1.

https://docs.docker.com/get-started/docker-concepts/running-containers/publishing-ports/

I followed the dockers inbuilt tutorial to build a test website. Here shows the port mapped to 8080

here I ran the following

A screenshot of a computer

AI-generated content may be incorrect.

This made a new build that used a specified port.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Said website, deployed on the chosen po

2.

https://docs.docker.com/get-started/docker-concepts/running-containers/overriding-container-defaults/

1. Ran a Postgres container on the default port:

docker run -d -e POSTGRES\_PASSWORD=secret -p 5432:5432 postgres

Resulted in the container brave\_snyder.

1. Ran another Postgres container mapped to a different host port (5433):

docker run -d -e POSTGRES\_PASSWORD=secret -p 5433:5432 postgres

Resulted in the container xenodochial\_goldstine.

1. Created a custom Docker network:

docker network create mynetwork

1. Attached a new Postgres container to the custom network, mapped to host port 5434:

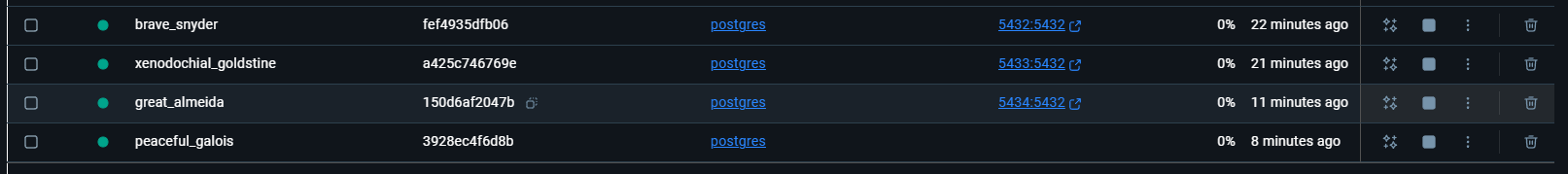
docker run -d -e POSTGRES\_PASSWORD=secret -p 5434:5432 --network mynetwork postgres

Resulted in the container great\_almeida.

1. Started a resource-limited Postgres container (no published port):

docker run -d -e POSTGRES\_PASSWORD=secret --memory="512m" --cpus=".5" postgres

Resulted in the container peaceful\_galois, likely running internally only.



3.

https://docs.docker.com/get-started/docker-concepts/running-containers/persisting-container-data/

A black background with text on it

AI-generated content may be incorrect.

1. Created a Docker volume to persist Postgres data:  
   docker volume create postgres\_data
2. Verified that the volume was created:  
   docker volume ls
3. Ran a Postgres container with the volume mounted to the correct data path:  
   docker run --name=db -e POSTGRES\_PASSWORD=secret -d -v postgres\_data:/var/lib/postgresql/data postgres
4. Connected to the running container’s Postgres CLI:  
   docker exec -ti db psql -U postgres
5. Later, launched a new Postgres container using the same volume (to simulate container deletion and persistence):  
   docker run --name=new-db -d -v postgres\_data:/var/lib/postgresql/data postgres
6. Verified that the previously created data was still there:  
   docker exec -ti new-db psql -U postgres -c "SELECT \* FROM tasks"

Result:  
Data (Finish work, Have fun) persisted successfully across container deletion — confirms volume was used correctly.

here is my volumes contents

A black rectangular object with white lines

AI-generated content may be incorrect.

4.

https://docs.docker.com/get-started/docker-concepts/running-containers/sharing-local-files/

1. Pull Required Docker Images

docker pull httpd:2.4

docker pull postgres

This downloads the latest versions of the httpd and postgres images from Docker Hub.

2. Start the Initial Apache Container

docker run -d -p 8080:80 --name my\_site httpd:2.4

This launches the httpd web server and maps port 8080 on your host to port 80 in the container.

3. Access the Web Server

Open a browser and navigate to:

http://localhost:8080

4. Stop and Delete the Container

Before using a bind mount, remove the container:

docker rm -f my\_site

5. Create a Host Directory with HTML Content

mkdir C:\Users\BBarr\public\_html

notepad C:\Users\BBarr\public\_html\index.html

Paste the whale-themed HTML content into index.html.

6. Run the Container with a Bind Mount

docker run -d --name my\_site -p 8080:80 -v C:/Users/BBarr/public\_html:/usr/local/apache2/htdocs/ httpd:2.4

This mounts your local folder public\_html into the container’s Apache web directory, replacing the default content.

7. Verify the HTML File is Served

http://localhost:8080/index.html

You should now see your custom whale page.

8. View Mounted Files in Docker Desktop

Open Docker Desktop.

Go to Containers > my\_site > Files.

Navigate to /usr/local/apache2/htdocs/.

Confirm index.html appears there.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

5.

<https://docs.docker.com/get-started/docker-concepts/running-containers/multi-container-applications/>

1. Clone Sample Application

git clone https://github.com/dockersamples/nginx-node-redis

cd nginx-node-redis

1. Build Custom Nginx Image

cd nginx

docker build -t nginx .

cd ..

1. Build Web App Image (Node.js)

cd web

docker build -t web .

1. Create Docker Network

docker network create sample-app

1. Start Redis Container

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

1. Start Web Containers

docker run -d --name web1 -h web1 --network sample-app --network-alias web1 web

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

1. Start Nginx Container (Expose Port 80)

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

1. Verify All Containers Are Running

docker ps

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.