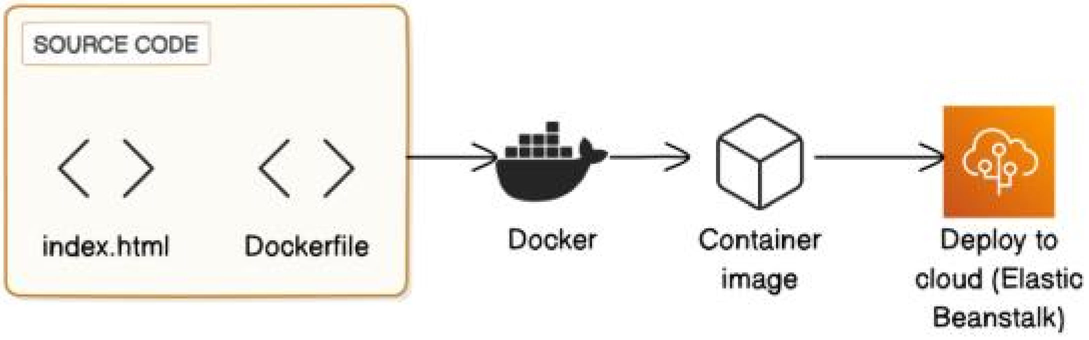
Containers on Elastic Beanstalk



What you're building in this project today

Introducing Today's Project!

What is Docker?
Docker is a platform for packaging apps into containers, making them portable and consistent. In today's project, I used Docker to build a custom container image with Nginx, run it locally, and deploy the containerized app on AWS Elastic Beanstalk.

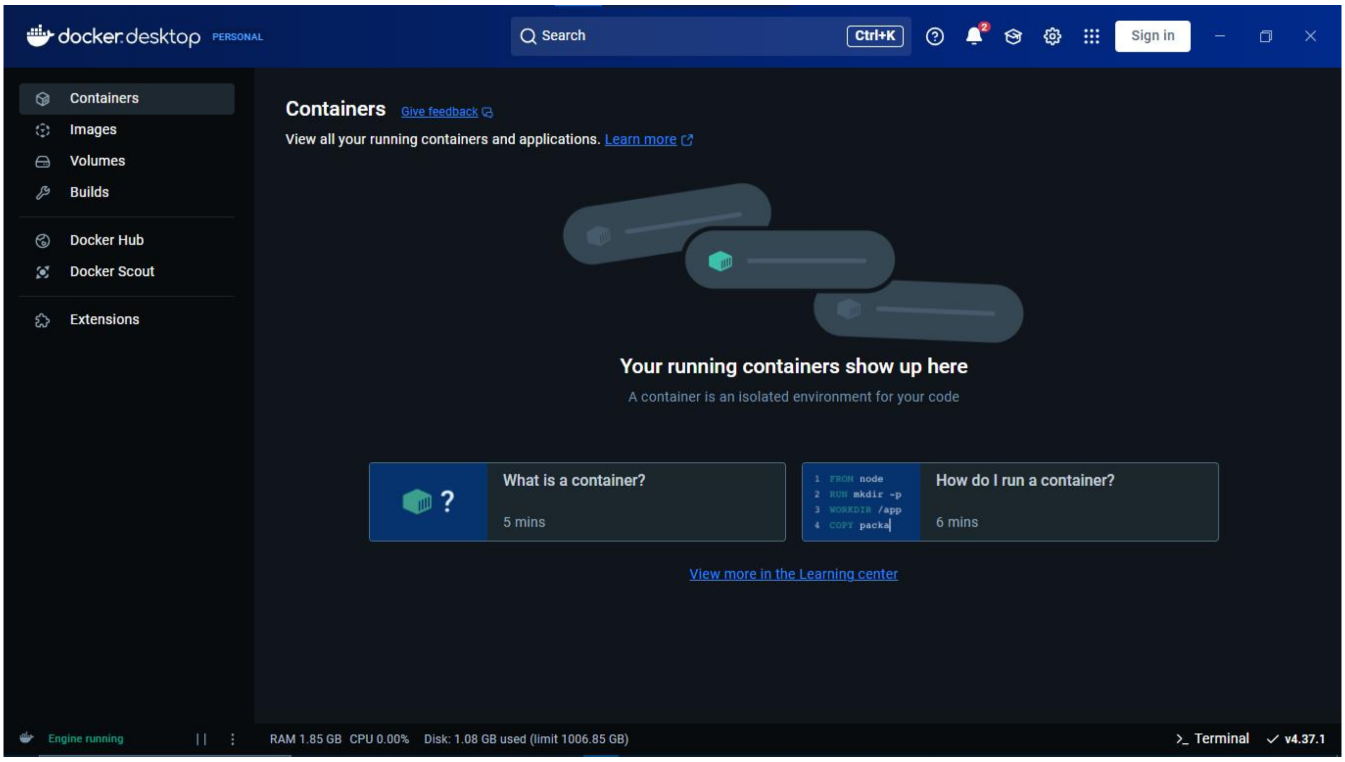
One thing I didn't expect...
One thing I didn't expect in this project was running into a port conflict issue when trying to deploy my custom container. I had to stop the existing container using port 80 before I could successfully run and deploy my custom image.

This project took me...
This project took me around 1.5 to 2 hours. This includes setting up Docker, building and running the custom container locally, troubleshooting port conflicts, deploying the app to Elastic Beanstalk, and making sure everything was working correctly.

Understanding Containers and Docker

Containers
Containers are lightweight, isolated environments that bundle applications with their dependencies. They are useful because they ensure consistent performance across systems, simplify deployment, and improve scalability and resource efficiency.
A container image is a lightweight, standalone package containing everything needed to run an application, including code, runtime, libraries, and dependencies. It serves as a blueprint to create containers, ensuring consistency across environments.

Docker
Docker is an open-source platform for building, running, and managing containerized applications. Docker Desktop is a developer tool for macOS and Windows that provides a GUI and CLI to manage containers, images, and environments locally.
The Docker daemon is a background process that manages Docker containers, images, networks, and volumes on a host system. It listens for Docker API requests, handles container creation, and monitors container execution, ensuring smooth operation.



Containers Give feedback

View all your running containers and applications. Learn more

Your running containers show up here

A container is an isolated environment for your code

What is a container? 5 mins

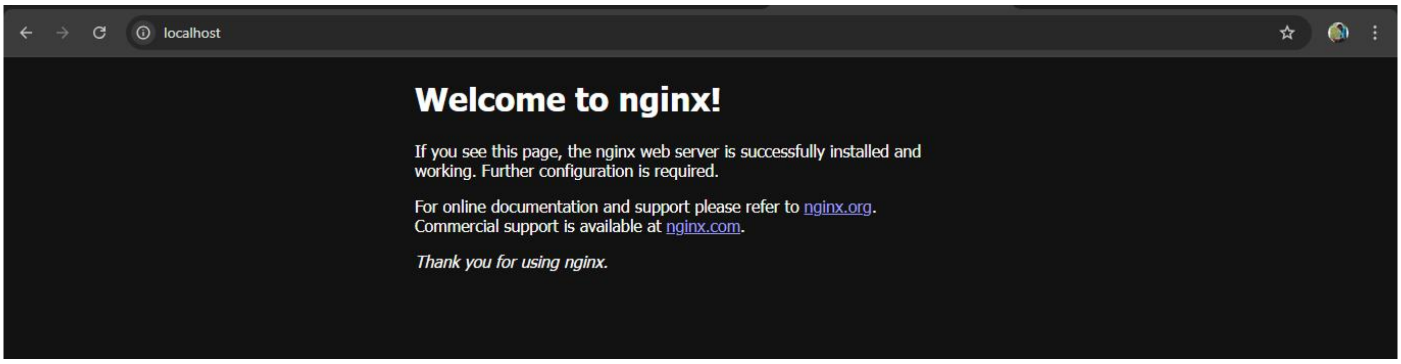
How do I run a container? 6 mins

View more in the Learning center

RAM 1.85 GB CPU 0.00% Disk: 1.08 GB used (limit 1006.85 GB)

Running an Nginx Image

Nginx is an open-source web server and reverse proxy server, known for its high performance and low resource usage. It is commonly used for serving static content, load balancing, and acting as a reverse proxy for web applications and APIs. The command I ran to start a new container was: docker run -d -p 80:80 nginx. This command starts a new container in detached mode, maps port 80 on my machine to port 80 in the container, and uses the Nginx image to serve a webpage.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

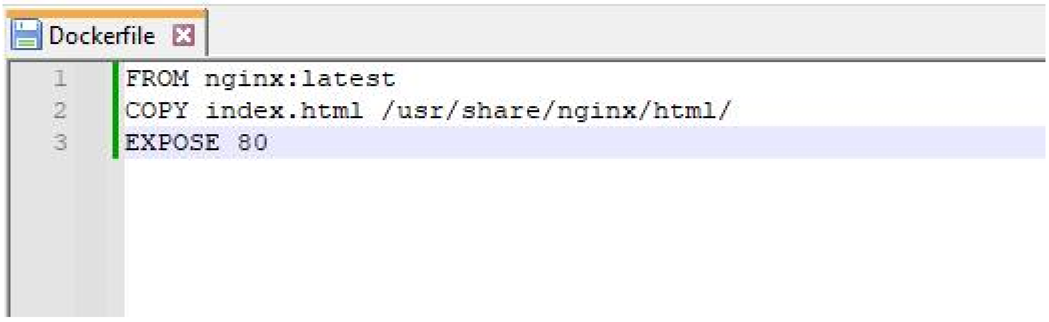
Thank you for using nginx.

Creating a Custom Image

The Dockerfile is a text file containing instructions for Docker to build a custom container image. It defines the base image, dependencies, files to copy, environment variables, and commands to run, ensuring consistent and automated image creation.

My Dockerfile tells Docker three things:
1. FROM nginx:latest – Use the latest Nginx image as the base.
2. COPY ./index.html /usr/share/nginx/html/ – Copy a custom `index.html` to the container.
3. EXPOSE 80 – Expose port 80 for the web server.

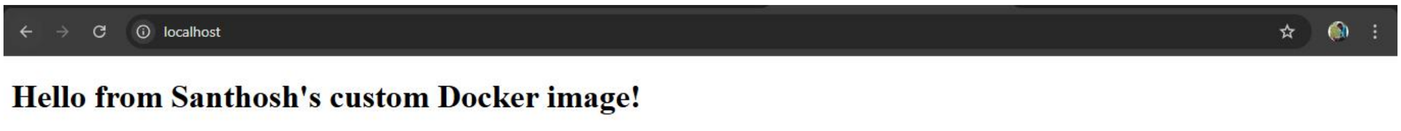
The command I used to build a custom image with my Dockerfile was: docker build -t my-custom-nginx . The `.` at the end means the current directory, telling Docker to use the Dockerfile and any necessary files from this location to build the image.



```
FROM nginx:latest
COPY index.html /usr/share/nginx/html/
EXPOSE 80
```

Running My Custom Image

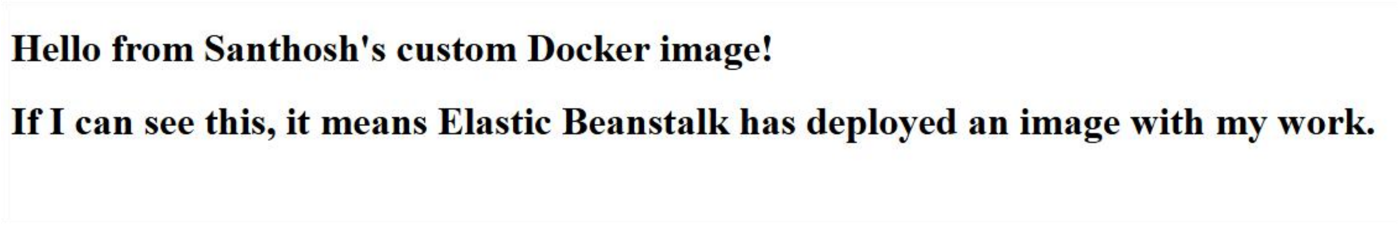
There was an error when I ran my custom image because port 80 was already in use by another container (the Nginx container). I resolved this by stopping the Nginx container using Docker Desktop or `docker stop` and then running my custom container.
In this example, the container image is `my-custom-nginx`, which includes the Nginx server and custom `index.html`. The container is the running instance of that image, executing and serving the webpage through port 8080 on your system.



Hello from Santhosh's custom Docker image!

Elastic Beanstalk

Elastic Beanstalk is a fully managed service from AWS that simplifies deploying, managing, and scaling web applications. It supports various languages and platforms, handling infrastructure provisioning, load balancing, and scaling automatically. Deploying my custom image with Elastic Beanstalk took me around 10-15 minutes. This time includes configuring the environment, uploading the image, and Elastic Beanstalk provisioning the necessary resources like EC2 instances and the load balancer.



Hello from Santhosh's custom Docker image!

If I can see this, it means Elastic Beanstalk has deployed an image with my work.

Today you've learned how to:

- **Install and run Docker:** You installed Docker on your local machine, which lets you create, manage, and run containers for your applications.
- **Build a custom Docker image:** You used a Dockerfile to define a Docker image that packages up your index.html file.
- **Deploy your image to AWS Elastic Beanstalk:** You uploaded your custom Docker image to AWS Elastic Beanstalk, which served your application in the cloud so you can access it from the internet!