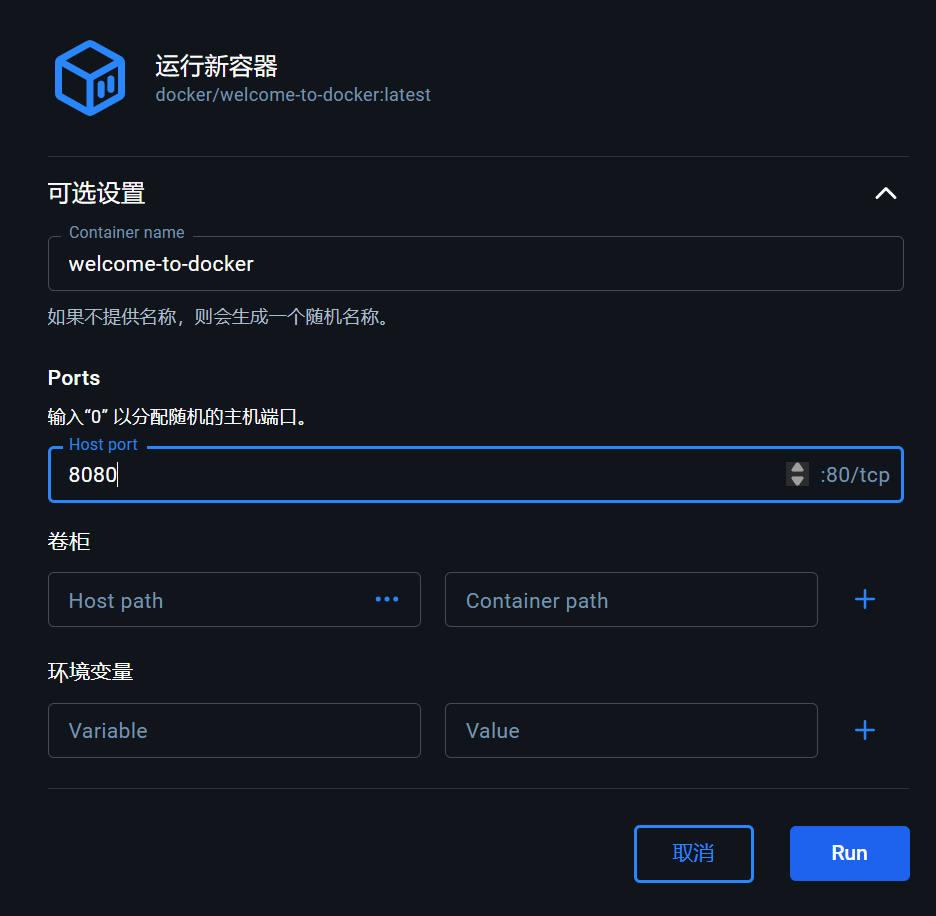
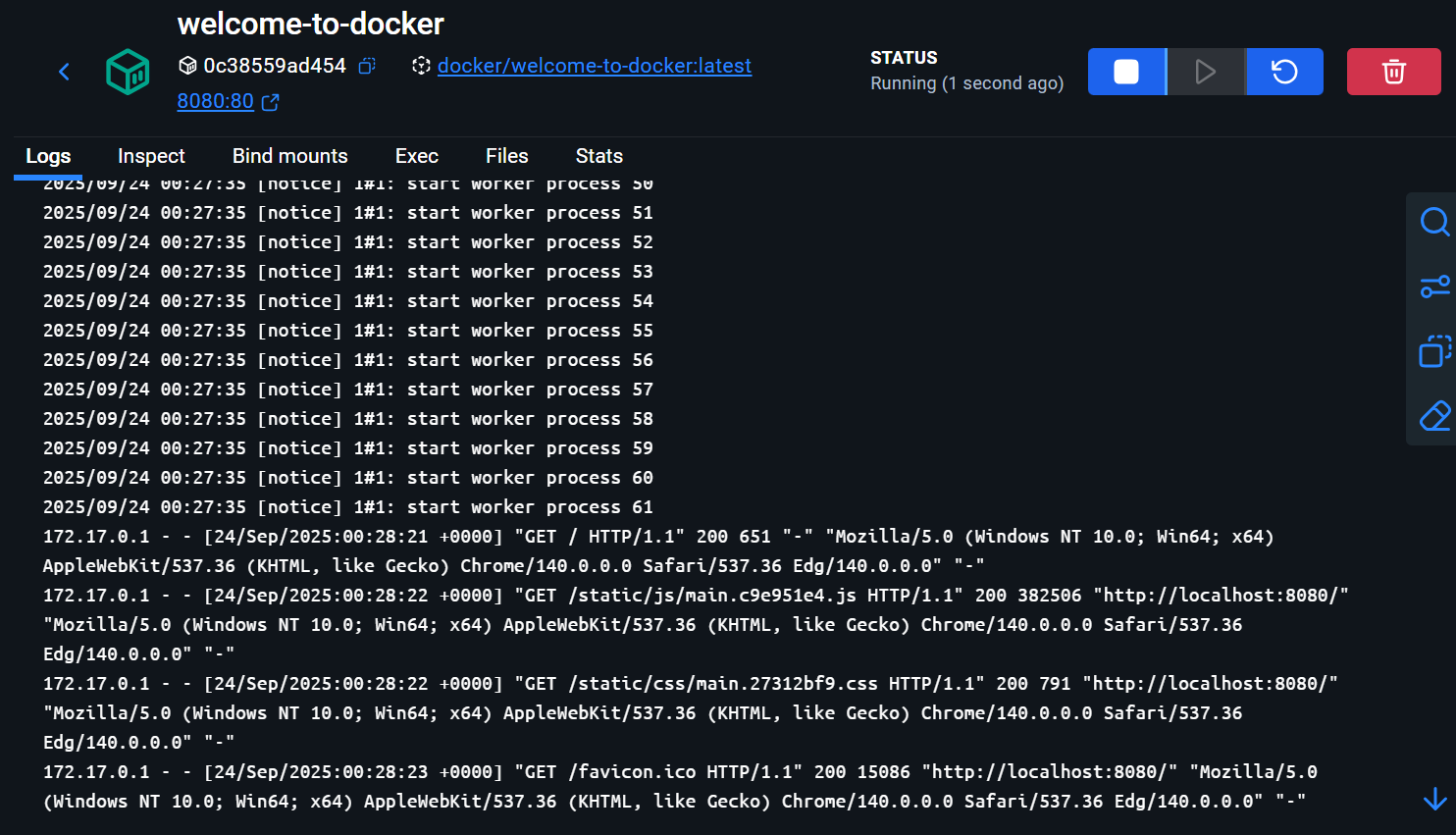
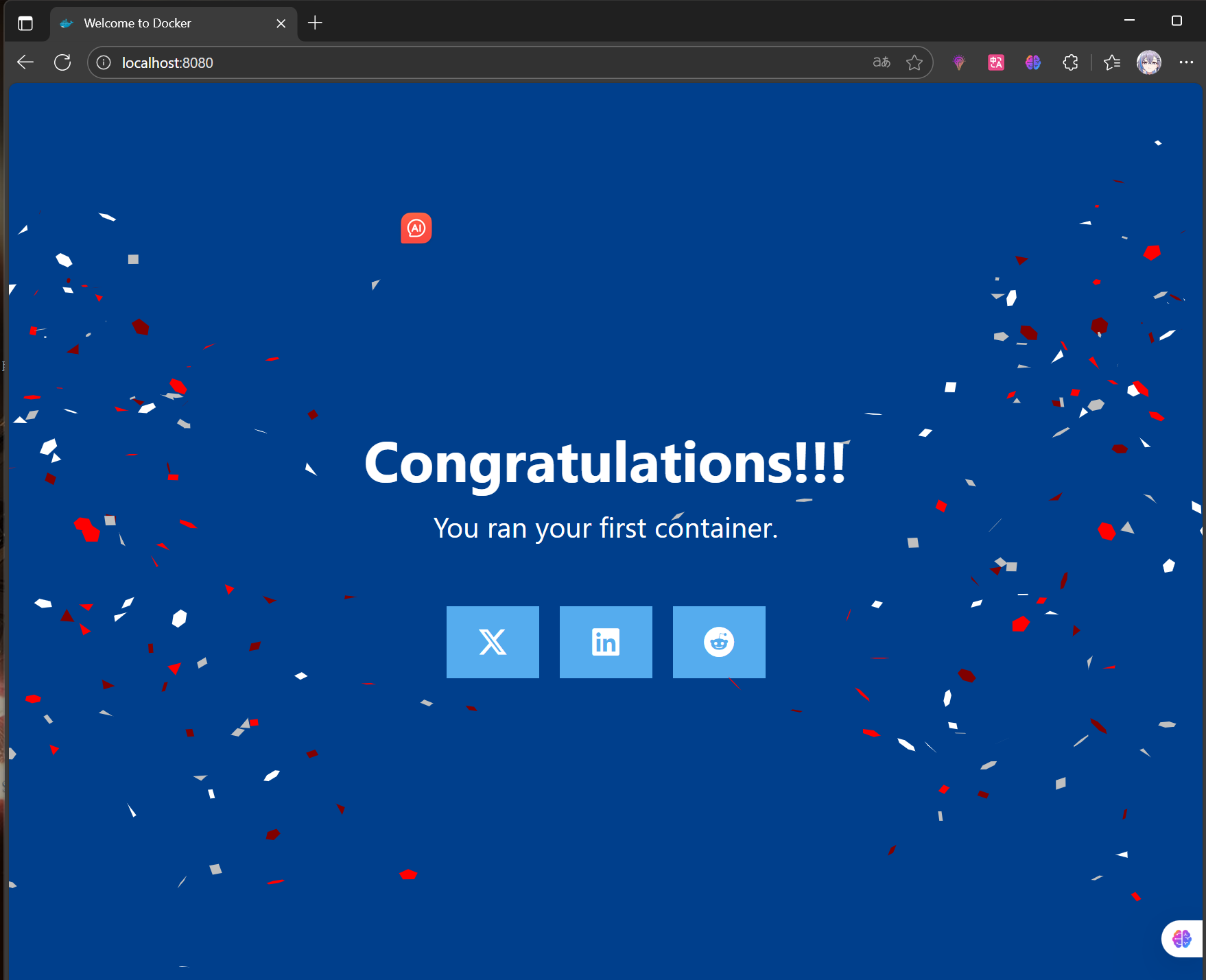
Search for containers

Deploy containers

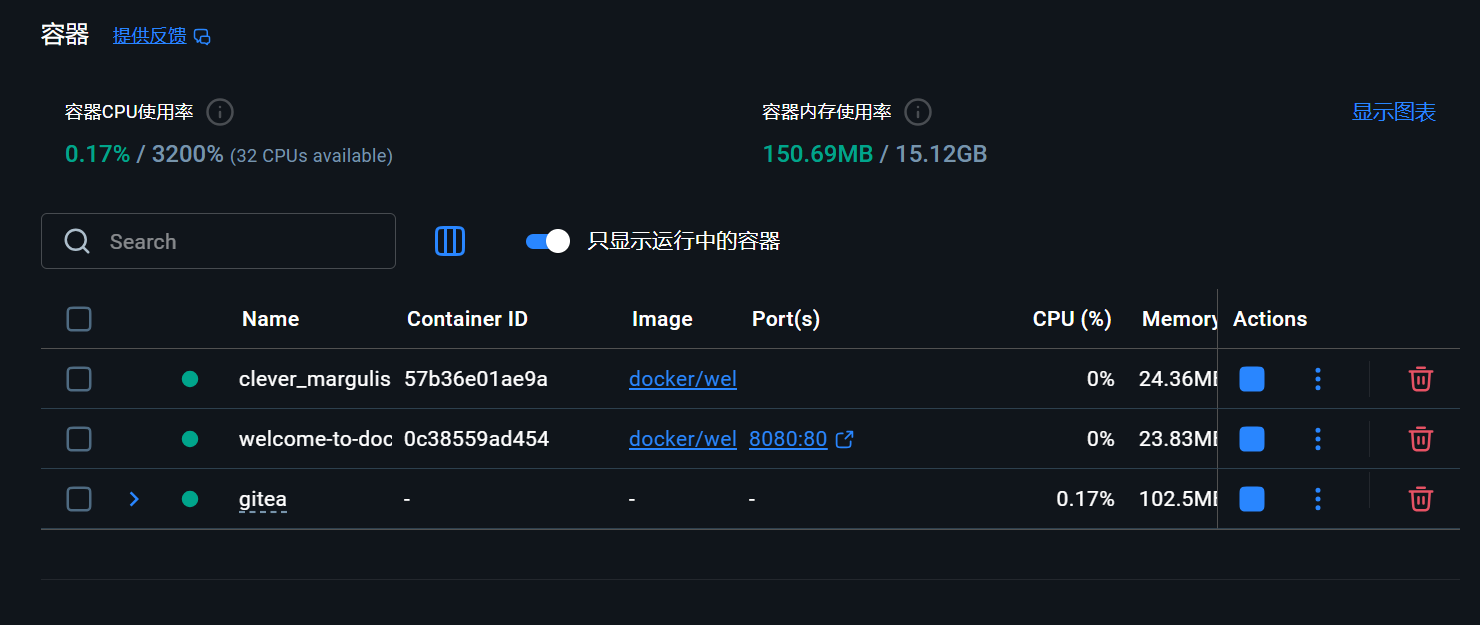


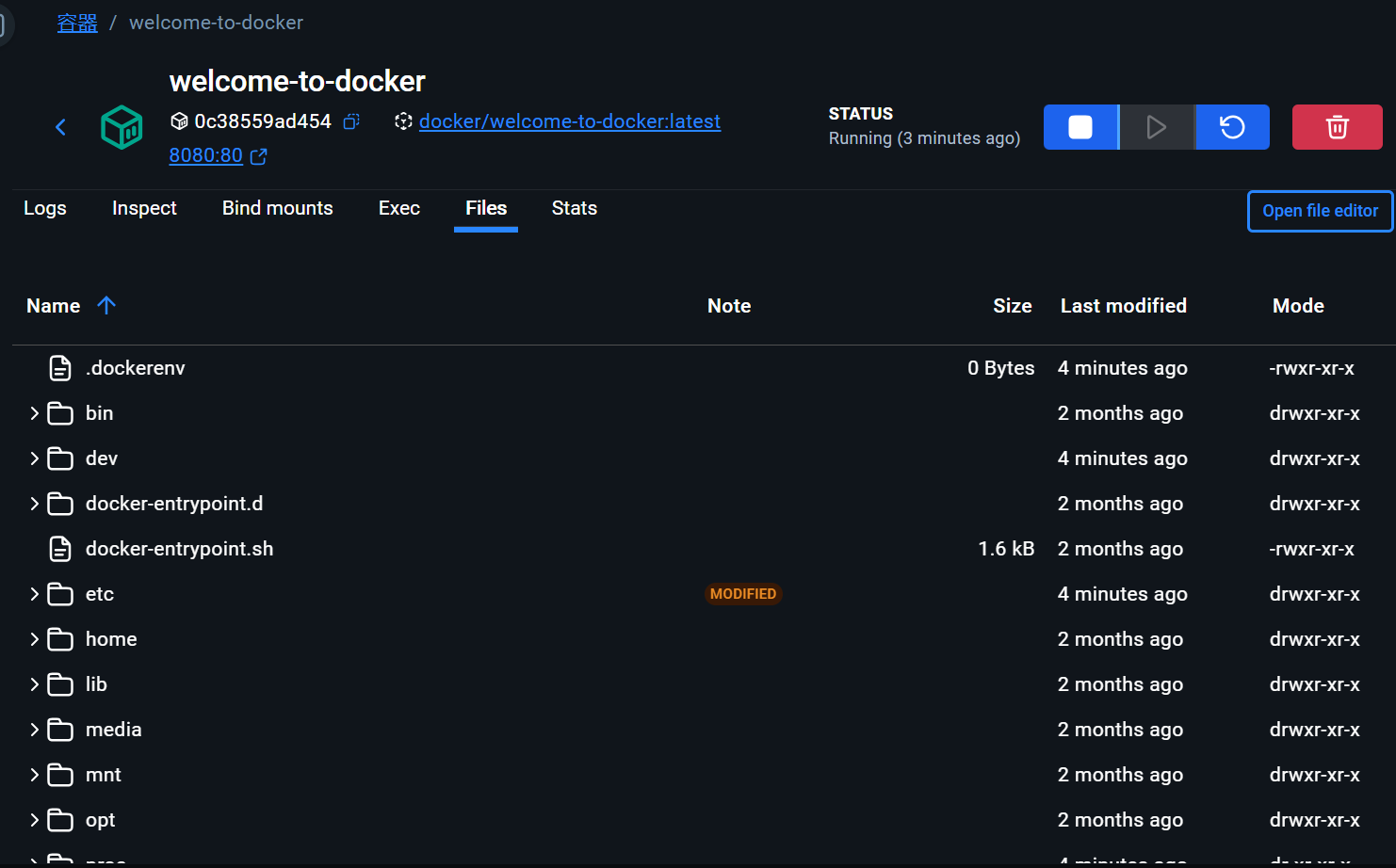


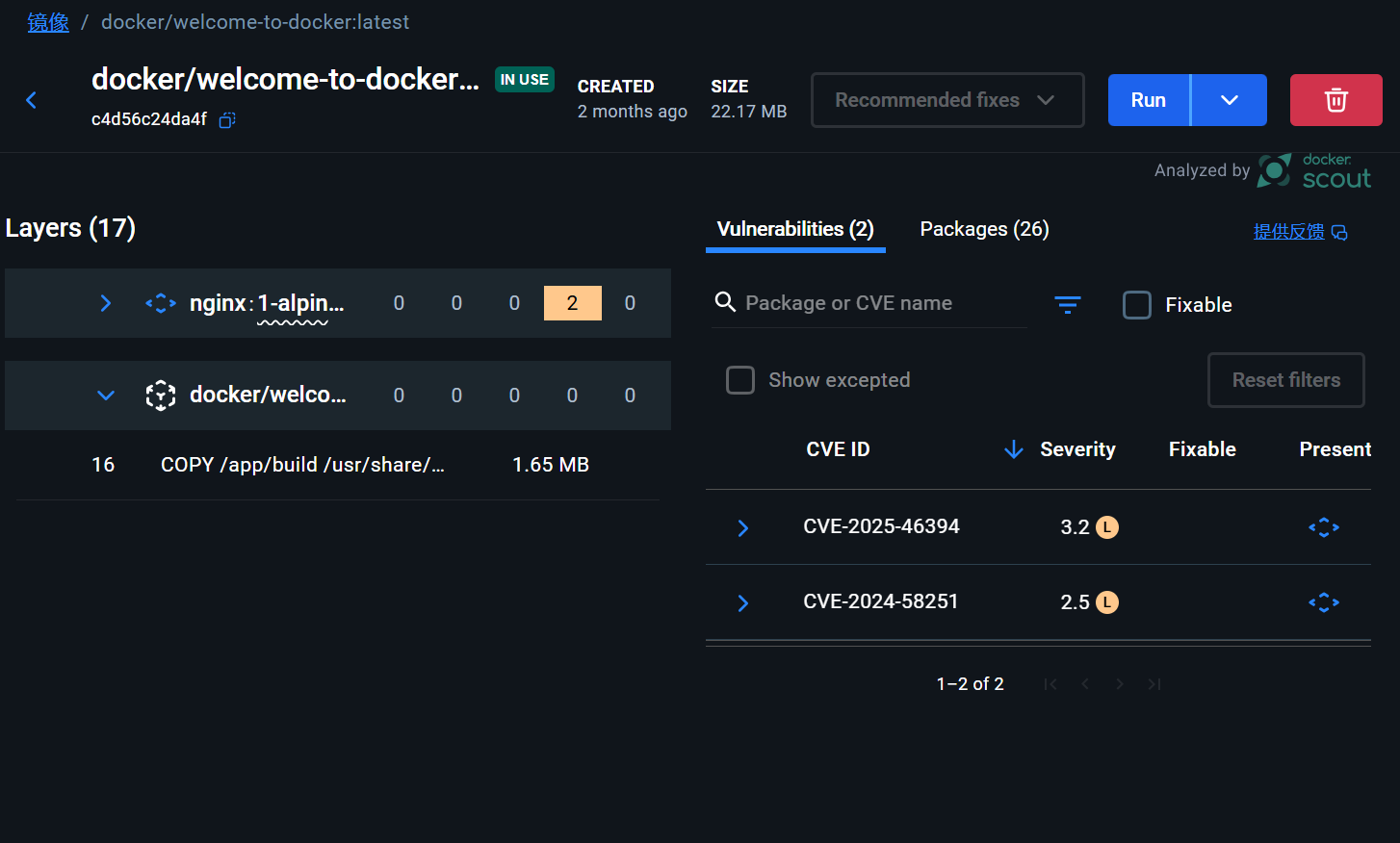
Access port

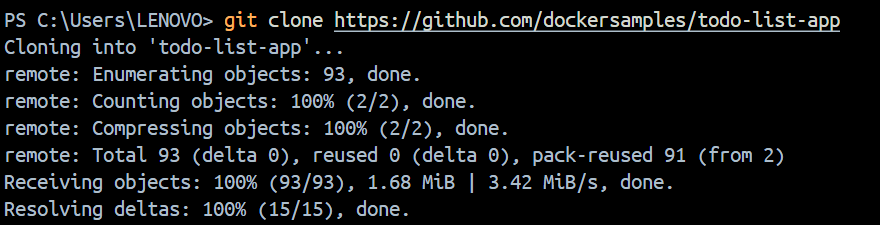


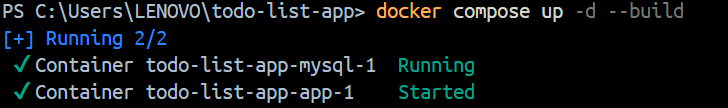
Container interface





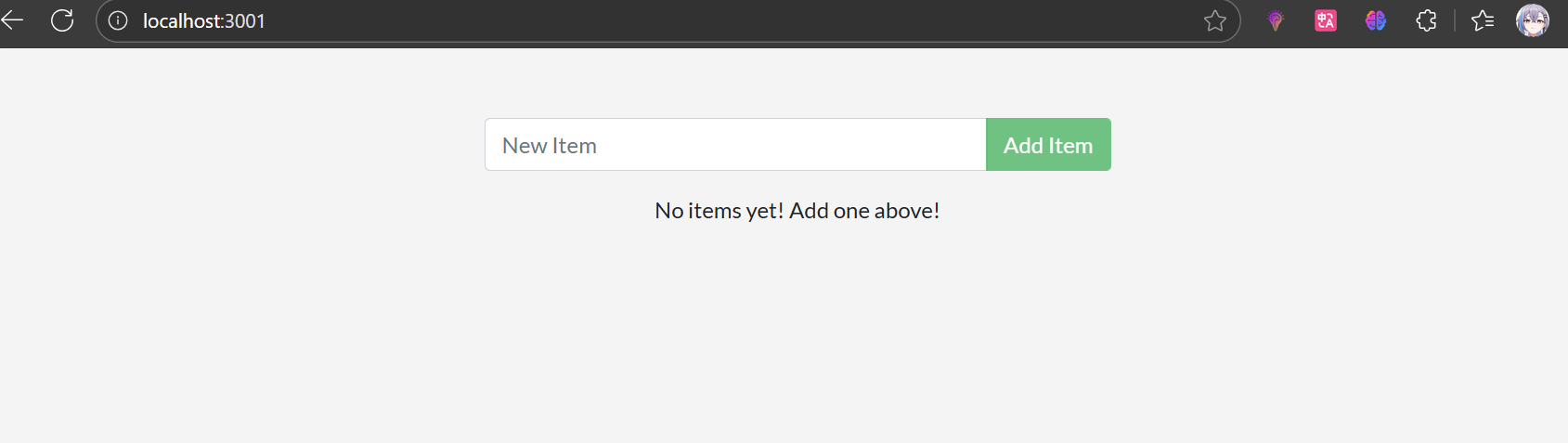


clone

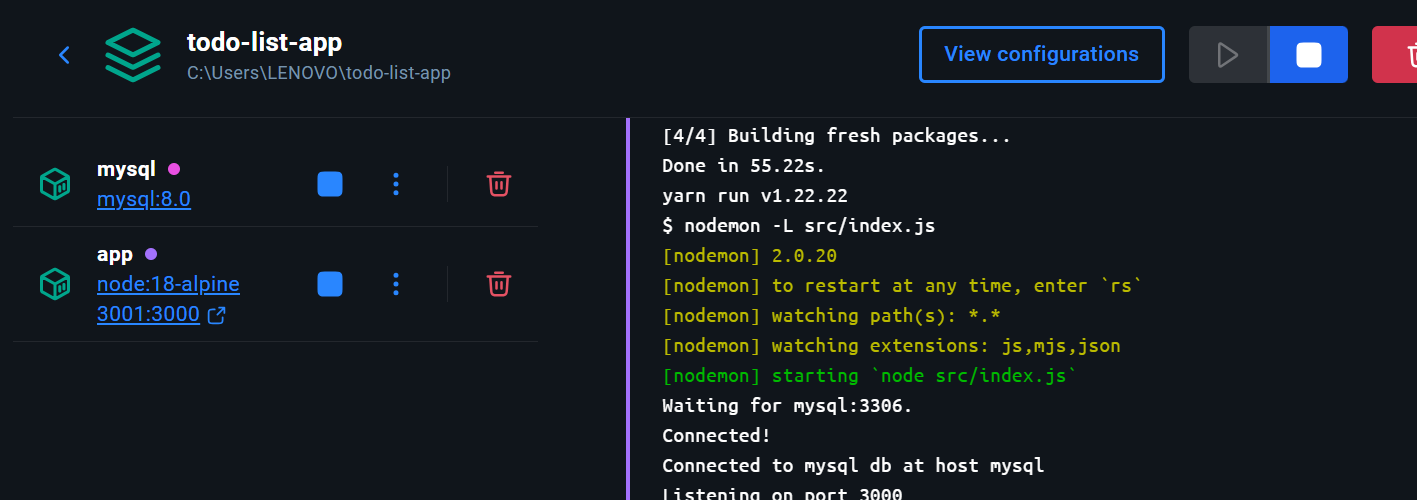
Configuration



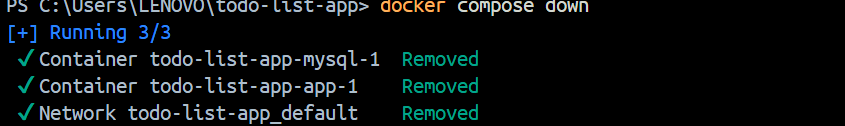
Application operation interface

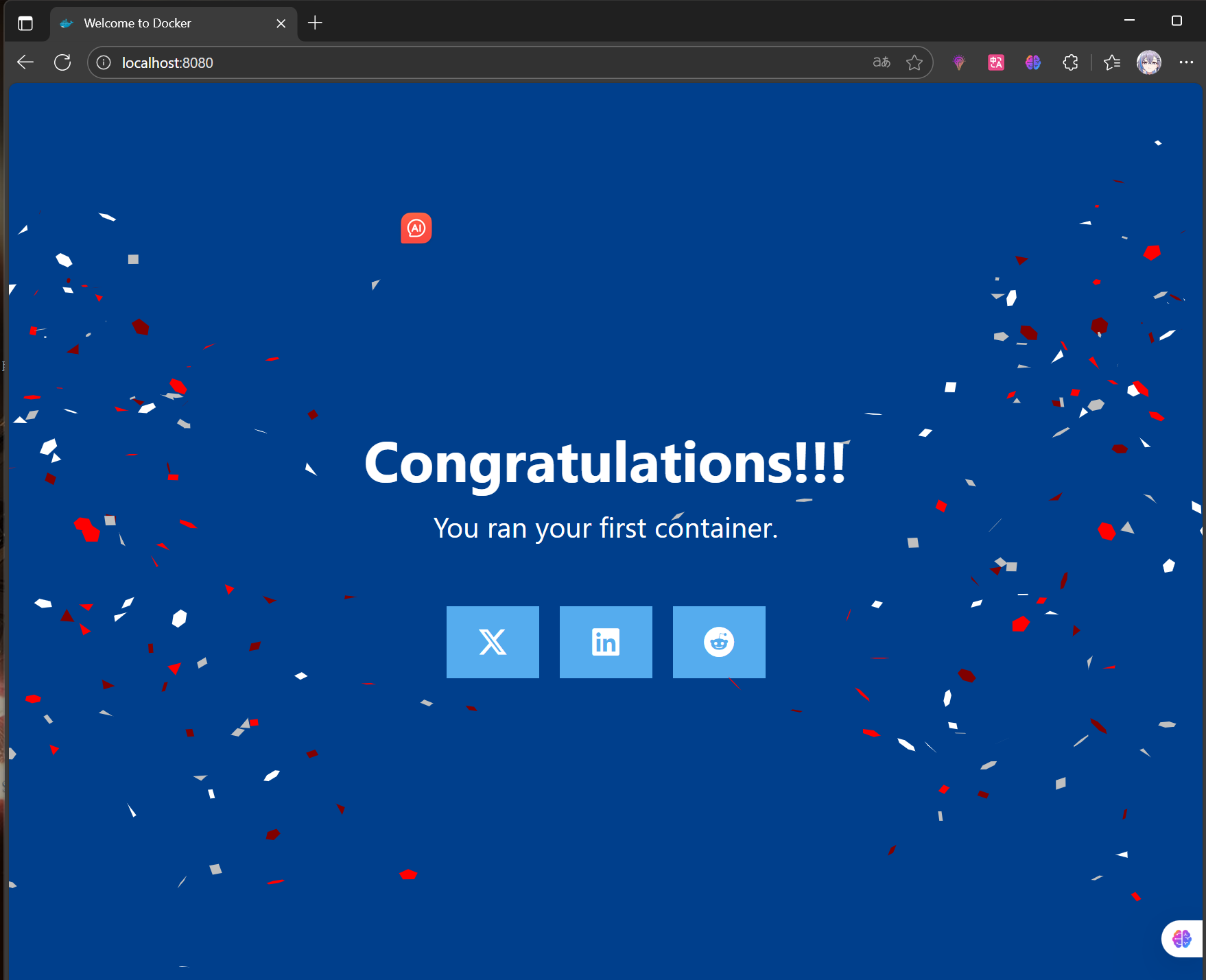


View the graphical interface



Uninstall



Publishing port

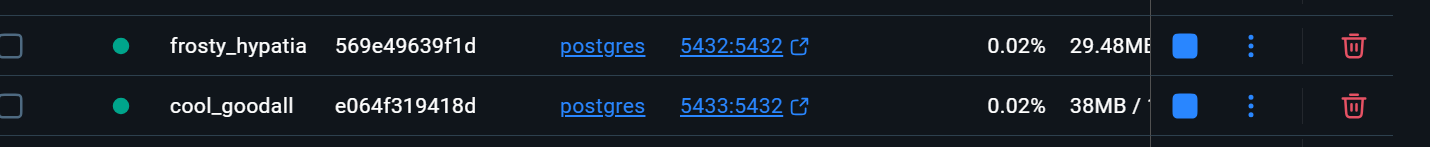
**Start a container using the Postgres image with the following command:**



Start a second Postgres container mapped to a different port.

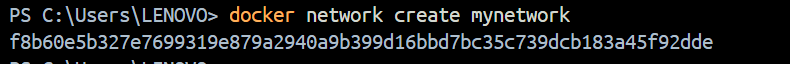


Verify that both containers are running by going to the **Containers** view in the Docker Desktop Dashboard.

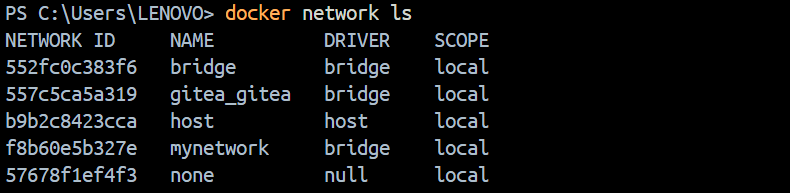


Run Postgres container in a controlled network

Create a new custom network by using the following command:



Verify the network by running the following command:

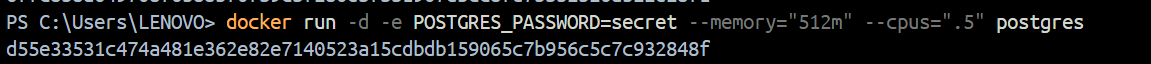


Connect Postgres to the custom network by using the following command



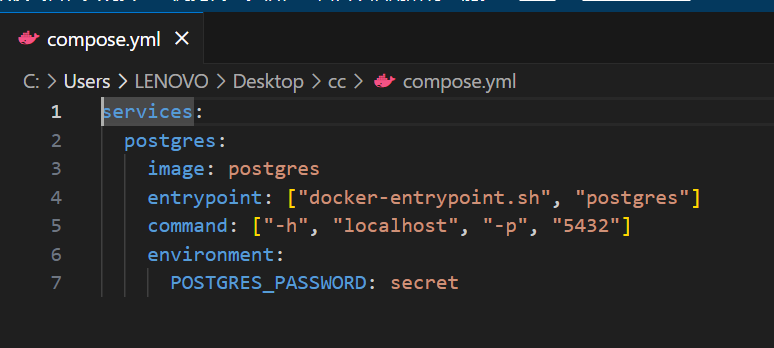
This will start Postgres container in the background, mapped to the host port 5434 and attached to the mynetwork network. You passed the --network parameter to override the container default by connecting the container to custom Docker network for better isolation and communication with other containers.

Manage the resources

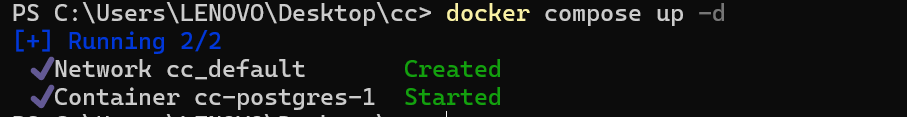


Override the default CMD and ENTRYPOINT in Docker Compose

Create a compose.yml file with the following content:

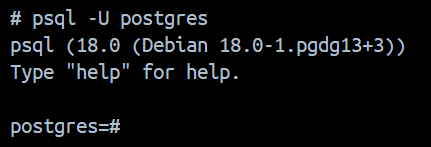


Bring up the service by running the following command:



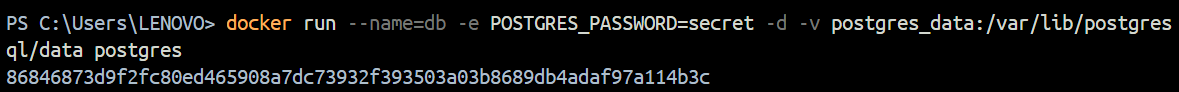
Verify the authentication with Docker Desktop Dashboard.

type the following command to connect to the Postgres database:

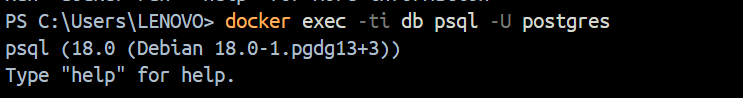


Use volumes

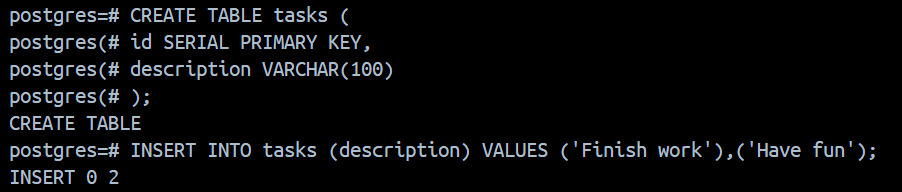
Start a container using the Postgres image with the following command:



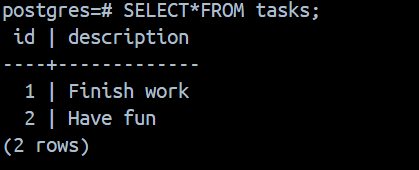
Connect to the database by using the following command:



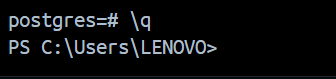
In the PostgreSQL command line, run the following to create a database table and insert two records:



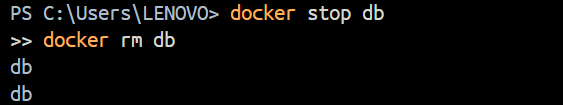
Verify the data is in the database by running the following in the PostgreSQL command line



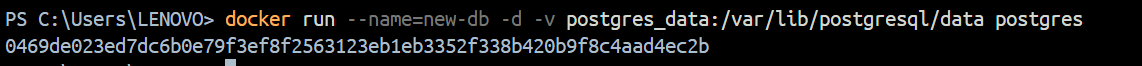
Exit out of the PostgreSQL shell by running the following command:



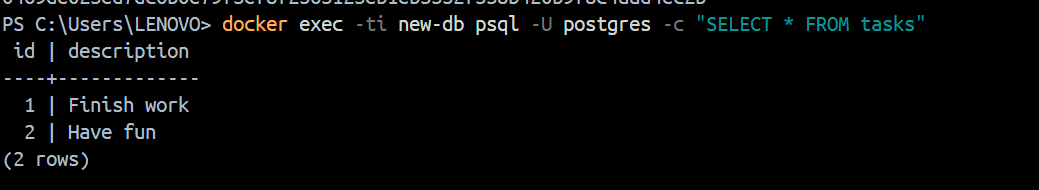
Stop and remove the database container. Remember that, even though the container has been deleted, the data is persisted in the postgres\_data volume.



Start a new container by running the following command, attaching the same volume with the persisted data:

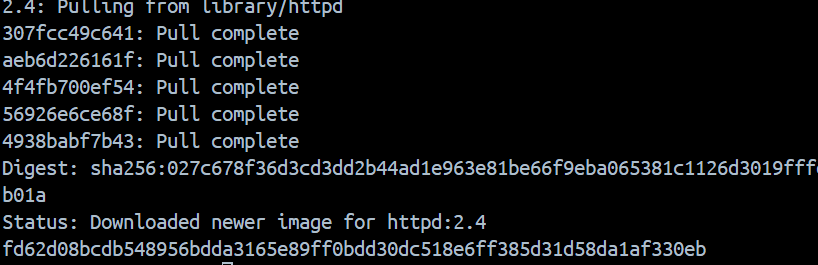


Verify the database still has the records by running the following command:

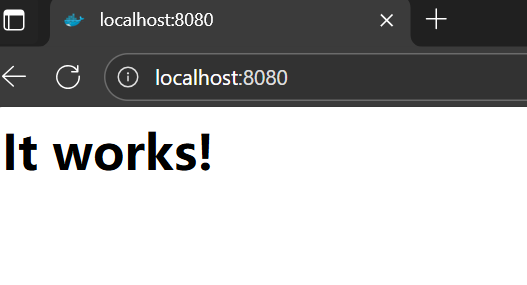


**Sharing local files with containers**

Start a container using the httpd image with the following command:

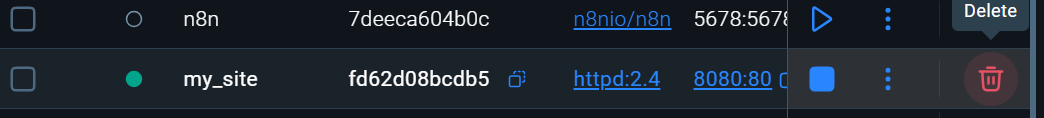


Open the browser and access http://localhost:8080

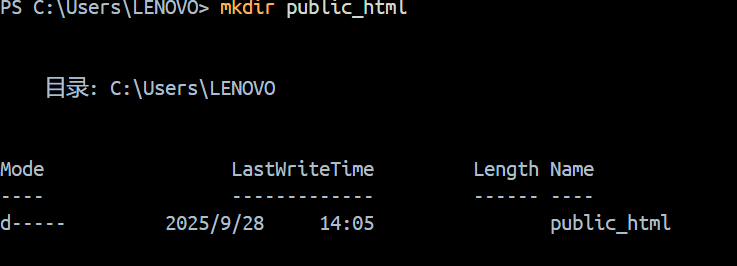


Use a bind mount

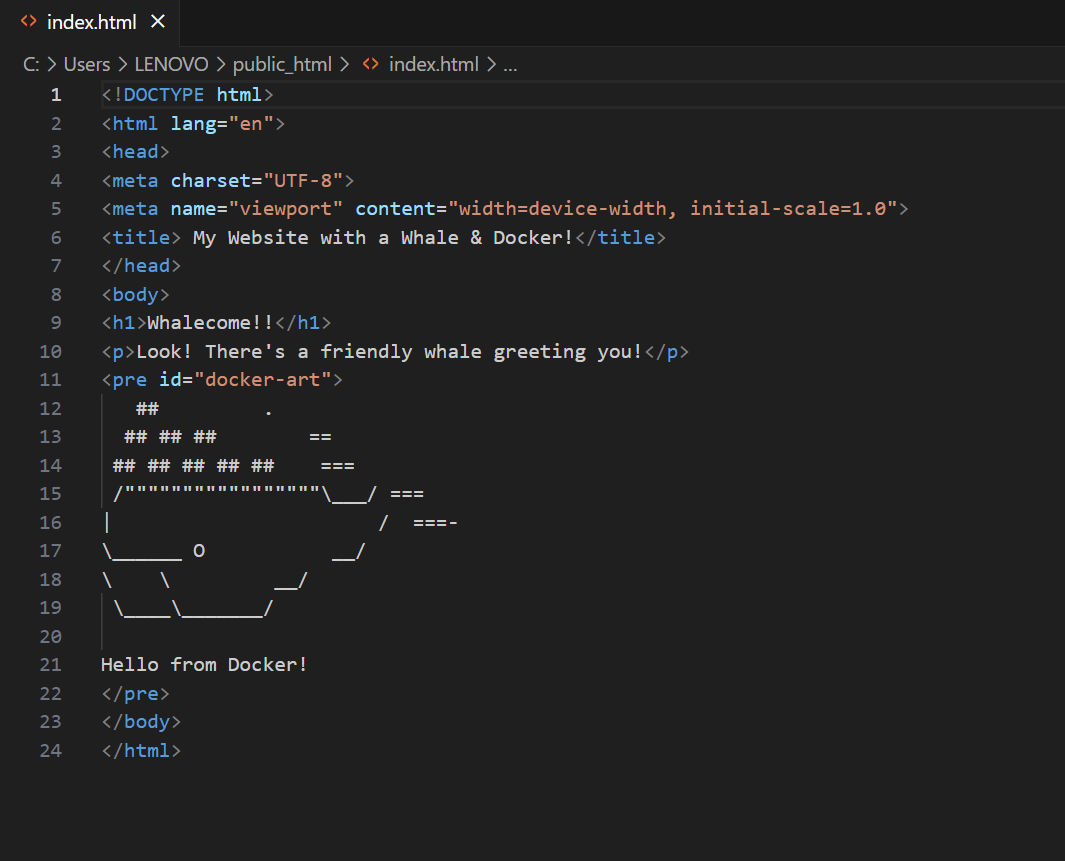
Delete the existing container by using the Docker Desktop Dashboard:



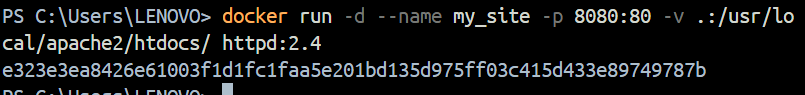
Create a new directory called public\_html on your host system.



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

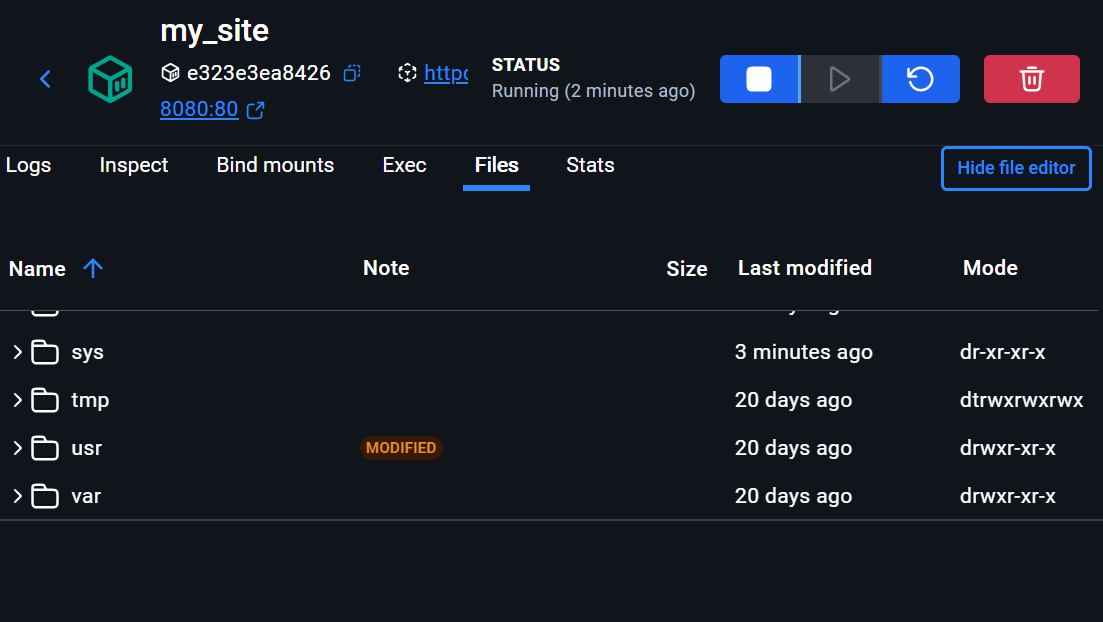


 run the container





Access the file on the Docker Desktop Dashboard

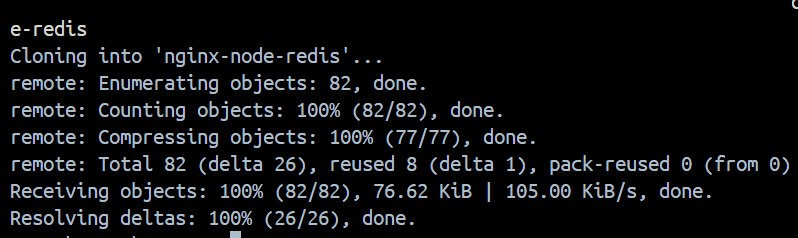


Delete the file on the host and verify the file is also deleted in the container. You will find that the files no longer exist under **Files** in the Docker Desktop Dashboard.

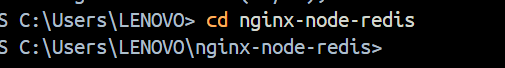
Recreate the HTML file on the host system and see that file re-appears under the **Files** tab under **Containers** on the Docker Desktop Dashboard. By now, you will be able to access the site too.

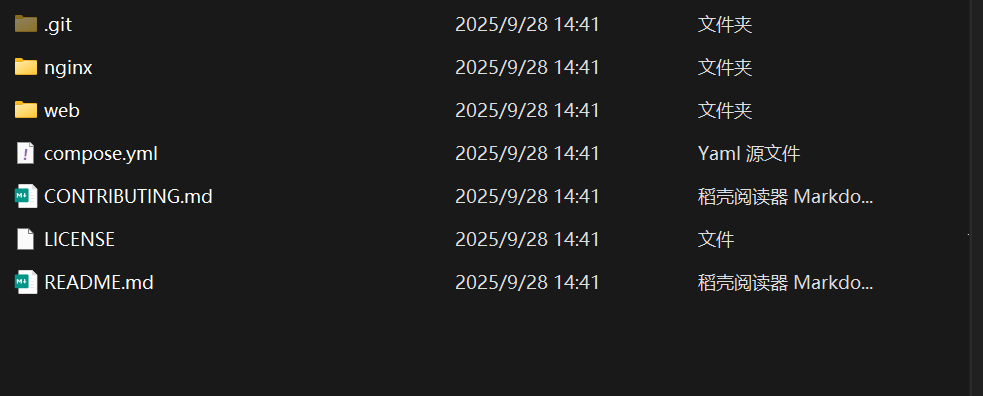
 build and run a counter web application based on Node.js, an Nginx reverse proxy, and a Redis database using the docker run commands.

Use the following command in a terminal to clone the sample application repository.

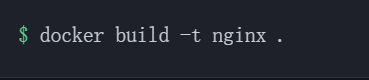


Navigate into the nginx-node-redis directory:





Navigate into the nginx directory to build the image by running the following command:



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



 create a network for them all to communicate through.



Start the Redis container by running the following command, which will attach it to the previously created network and create a network alias

docker run -d --name redis --network sample-app --network-alias redis redis

Start the first web container by running the following command:  
docker run -d --name web1 -h web1 --network sample-app --network-alias web1 web

Start the second web container by running the following:

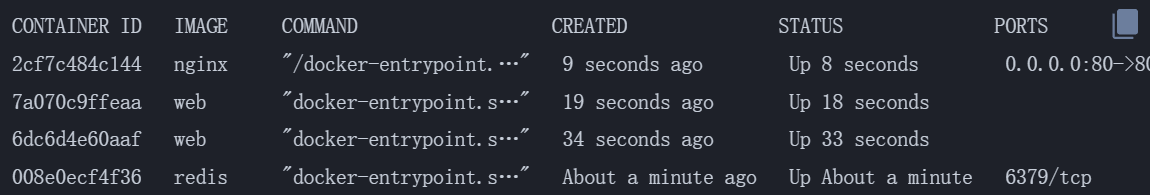
docker run -d --name web2 -h web2 --network sample-app --network-alias web2 web

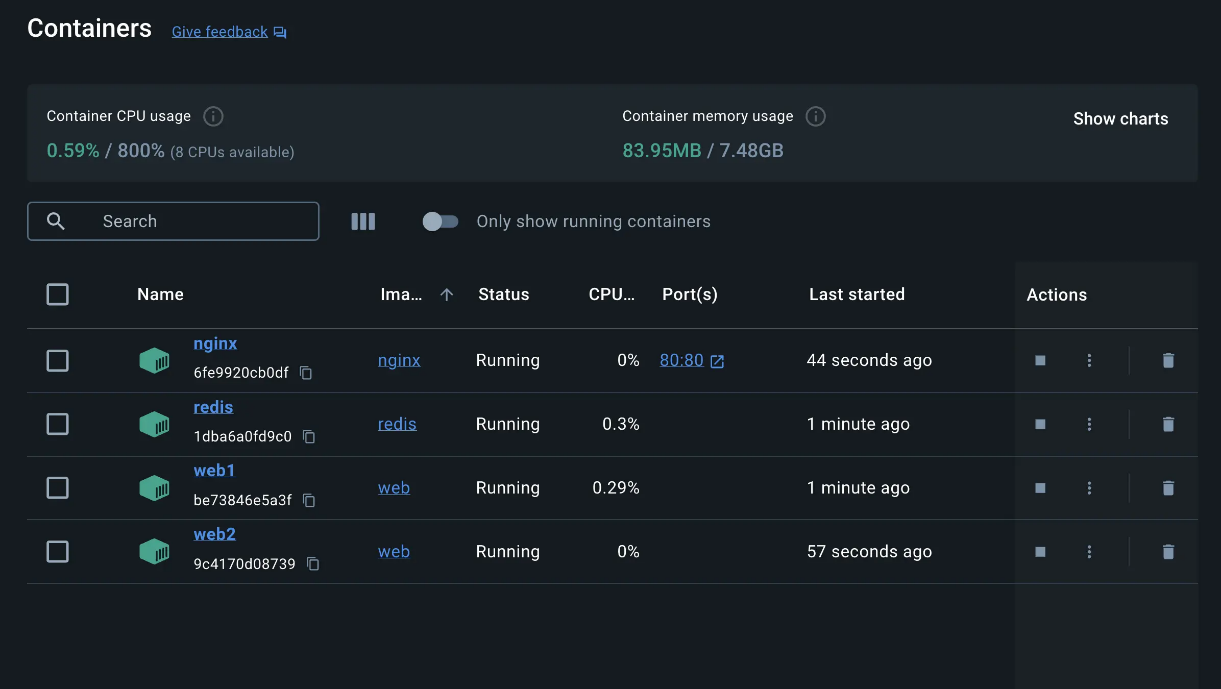
Start the Nginx container by running the following command:

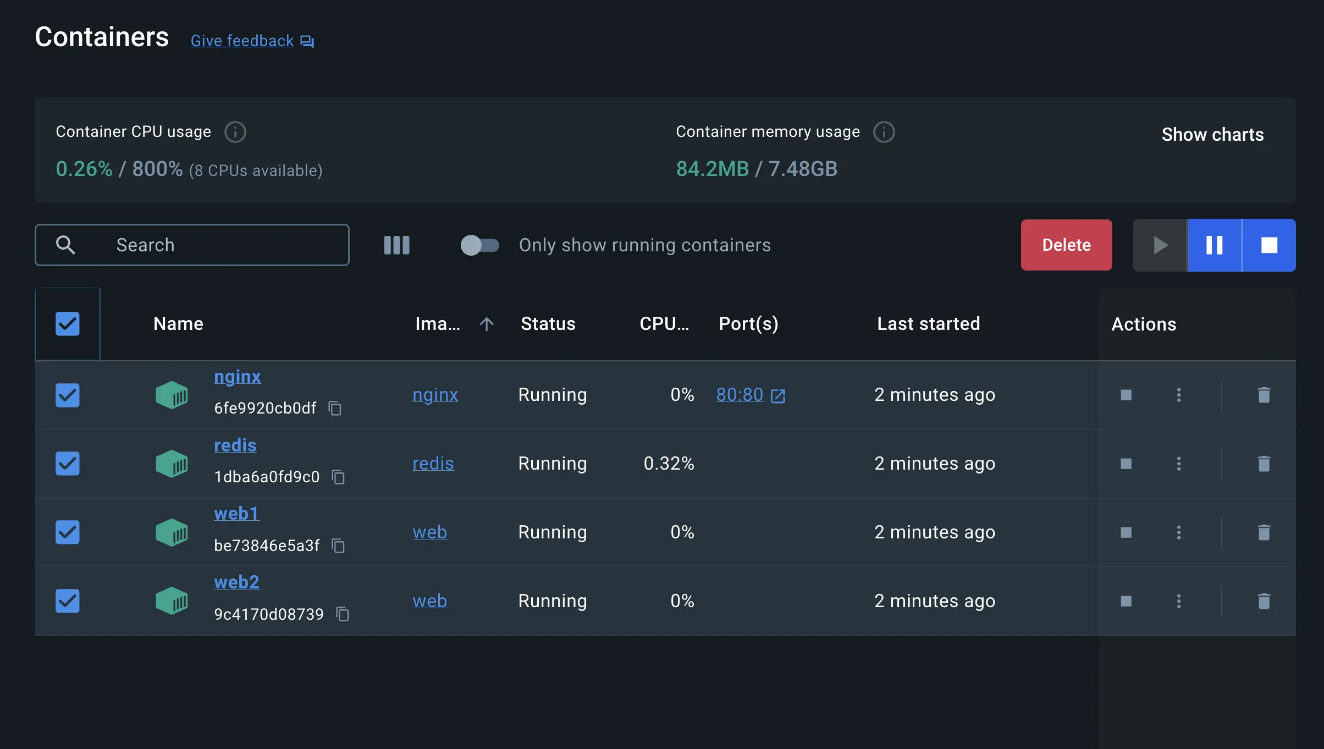
docker run -d --name nginx --network sample-app -p 80:80 nginx

Verify the containers are up by running the following command:

docker ps

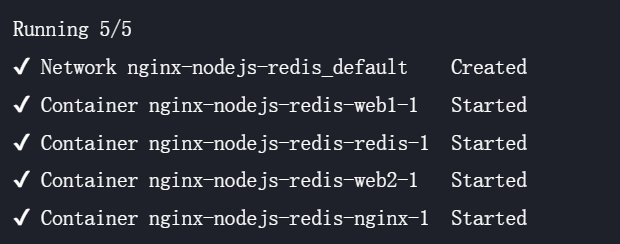






Use the docker compose up command to start the application

docker compose up -d –build



If you look at the Docker Desktop Dashboard, you can see the containers and dive deeper into their configuration.

