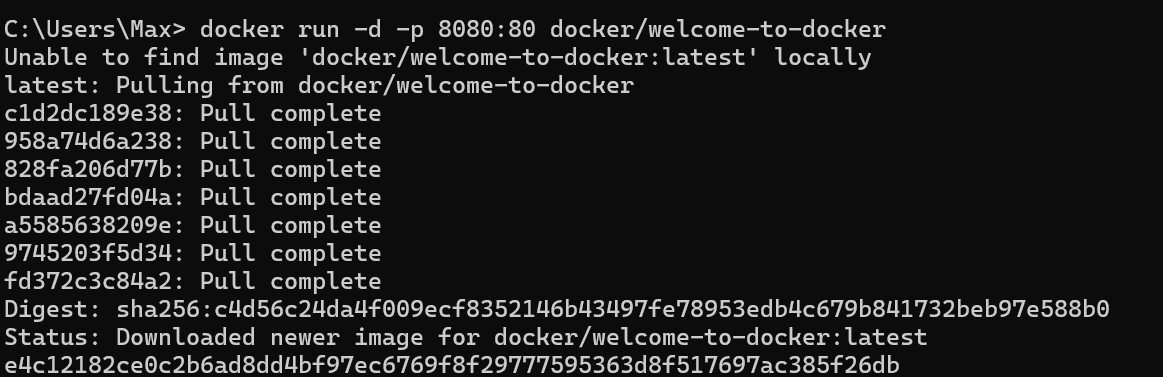
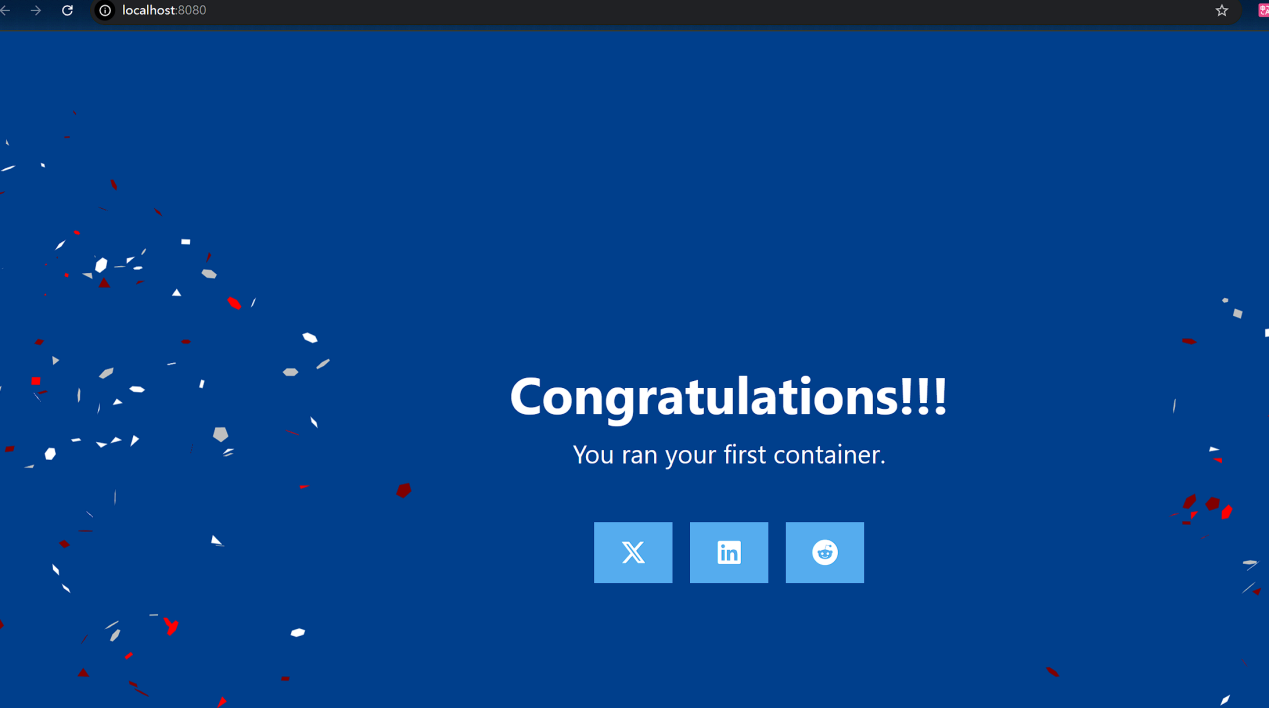
What is container?

Open the CLI terminal and we can start a container by using the docker run command

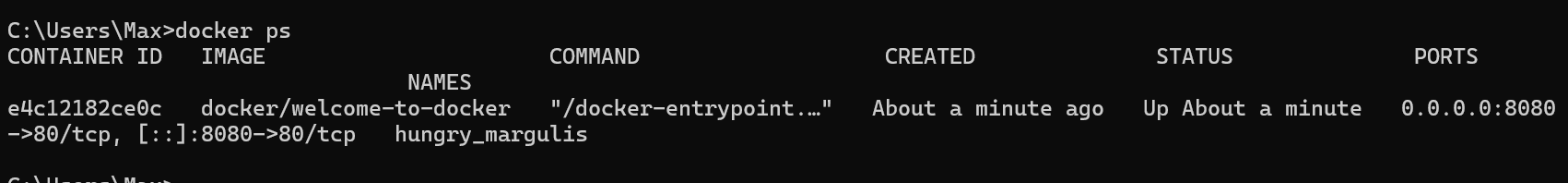
It will output the full container ID



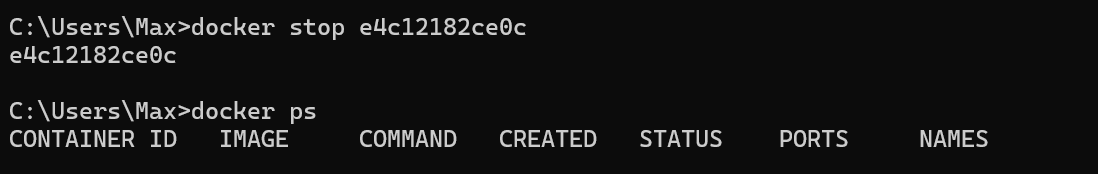


We can use docker ps command to verify if the container is up and running and get the ID of the container

If we want to see stopped containers ,we need to use docker ps -a



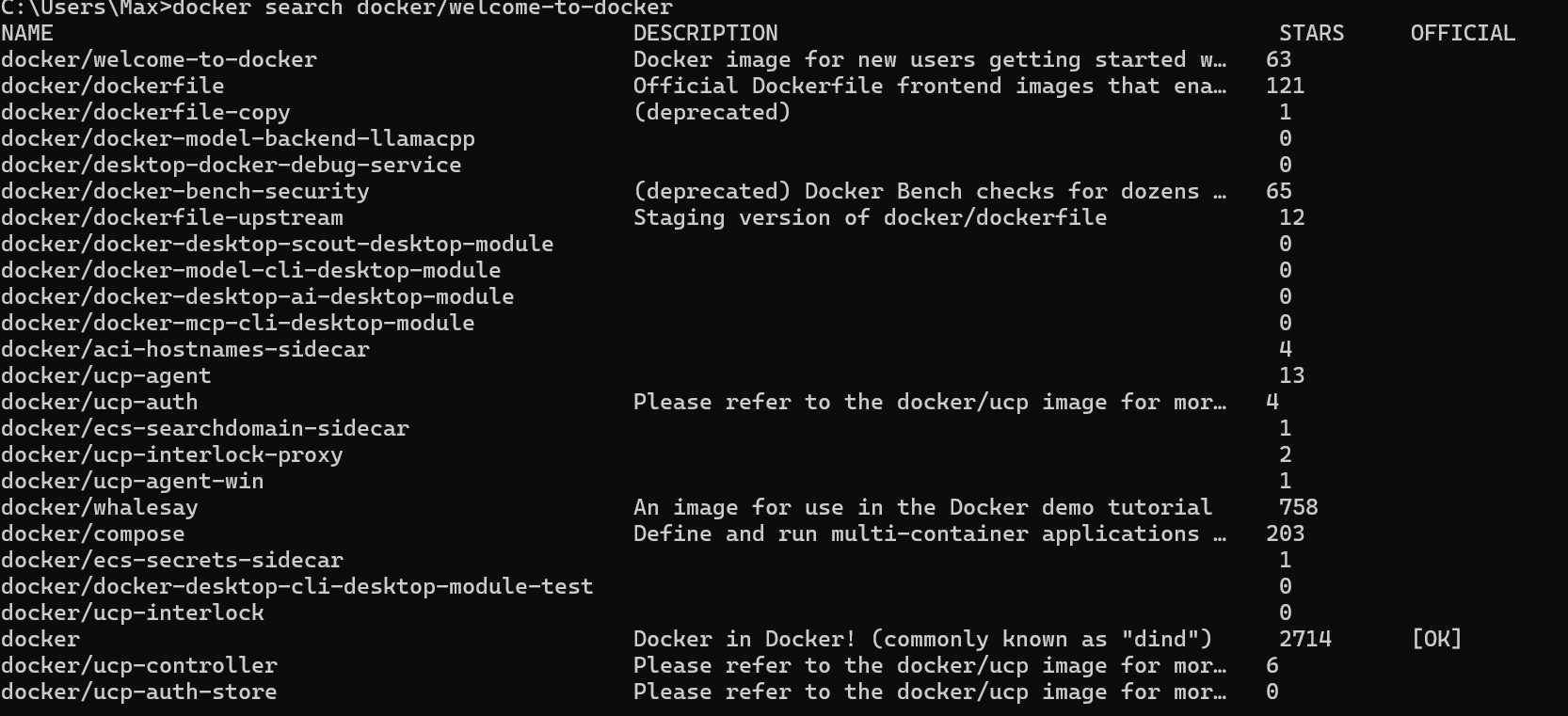
We use “docker stop <the-container-id>” to stop a container



What is an image?

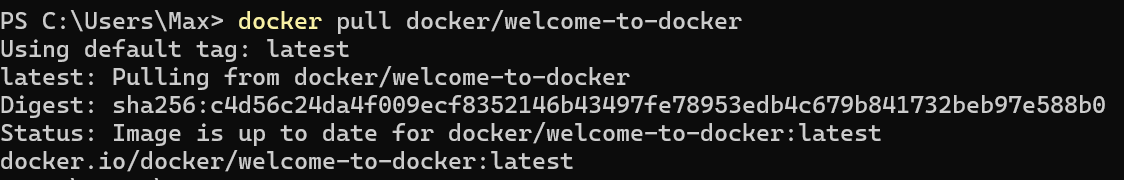
We use “docker search docker/welcome-to-docker” to search for images

And the output will display information about relevant images

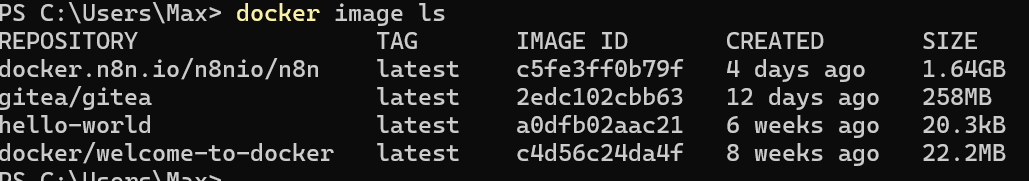


We use “docker pull” to pull the image

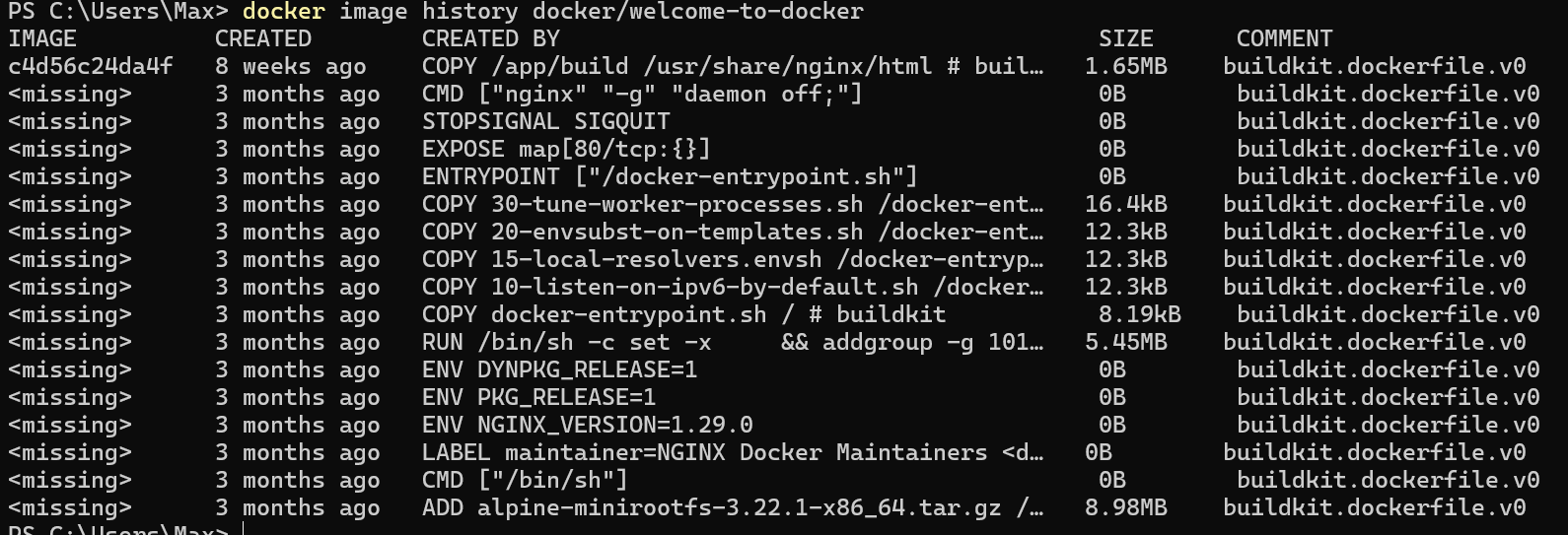
Each line represents a different layer of the image



We use “docker image ls” to list downloaded images



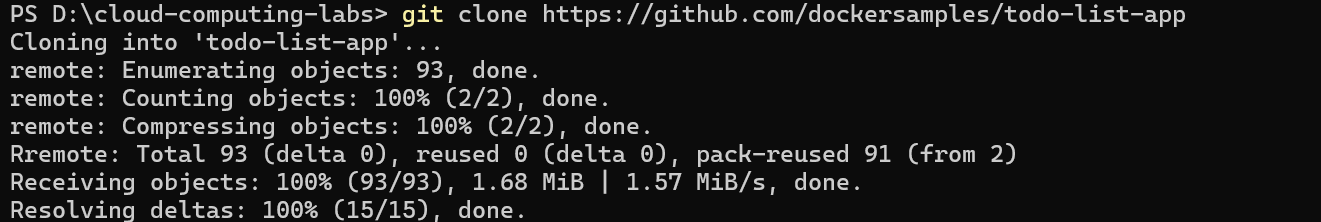
We use “docker image history” to list the image’s layers

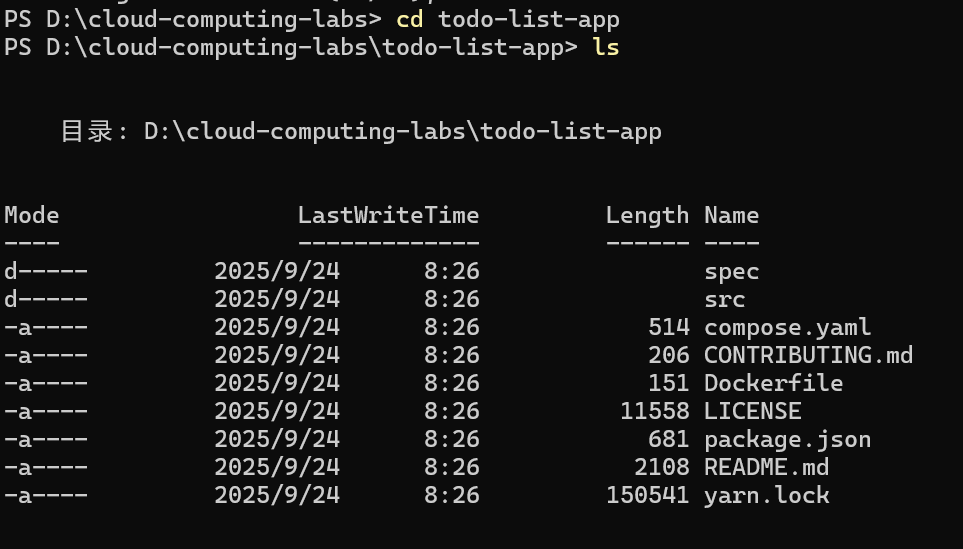


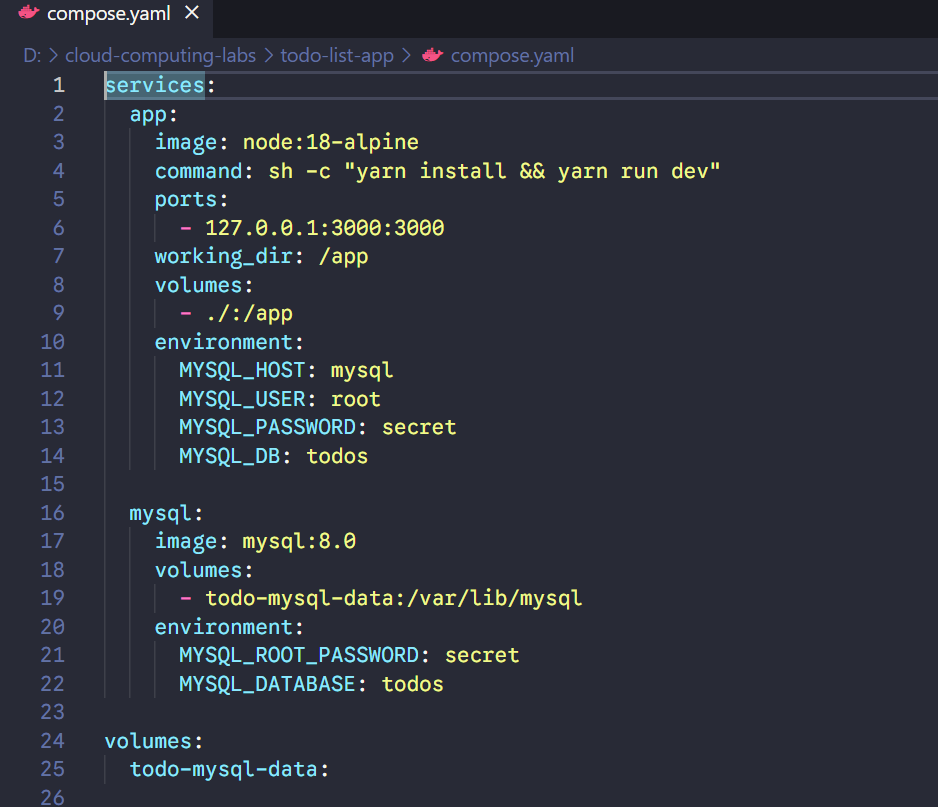
What is Docker Compose?

With Docker Compose, we can define all of the containers and their configurations in a single yaml file.

Clone a example application







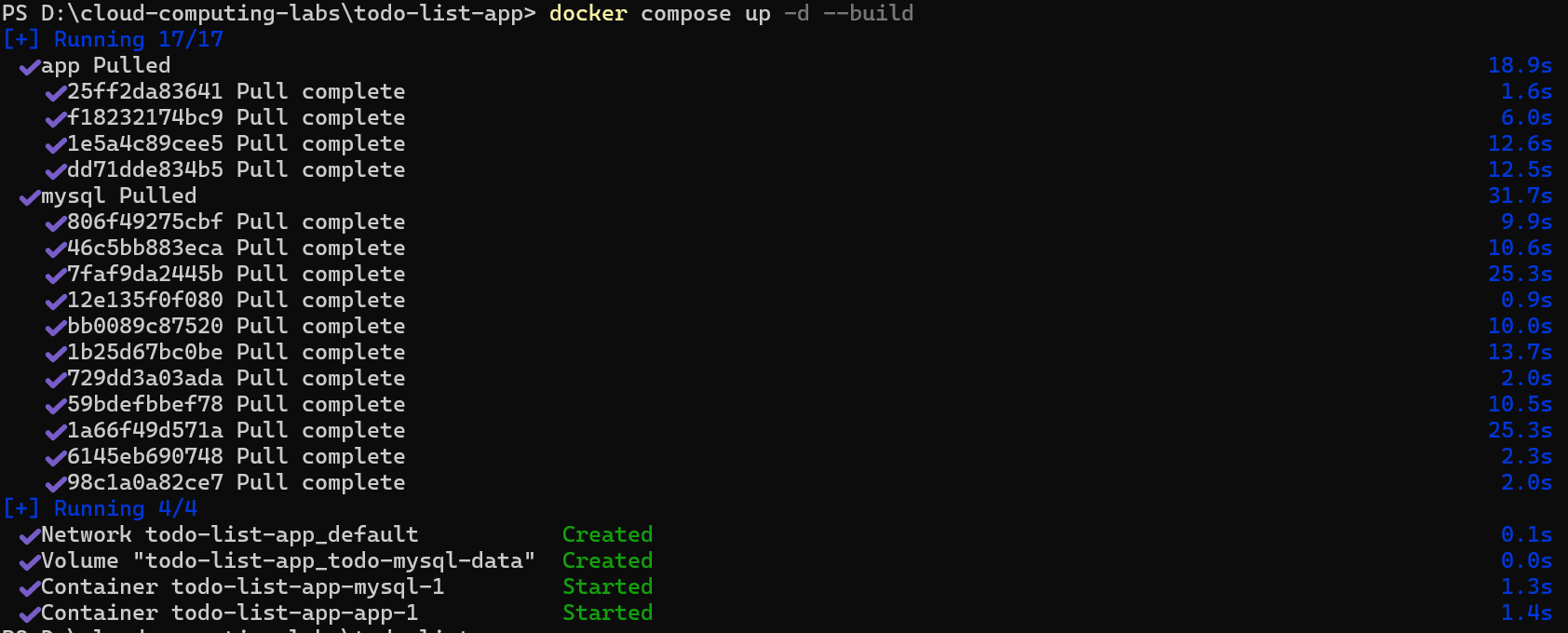
We use “docker compose up” to start the application

Two container images were downloaded from docker-hub :node and mysql

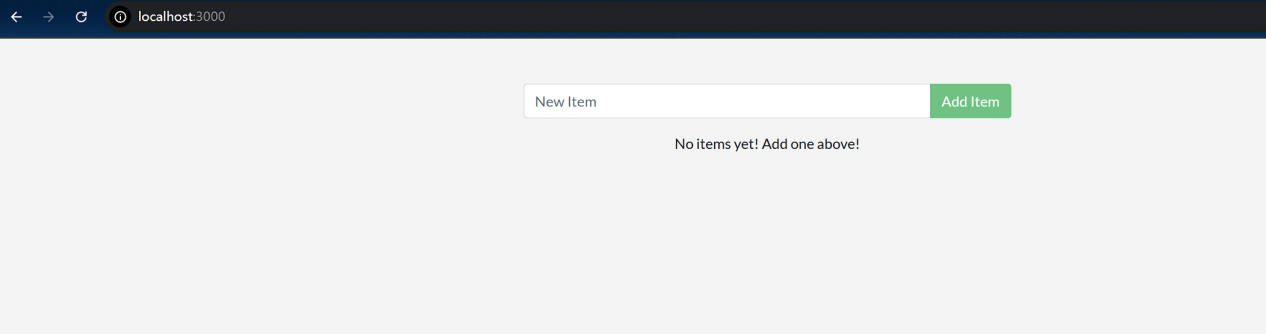
A network was created for the application

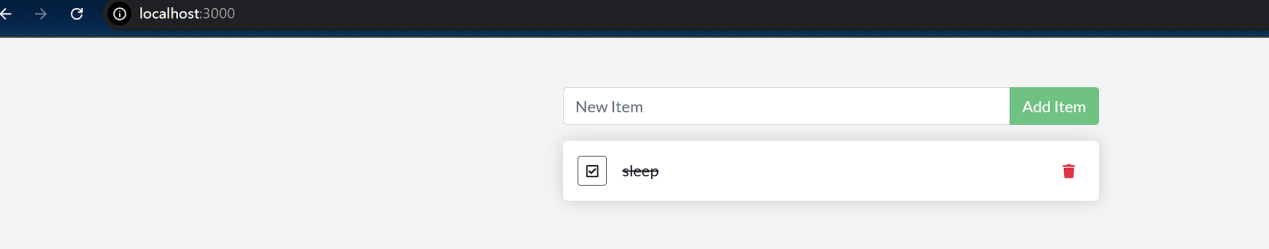
A volume was created to persist the database files between container restarts

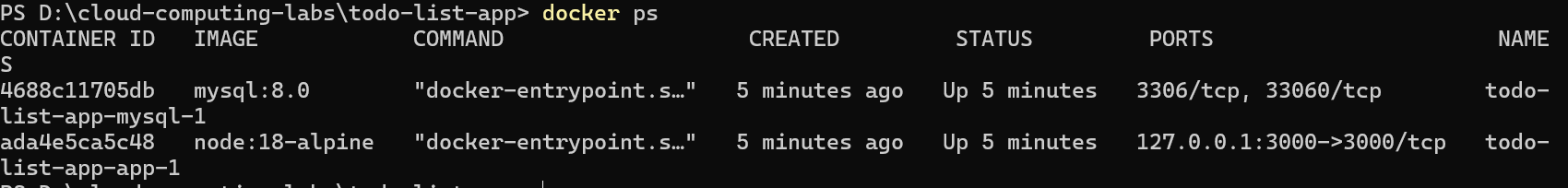
Two containers were started with all of their necessary config



We can open <http://localhost:3000> in the brower to see the website

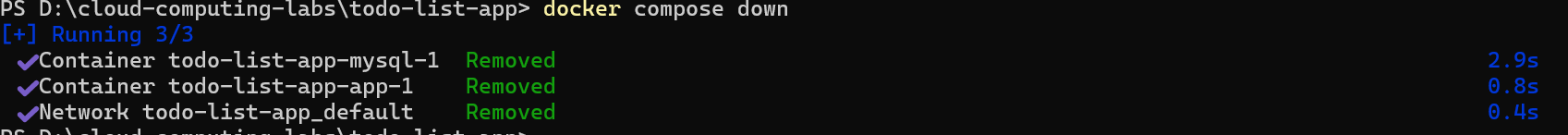


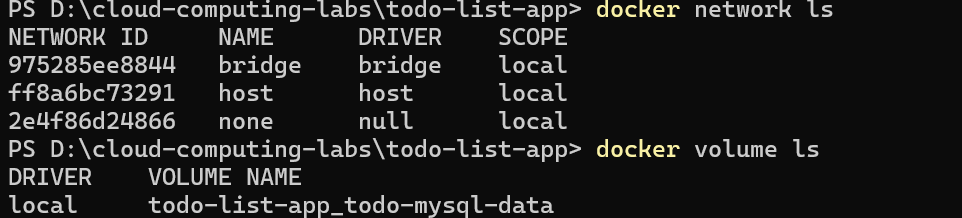




We can use “docker compose down” to remove everything

We can also select the delete button in the GUI to remove the containers

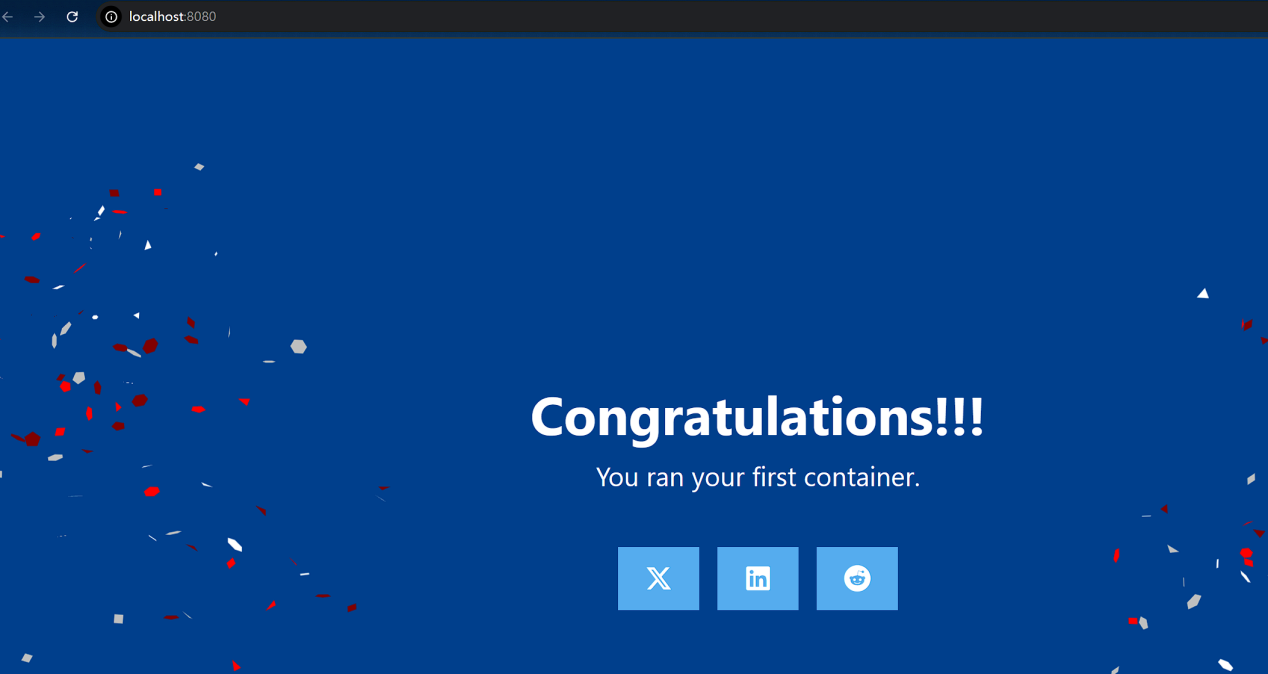




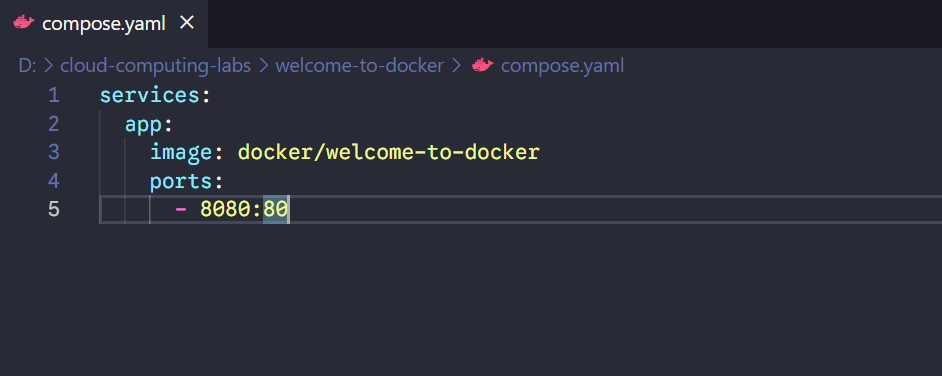
Publishing and exposing ports

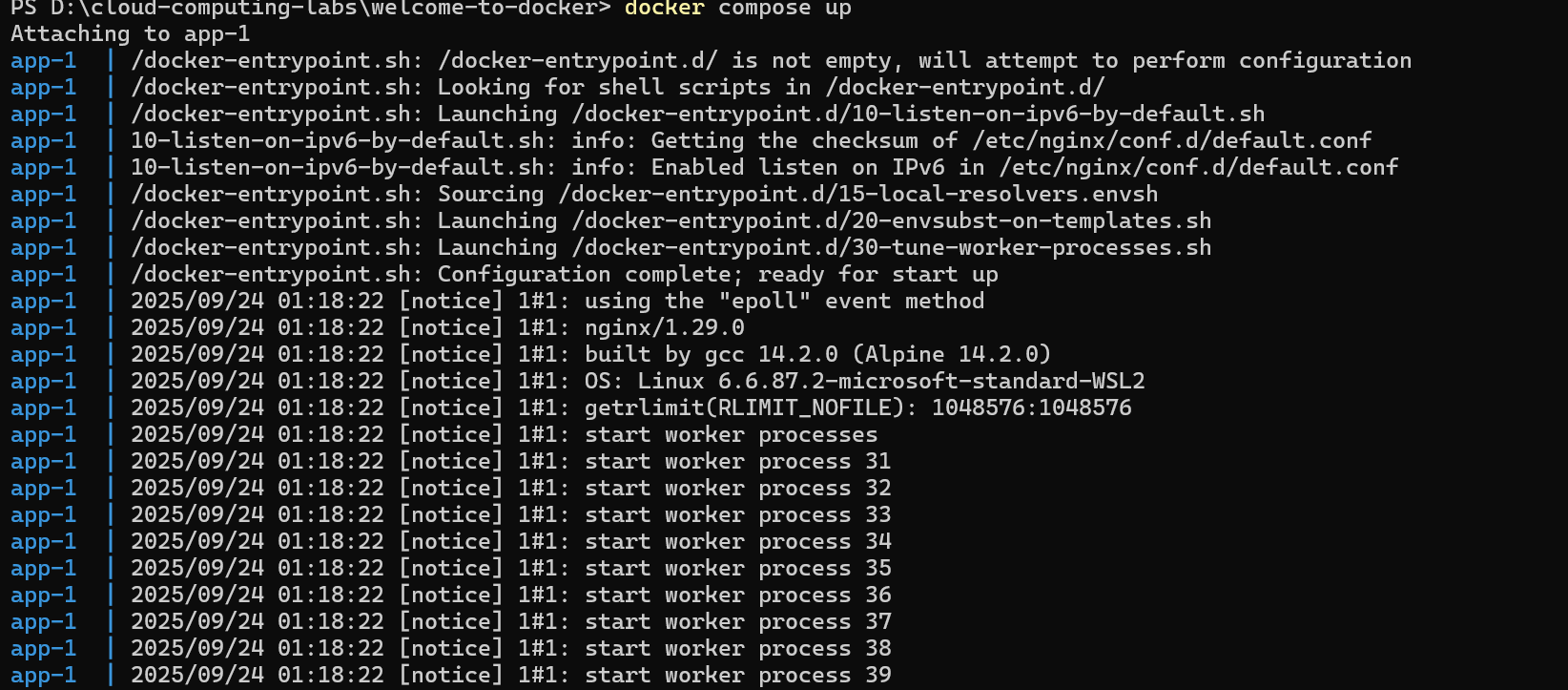


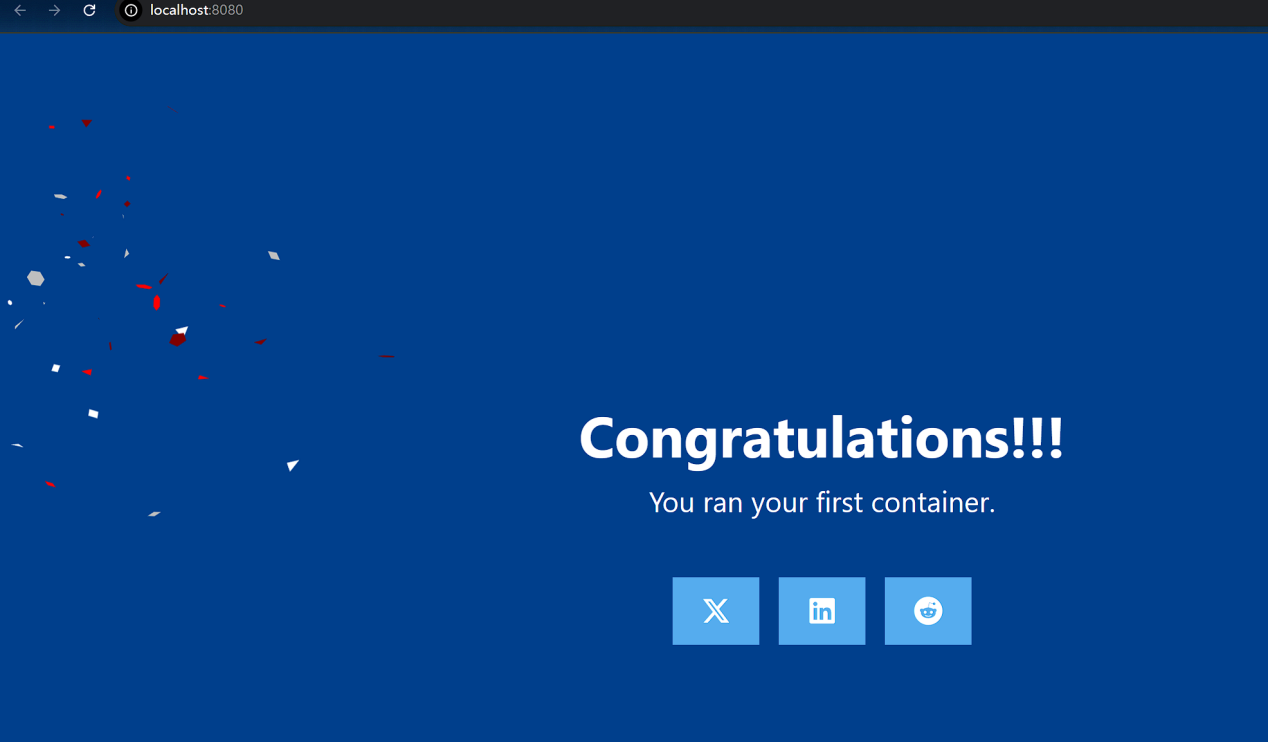
The first 8080 refers to the host port. This is the port on your local machine that will be used to access the application running inside the container. The second 80 refers to the container port.

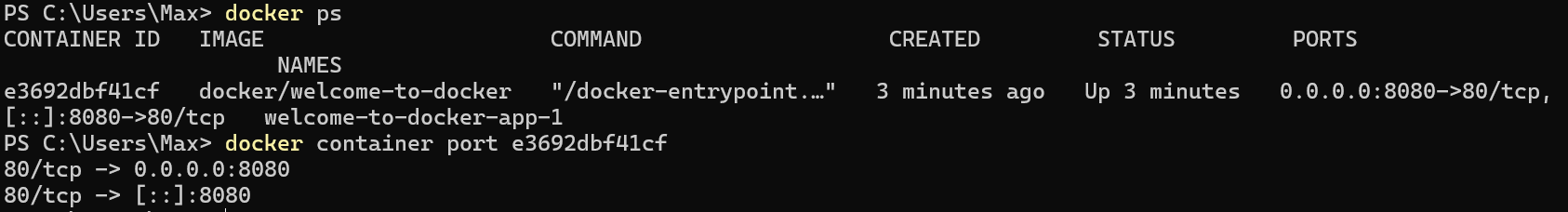


Launch the application using Docker Compose







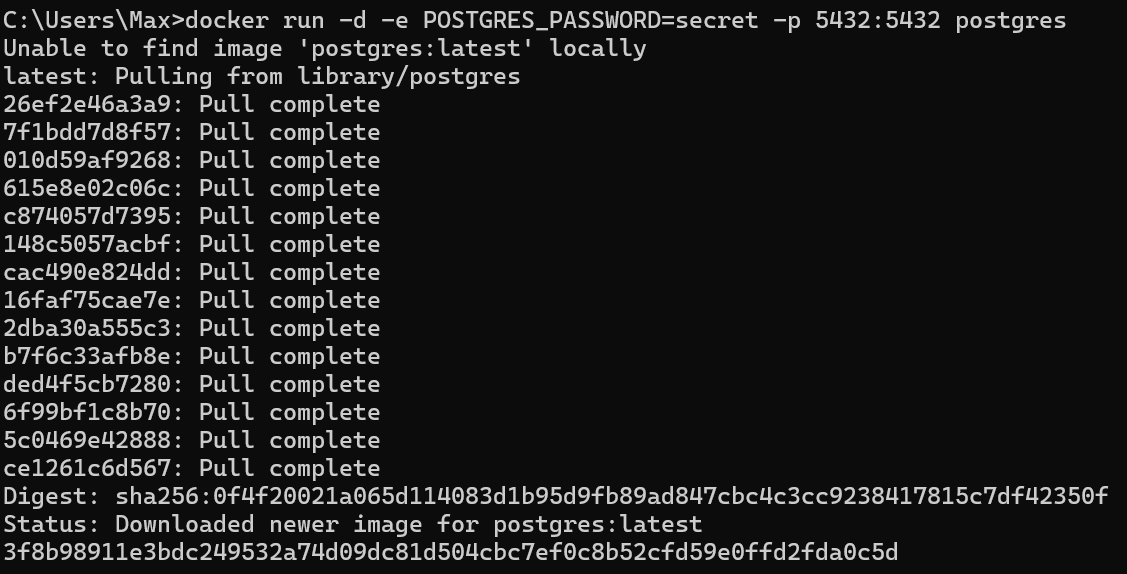


Overriding Container defaults

We can use the **docker stats** command to monitor the real-time resource usage of running containers.

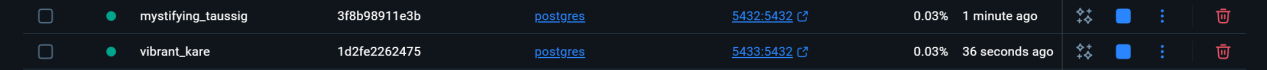
Run multiple instances of the Postgres database

start the Postgres database in the background, listening on the standard container port 5432 and mapped to port 5432 on the host machine



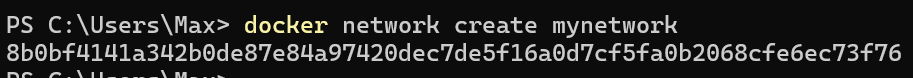
This will start another Postgres container in the background, listening on the standard postgres port 5432 in the container, but mapped to port 5433 on the host machine. You override the host port just to ensure that this new container doesn't conflict with the existing running container.



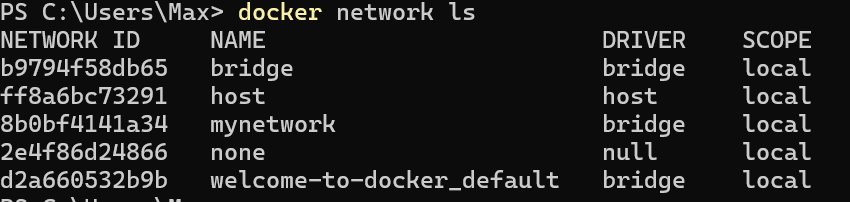


Bridge network acts like a virtual bridge, allowing containers on the same host to communicate with each other while keeping them isolated from the outside world and other hosts.

Create a new custom network by using the following command



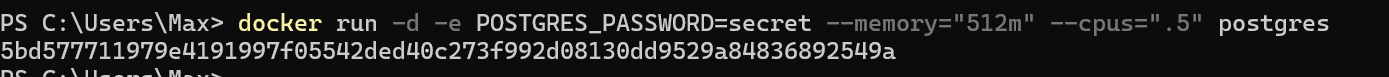
Verify the network by running the following command:



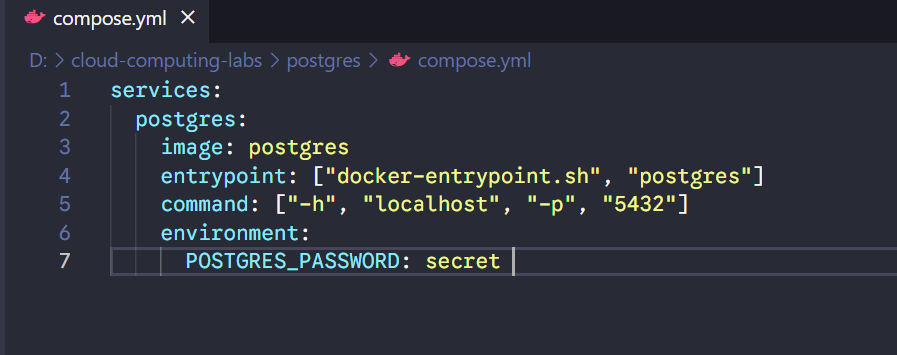
Connect Postgres to the custom network by using the following command



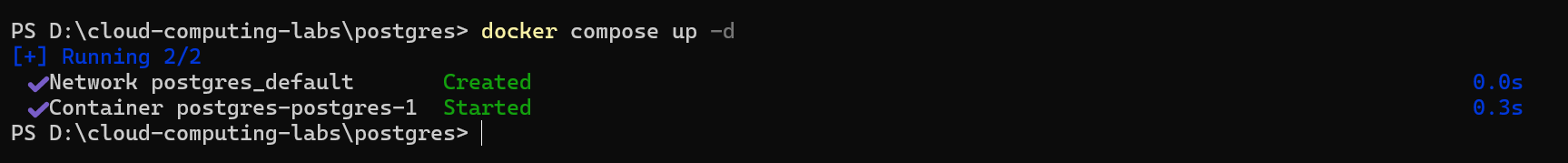
The --cpus flag specifies the CPU quota for the container. Here, it's set to half a CPU core (0.5) whereas the --memory flag specifies the memory limit for the container. In this case, it's set to 512 MB.



Override the default CMD and ENTRYPOINT in Docker Compose



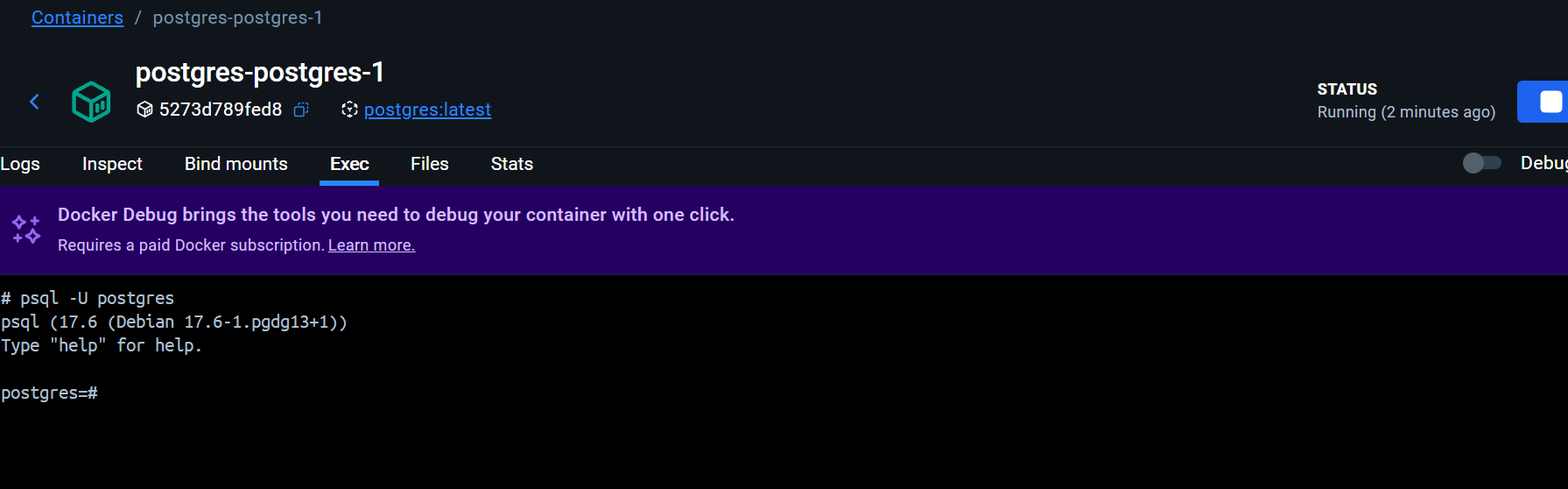
The Compose file defines a service named postgres that uses the official Postgres image, sets an entrypoint script, and starts the container with password authentication.



This command starts the Postgres service defined in the Docker Compose file.



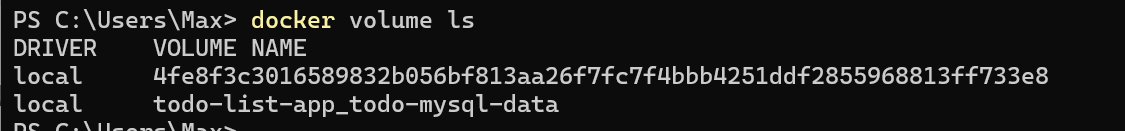
Verify the authentication with Docker Desktop Dashboard.



Persisting container data

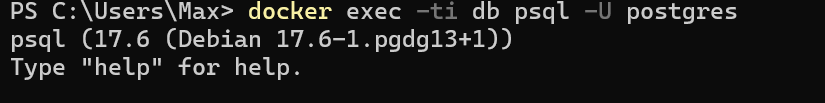
Volumes are a storage mechanism that provide the ability to persist data beyond the lifecycle of an individual container.

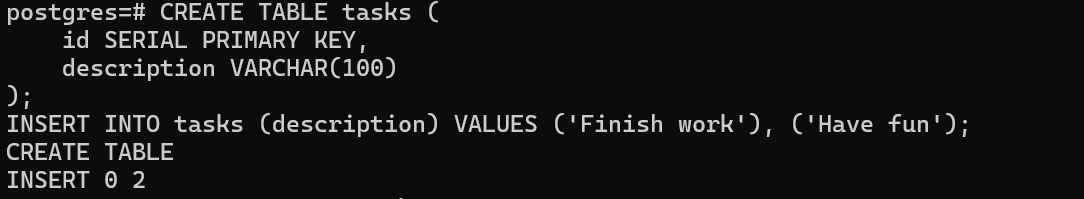
**docker volume ls** - list all volumes

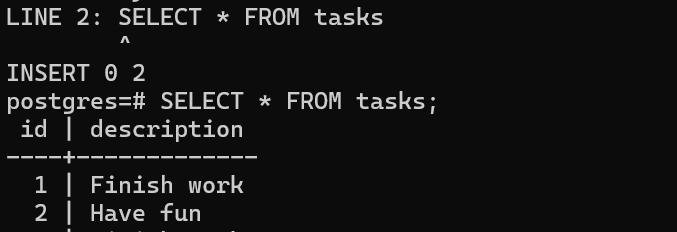




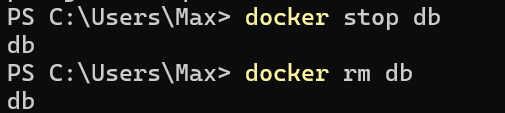
This will start the database in the background, configure it with a password, and attach a volume to the directory PostgreSQL will persist the database files.



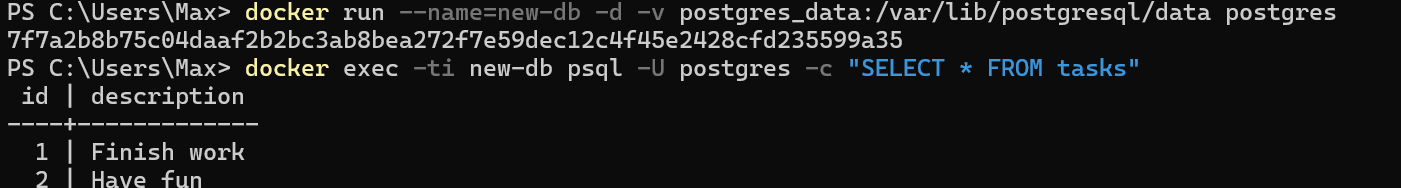


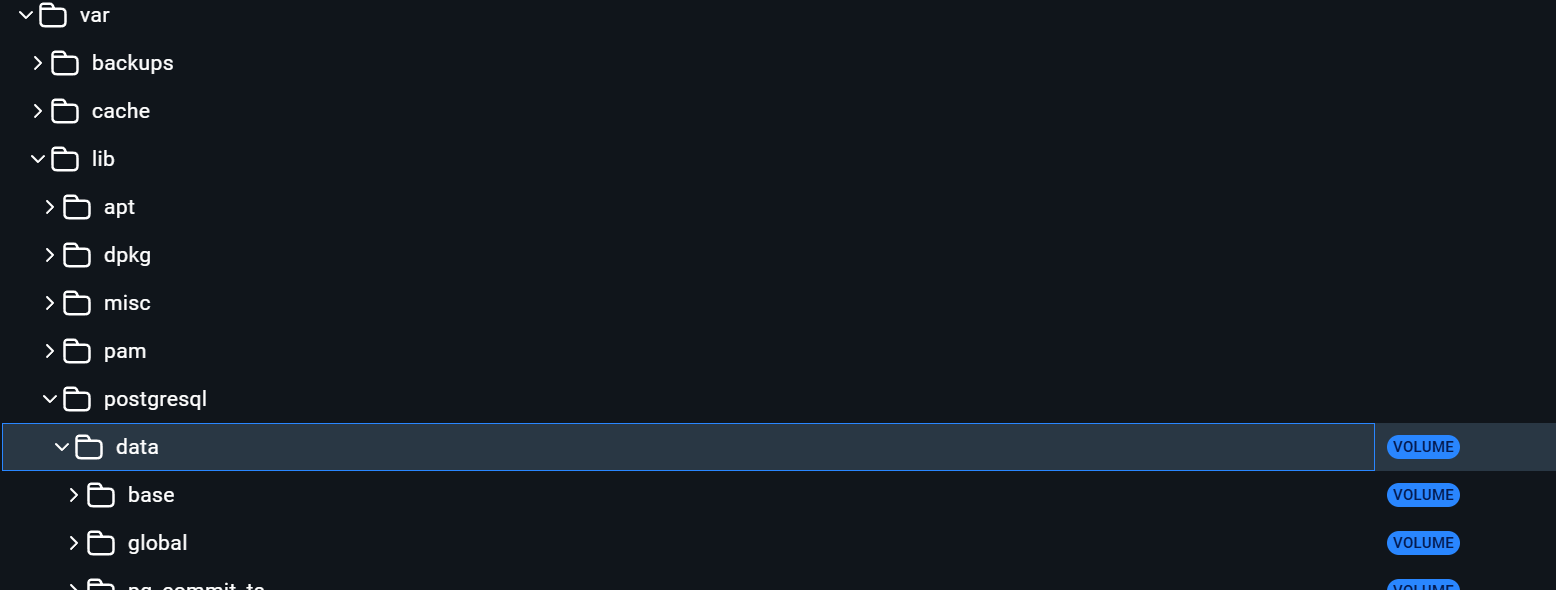


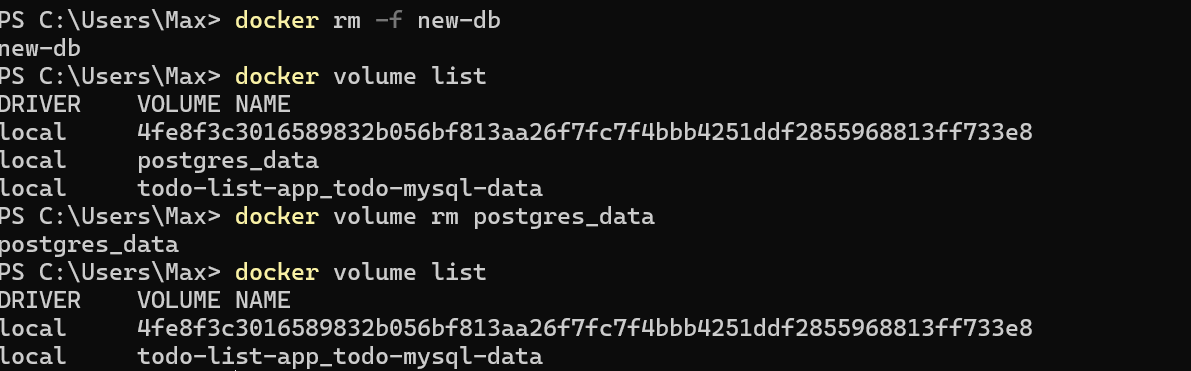
Stop and remove the database container. Remember that, even though the container has been deleted, the data is persisted in the postgres\_data volume.









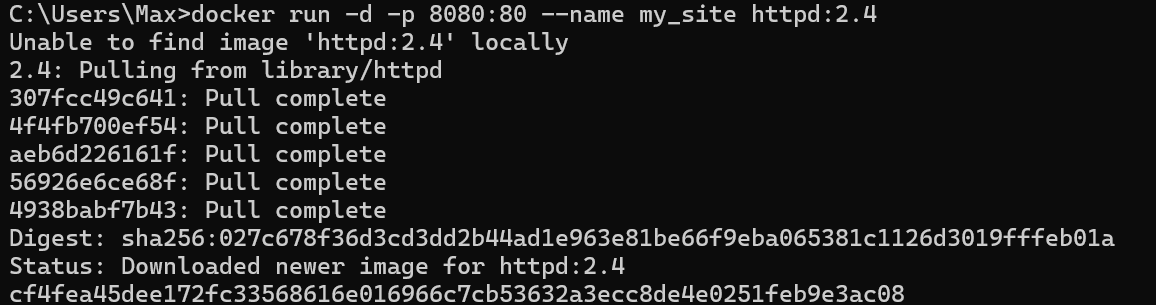


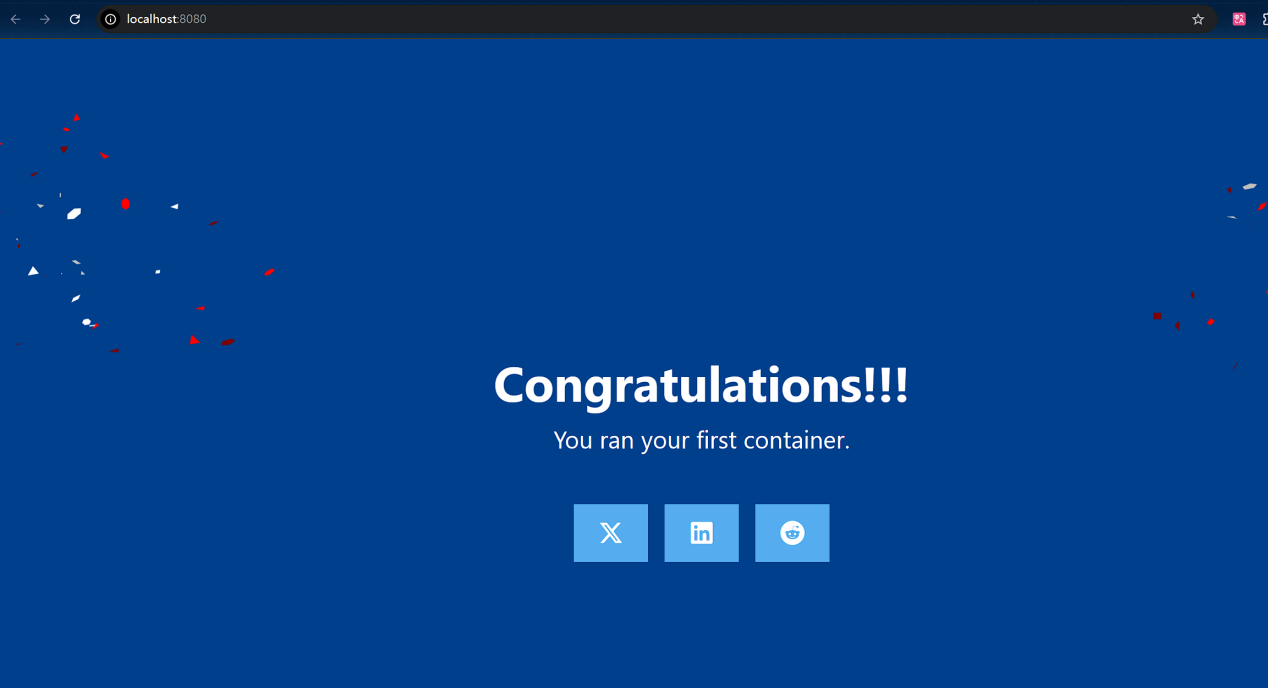
Sharing local files with containers

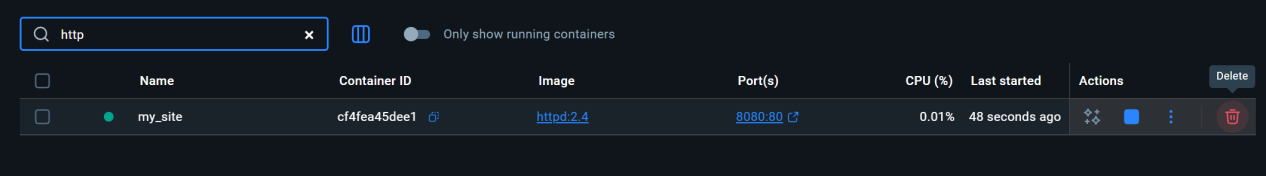
Docker offers two primary storage options for persisting data and sharing files between the host machine and containers: **volumes and bind mounts**.

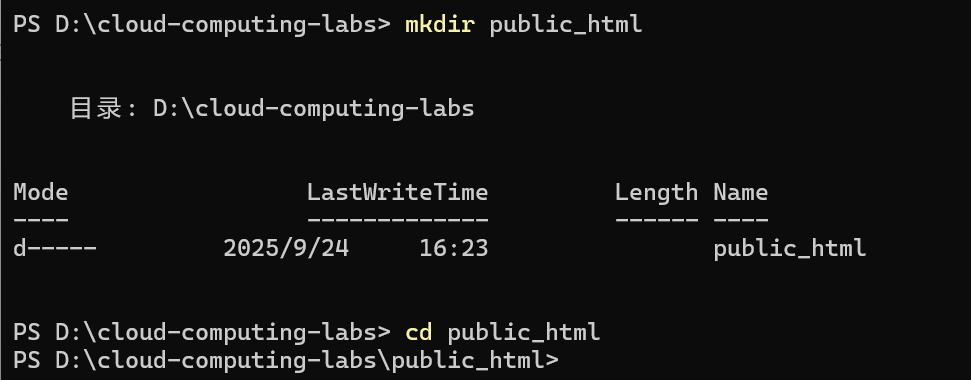
The -v flag is simpler and more convenient for basic volume or bind mount operations. If the host location doesn’t exist when using -v or --volume, a directory will be automatically created.

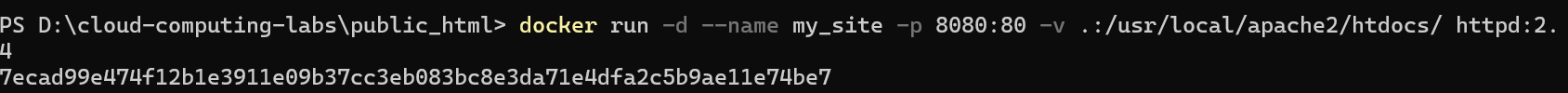
The --mount flag offers more advanced features and granular control, making it suitable for complex mount scenarios or production deployments. If you use --mount to bind-mount a file or directory that doesn't yet exist on the Docker host, the docker run command doesn't automatically create it for you but generates an error.

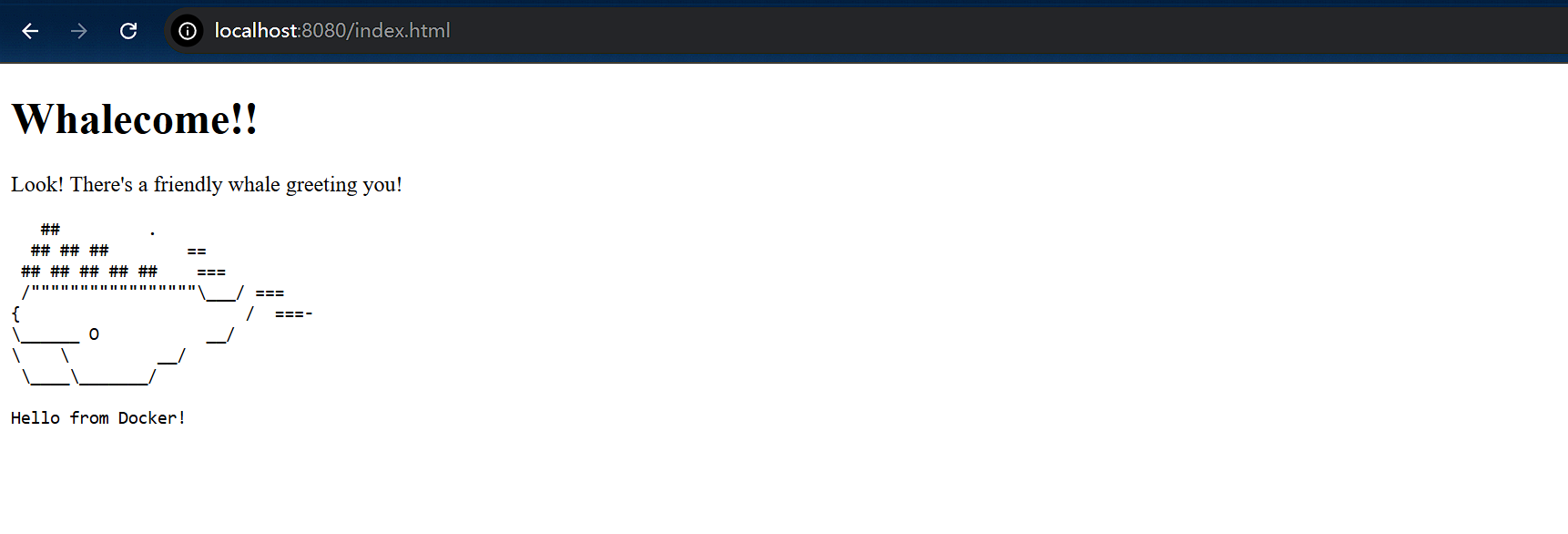


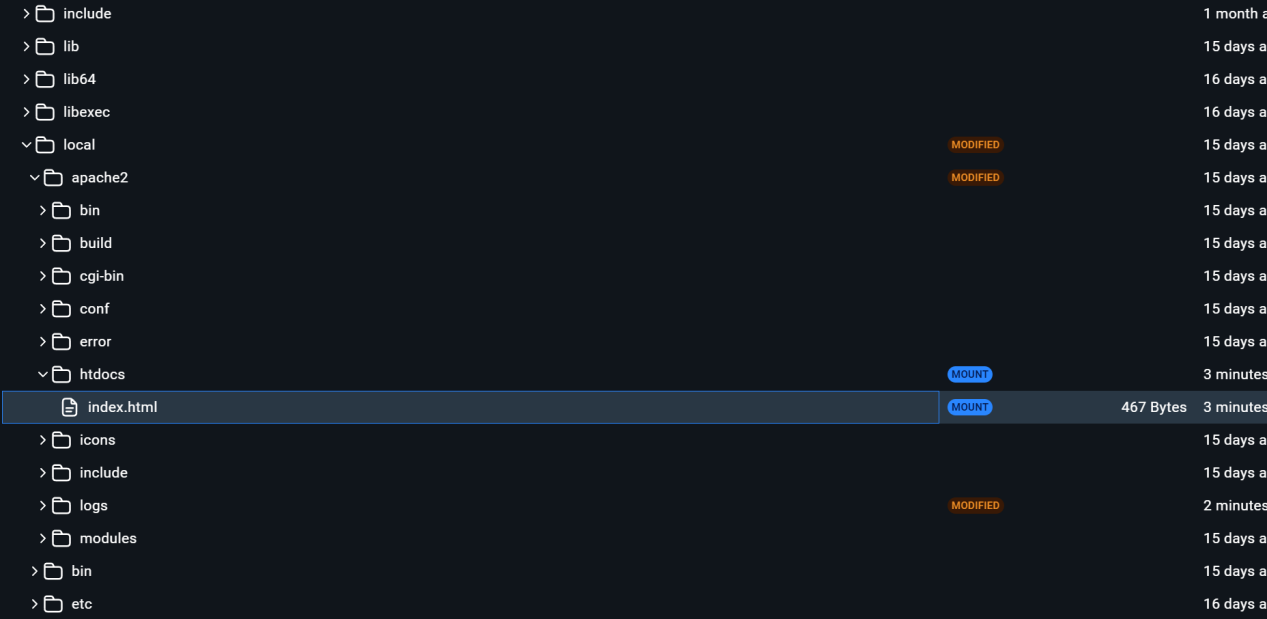


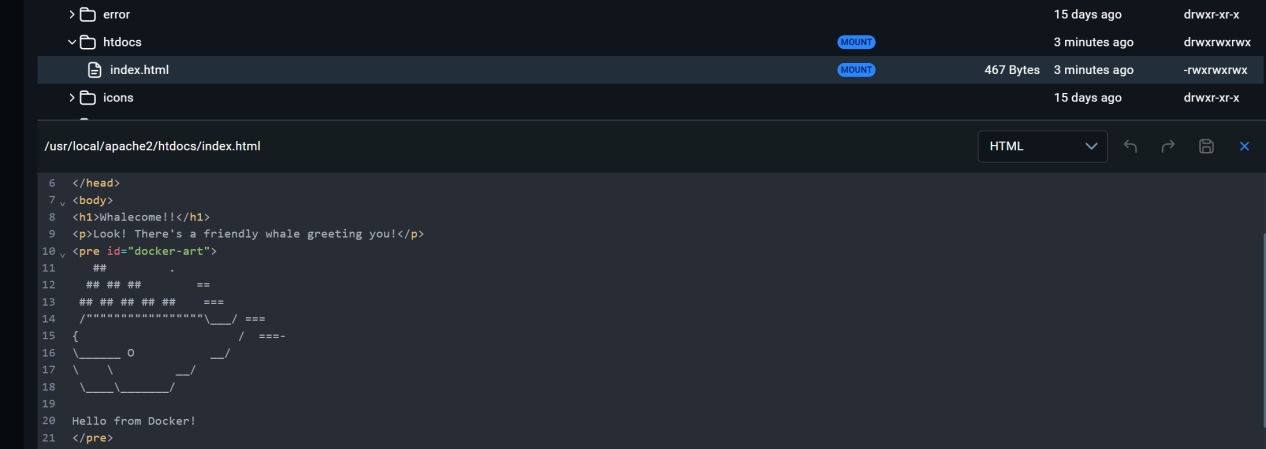


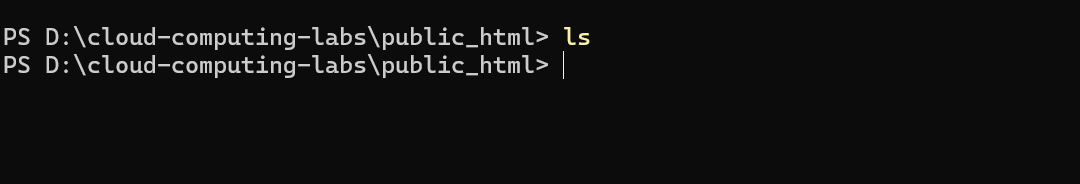


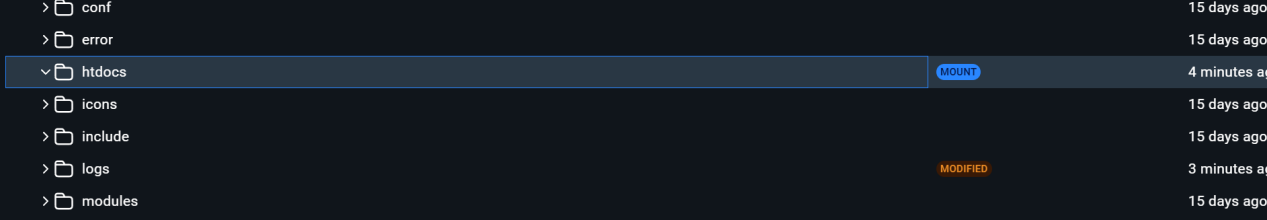












Multi-container applications

Docker Compose defines your entire multi-container application in a single YAML file called compose.yml. This file specifies configurations for all your containers, their dependencies, environment variables, and even volumes and networks.

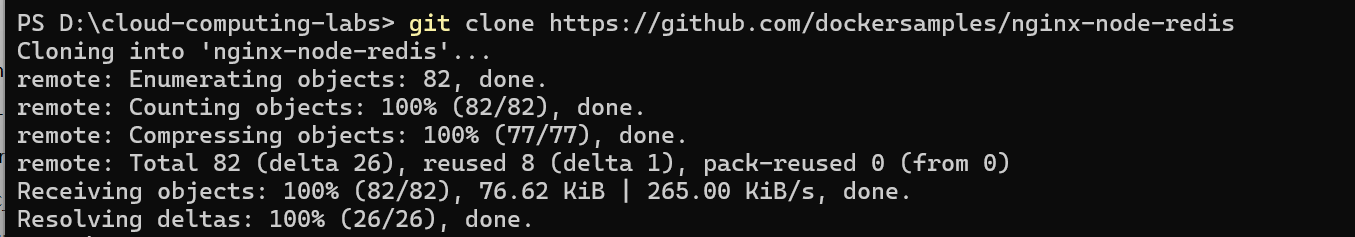
You don't need to run multiple docker run commands. All you need to do is define your entire multi-container application in a single YAML file. This centralizes configuration and simplifies management.

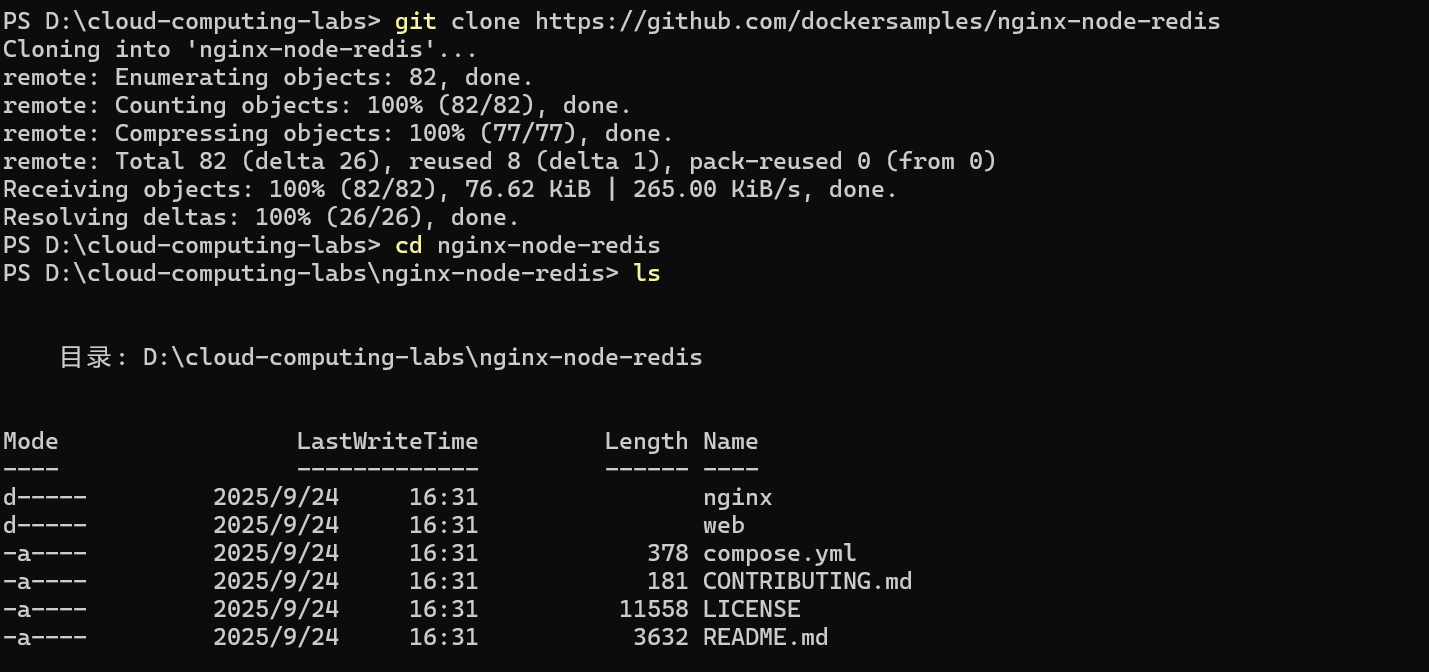
You can run containers in a specific order and manage network connections easily.

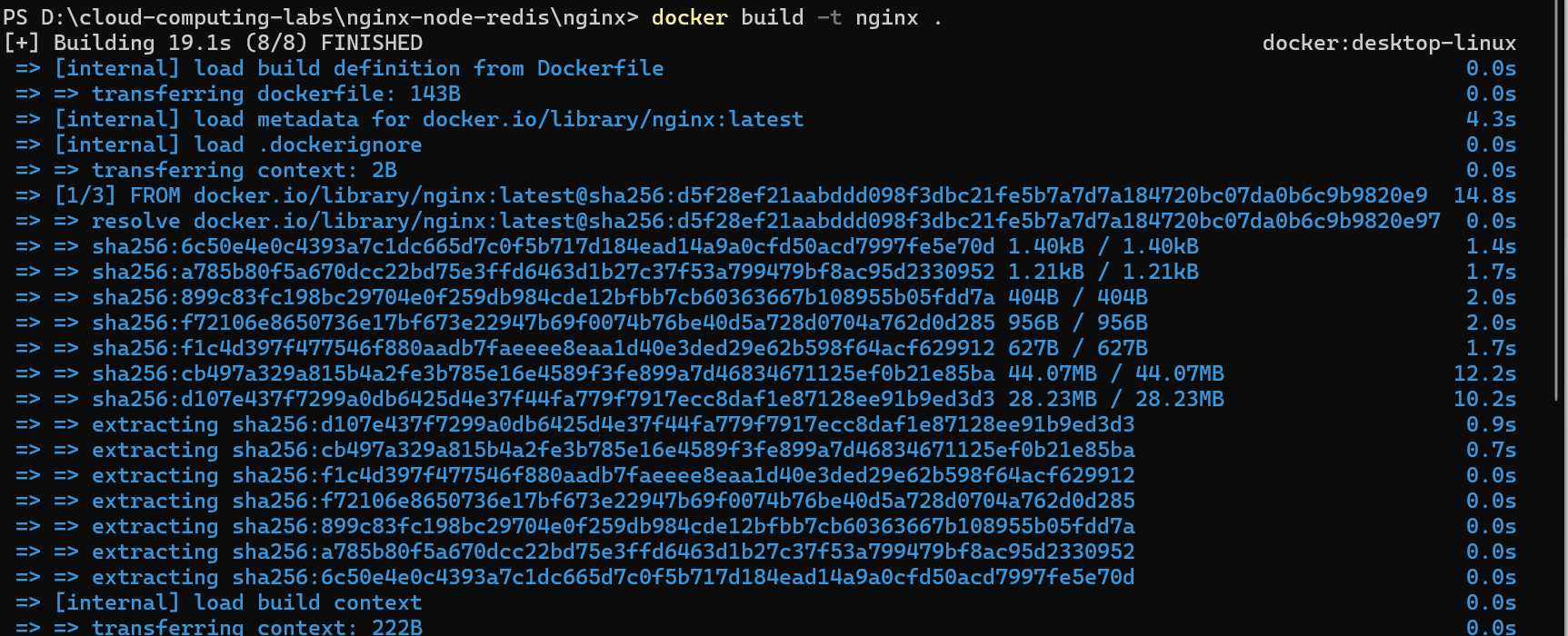
You can simply scale individual services up or down within the multi-container setup. This allows for efficient allocation based on real-time needs.

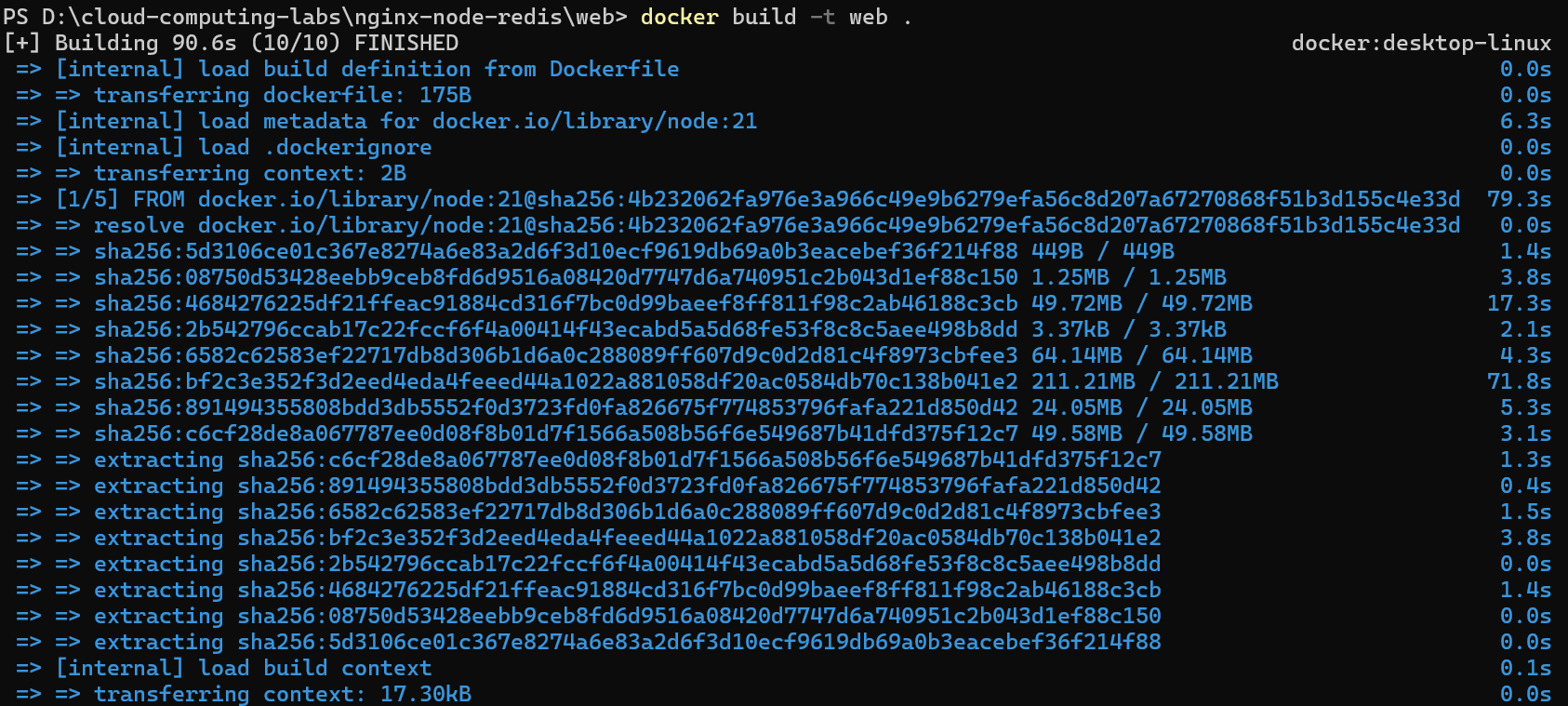
You can implement persistent volumes with ease.

It's easy to set environment variables once in your Docker Compose file.





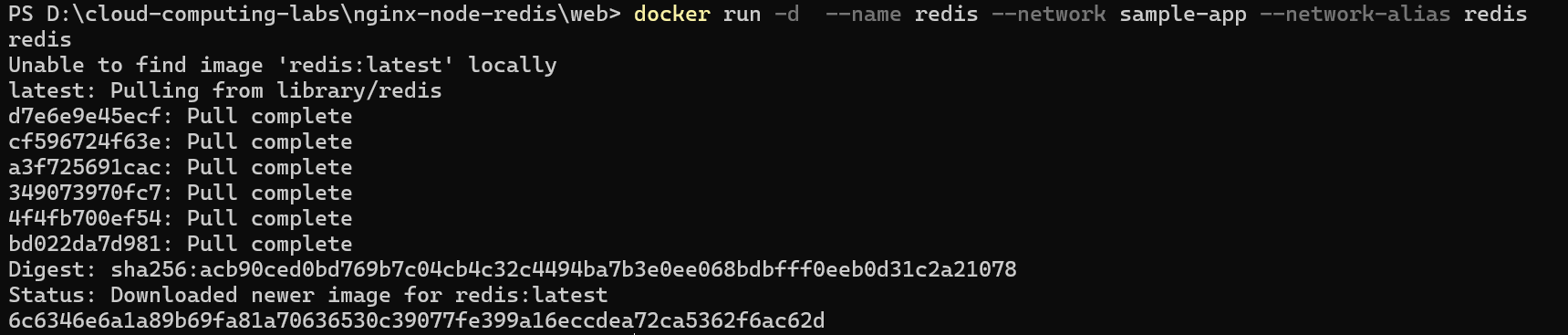




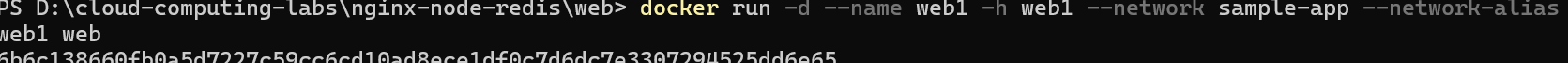
create a network for multi-container application to communicate through.



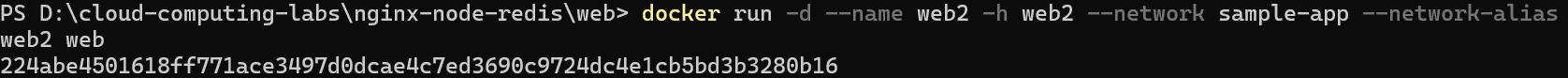
Start the Redis container by running the following command, which will attach it to the previously created network and create a network alias (useful for DNS lookups):



Start the first web container by running the following command:

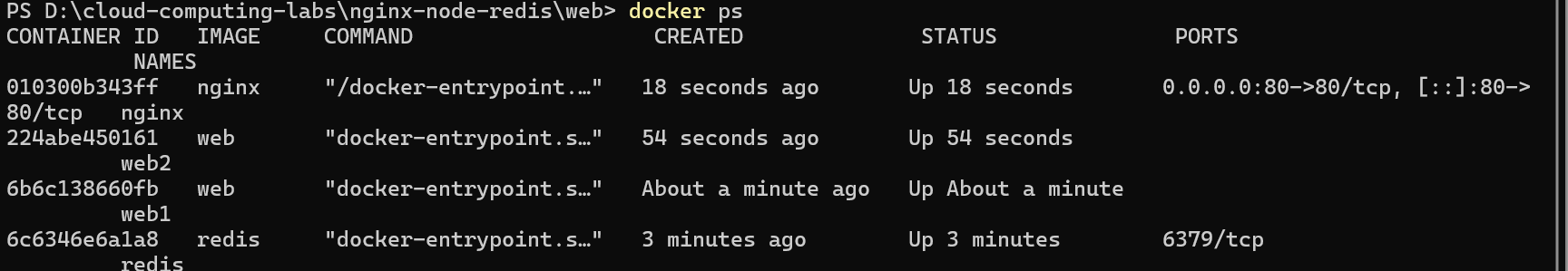


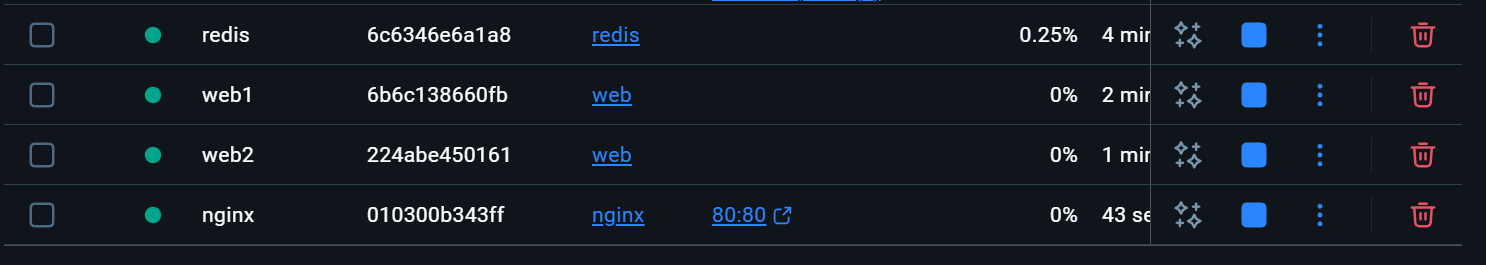
Start the second web container by running the following:

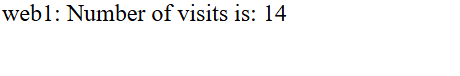


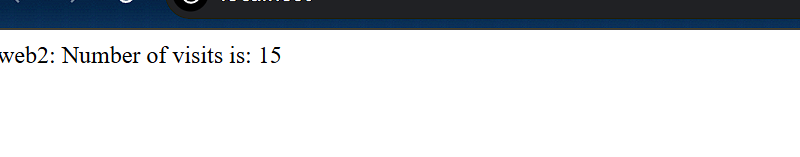
Start the Nginx container by running the following command:

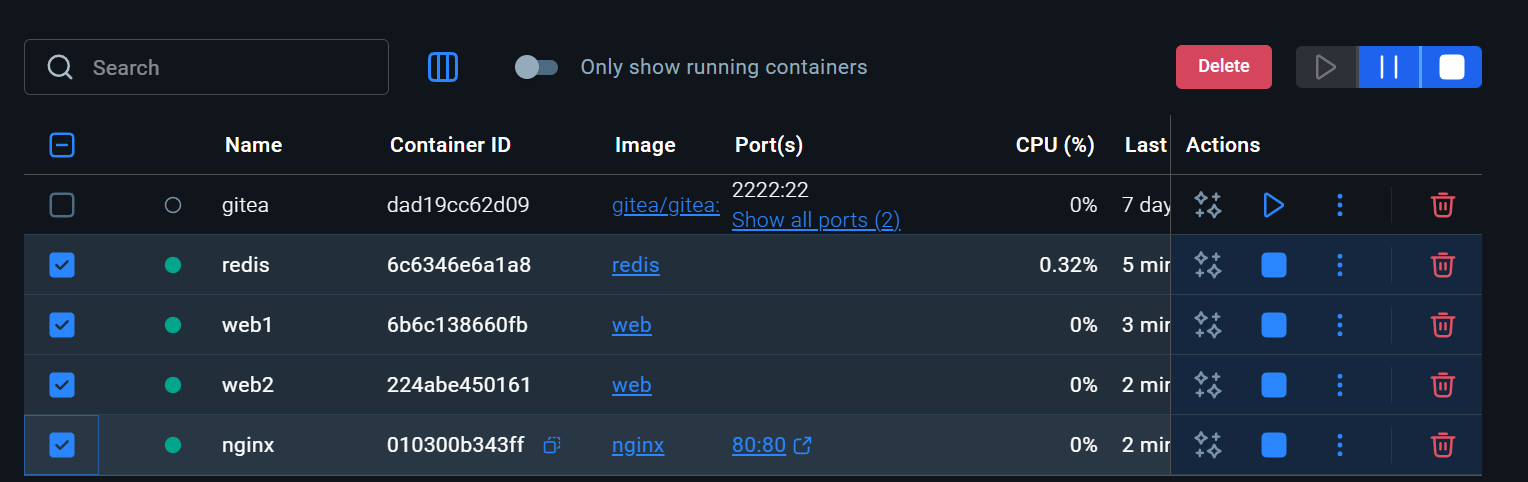












Simplify the deployment using Docker Compose

