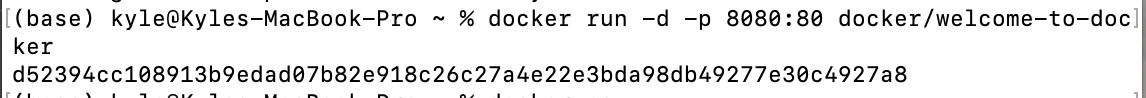
Open the CLI terminal and start the container using the command docker run  
docker run -d -p 8080:80 docker/welcome-to-docker  
The output is the full container ID

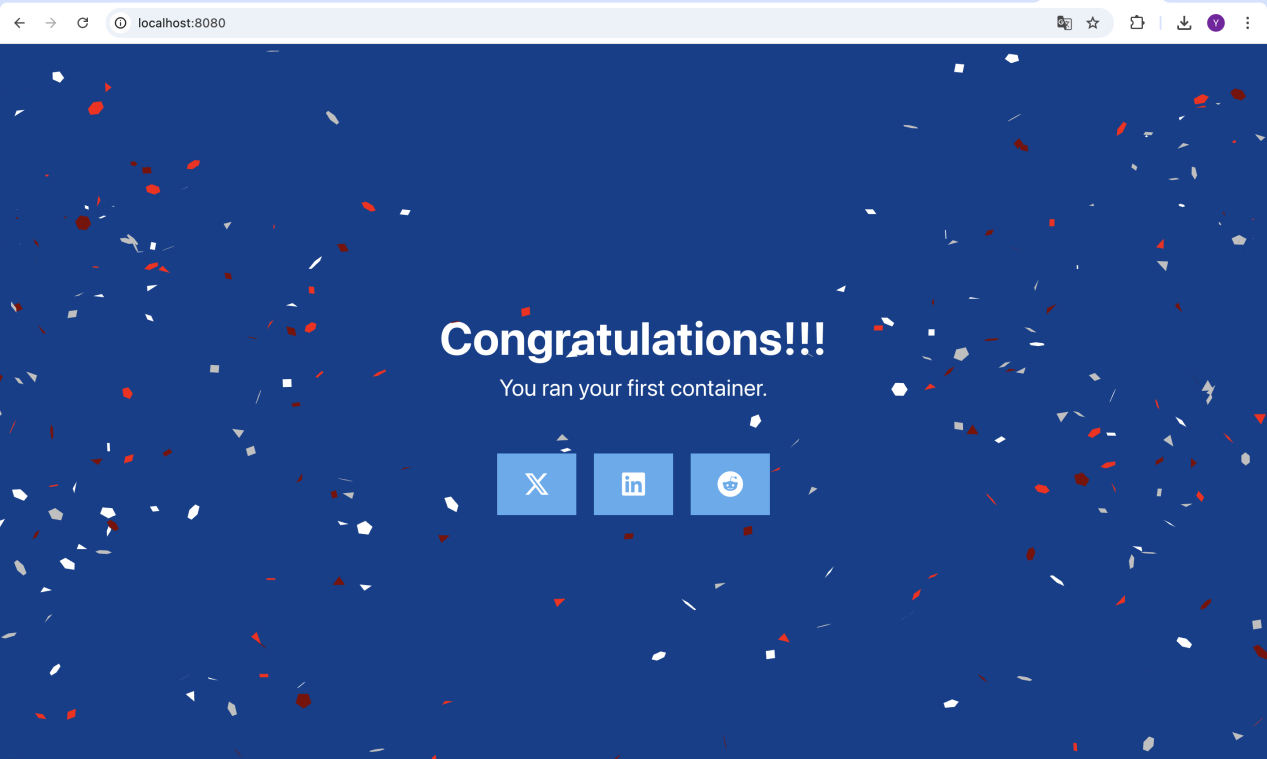


Verify that the container is up and running

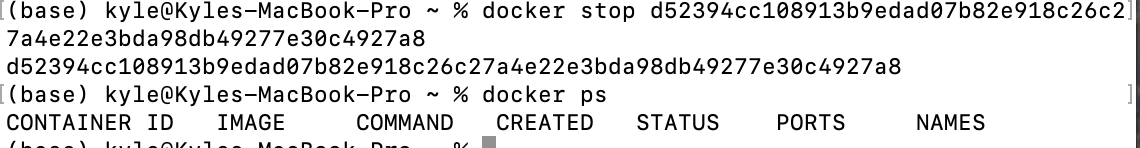
docker ps



Visit http://localhost:8080 in my browser

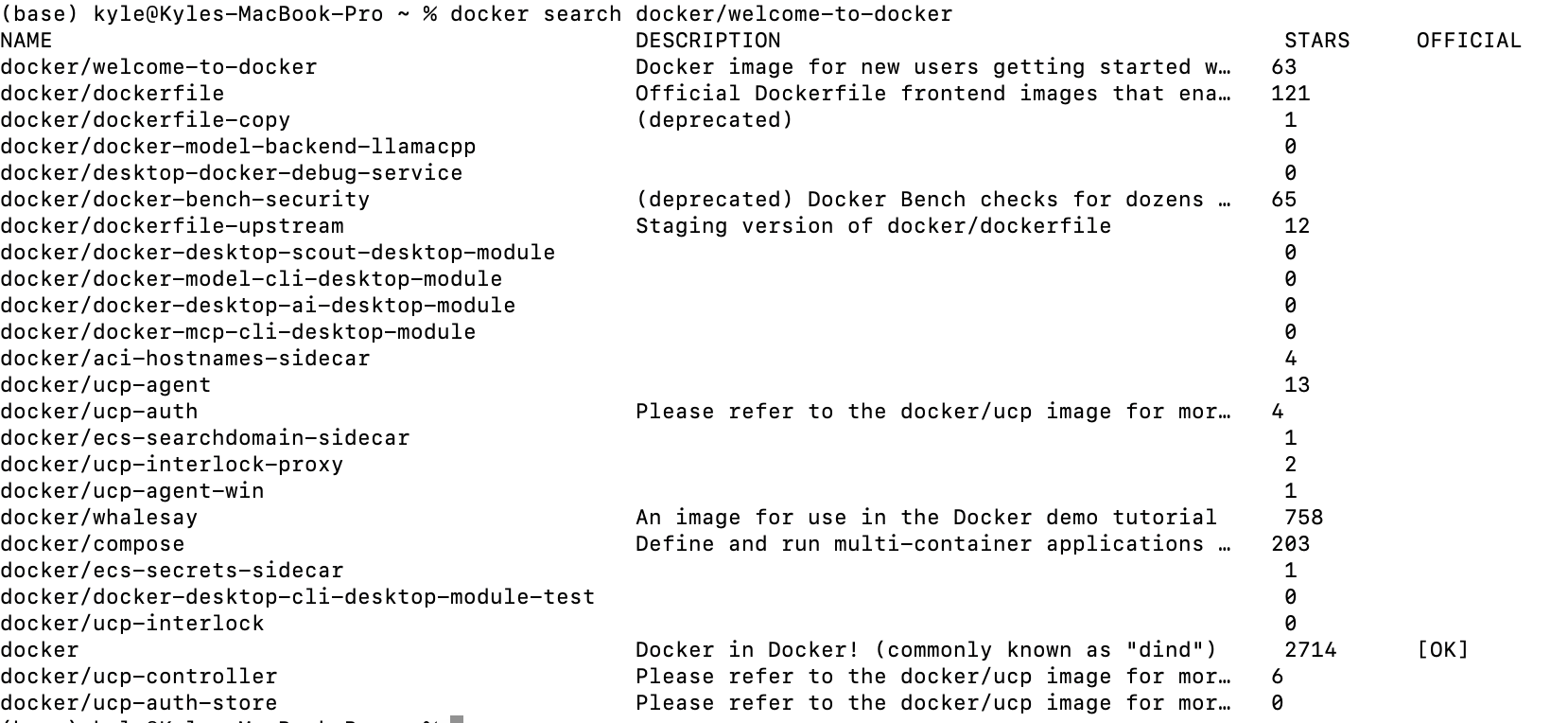


Stop my docker by supply id to the command



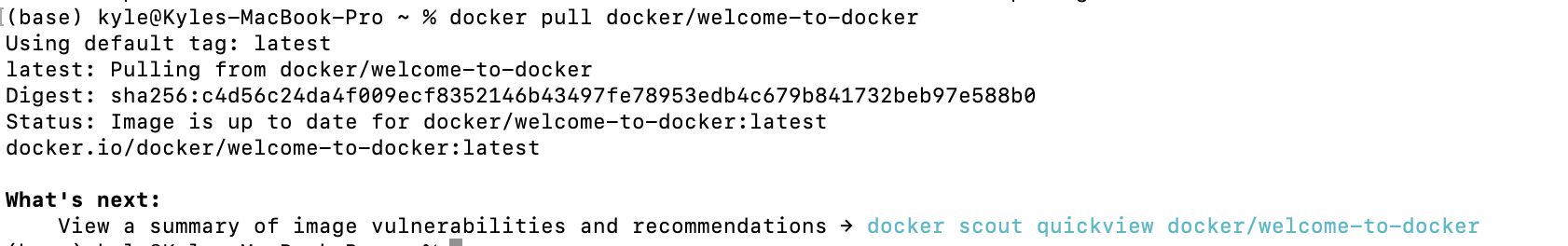
search for images using the [docker search](https://docs.docker.com/reference/cli/docker/search/) command

docker search docker/welcome-to-docker



Pull the image using the [docker pull](https://docs.docker.com/reference/cli/docker/image/pull/) command

docker pull docker/welcome-to-docker



List my downloaded images using the [docker image ls](https://docs.docker.com/reference/cli/docker/image/ls/) command:

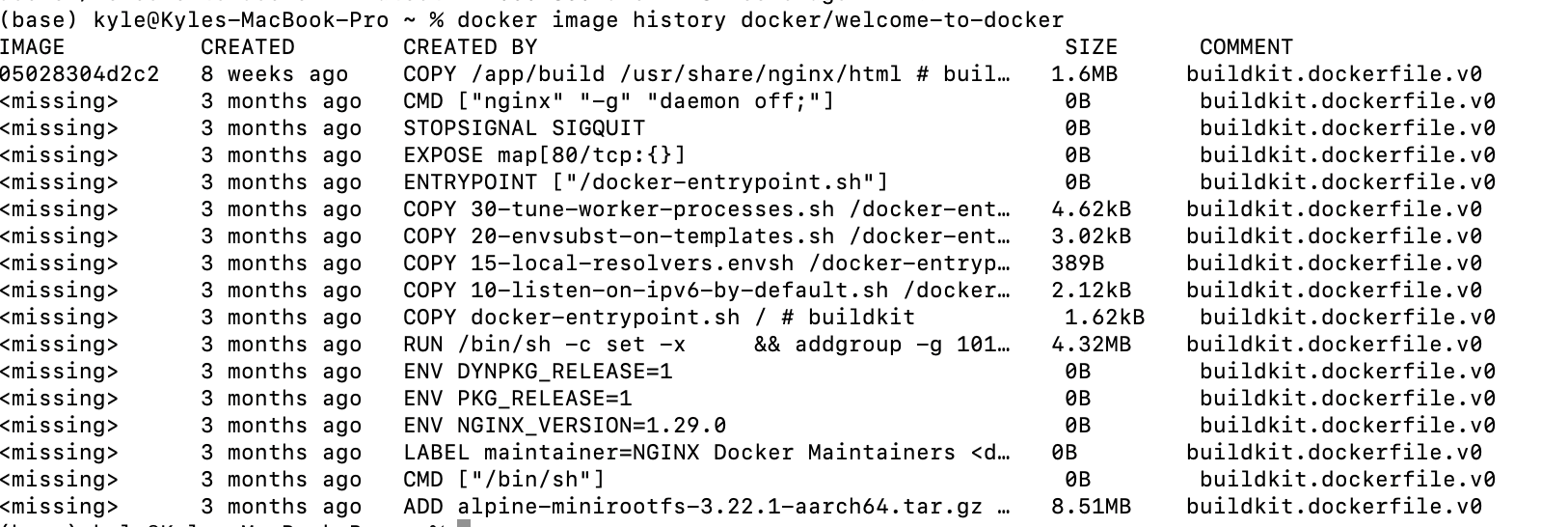
docker image ls



The image size represented here reflects the uncompressed size of the image, not the download size of the layers.

List the image's layers using the [docker image history](https://docs.docker.com/reference/cli/docker/image/history/) command:

docker image history docker/welcome-to-docker



Clone the GitHub repository using the following command:

git clone <https://github.com/dockersamples/helloworld-demo-node>

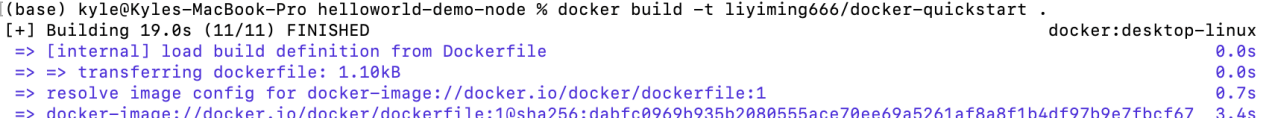


Navigate into the newly created directory.

cd helloworld-demo-node

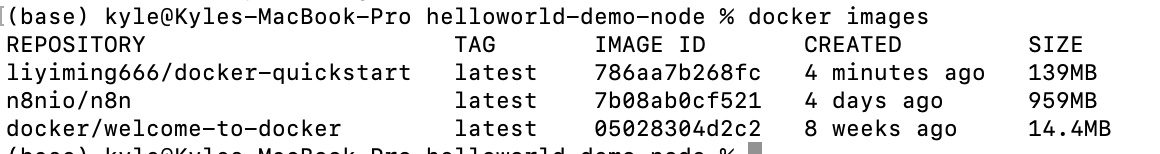
Run the following command to build a Docker image

docker build -t liyiming666/docker-quickstart .



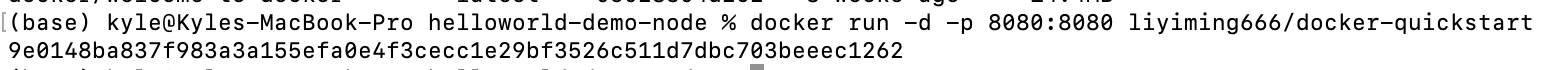
Run the following command to list the newly created Docker image:

docker images

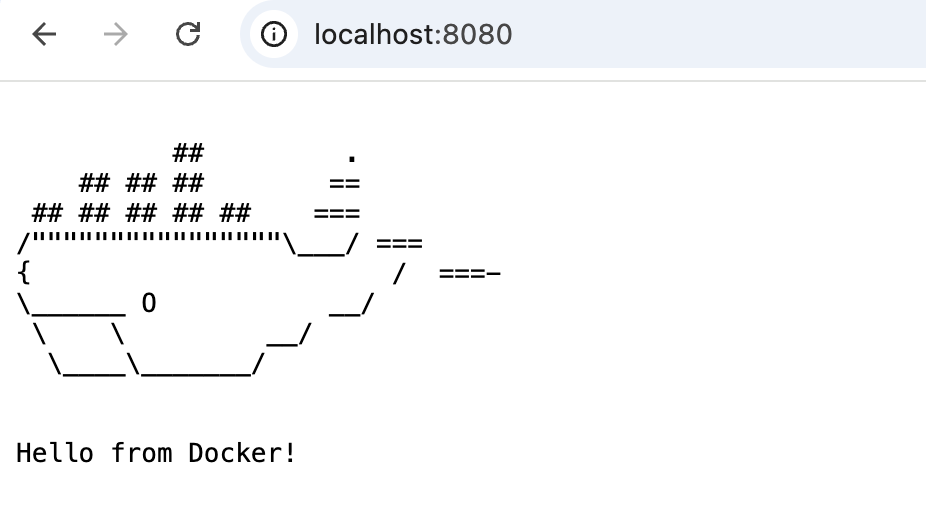


Start a container to test the image

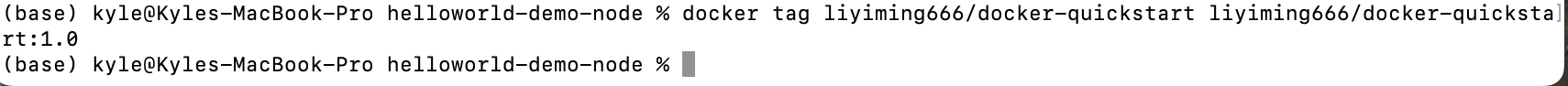
docker run -d -p 8080:8080 liyiming666/docker-quickstart



verify if the container is working by visiting [http://localhost:8080](http://localhost:8080/)

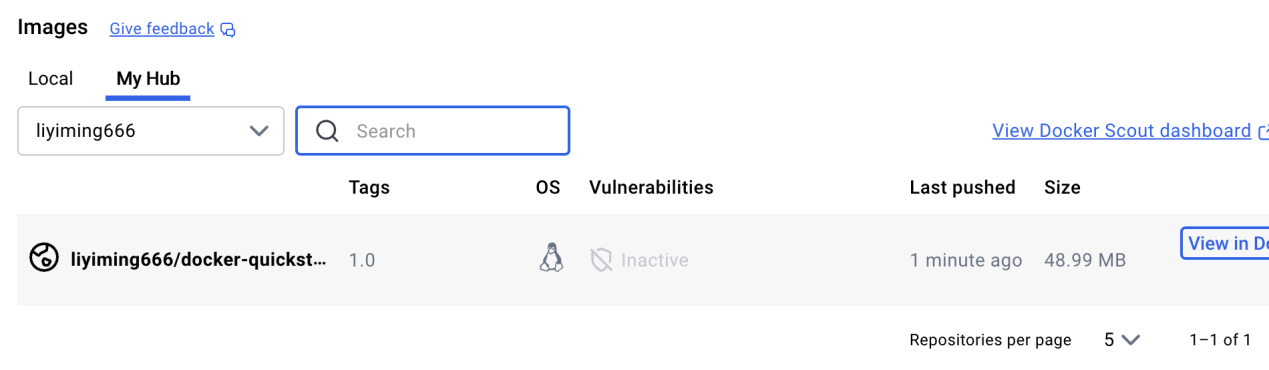


Use the [docker tag](https://docs.docker.com/reference/cli/docker/image/tag/) command to tag the Docker image.



Finally, it's time to push the newly built image to your Docker Hub repository by using the [docker push](https://docs.docker.com/reference/cli/docker/image/push/) command

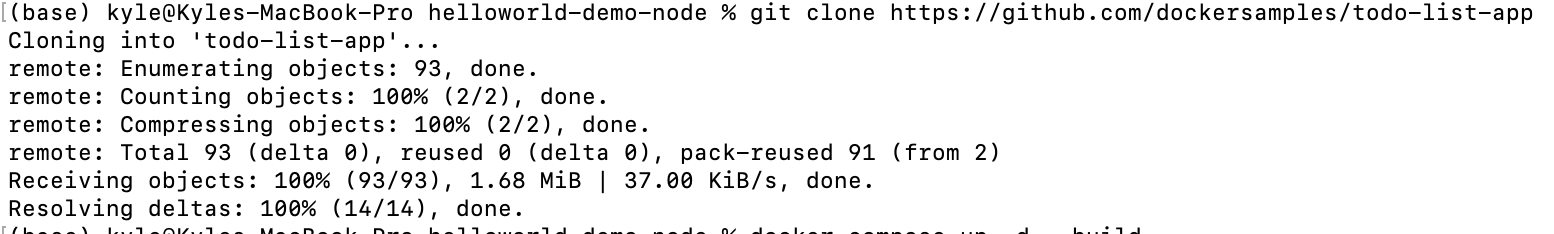




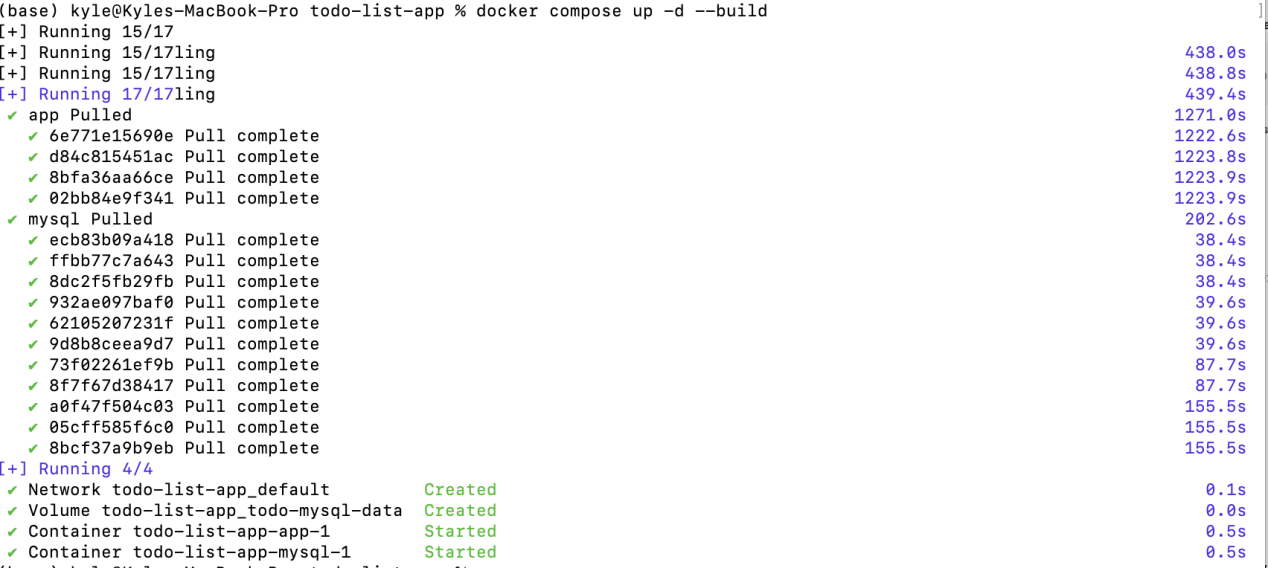
**What is Docker Compose?**

[Start the application](https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-docker-compose/" \l "start-the-application)

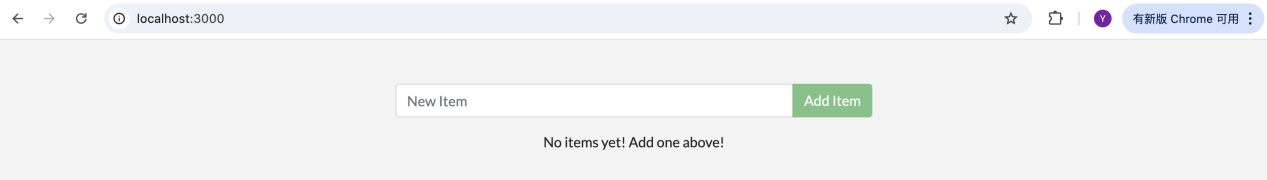
Open a terminal and [clone this sample application](https://github.com/dockersamples/todo-list-app).



Use the [docker compose up](https://docs.docker.com/reference/cli/docker/compose/up/) command to start the application:



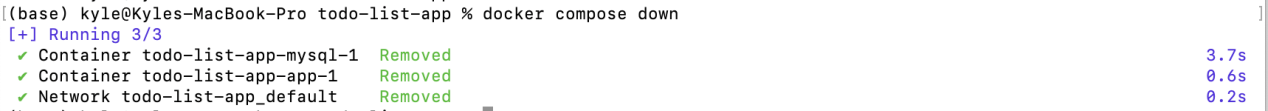
open [http://localhost:3000](http://localhost:3000/) in my browser to see the site.



Look at the Docker Desktop GUI, we can see the containers and dive deeper into their configuration.

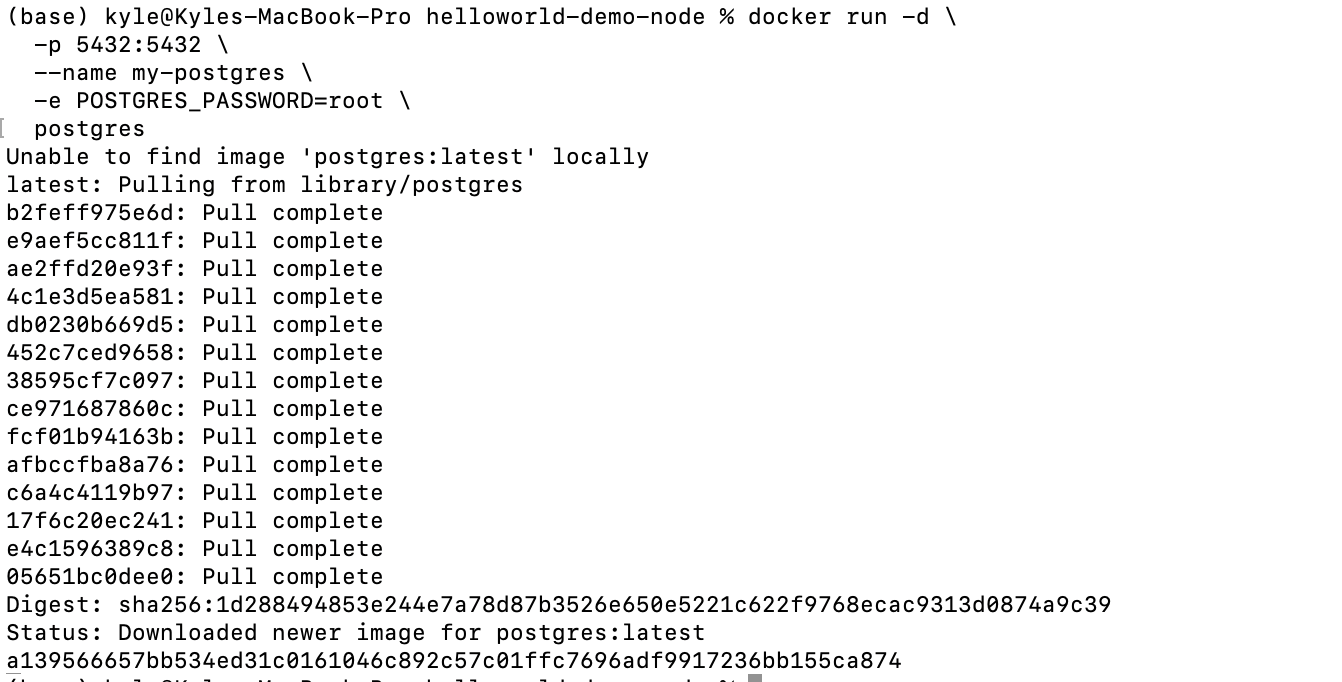


In the CLI, use the [docker compose down](https://docs.docker.com/reference/cli/docker/compose/down/) command to remove everything

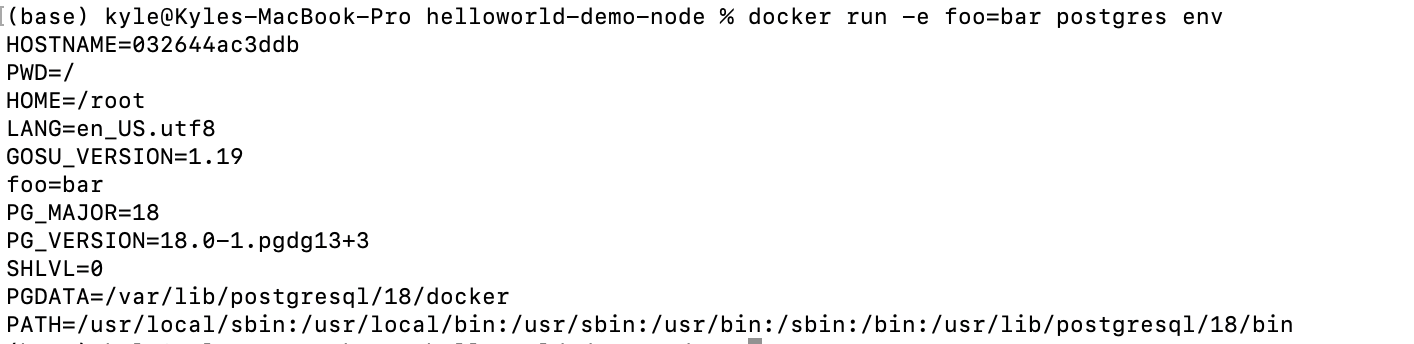


If we do want to remove the volumes, add the --volumes flag when running the docker compose down command:

docker compose down --volumes



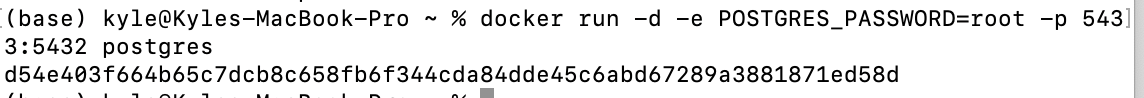
This option sets an environment variable foo inside the container with the value bar.



This command limits container memory usage to 512 MB and defines the CPU quota of 0.5 for half a core.



Start a second Postgres container mapped to a different port.



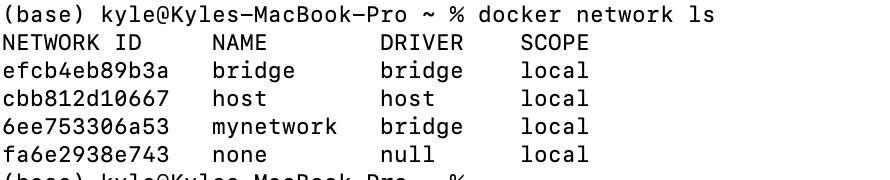
截屏2025-09-28 17.35.39截屏2025-09-28 17.35.47

Follow the steps to see how to connect a Postgres container to a custom network.

Create a new custom network by using the following command:

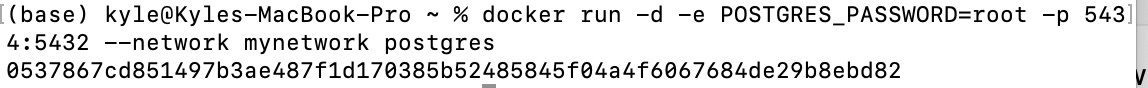
截屏2025-09-28 17.37.21

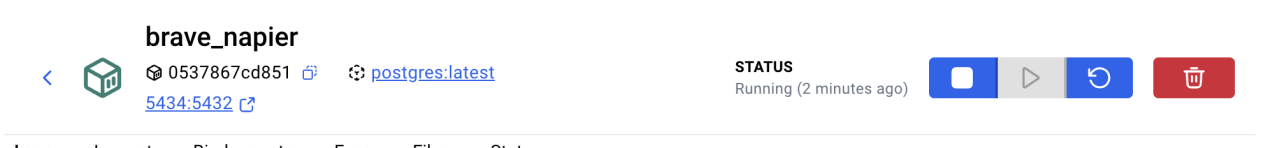
Verify the network by running the following command:



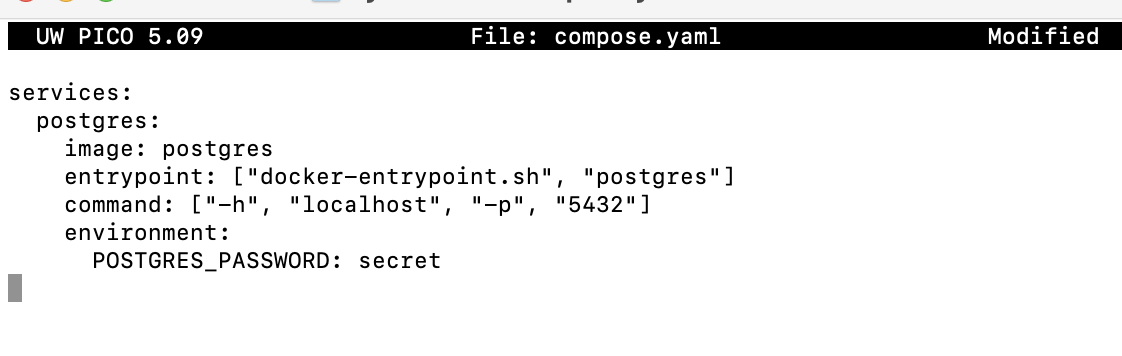
This command lists all networks, including the newly created "mynetwork".

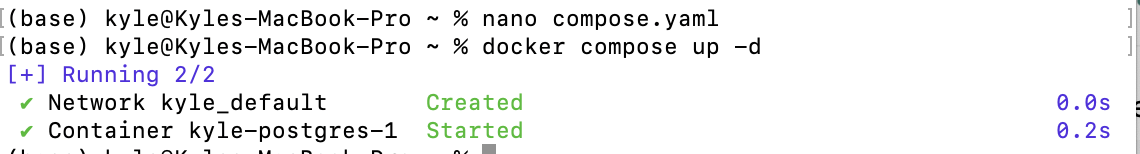
Connect Postgres to the custom network by using the following command:



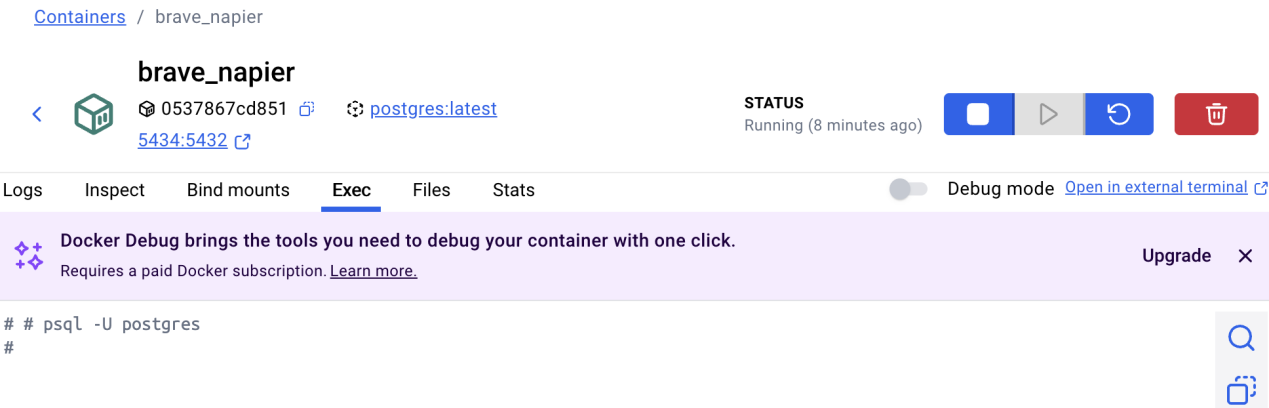


Create compose.yaml file

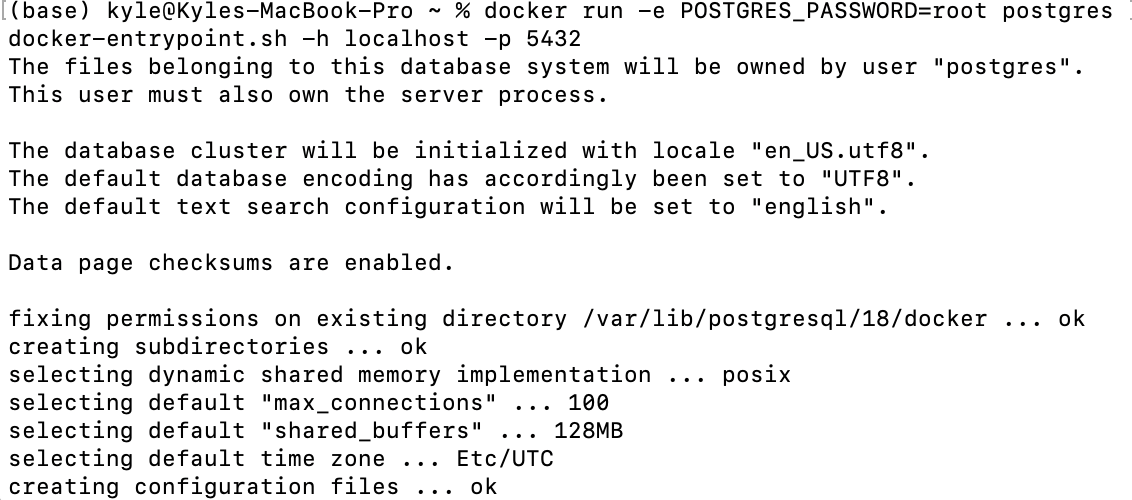




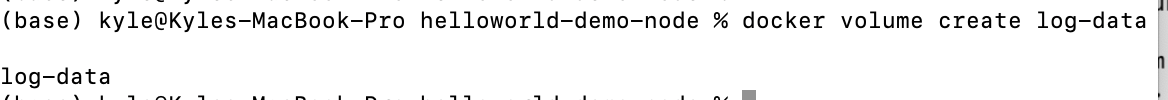
Open the Docker Desktop Dashboard, select the Postgres container and select Exec to enter into the container shell.



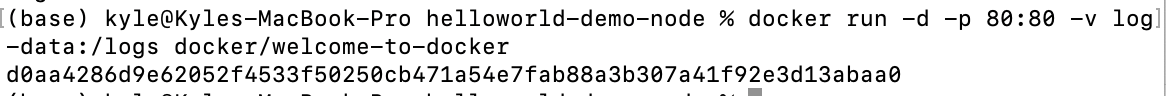
Runs a Postgres container, sets an environment variable for password authentication, overrides the default startup commands and configures hostname and port mapping.



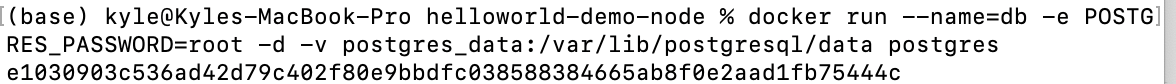
create a volume named log-data



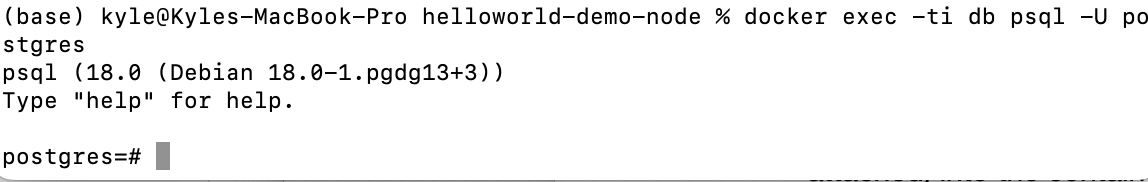
When starting a container with the following command, the volume will be mounted (or attached) into the container at /logs



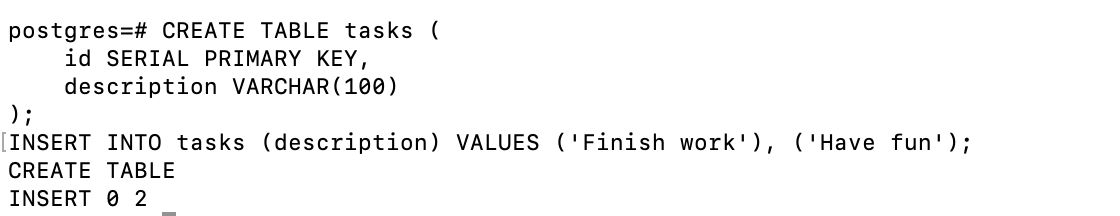
Start a container using the [Postgres image](https://hub.docker.com/_/postgres)



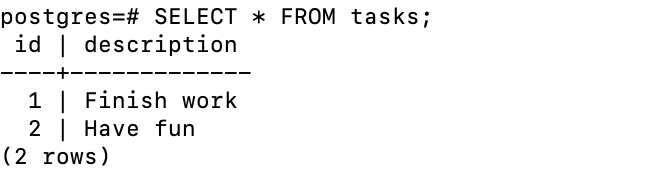
Connect to the database



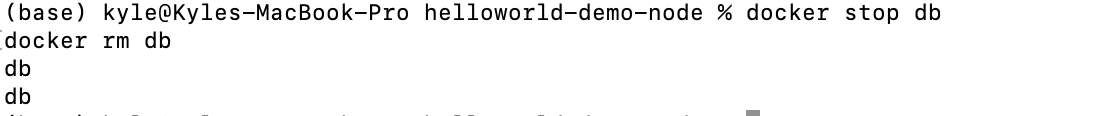
create a database table and insert two records:



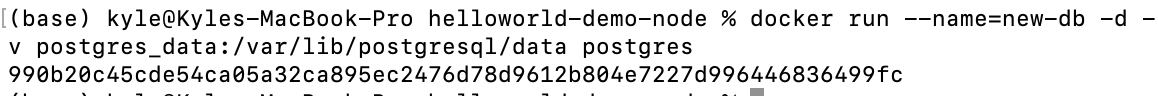
Verify the data is in the database

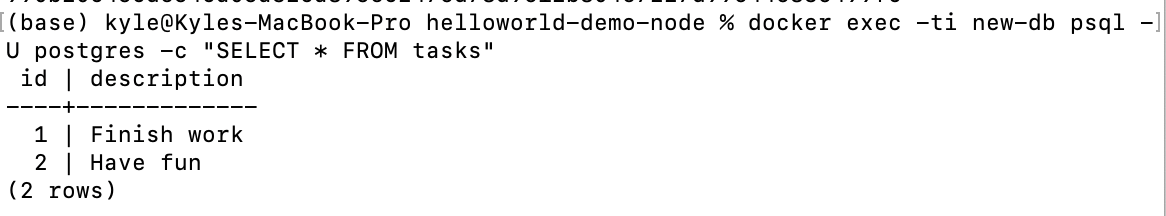


Stop and remove the database container.

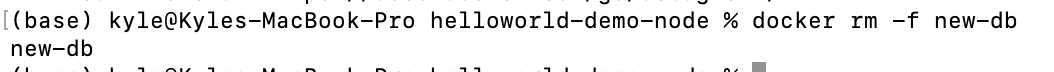


Start a new container,attaching the same volume with the persisted data:

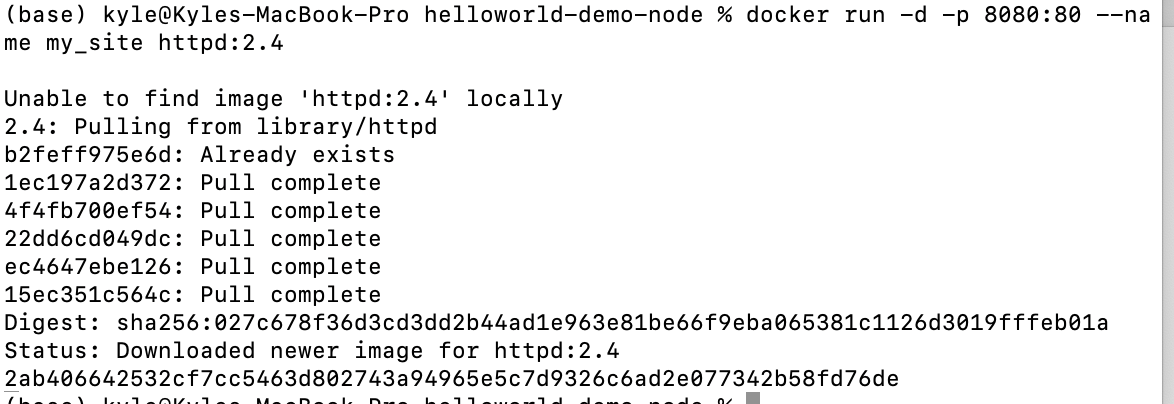
Verify the database still has the records

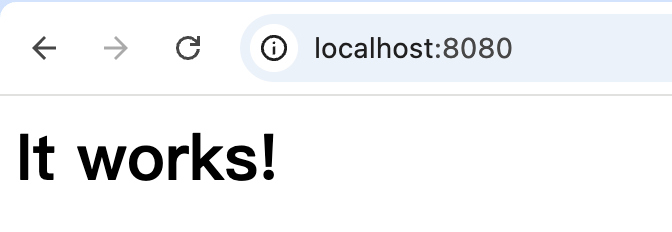


Before removing a volume, it must not be attached to any containers.



Start a container using the [httpd](https://hub.docker.com/_/httpd) image



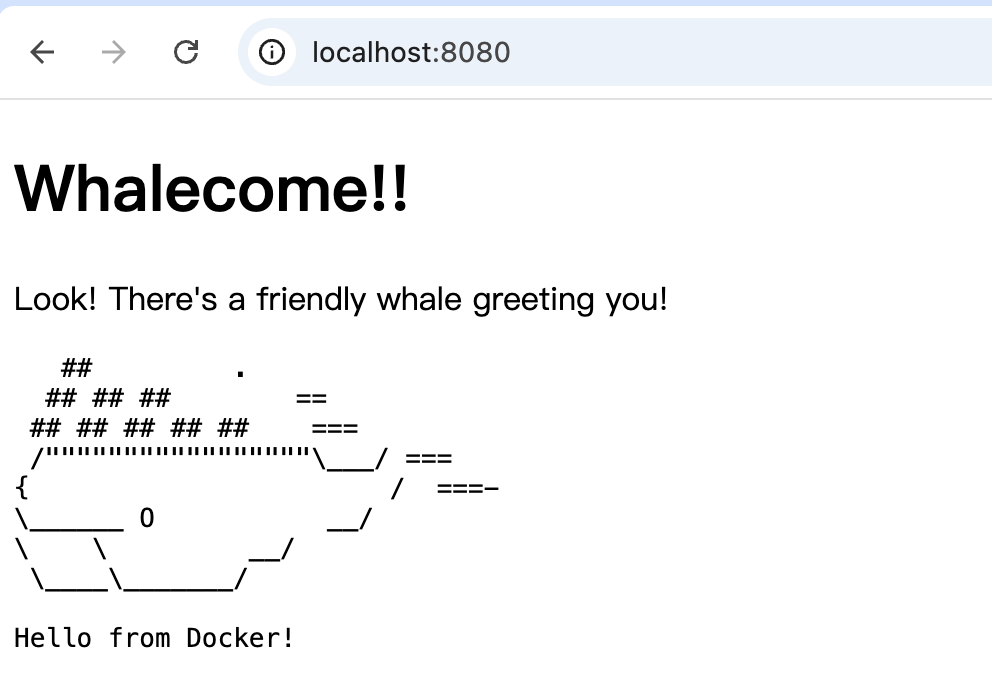


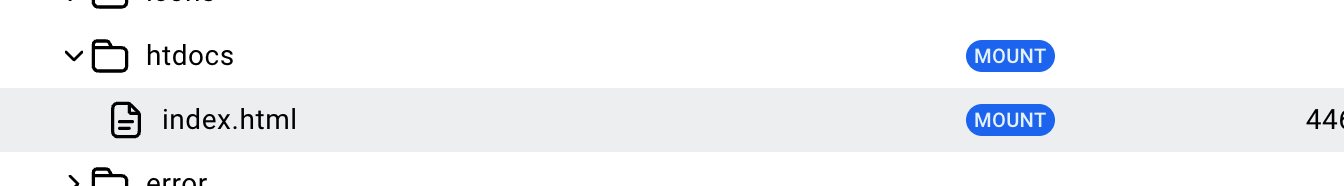
Create a new directory called public\_html

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 us with a friendly whale.

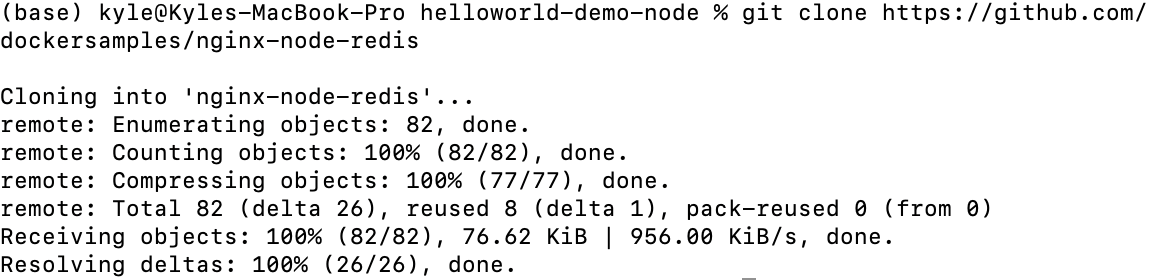


It's time to run the container.

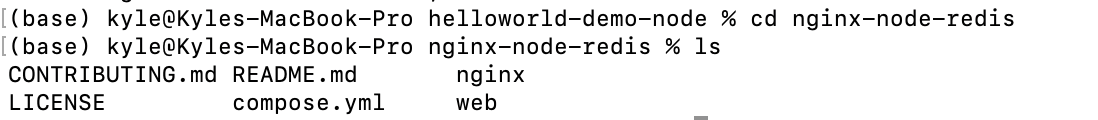
We can see from docker that index.html was mounted。



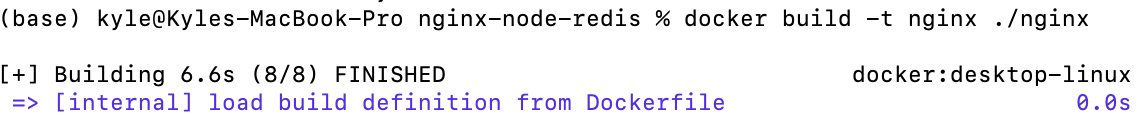
 clone the sample application repository.

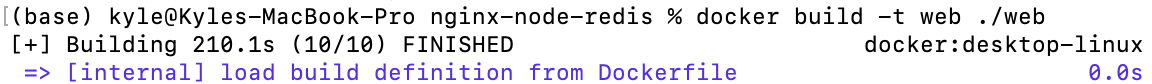


Navigate into the nginx-node-redis directory:



Build the nginx image and first web image





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



Start the Redis container



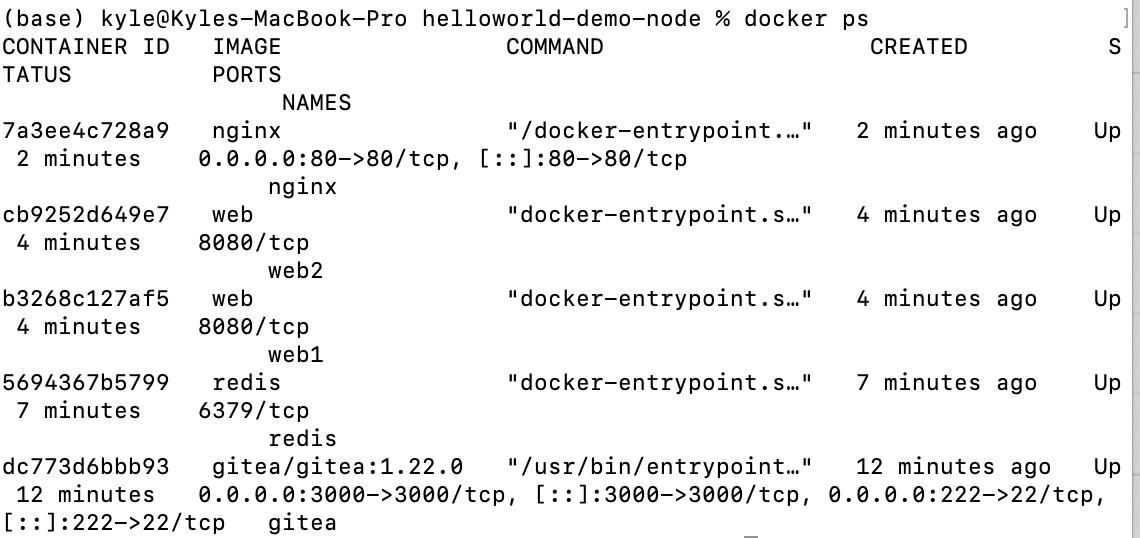
Start the first and second web container

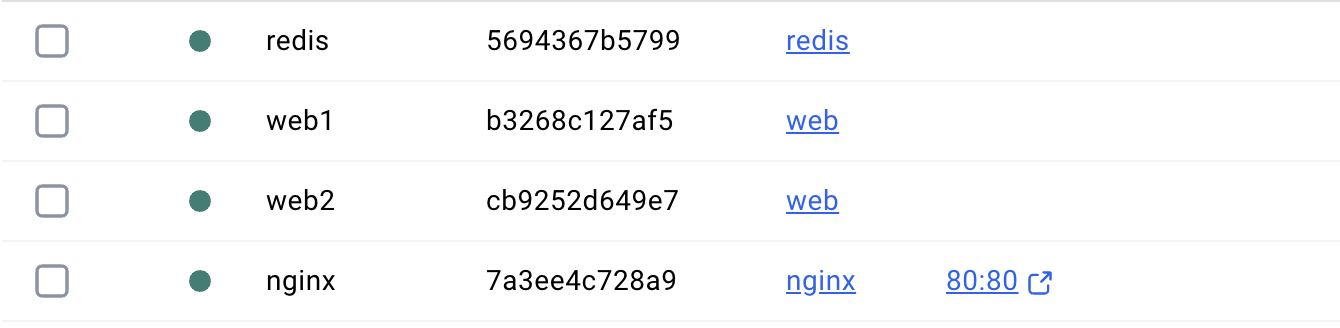


Start the Nginx container



Verify the containers are up





open [http://localhost](http://localhost/) in my browser to see the site.

