

End-to-End Deployment of a Containerized Web Application

Using Docker and AWS Elastic Beanstalk

Objective

This project aims to gain hands-on experience with containerization and cloud deployment using Docker and AWS Elastic Beanstalk. I built a **Docker** container for a static web application and deployed it to the cloud using **AWS Elastic Beanstalk**. This project helped me to understand the complete end-to-end workflow from creating a Docker image, running it locally, to deploying it in a scalable cloud environment. It strengthened my skills in cloud-native application deployment and containerized infrastructure.

Tools & Technologies Used

- **Docker** – I used Docker to containerize my static web application for consistent execution across environments.
- **Dockerfile** – I wrote a Dockerfile that defines how the application is built and run inside the container.
- **HTML (index.html)** – A basic HTML file served as the frontend content for my application.
- **AWS Elastic Beanstalk** – I used this AWS service to deploy my Docker container to the cloud with minimal infrastructure management.
- **AWS CLI** – I used the AWS Command Line Interface to interact with Elastic Beanstalk and manage the deployment process.

Why Do Containers Matter?

When developers create applications, they often use containers to package up their work and share it in a way that's easy for other people to run. Using containers has absolutely changed the game for making software fast and efficient, and it's also the reason why deploying software has become much simpler.

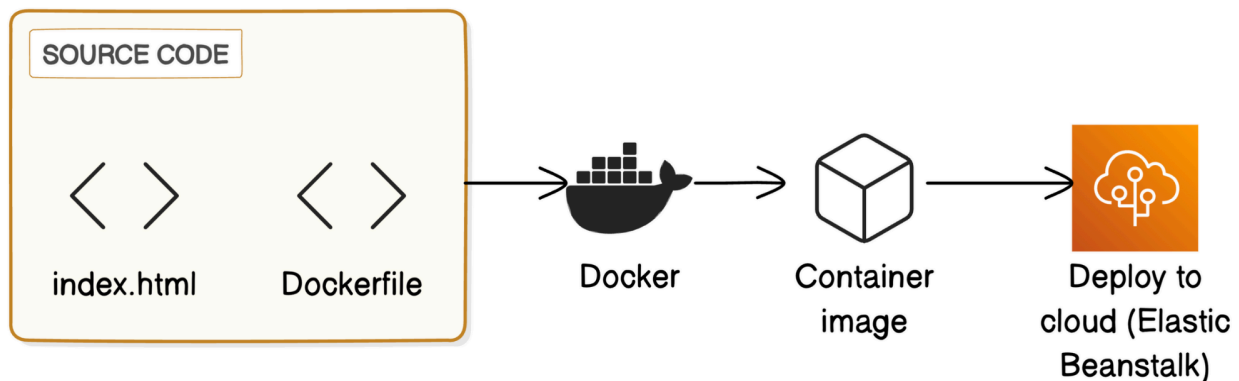


Project Workflow: Containerization and Deployment

The diagram below illustrates the complete process I followed to build and deploy a containerized web application:

Steps Involved:

1. Install and use Docker to manage containerized environments.
2. Build a custom container image from index.html using a Dockerfile.
3. Run the containerized application locally to ensure it works as expected.
4. Deploy the containerized application to the cloud using AWS Elastic Beanstalk, making it accessible via a live URL.



Project Workflow

Step-by-Step Process: From Code to Cloud

Step 1: Set Up an IAM User

What is an IAM user? Why am I setting one up?

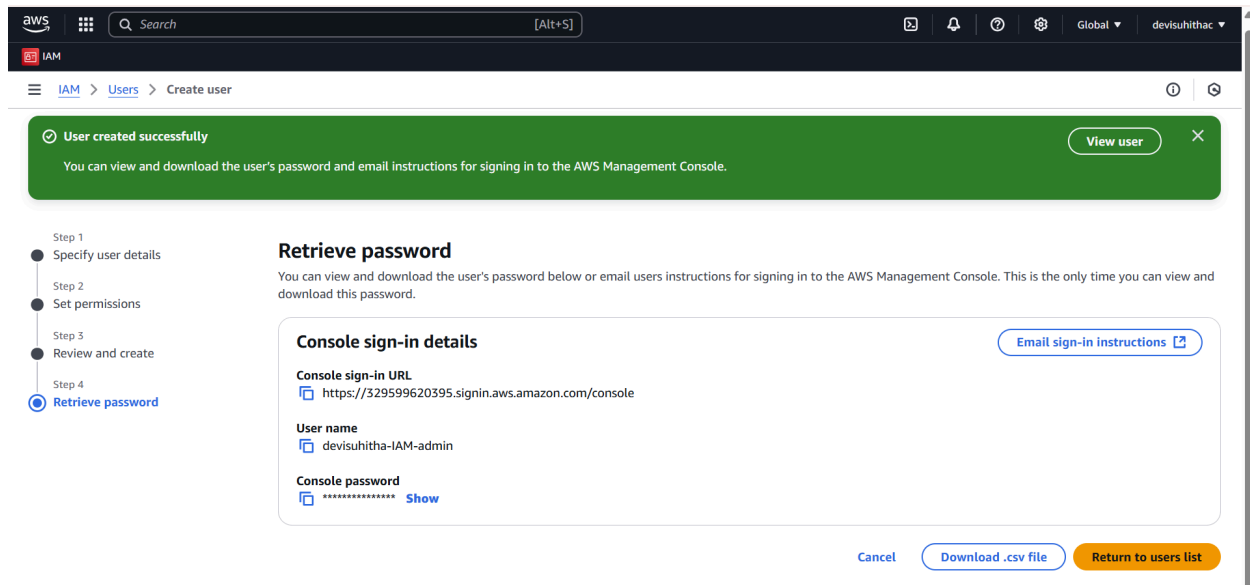
In AWS, a **user** is a person or a computer that can do things on the AWS cloud.

When I create an AWS account for the first time, the login I get is called the **root user** of the AWS account. The root user has complete access to all AWS services and resources in my account, including my billing and personal information.

AWS recommends that I do **not** use the root user for my everyday tasks to protect it from **security breaches**. Instead I should create an **IAM user**.

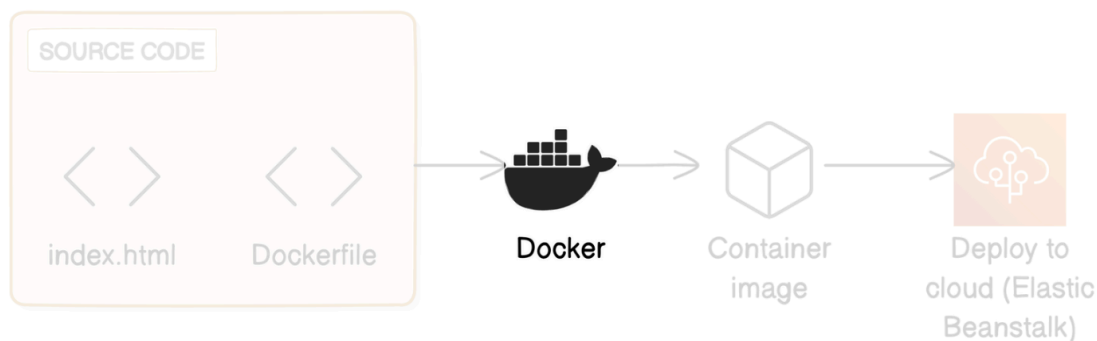
An IAM user is an additional user that gets access to my AWS Account's resources. When creating a User, I can specify in detail the level of access it has to my account's resources and services.

I created an IAM user with **Administrator Access**. This means my IAM user is allowed to perform all possible actions to all the resources in my account.



A new IAM user set up for my AWS Account

Step 2: Install Docker

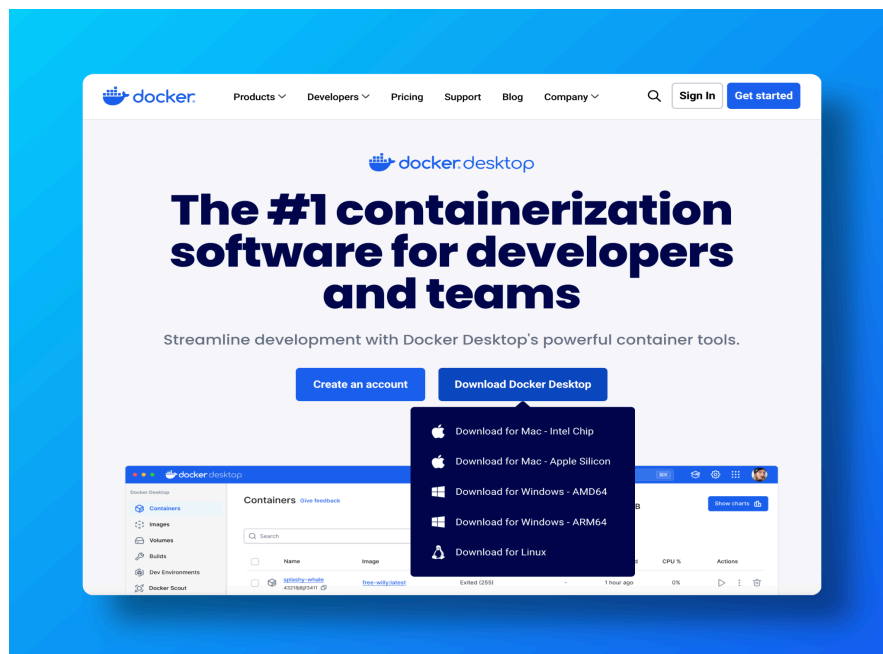


What I'm building in this step

Why are we installing Docker first? Docker is in the middle of my diagram

Docker is THE foundational tool. The first element of my diagram, the source code, represents an app that I'm about to build. Technically I could work on the source code first, but it'll actually be a lot easier to understand the source code if I install and use Docker first.

- Head over to the official [Docker website](https://www.docker.com).
- Select **Download Docker Desktop**, and then select the version of Docker that matches our local computer's operating system.



What is Docker Desktop ?

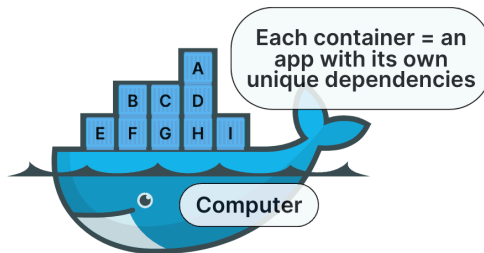
Docker Desktop is a program that makes it easy to work with Docker, a tool for creating and managing containers. Engineers use Docker Desktop to build, test, and deploy applications right from their computers and use Docker in a user-friendly way.

What does Docker do?

Docker helps us to create, manage, and deploy these containers efficiently. I can think of Docker as a tool that helps us to create the cargo and load them onto ships.

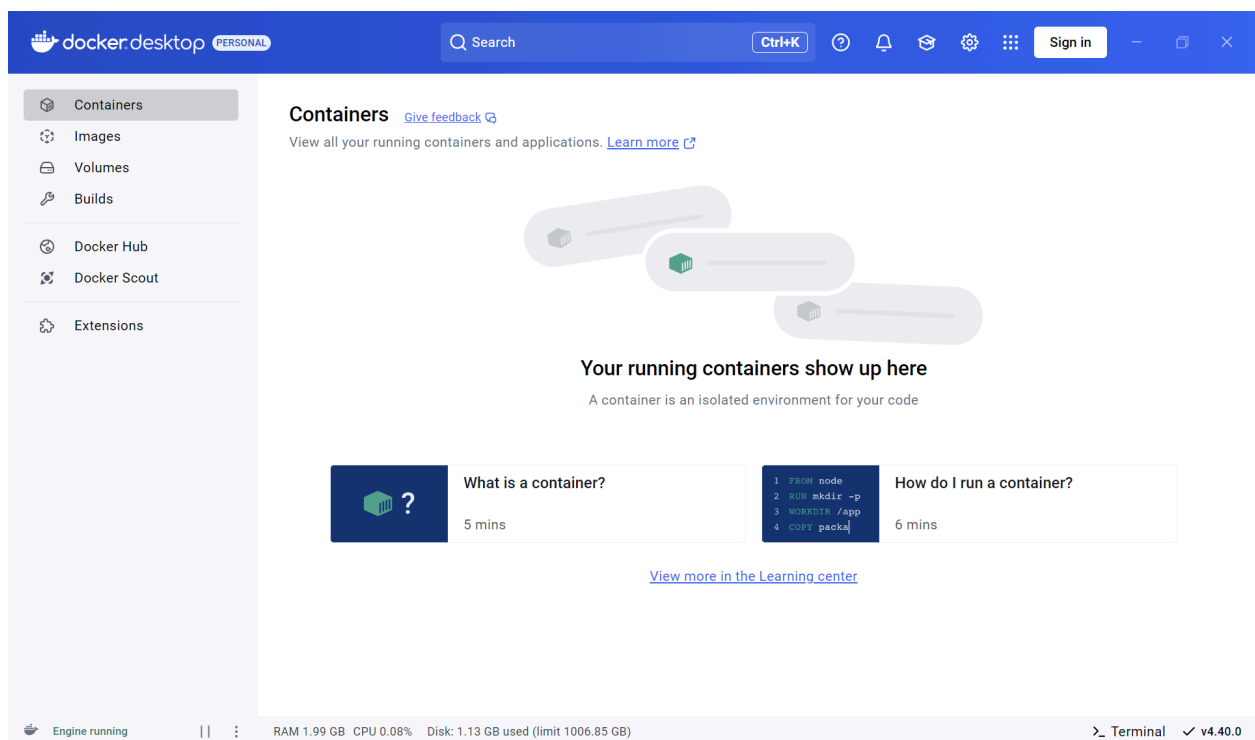


Just like how the same ship can carry lots of different shipping containers...



You can use the same computer to run lots of different containerized apps!

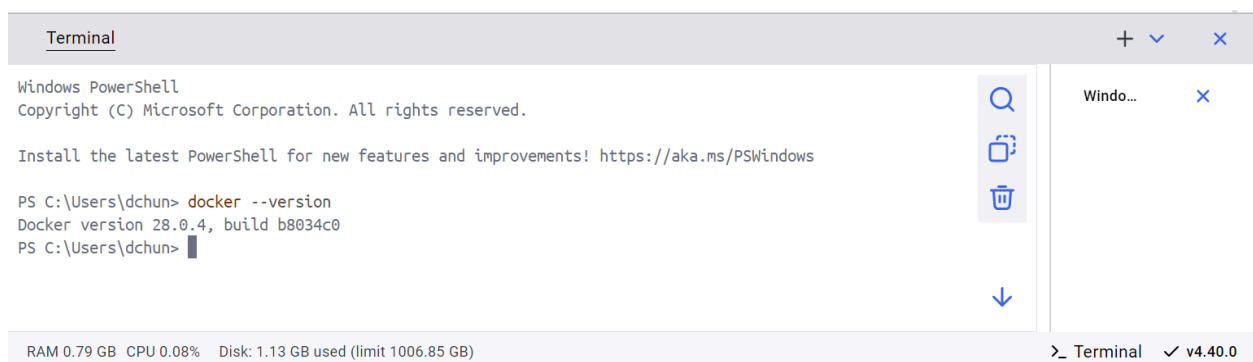
Containers are mostly used during the development phase of an app. For example, when a team is developing a complex web app, they might use containers to make sure everyone in the team is working in exactly the same environment, even though they are on different machines. That's why containers are such a popular DevOps tool i.e. a tool for making software development and releases more efficient!



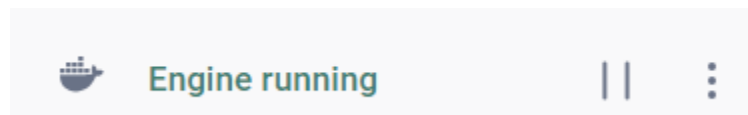
Why would I need to verify I've installed Docker? Isn't seeing Docker Desktop enough verification?

Think of **Docker** and **Docker Desktop** as two different things. Docker on its own is a tool for container management, while Docker Desktop is an app that makes using Docker more interactive and efficient. So downloading Docker Desktop doesn't necessarily mean you've installed Docker itself. It's best to check we've installed Docker using the terminal.

We can use command prompt or Open the **Docker Terminal**, which you can find at the bottom bar in your Docker Desktop. Run `docker --version` in your Docker Terminal.



Next, I verified that I've installed called a **Docker daemon**(a small docker icon)



What is the Docker daemon?

It is a background process that manages the Docker containers on my computer. It takes commands from the Docker client(i.e. Commands I typed into the terminal, or clicks I make through the Docker Desktop) and do the heavy lifting of building, running and distributing my containers.

Step 3: Build my Custom Image

What's a container image?

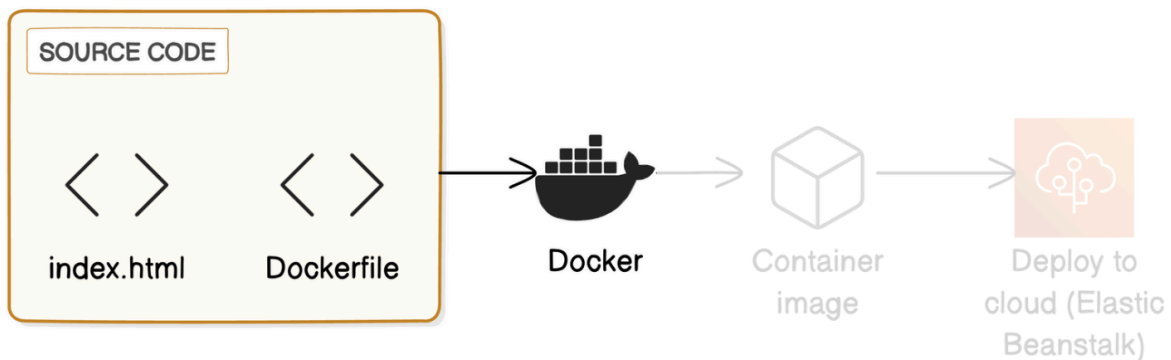
A container image is a **blueprint** or template for creating containers. It gives Docker instructions on what to include in a container, such as application code, libraries, dependencies, and necessary files needed to run the application. **Extra for Experts: What is a**

Docker Hub?

Docker Hub is a **container registry**, which means it's an online library where people can share and find Docker images. **AWS** also has its own container registry, called **Amazon ECR** (Elastic Container Registry).

In this step, I'm going to:

- Write a set of instructions that tells Docker how to build my own custom container image.
- Get Docker to read my instructions and build the image.



What I'm building in this step

In my local computer's Desktop, I created a folder and named it **Compute**.

- cd ~/Desktop - navigates my terminal to my Computer's Desktop
- mkdir Compute - Creates a new folder named Compute. mkdir stands for 'make directory' i.e. create a new folder.

```

PS C:\Users\dchun> cd ~/Desktop
PS C:\Users\dchun\Desktop> mkdir Compute

Directory: C:\Users\dchun\Desktop

Mode                LastWriteTime         Length Name
----                -
d-----          4/14/2025   6:26 PM             Compute
  
```

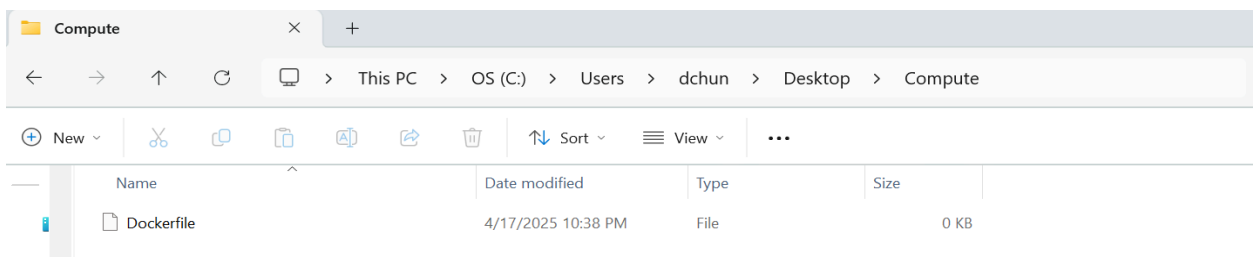
I created a new file in my **Compute** folder called **Dockerfile**.

- `cd ~/Desktop/Compute`
- `touch Dockerfile(Linux)` → creates an empty file called Dockerfile in the current directory i.e. the Compute folder.
- `PowerShell(Windows)` → `New-Item -Name "Dockerfile" -ItemType "File"`

```
PS C:\Users\dchun> cd ~/Desktop/Compute
PS C:\Users\dchun\Desktop\Compute> New-Item -Name "Dockerfile" -ItemType "File"

Directory: C:\Users\dchun\Desktop\Compute

Mode                LastWriteTime         Length Name
----                -
-a-----         4/17/2025  10:38 PM              0 Dockerfile
```



Confirm that I've created Dockerfile

What is a Dockerfile? → A Dockerfile is a document with all the instructions for building my Docker image. Docker would read a Dockerfile to understand how to set up my application's environment and which software packages it should install.

Now let's fill in Dockerfile

- Open the **Dockerfile**, where I used notepad++ on windows for the text editor and added the following lines and saved it.

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


What do these lines mean?

FROM nginx:latest → I'm starting with the official Nginx image as the base. Think of it like using a ready-made recipe. I don't need to build everything from scratch, just add what I need on top.

COPY index.html /usr/share/nginx/html/ → I'm adding my own index.html file to the Nginx server. This way, when someone visits the site, they'll see my custom content instead of the default Nginx page.

EXPOSE 80 → I'm telling Docker to make port 80 available so that people can access my website through their browser.

Creating the web page for my container

Now, I have my Dockerfile ready, what's the web page that my Nginx container should serve? I kept together a simple HTML file for this project.

I ran the commands in my terminal to create a new file named `index.html` in the same directory as my Dockerfile.

- `cd ~/Desktop/Compute`
- `New-Item -Name "index.html" -ItemType "File"`

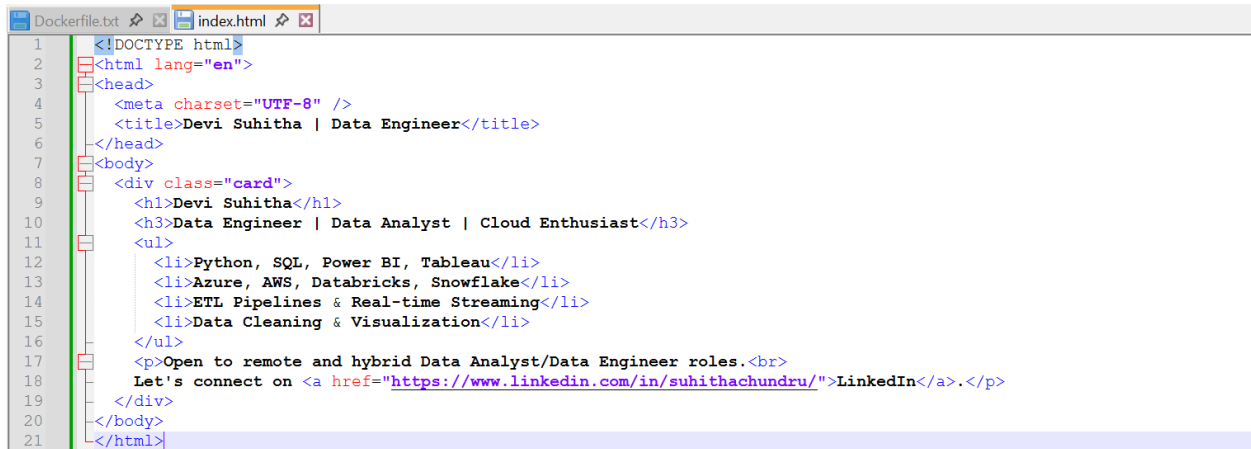
```
PS C:\Users\dchun> cd ~/Desktop/Compute
PS C:\Users\dchun\Desktop\Compute> New-Item -Name "index.html" -ItemType "File"

Directory: C:\Users\dchun\Desktop\Compute

Mode                LastWriteTime         Length Name
----                -
-a-----         4/18/2025  12:21 PM              0 index.html
```

What is an HTML file? This file determines what my webpage looks like. It's a document that lays out my webpage's layout, text, images and links. When Nginx opens my HTML file, it will read the code inside and display it on the user's browser.

I opened my index.html file and added my HTML content.



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Devi Suhitha | Data Engineer</title>
6 </head>
7 <body>
8   <div class="card">
9     <h1>Devi Suhitha</h1>
10    <h3>Data Engineer | Data Analyst | Cloud Enthusiast</h3>
11    <ul>
12      <li>Python, SQL, Power BI, Tableau</li>
13      <li>Azure, AWS, Databricks, Snowflake</li>
14      <li>ETL Pipelines & Real-time Streaming</li>
15      <li>Data Cleaning & Visualization</li>
16    </ul>
17    <p>Open to remote and hybrid Data Analyst/Data Engineer roles.<br>
18    Let's connect on <a href="https://www.linkedin.com/in/suhithachundru/">LinkedIn</a>.</p>
19  </div>
20 </body>
21 </html>

```

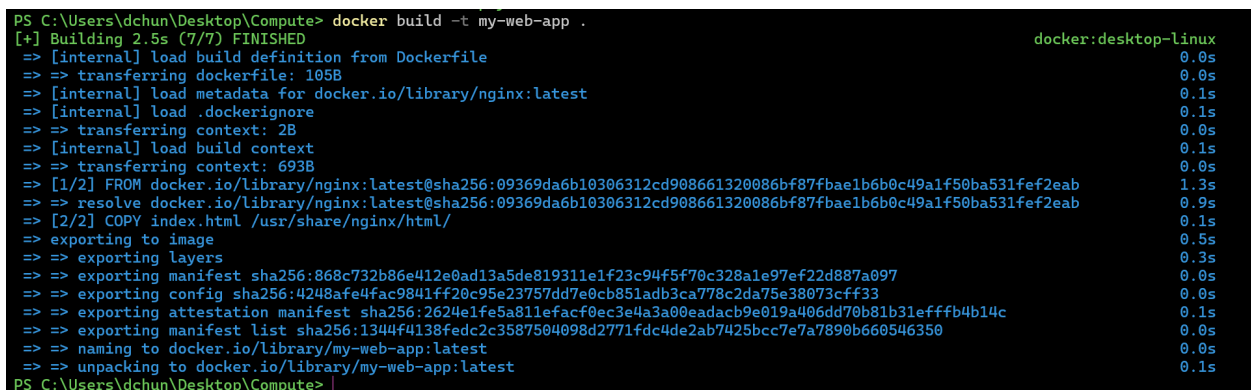
Building the Image

Now, I built the image! By heading back to my local terminal, I make sure I was still in the Compute directory, and run:

command : docker build -t my-web-app .

What does building an image mean? → Building an image means I created a Docker image using the instructions in a Dockerfile. Docker will then use the built image as the blueprint to create containers that run my application anywhere.

What do these commands do? → **-t my-web-app** names my image **my-web-app**, and the **(.)** tells Docker to find the Dockerfile in the current directory i.e. the Compute folder.



```

PS C:\Users\dchun\Desktop\Compute> docker build -t my-web-app .
[+] Building 2.5s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 105B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 693B
=> [1/2] FROM docker.io/library/nginx:latest@sha256:09369da6b10306312cd908661320086bf87fbae1b6b0c49a1f50ba531fef2eab
=> resolve docker.io/library/nginx:latest@sha256:09369da6b10306312cd908661320086bf87fbae1b6b0c49a1f50ba531fef2eab
=> [2/2] COPY index.html /usr/share/nginx/html/
=> exporting to image
=> exporting layers
=> exporting manifest sha256:868c732b86e412e0ad13a5de819311e1f23c94f5f70c328a1e97ef22d887a097
=> exporting config sha256:4248afe4fac9841fff20c95e23757dd7e0cb851adb3ca778c2da75e38073cfff33
=> exporting attestation manifest sha256:2624e1fe5a811efacf0ec3e4a3a0eadacb9e019a406dd70b81b31efffb4b14c
=> exporting manifest list sha256:1344f4138fedc2c35875040998d2771fdc4de2ab7425bcc7e7a7890b660546350
=> naming to docker.io/library/my-web-app:latest
=> unpacking to docker.io/library/my-web-app:latest
PS C:\Users\dchun\Desktop\Compute>

```

What does the response from the terminal mean?

The terminal response shows the steps Docker took to build my image from the Dockerfile:

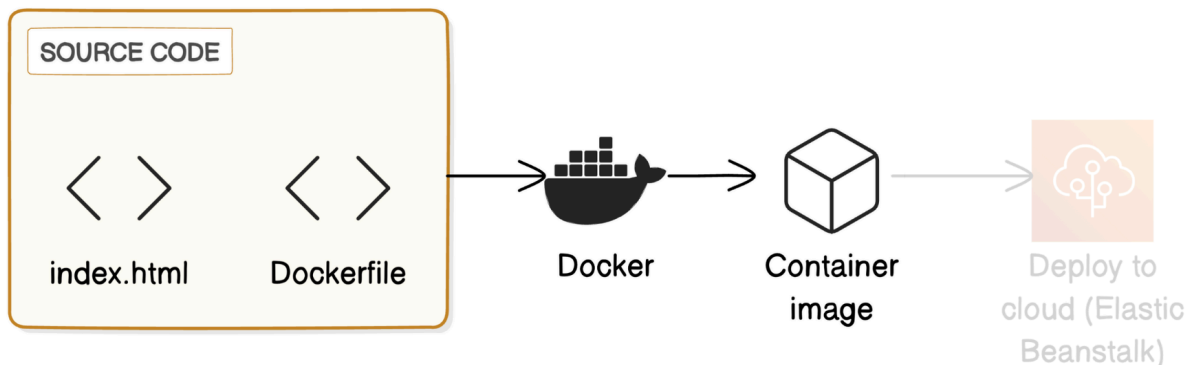
- **Loading build definition:** Docker is loading the instructions in my Dockerfile to understand how to build the image.
- **[1/2] FROM docker.io...:** Docker is processing the first line in my Dockerfile.
- **[2/2] COPY index.html...:** Docker is processing the second line in my Dockerfile, which tells it to copy my index.html file into the container's file system.
- **Exporting to image:** Docker is combining all the instructions applied to the image together, making it ready to run containers.

Step 4: Run the Container with my Custom Image

- I'm done with my image building. Next I will run the container with the image I built.

In this step, I'm going to

- Run my custom container image, so I can see my webpage in action.






What I'm building in this step

To run an image, I used below command:

```
docker run -d -p 80:80 my web-app
```

What does this command do ? → I'm asking Docker to run **my web-app** image as a container in the background(**-d**), making it accessible through port 80 on my local machine(**-p 80:80**).

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (Actions
<input type="checkbox"/>	cranky_borg	ba4dfd1918a9	my-web-app	80:80		  

There was an **error** when I ran my custom image because I tried to map my port 80 with the new container's port 80, but a running container was already using port 80. I resolved this by **stopping the running container so that I can start a new one.**

- Another way to find the container is by running **docker ps --filter "publish=80"** in our local terminal. Our terminal should show us the id under CONTAINER ID.
- We can then stop the container by running **docker stop {container_id}** Replace **{container_id}** with the actual ID of the other container.

At last, I opened my web browser and navigated to **http://localhost** again. I've done everything correctly, So I should see the content of my `index.html` file.



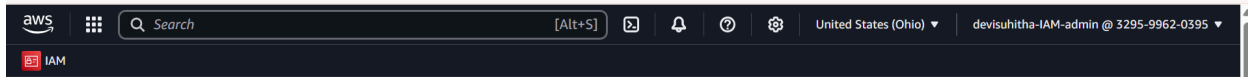
This is my web page !

Difference between containers and container images?

The container image is the blueprint for creating a new container running an Nginx server that serves my custom `index.html` file. **The container** is the actual software that's created from this image and running the web server displaying my `index.html`.

Real time Example: If Docker containers are apartment units within a building, Docker images serve as the floor plan. It specifies the components required to build the room and the furniture needed for the tenant to live comfortably.

Step 5: Log in to the AWS Management Console with IAM Admin User

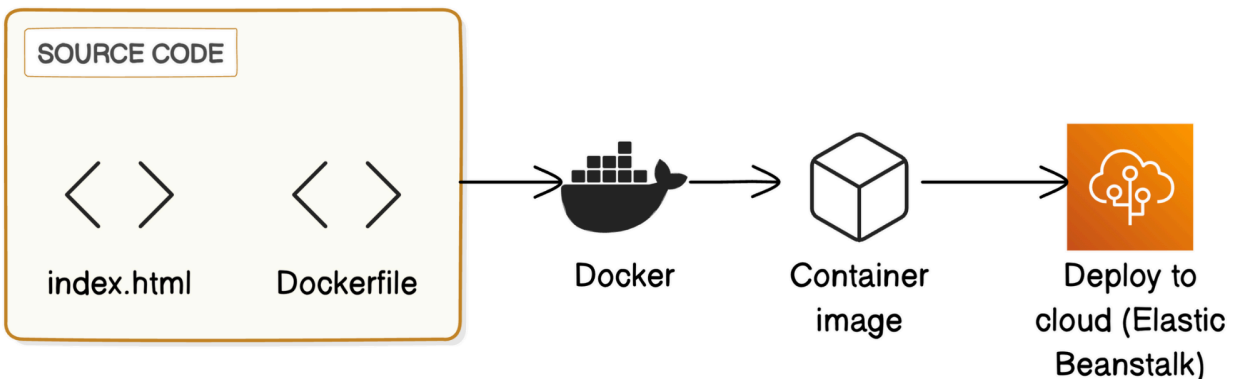


Step 6: Deploy my custom image to Elastic Beanstalk

I'll deploy my custom Docker image to AWS Elastic Beanstalk, a service that makes it easy for developers to deploy applications in the cloud.

In this step, I'm going to:

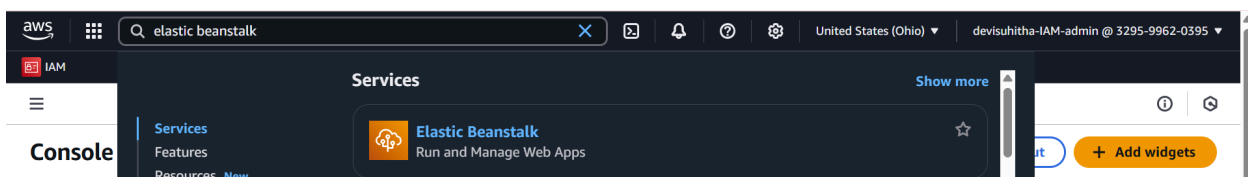
- Set up an application in Elastic Beanstalk.
- Deploy my application and make it public on the internet!



What I'm building in this step

Creating an Elastic Beanstalk Application

- Head to the [AWS Management Console](#) and log in as an IAM user.
- Search for Elastic Beanstalk and click on the service.



What is Elastic BeanStalk ?

Amazon Elastic Beanstalk is an easy-to-use service for **deploying and scaling web applications** and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, **Nginx**, Passenger, and IIS.

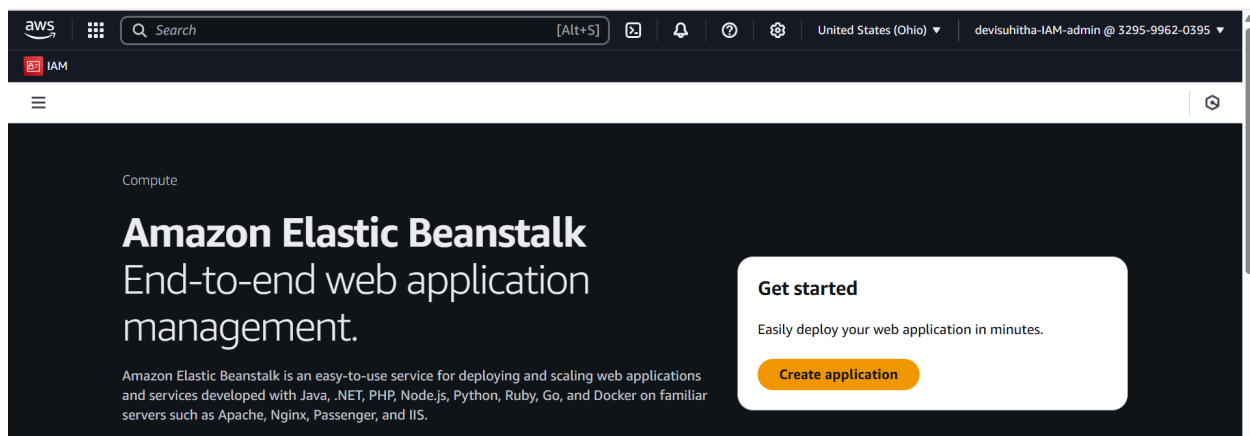
In more technical terms, Elastic Beanstalk handles things like:

- Capacity provisioning
- Load balancing
- Auto-scaling
- Application health monitoring

How is Elastic Beanstalk related to containers and Docker?

Elastic Beanstalk can run applications that are packaged as Docker containers. This means we can develop our application on our local machine and create a Docker container image to package everything needed to run it. Once our container image is ready, we can easily deploy it to Elastic Beanstalk.

Click on **Create Application** in the Elastic Beanstalk homepage.



In the **Configure Environment** page:

- I left the **Environment tier** as default.

Step 1 **Configure environment**

Step 2 Configure service access

Step 3 - optional Set up networking, database, and tags

Step 4 - optional Configure instance traffic and scaling

Step 5 - optional Configure updates, monitoring, and logging

Step 6 Review

Configure environment [Info](#)

Environment tier [Info](#)

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

☒ **Web server environment**
Run a website, web application, or web API that serves HTTP requests. [Learn more](#)

☐ **Worker environment**
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. [Learn more](#)

Application information [Info](#)

Application name

Suhitha Chundru Web App

Maximum length of 100 characters.

What is an environment in Elastic Beanstalk?

An environment in Elastic Beanstalk is like the settings for the virtual machines that will run our application. When we configure our environment, we're deciding the instance type (the type of EC2 instance that will be running the container), scaling options (how many instances to run), network settings, and more.

Virtual Machines? EC2? How does that relate to Elastic Beanstalk?

Elastic Beanstalk uses EC2 instances (i.e. virtual servers) under the hood to run our applications. Even though I'm deploying a Docker container, it ultimately runs on an EC2 instance managed by Elastic Beanstalk.

- I entered the Suhitha Chundru Web App as the application name.
- I left the **Environment information** section with the default values.
- Under the **Platform** section, select **Docker** as my Platform.

Platform [Info](#)

Platform type

☒ **Managed platform**
Platforms published and maintained by Amazon Elastic Beanstalk. [Learn more](#)

☐ **Custom platform**
Platforms created and owned by you. This option is unavailable if you have no platforms.

Platform

Docker

Platform branch

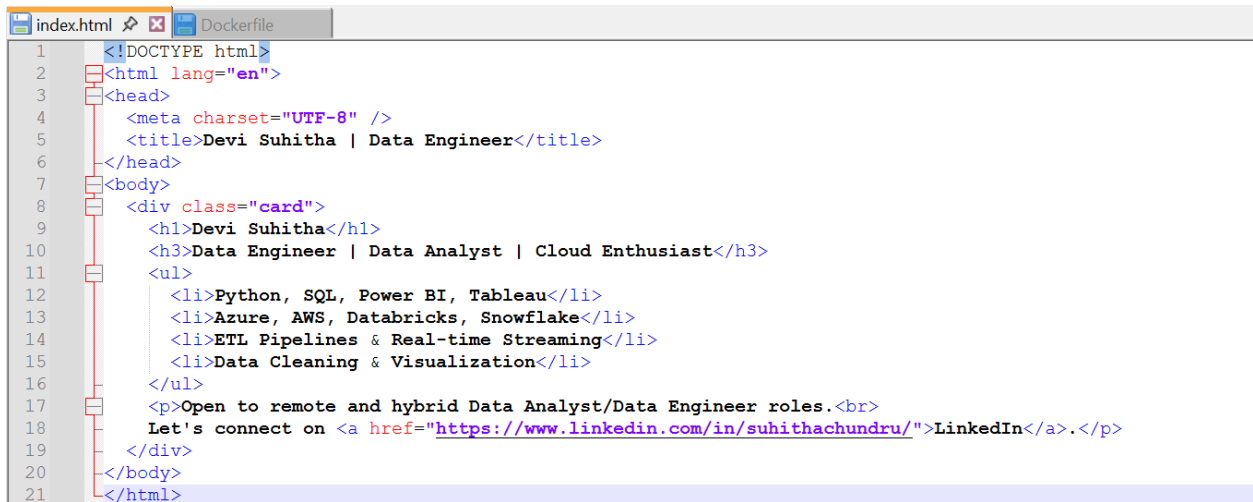
Docker running on 64bit Amazon Linux 2023

Platform version

4.5.0 (Recommended)

In the **Application code** section, select Upload your code.

Before I upload my code, head back to the Compute folder in my desktop and open up index.html with my text editor.



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Devi Suhitha | Data Engineer</title>
6 </head>
7 <body>
8   <div class="card">
9     <h1>Devi Suhitha</h1>
10    <h3>Data Engineer | Data Analyst | Cloud Enthusiast</h3>
11    <ul>
12      <li>Python, SQL, Power BI, Tableau</li>
13      <li>Azure, AWS, Databricks, Snowflake</li>
14      <li>ETL Pipelines & Real-time Streaming</li>
15      <li>Data Cleaning & Visualization</li>
16    </ul>
17    <p>Open to remote and hybrid Data Analyst/Data Engineer roles.<br>
18    Let's connect on <a href="https://www.linkedin.com/in/suhithachundru/">LinkedIn</a>.</p>
19  </div>
20 </body>
21 </html>

```

Just **updated** my index.html by adding a new line.

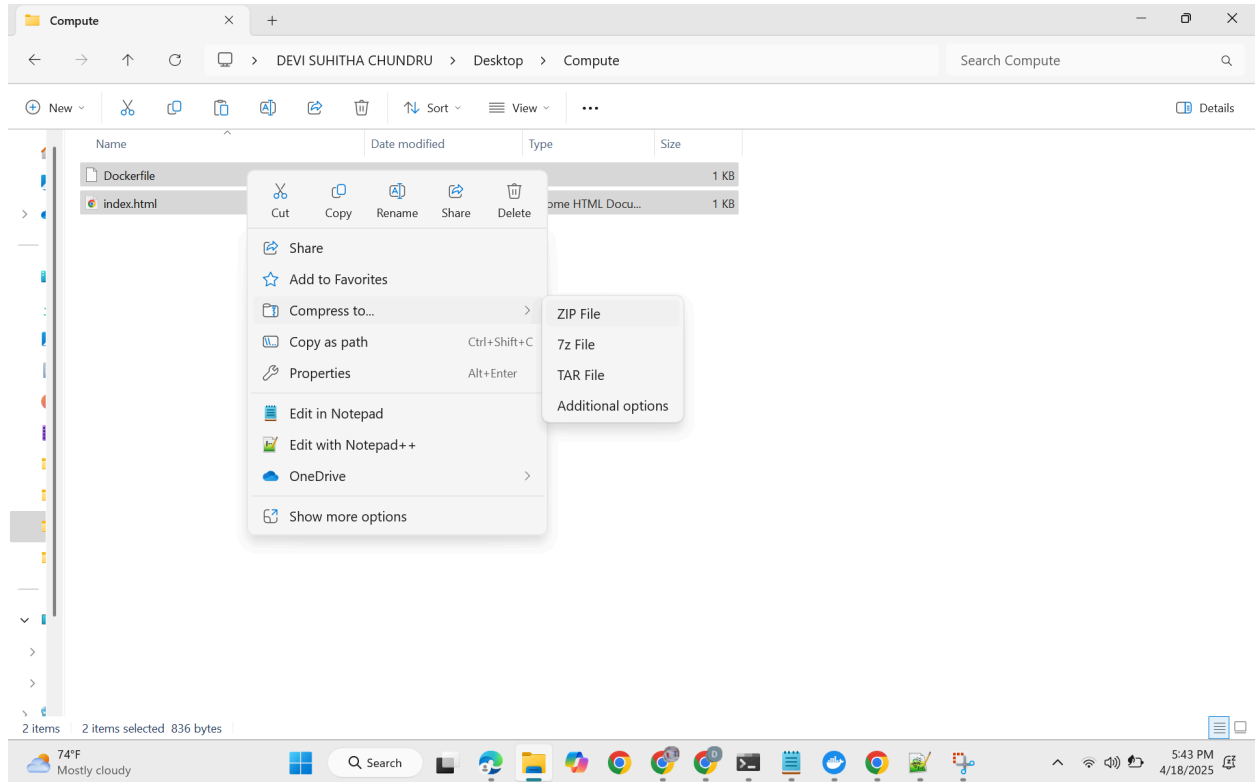


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Devi Suhitha | Data Engineer</title>
6 </head>
7 <body>
8   <div class="card">
9     <h1>Devi Suhitha</h1>
10    <h3>Data Engineer | Data Analyst | Cloud Enthusiast</h3>
11    <ul>
12      <li>Python, SQL, Power BI, Tableau</li>
13      <li>Azure, AWS, Databricks, Snowflake</li>
14      <li>ETL Pipelines & Real-time Streaming</li>
15      <li>Data Cleaning & Visualization</li>
16    </ul>
17    <p>Open to remote and hybrid Data Analyst/Data Engineer roles.<br>
18    Let's connect on <a href="https://www.linkedin.com/in/suhithachundru/">LinkedIn</a>.</p>
19    <h1>
20    If I can see this, it means Elastic Beanstalk has deployed an image with my work.
21  </h1>
22 </div>
23 </body>
24 </html>

```

Elastic Beanstalk needs my source code to be in a ZIP file. Created a ZIP file containing **Dockerfile** and **index.html**. My files are at the root level of the ZIP file, not inside a subfolder. To create my ZIP file, in my **Compute** folder, I multi-selected both **Dockerfile** and **index.html**, then I compressed them together.



- Again I go back to my Elastic Beanstalk setup page, enter **Version One** as the Version label.
- Select **Local file**.
- Under **Upload application**, select **Choose file**.
- Upload the ZIP file I created earlier. This uploads my application code to Elastic Beanstalk.

Application code [Info](#)

- ☐ Sample application
- ☐ Existing version
Application versions that you have uploaded.
- ☒ Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

Version label

Unique name for this version of your application code.

Source code origin. Maximum size 500 MB

- ☒ Local file

Upload application

[Choose file](#)

✓ File name: **Dockerfile.zip**

File must be less than 500MB max file size

- ☐ Public S3 URL

In the **Presets** section, I chosen **Single instance (Free tier eligible)** because by selecting a preset, it means selecting a predefined way to set up Elastic Beanstalk recommended by AWS and **The Single instance** preset creates a single EC2 instance, which is great for testing and is free tier eligible → Select Next.

Presets [Info](#)

Start from a preset that matches your use case or choose custom configuration to unset recommended values and use the service's default values.

Configuration presets

- ☒ Single instance (free tier eligible)
- ☐ Single instance (using spot instance)
- ☐ High availability
- ☐ High availability (using spot and on-demand instances)
- ☐ Custom configuration

In the **Configure service** access page:

I selected **"Create and use a new service role"** to let Elastic Beanstalk create a new IAM role for me. I used the default service role name, which was something like **aws-elasticbeanstalk-service-role**.

What is a service role?

A service role is an AWS IAM role that gives Elastic Beanstalk permission to call other AWS services on our behalf..

Why does Elastic Beanstalk need a new service role?

One of the benefits of Elastic Beanstalk is that it can handle our application's cloud infrastructure for us, and it can only do that if it has the permission to use our AWS resources automatically e.g. creating a new EC2 instance. It also helps with automating tasks like scaling and health monitoring!

The screenshot shows the AWS IAM console interface. On the left, a navigation pane lists steps: Step 1: Configure environment, Step 2: Configure service access (selected), Step 3: optional: Set up networking, database, and tags, Step 4: optional: Configure instance traffic and scaling, Step 5: optional: Configure updates, monitoring, and logging, and Step 6: Review. The main content area is titled 'Configure service access' with an 'Info' link. It contains a 'Service access' section explaining that IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage the environment. Below this, the 'Service role' section has two options: 'Create and use new service role' (selected) and 'Use an existing service role'. The 'Service role name' field is populated with 'aws-elasticbeanstalk-service-role' and has a 'View permission details' link. The 'EC2 key pair' section prompts the user to select an EC2 key pair to securely log in to their EC2 instances, with a 'Choose a key pair' dropdown and a 'View permission details' link. The 'EC2 instance profile' section prompts the user to choose an IAM instance profile with managed policies that allow their EC2 instances to perform required operations, with a dropdown showing 'ec2instanceRole' and a 'View permission details' link. At the bottom, there are navigation buttons: 'Cancel', 'Skip to review', 'Previous', and 'Next'.

What's an instance profile?

An instance profile is another IAM role (similar to service roles), but they're designed to give EC2 instances the permission to access other AWS services that our application might need to work.

In the **Set up networking, database, and tags** page:

- Under **Instance settings**, check **Activated** for the Public IP address option.

Instance settings

Choose a subnet in each AZ for the instances that run your application. To avoid exposing your instances to the Internet, run your instances in private subnets and load balancer in public subnets. To run your load balancer and instances in the same public subnets, assign public IP addresses to the instances. [Learn more](#)

Public IP address

Assign a public IP address to the Amazon EC2 instances in your environment.

☒ Activated

What does activating a Public IP address do?

This makes my EC2 instances and my application accessible from the internet - exciting!

For the **Configure instance traffic and scaling** page:

- Under the **Instances** section, for my **Root volume type** I selected **General Purpose 3 (SSD)**.
- For the **Size** select 10 GB.

▼ Instances [Info](#)

Configure the Amazon EC2 instances that run your application.

Root volume (boot device)

Root volume type

General Purpose 3(SSD) ▼

Size

The number of gigabytes of the root volume attached to each instance.

10

GB


What does root volume type mean?

The root volume type is my EC2 instance's storage space, just like how my local computer would have its own storage space for the operating system and files I keep locally.

General Purpose 3 (SSD) is Free Tier eligible and is fit for most applications.

Under **Instance metadata service (IMDS)**, I made sure that **IMDSv1 is deactivated** by keeping the "**Deactivated**" checkbox checked.

Instance metadata service (IMDS)

Your environment's platform supports both IMDSv1 and IMDSv2. To enforce IMDSv2, deactivate IMDSv1. [Learn more](#) 

IMDSv1

With the current setting, the environment enables only IMDSv2.

☒ Deactivated

What is Instance metadata service (IMDS)?

IMDS is a service that gives applications temporary credentials to your EC2 instances. For example, you can use IMDS to give your application the permission to access S3 to get data.

Since my app won't need access to other AWS services, I don't need IMDS.

I also don't need to configure anything else, so I skipped the **Security Groups** and **Capacity** sections.

What do these settings do?

Security groups control traffic, while Capacity controls the compute capacity of our environment and auto scaling settings to optimize the number of instances used.

In the **Configure updates, monitoring, and logging** page:

- Under the **Monitoring** section, I selected **Basic** for my system.
- I made sure to choose **Basic** to keep my project free.

What are monitoring and health reporting?

Monitoring and health reporting tools give us performance and status updates on our Elastic Beanstalk application, so we can spot any performance issues before they affect our users.

- I scrolled to the **Managed platform updates** section and made sure to **uncheck** the "**Activated**" checkbox for the **Managed updates** option.

▼ Managed platform updates [Info](#)

Activate managed platform updates to apply platform updates automatically during a weekly maintenance window that you choose. Your application stays available during the update process.

Managed updates

☐ Activated

Weekly update window

Thursday ▼ at 16 ▼ : 24 ▼ UTC

Update level

Minor and patch ▼

Instance replacement

If enabled, an instance replacement will be scheduled if no other updates are available.

☐ Activated

What do managed updates do?

Managed updates automatically update the software running our application in Elastic Beanstalk e.g. Docker. This makes sure our application stays secure and stable without needing manual updates.

Now in the **Rolling updates and deployments** section, I accepted the default **All at once** deployment policy.

What are application deployments?

Application deployments move a new version of my software into production on Elastic Beanstalk. This includes uploading my code, setting configuration options, and finally, replacing the old version with the new one on the live environment that users see.

There are two main types of deployments, also called **deployment policies**:

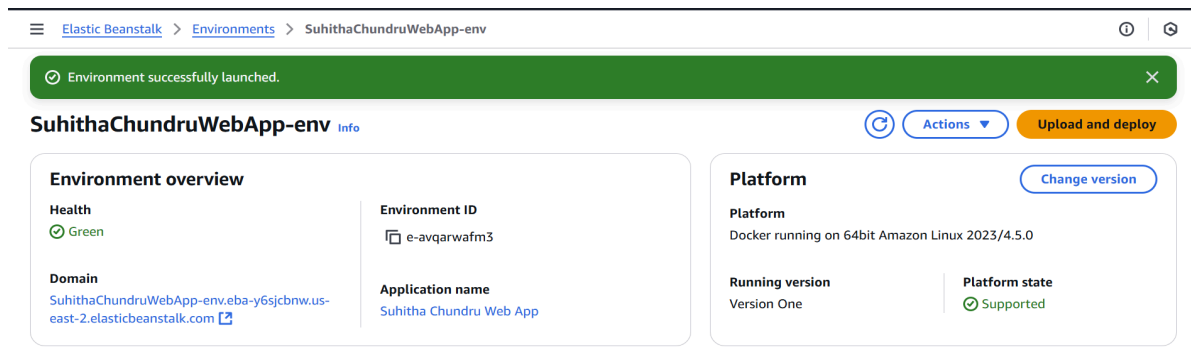
- **All at once** updates all instances at the same time, resulting in a brief period of downtime.
- **Rolling** updates instances gradually, which minimises downtime.

Start → Create Elastic Beanstalk App → Name: NextWork App → Upload: ZIP file (Dockerfile + index.html) → Choose Platform: Docker → Set Environment Type: Single Instance → Set EC2 Instance Profile: ecsInstanceRole → Disable IMDSv1 (check Deactivated) → Skip Security Groups & Capacity Sections → Set Monitoring: Basic (to stay free) → Disable Managed Platform Updates → Enable Public IP Address

→ Set Root Volume Type: gp3 → Set Deployment Policy: AllAtOnce → Review & Click Submit → ⌚ Wait a few minutes...



🎉 Environment Launch Successful!



I clicked on the **Domain** link on the environment dashboard to see my environment in action.

Environment overview

Health

🟢 Green

Domain

SuhithaChundruWebApp-env.eba-y6sjcbnw.us-east-2.elasticbeanstalk.com

Environment ID

🏠 e-avqarwafm3

Application name

Suhitha Chundru Web App

Wow! The domain opened and displayed my updated HTML file, which means Elastic Beanstalk successfully deployed the contents of my ZIP file as a container image.



Devi Suhitha

Data Engineer | Data Analyst | Cloud Enthusiast

- Python, SQL, Power BI, Tableau
- Azure, AWS, Databricks, Snowflake
- ETL Pipelines & Real-time Streaming
- Data Cleaning & Visualization

Open to remote and hybrid Data Analyst/Data Engineer roles.
Let's connect on [LinkedIn](#).

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

