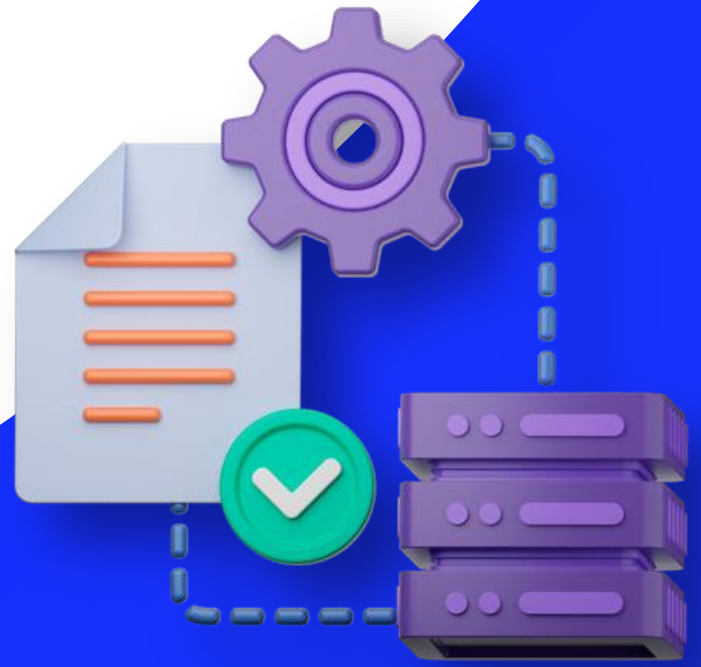
Institute of Data

Software Engineering

Module 10 – Part 2 Deployment and Maintenance



Agenda

- Section 1 — Introduction to AWS EC2
- Section 2 — Environment Variables
- Section 3 — Deploying application in EC2
- Section 4 — Beanstalk

Section 1:

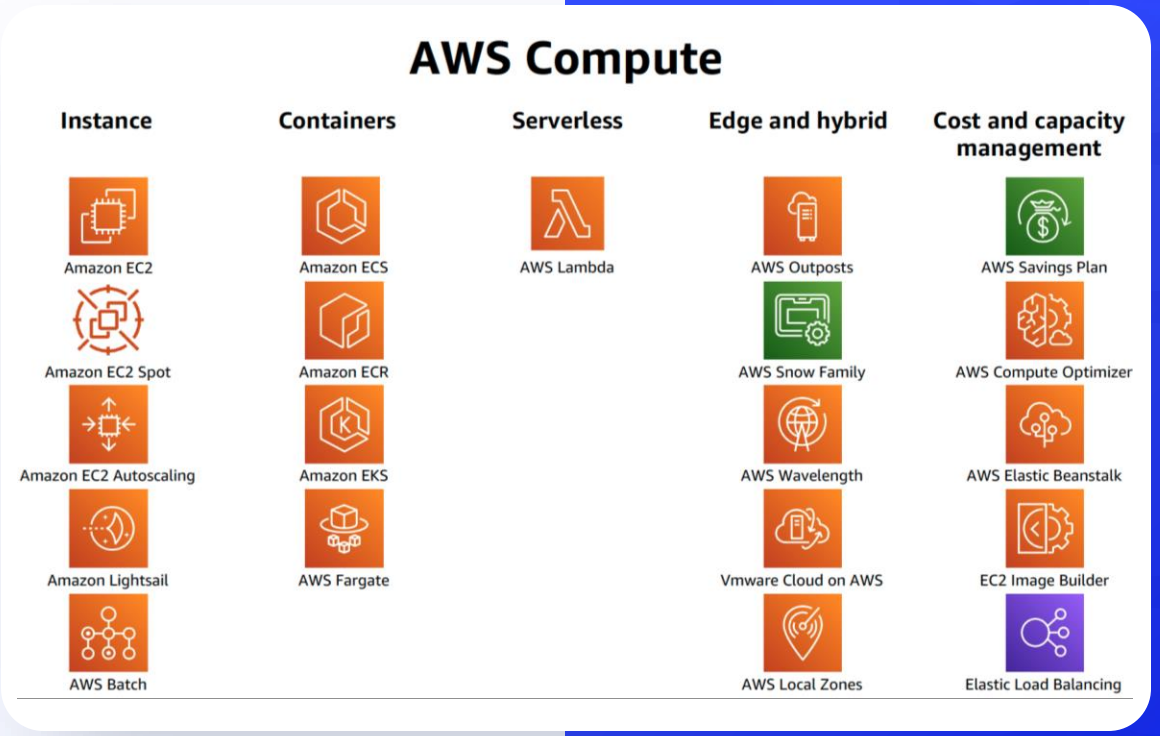
Introduction to AWS EC2

AWS (Amazon Web Services) is probably the main player in the field of cloud-based computing services. They offer a huge range of products and services, covering things such as domains, emails, SMS, servers, databases, load balancing, static storage and so much more. They specialize in **security** and **performance**, as well as **'elastic'** products that stretch/scale to fit your budget and needs.

Amazon EC2 (elastic cloud computing) offers a range of instance types that are tailored to certain use cases. Instance types are different combinations of **CPU**, **memory**, **storage**, and **networking** capabilities that allow you to choose the best resource mix for your applications. Each instance type has one or more instance sizes, allowing you to **scale** your resources to your target workload needs.

Compute using Amazon EC2

Whether you're building corporate, cloud-native, or mobile apps, or running enormous clusters to fuel analysis workloads, establishing and running your business starts with **compute**. AWS provides a comprehensive set of compute services that enable you to build, launch, run, and expand your applications on the world's most powerful, secure, and innovative cloud.



AWS computing services provide:

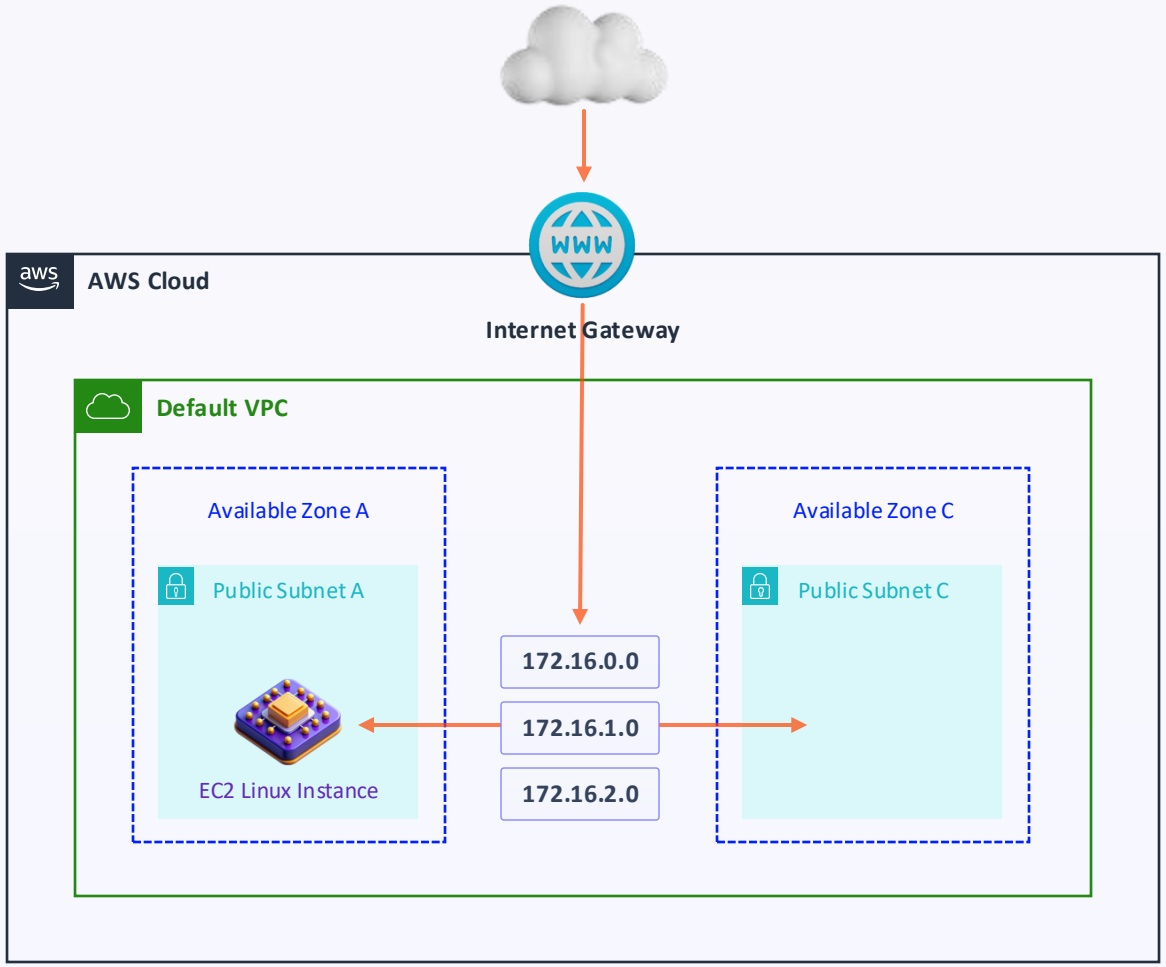
- Right compute for your workloads
- Tools to accelerate from idea to market
- Built-in security
- Flexibility to optimize costs
- Compute resource where you need it

EC2 Overview

In the Amazon Web Services (AWS) Cloud, Amazon EC2 delivers **scalable** computing capability. Using Amazon EC2 reduces the requirement for upfront hardware investment, allowing you to develop and deploy apps more quickly. EC2 allows you to create as many or as few virtual servers as you need, as well as establish security and networking and manage storage. You can scale up or down on EC2 to manage variations in demand or popularity spikes, decreasing the need to forecast traffic.

First create a new AWS account, then create your own web server by going through these 4 labs in the following order:

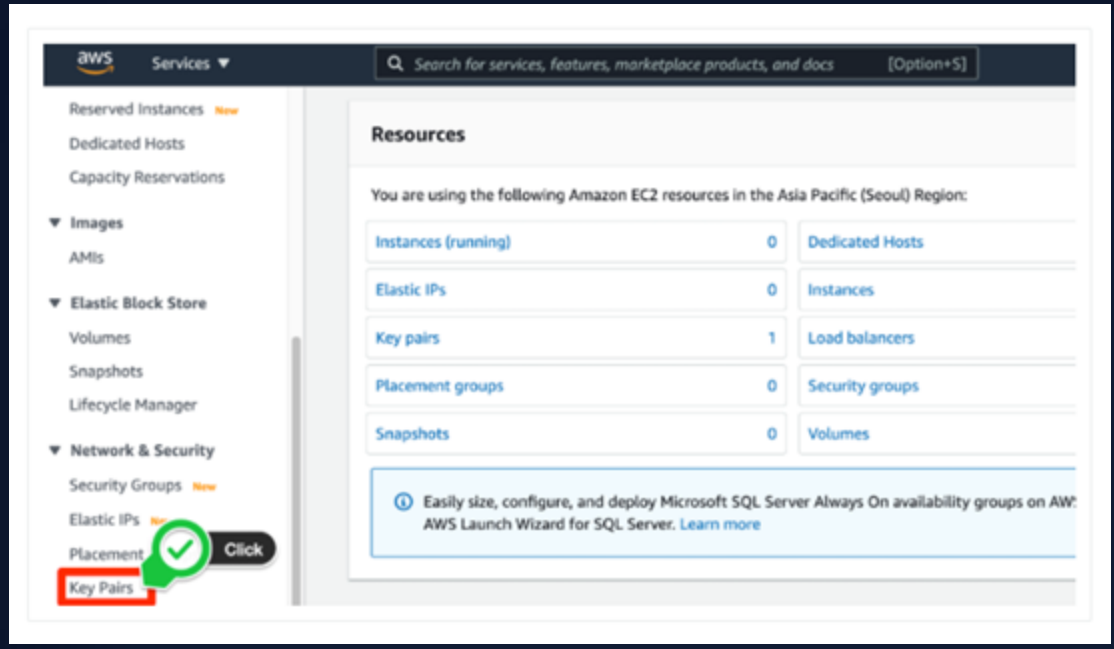
1. Create a new key pair
2. Launch a Web Server Instance
3. Connect to your linux instance
4. Connect to EC2 Instance using PuTTy (Optional)

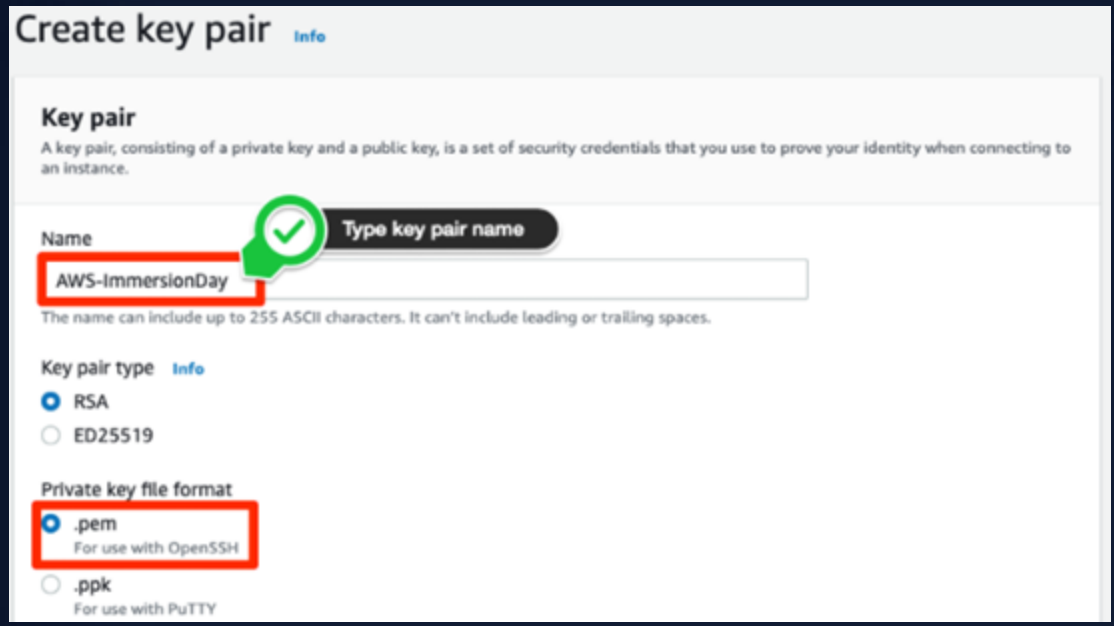


Create a key pair

In this lab, you will need to create an SSH keypair to use for connecting to your EC2 instance:

1. Sign into the AWS Management Console and open the Amazon EC2 console. In the upper-right corner of the AWS Management Console, confirm you are in the closest AWS region.
2. Click on **Key Pairs** in the Network & Security section near the bottom of the leftmost menu. This will display a page to manage your SSH key pairs.
3. To create a new SSH key pair, click the **Create key pair** button at the top of the browser window.
4. Type **[Your Name]-ImmersionDay** into the Key Pair Name: text box and click **Create key pair** button.
   *For Windows and PuTTY users, please select **ppk** for file format. The **pem** format should be used for Mac or SSH users.*
5. The page will download the file [**Your Name]-ImmersionDay.pem** (or ppk) to the local drive. Follow the browser instructions to save the file to the default download location. *Remember the full path to the key pair file you just downloaded.*





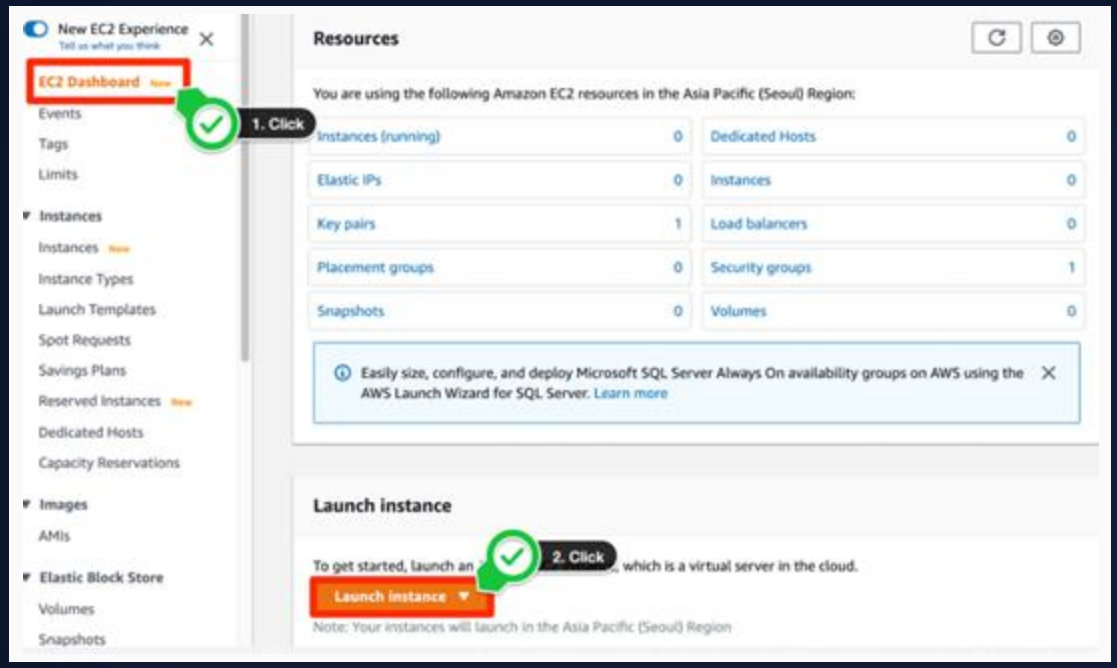
Launch a Web server

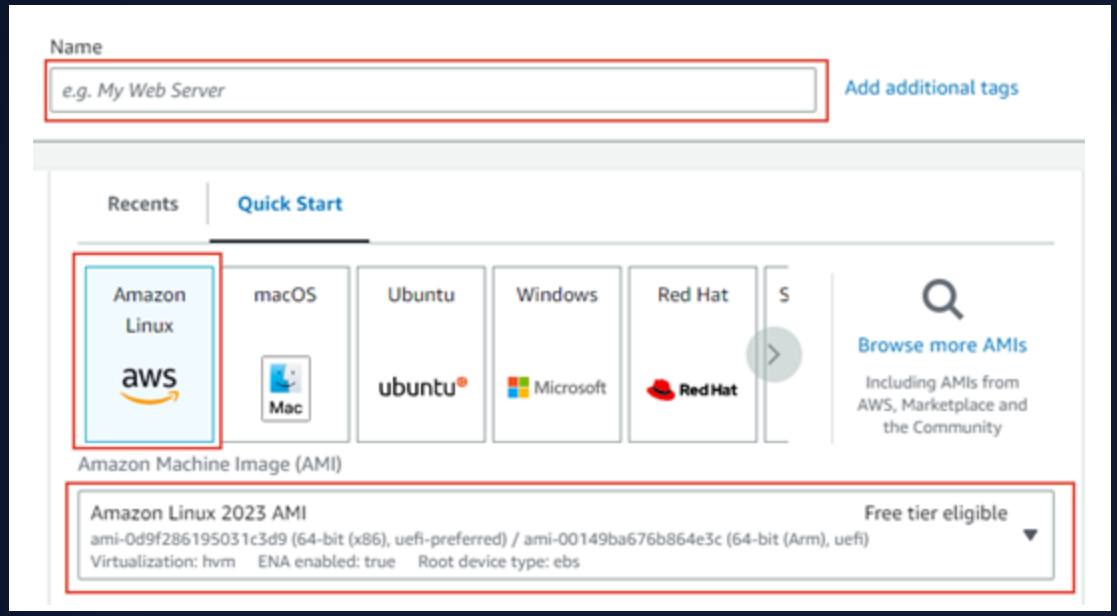
Next we will launch an Amazon Linux 2 EC2 instance:

1. Click on **EC2 Dashboard** near the top of the leftmost menu. And click on **Launch instances**.

2. Enter a name for your instance and continue with the default Amazon Linux AMI (Amazon Machine Image) from the Quick Start tab:

   (yours may look slightly different as they update them frequently)

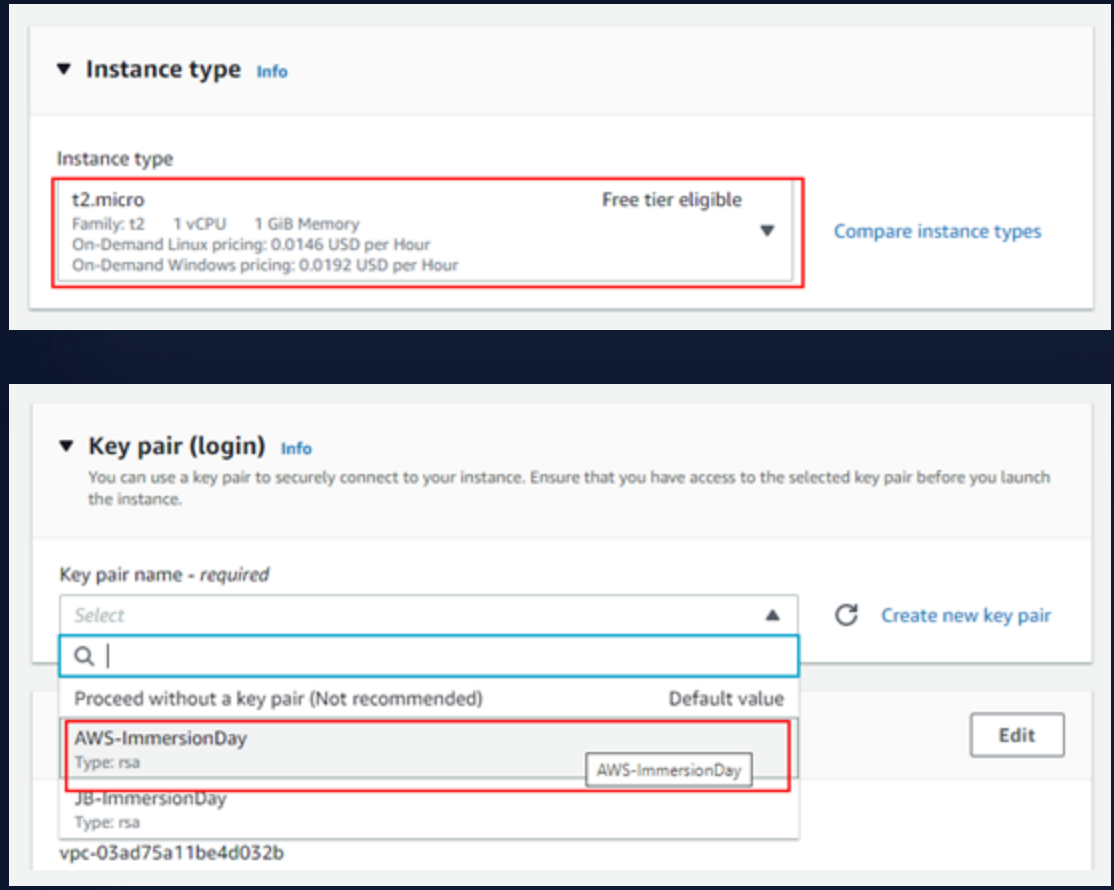




Launch a Web server

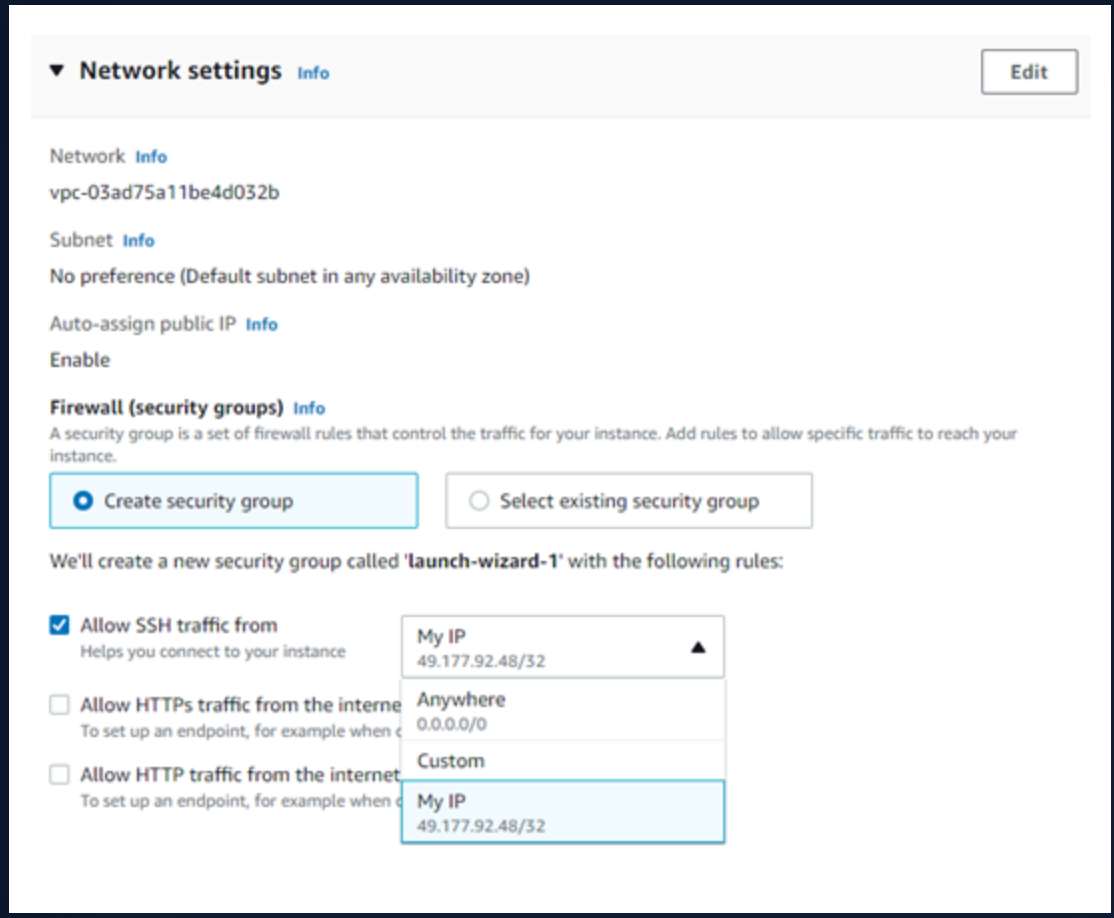
3. Next choose an Instance Type, select the **t2.micro** instance size which is eligible for the Free Tier.

4. Next choose the key pair that you created back on slide 7. You will also need the local copy of this key to connect to your instance via PuTTY / SSH



Network & Storage

5. To ensure the right access is granted to your instance, make the following changes in Network settings:
   - **Allow SSH traffic** from My IP (*if your IP changes and you need to connect via SSH/PuTTY later on, remember to update this*)
   - **Tick Allow HTTPs traffic** from the internet
   - **Tick Allow HTTP traffic** from the internet
6. Keep the default storage option

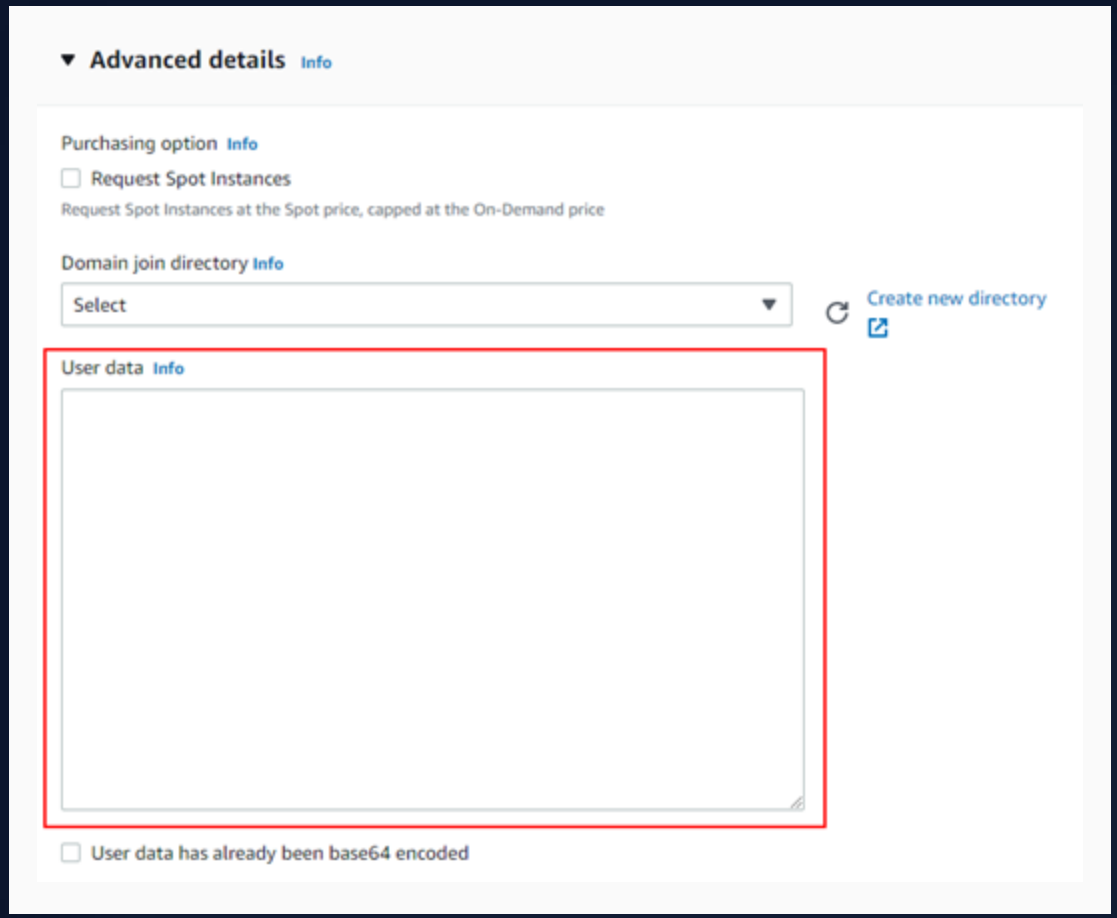


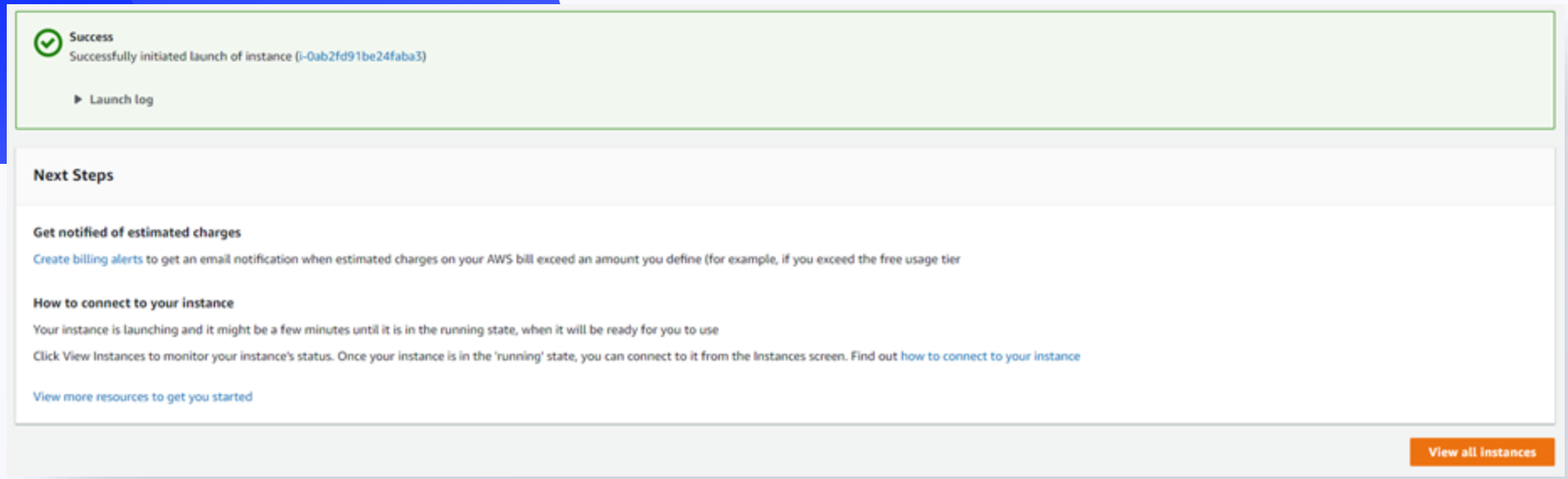
Advanced Startup Script

7. In the Advanced tab, leave most settings unchanged, but scroll down to the **User data** section near the bottom. In here, paste the following startup script (taken from here)

```
#!/bin/sh

# Install a LAMP stack
amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
yum -y install httpd php-mbstring

# Start the web server
chkconfig httpd on
systemctl start httpd

# Install the web pages for our lab
if [ ! -f /var/www/html/immersion-day-app-php7.tar.gz ]; then
   cd /var/www/html
   wget https://aws-joozero.s3.ap-northeast-2.amazonaws.com/immersion-day-app-php7.tar.gz
   tar xvfz immersion-day-app-php7.tar.gz
fi

# Install the AWS SDK for PHP
if [ ! -f /var/www/html/aws.zip ]; then
   cd /var/www/html
   mkdir vendor
   cd vendor
   wget https://docs.aws.amazon.com/aws-sdk-php/v3/download/aws.zip
   unzip aws.zip
fi

# Update existing packages

yum -y update
```

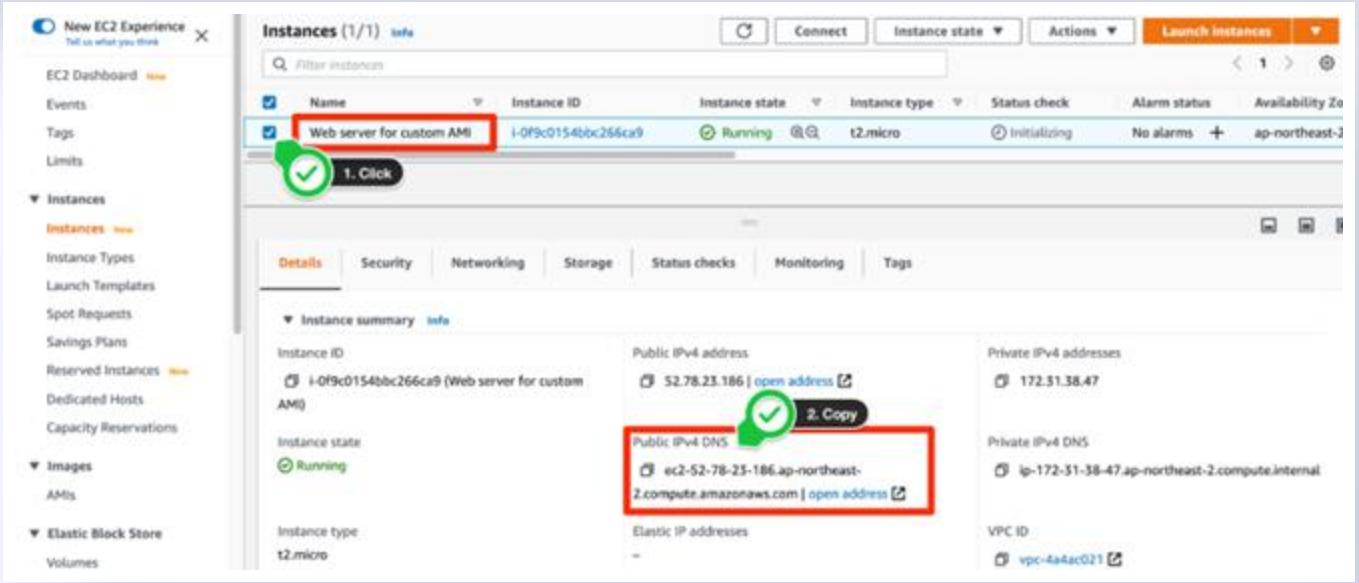
8. Finally click 'Launch Instance' from the Summary on the right:





9. Your new instance will now launch, and will take a few minutes to initialize and start. Click **View all instances** to check progress.

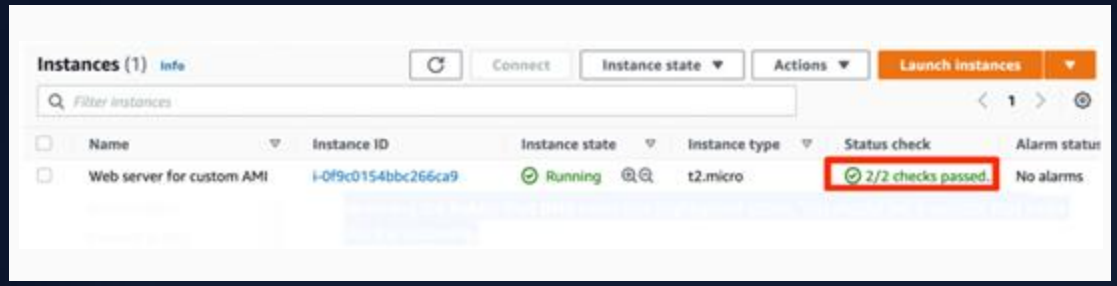
10. Once your instance has launched, you will see your Web Server as well as the Availability Zone the instance is in, and the publicly routable **DNS name**. Click the checkbox next to your web server to view details about this EC2 instance.



View your website live

11. Wait for the instance to pass the Status Checks to finish loading.
12. Open a new browser tab and browse the Web Server by entering the EC2 instance's **Public DNS name** into the browser. This name can be found in the console by reviewing the **Public IPv4 DNS** name line highlighted above. You should see a website that looks like the following.

If it doesn't load, try changing the address from https to http in the browser address bar.

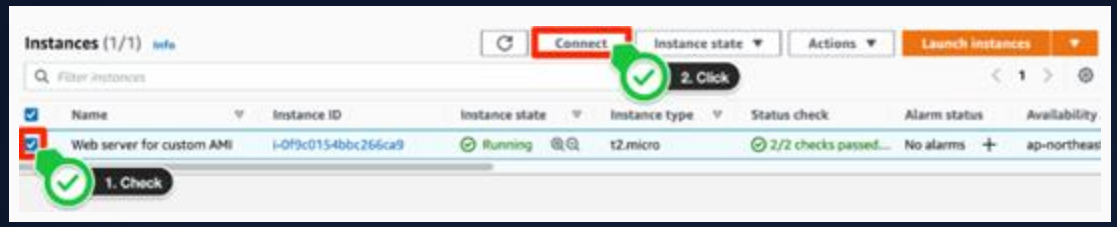


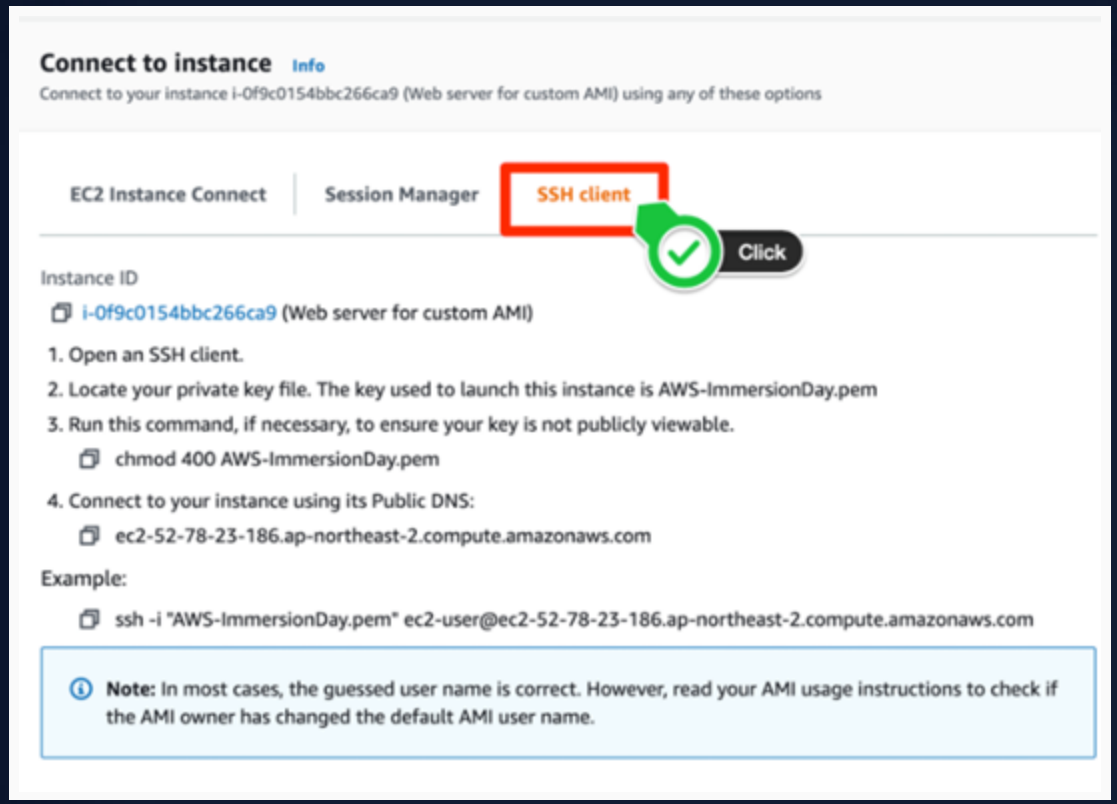
Connect to your Linux instance

1. In the EC2 instance console, select the instance you want to connect to, and then click the **Connect** button.
2. In the **Connect to instance** page, select **SSH client**. Follow the instructions provided.
3. If you are using Windows use **PuTTy** (next slide) or for Mac users, copy the **ssh** command from step 4 (see right).

You will need to run the ssh command from the same folder where you stored your .pem key file.

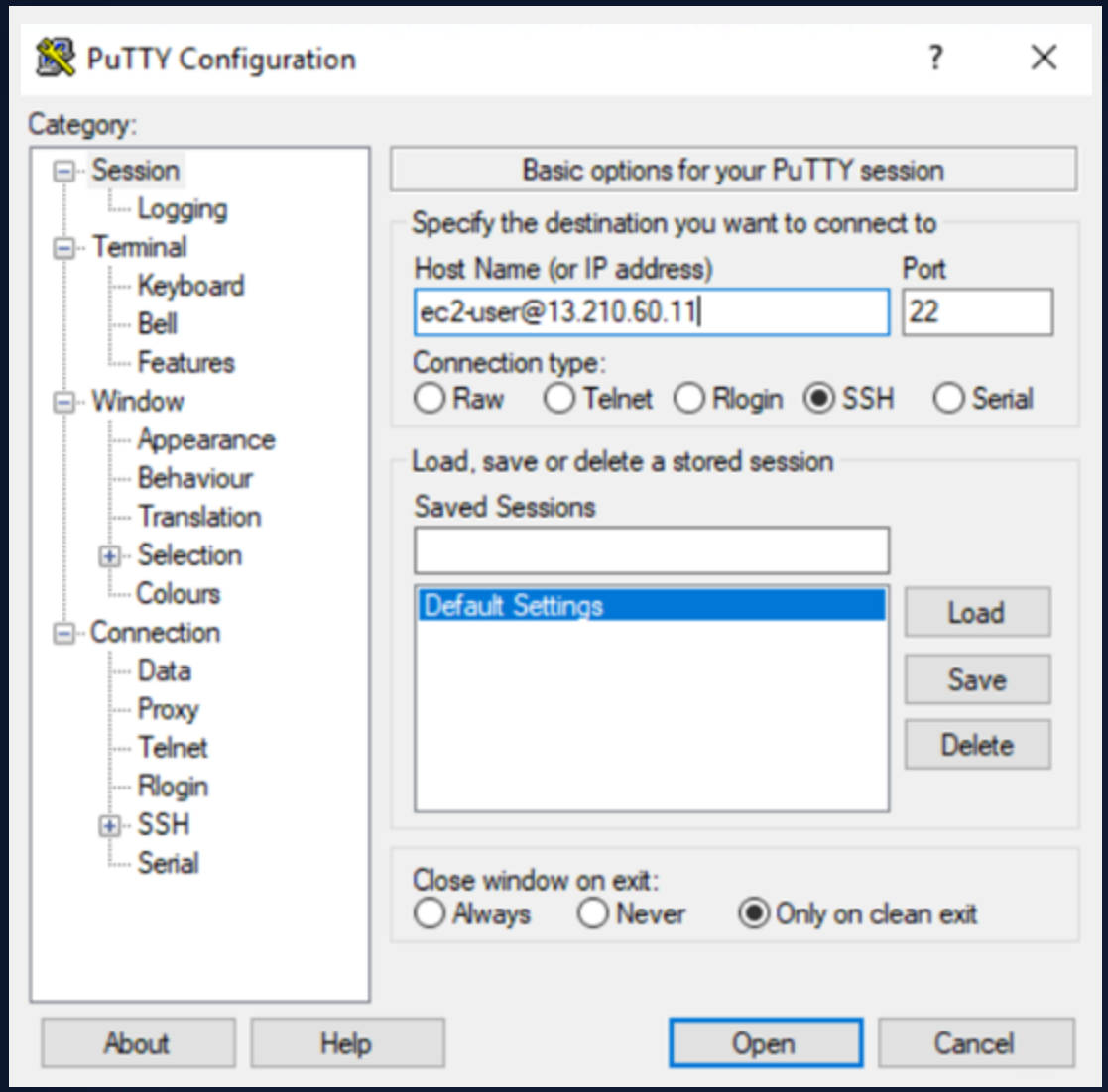
You will also need to run the command as sudo, or chmod the .pem file for it to work.





Connect using PuTTy (Windows)

1. Start PuTTy (if you need, to download PuTTY first).
2. In the **Category** pane, choose **Session**.
3. In the **Host Name** box enter **ec2-user@[your public IP of EC2 that you created]**.
4. Set the Port value to 22.
5. Under **Connection type**, select **SSH**.
6. In the **Category** pane, expand Connection, expand **SSH**, and then choose **Auth**. Complete the following:
   - Choose **Browse**.
   - Select the **.ppk** file that you generated for your key pair
7. Go back to the Session tab, put a name in the box below Saved Sessions, and click Save. Next time you open PuTTY you can reuse this saved session.
8. Click **Open**. *If this is the first time you have connected to this instance, PuTTY displays a security alert dialog box that asks whether you trust the host to which you are connecting.* Choose **Yes**. A window opens and login as **ec2-user** and you are connected to your instance.



Exercise 1

Try creating your own EC2 ubuntu instance on AWS, following the steps in the previous slides. Remember to create only a free tier server so that AWS does not charge you anything for your instance.

Section 2:

Environment Variables

Environment variables are key-value pairs supplied to a specific program, used for storing data specific to an environment. You would typically have different environment variables for a **development** environment running on **localhost**, a **staging** environment for **testing** changes, and different variables again for a production environment. Values such as **database** connection details, **ports**, **API keys**, and **paths** are all commonly stored in environment variables.

Both keys and values are always **strings**. These named values are then available to the code of the **back-end** application, allowing it to connect to any database or use other settings that vary for different environments, wherever the code is deployed.

However, **front-end** projects are not as simple. Browsers do not support runtime environment variables because they are different for each user/client. Instead developers can use bundlers like Webpack or Vite to define environment variables at compile-time, but these are still sent to the browser and are therefore less secure.

Environment Variables

The purpose of using environment variables is to **separate configuration from code**, ensuring a clean and secure setup. Modern applications often follow the **Infrastructure-as-Code** (IaC) model, where infrastructure configuration is managed through machine-readable files (often in **YAML**, **JSON** or **HCL** formats) for automated, consistent, and version-controlled deployments. Environment variables can be stored in **files** (e.g., .env files) or directly in environment **settings**, providing flexibility in managing configurations across development, staging, and production environments.

Why use environment variables instead of JSON files? While JSON supports nested structures, simplicity is key, and most settings don't require nesting. Additionally, storing sensitive credentials in project repositories is insecure, and environment variables allow for better control over the settings passed to the application.

Dotenv NPM

Dotenv is a module that reads `.env` files and loads environment variables into `process.env`. It is one of the most used npm packages for reading environment variables, and you should have already used it in previous modules to separate configuration from the code. This file should not be included in your Github repository - make sure you add it to the `.gitignore file, along with node_modules`.

First install the package:

```
npm install dotenv
```

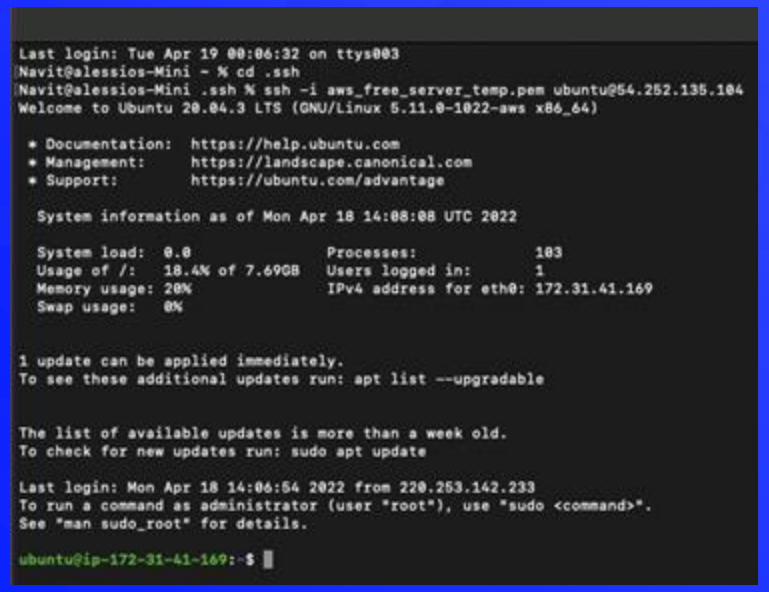
To use dotenv, simply add this code:

```
require('dotenv').config();
```

`process.env` is a global Node.js object that stores all of the environment variables provided by the system or runtime. The `dotenv` package enhances this by loading variables from a `.env` file into `process.env`, while system-defined variables take precedence. This means we can configure our system either with a `.env` file, or with environment settings configured when running the app, eg. from an AWS EC2 server - all without changing any code.

Section 3:

Deploying application in EC2



After creating your ec2 instance on the previous slides, you should be able to connect to it using ssh.

Follow the steps from slide 14, using either **PuTTY** on Windows with your **PPK** format key, or **SSH** on Mac with your **PEM** format key.

The username should be `ec2-user` and the key is used as the authentication mechanism instead of a password.

```
ssh -i "NewKeyPair.pem" ec2-user@ec2-54-206-120-79.ap-southeast-2.compute.amazonaws.com
```

Deploying application in EC2

Next what we need to do is to install Docker on our EC2 instance. The commands for this can vary depending on the type of linux installation used. If the left set of commands below are not working, try the right:

```
sudo apt update
sudo apt install docker
```

```
sudo yum update
sudo yum install docker
```

Once we have Docker installed on our server, we can check if it was successfully installed and start it by using the commands:

```
docker --version
sudo systemctl start docker.service
```

We can also see if Docker is running or not by using the command:

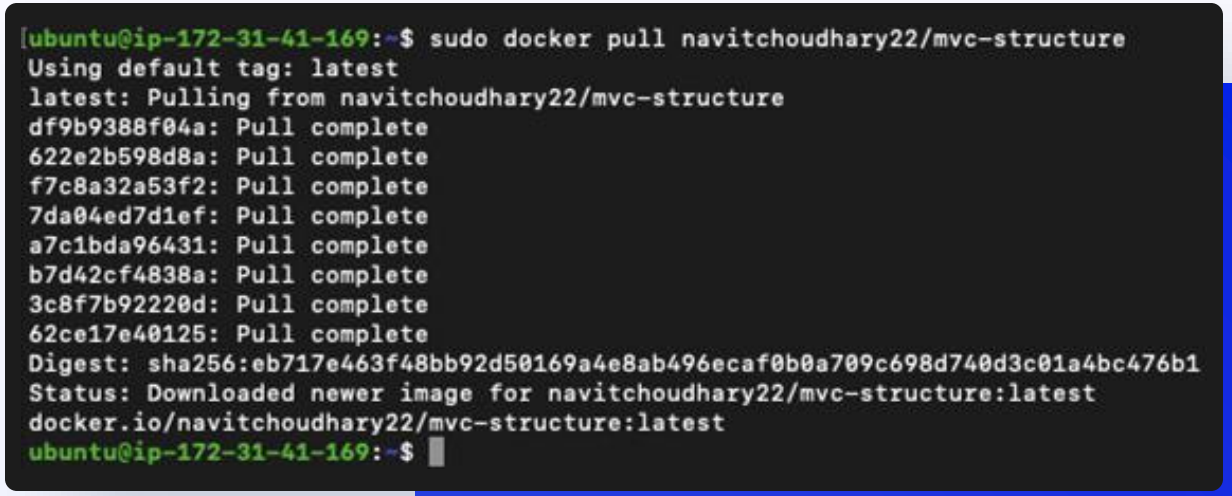
```
sudo systemctl status docker
```

Deploying application in EC2

Now that we have Docker installed, we can use the Docker image of our previous application hosted on DockerHub. First we need to pull it onto the EC2 instance (substitute your own image name in the command below):

```
sudo docker pull <image name>
```

```
sudo docker pull navitchoudhary22/mvc-structure
```



Next, stop the existing application running on port 80 by running the following command:

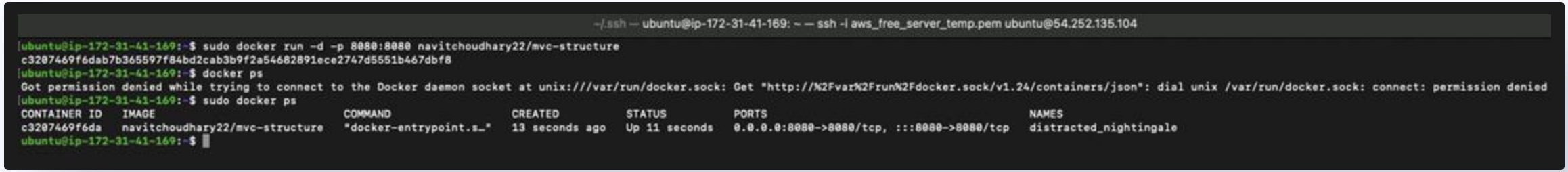
```
sudo systemctl stop httpd
```

Deploying application in EC2

After successfully cloning the image on our EC2 server, all we now need to do is to run the docker image:

```
sudo docker run -d -p 80:8080 <image name>
```

```
sudo docker run -d -p 80:8080 navitchoudhary22/mvc-structure
```



Make sure to use 80 as the first port before the colon, and the right internal port for your application as the second port after the colon, and substitute your own image name.

Our application should now be successfully running on our EC2 instance. You can see it running in your browser by using the public URL for your EC2 instance.

Exercise 2

Try hosting the docker image of your NodeJS application that you previously configured to use GitHub actions to deploy on DockerHub.

Share the link of your hosted application to your trainer.

Section 4:

AWS Elastic Beanstalk

The entire application development process is being reshaped by cloud computing. A variety of cloud providers, such as Amazon Web Services and Microsoft Azure, provide development tools to make the process easier and more secure. The AWS Elastic Beanstalk development tool is an example of a PaaS-based (platform-as-a-service) development tool.

AWS Elastic Beanstalk is a simple tool for delivering and scaling web applications and services written in Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on well-known servers like Apache, Nginx, Passenger, and IIS.

A developer can use AWS Elastic Beanstalk to launch an application without having to manually provision the underlying infrastructure while yet retaining high availability.

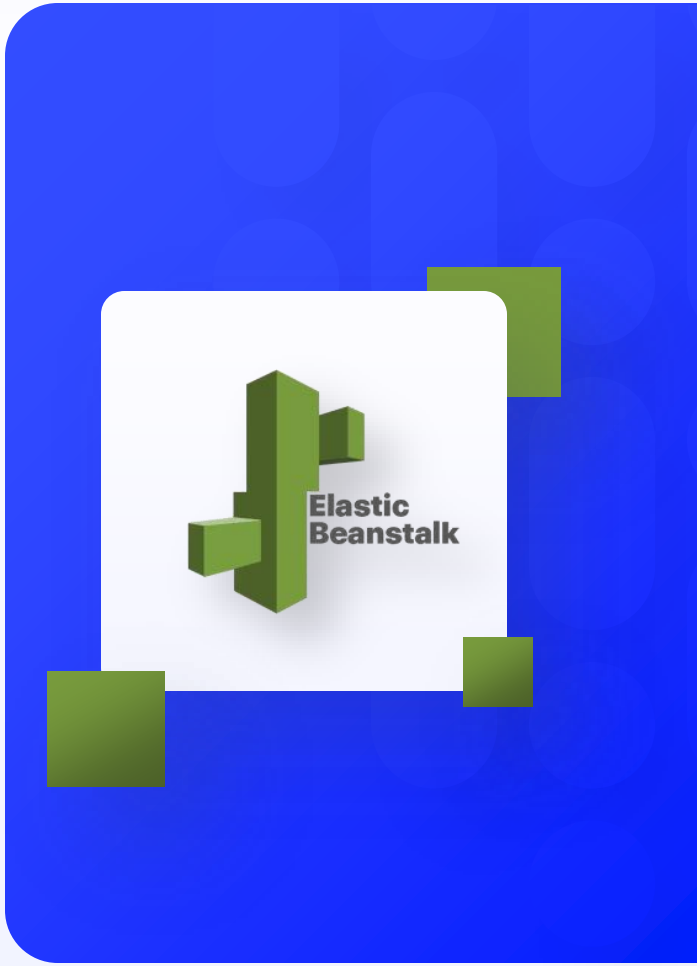
Benefits of Elastic Beanstalk

Offers Quicker Deployment: Elastic Beanstalk allows developers to quickly and simply deploy their apps. Users will not need to worry about the underlying infrastructure or resource settings because the application will be ready to use in minutes

Supports Multi-Tenant Architecture: Customers can use AWS Elastic Beanstalk to distribute their programmes across numerous devices while ensuring scalability and security. It creates a detailed report on app usage and user profiles.

Simplifies Operations: Beanstalk is in charge of the application stack as well as the infrastructure provisioning and management. Developers must concentrate only on writing code for their application rather than managing and configuring servers, databases, firewalls, and networks.

Offers Complete Resource Control: Developers can use Beanstalk to select the appropriate AWS resources for their application, such as the EC2 instance type. It allows developers to have complete control over AWS resources and access them at any time.



Elastic Beanstalk Components

When deploying an application on Beanstalk there are certain terms that will come up frequently. Let us look at those concepts:

Application:

- In Elastic Beanstalk, an application is conceptually comparable to a folder.
- An application is made up of various components such as environments, versions, and configurations.

Application Version:

- A specific, identified iteration of deployable code for a web application is referred to as an application version.
- An Amazon S3 object containing deployable code, such as a Java WAR file, is referenced by an application version.

Environment:

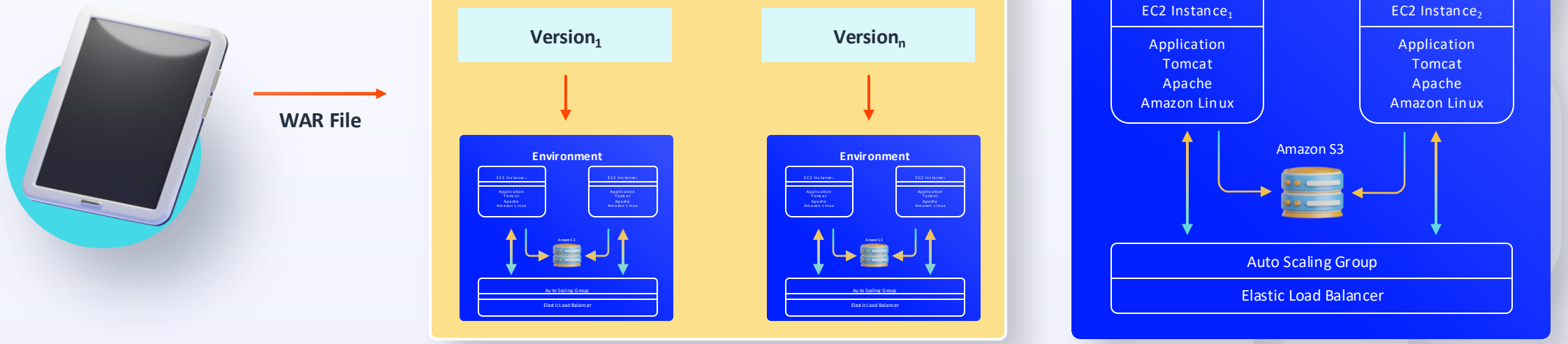
- The current version of the Elastic Beanstalk Application will be active in environments within the Elastic Beanstalk Application.
- At any one time, each environment only runs one application version. However, the same or different versions of an application can be operated in many settings at the same time.

Elastic Beanstalk Components

Environment Tier:

Based on requirements, Beanstalk offers two different Environment tiers: Web Server Environment, Worker Environment.

- Web Server Environment: Handles HTTP requests from clients (our application will use this)
- Worker Environment: Processes background tasks which are resource-consuming and time-intensive



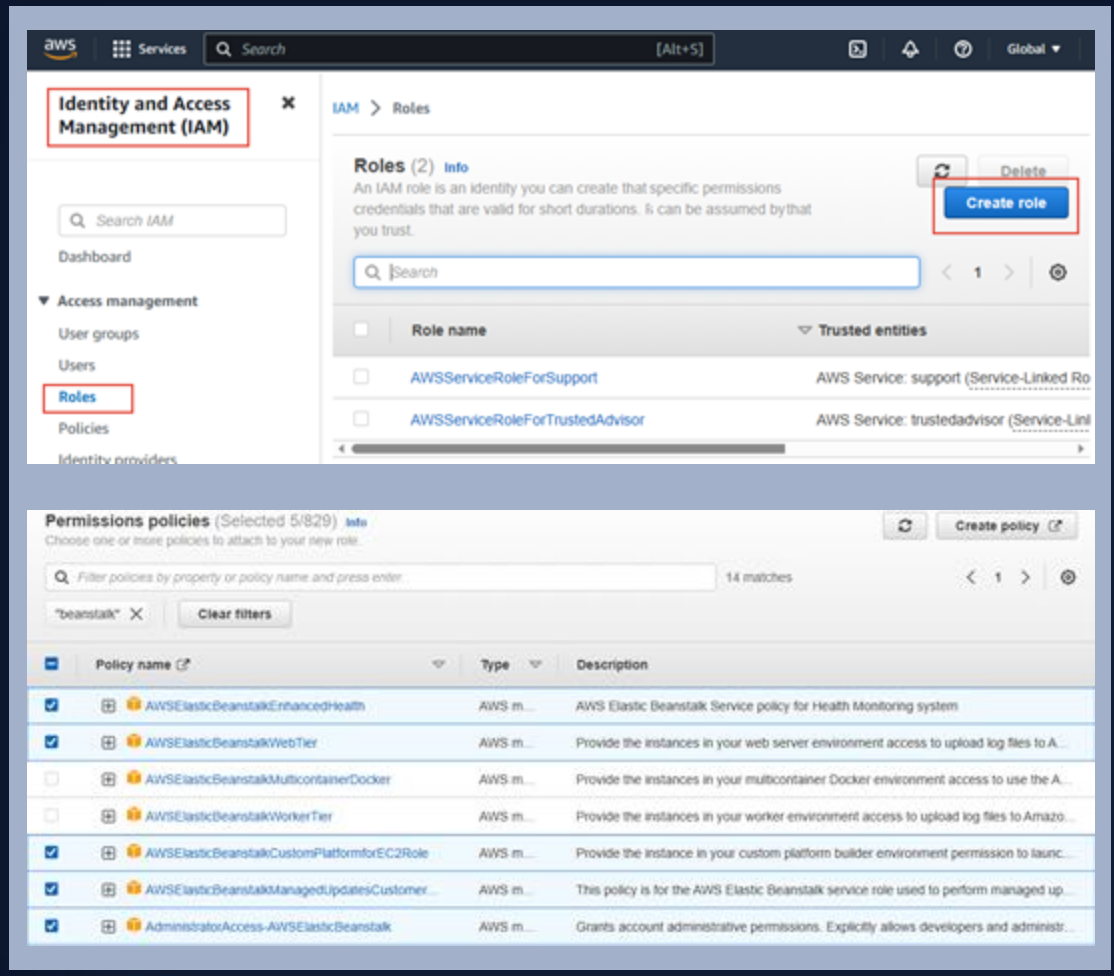
Setting up Elastic Beanstalk Access

1. Our first step is to set up security and access permissions for our Beanstalk environment to use.
   Search for **IAM** (Identity and Access Management) in the top bar, then click **Roles** and '**Create role**'.

2. Choose **AWS service** as the trusted entity type, with **EC2** as the use case.

3. In **Permissions policies**, search for 'beanstalk' to find the required policies, then tick AWSElasticBeanstalkEnhancedHealth, **AWSElasticBeanstalkWebTier**, **AWSElasticBeanstalkCustomPlatformforEC2Role**, **AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy**, and **AdministratorAccess-AWSElasticBeanstalk**.

4. Name the new role 'aws-elasticbeanstalk-service-role' and click Create.



Setting up Elastic Beanstalk Server

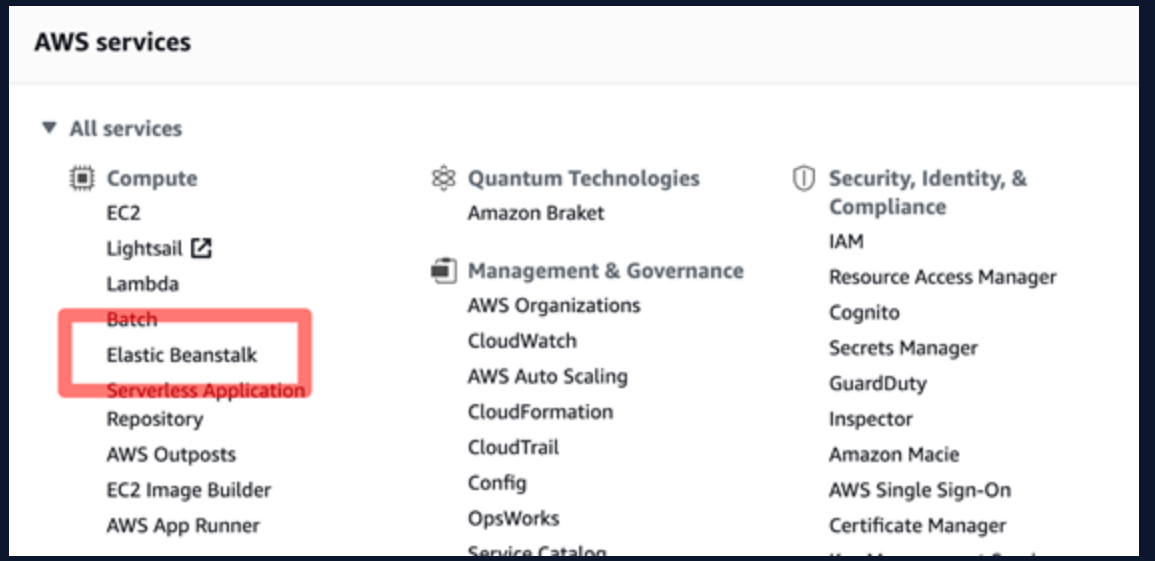
1. Now we can set up our Elastic Beanstalk server on Amazon AWS. Login to your AWS Management Console and click on “Elastic Beanstalk” under services (or search for it)

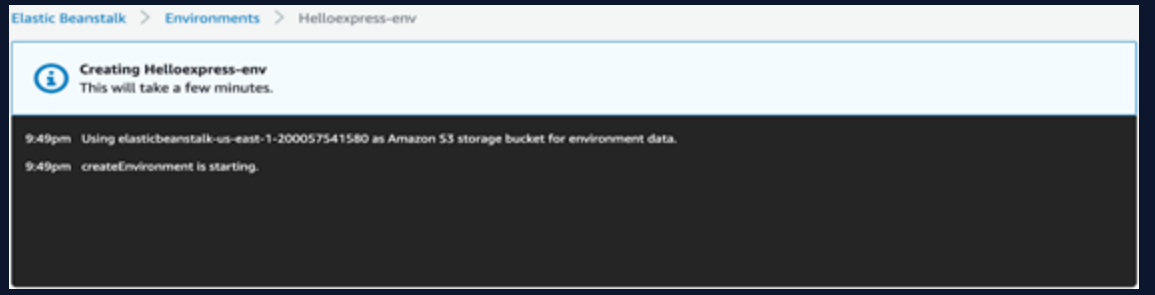
2. Next step is to create an Amazon Elastic Beanstalk application. Click on the “Create Application” button and choose a web server environment.

In the create application page fill out the details as shown below (leave the rest as default):

- **Application Name** : Hello-Express
- **Platform** : Node.js
- **Platform branch** : (default)
- **Platform version** : (default)
- **Application Code** : Sample application
- **Configuration presets** : Single instance (free tier)

Click Next.





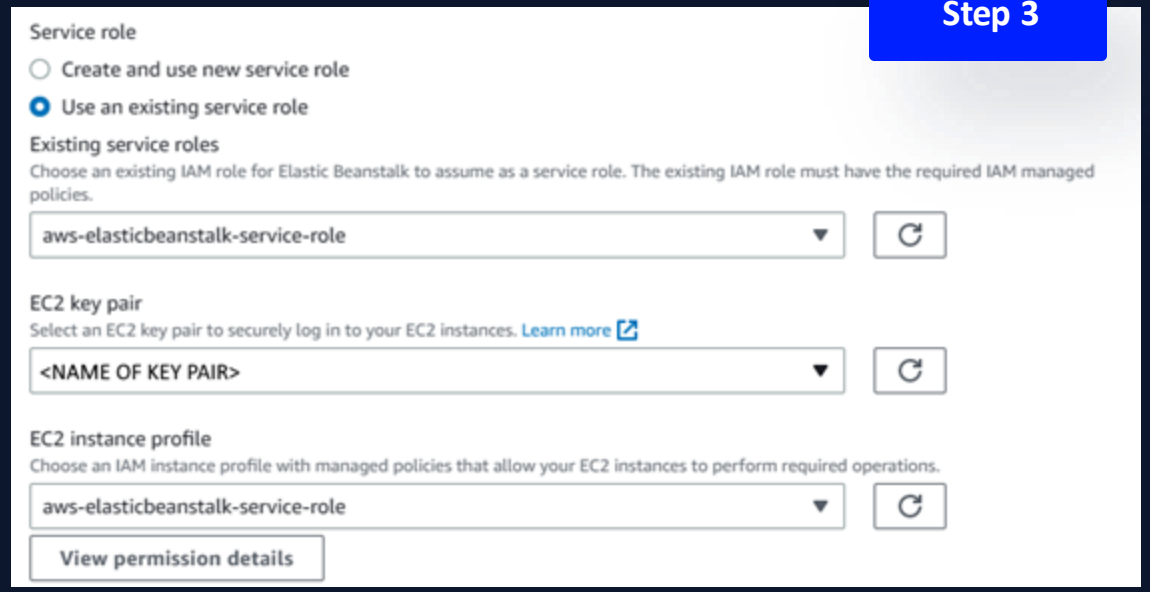
Setting up Elastic Beanstalk Server

3. Select your new **aws-elasticbeanstalk-service-role** for both the ‘Existing service role’ and ‘EC2 instance profile’ dropdowns, and choose your previously created EC2 key pair. Click Next.

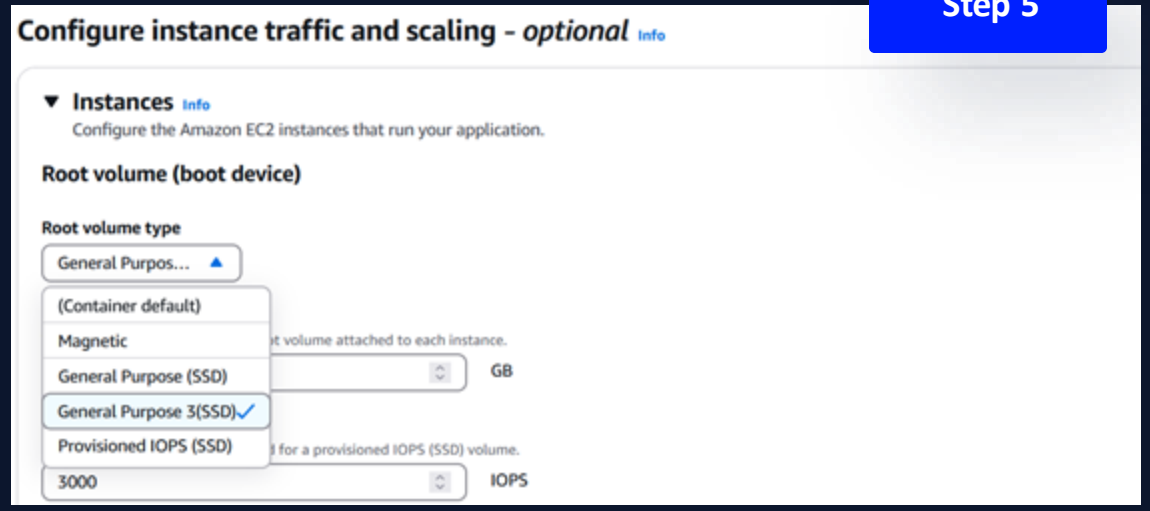
4. Skip the ‘Set up networking, database, and tags’ section settings and click Next.

5. In the ‘Configure instance traffic and scaling’ settings, choose **General Purpose 3 (SSD)** for the ‘Root volume type’.

Step 3



Step 5

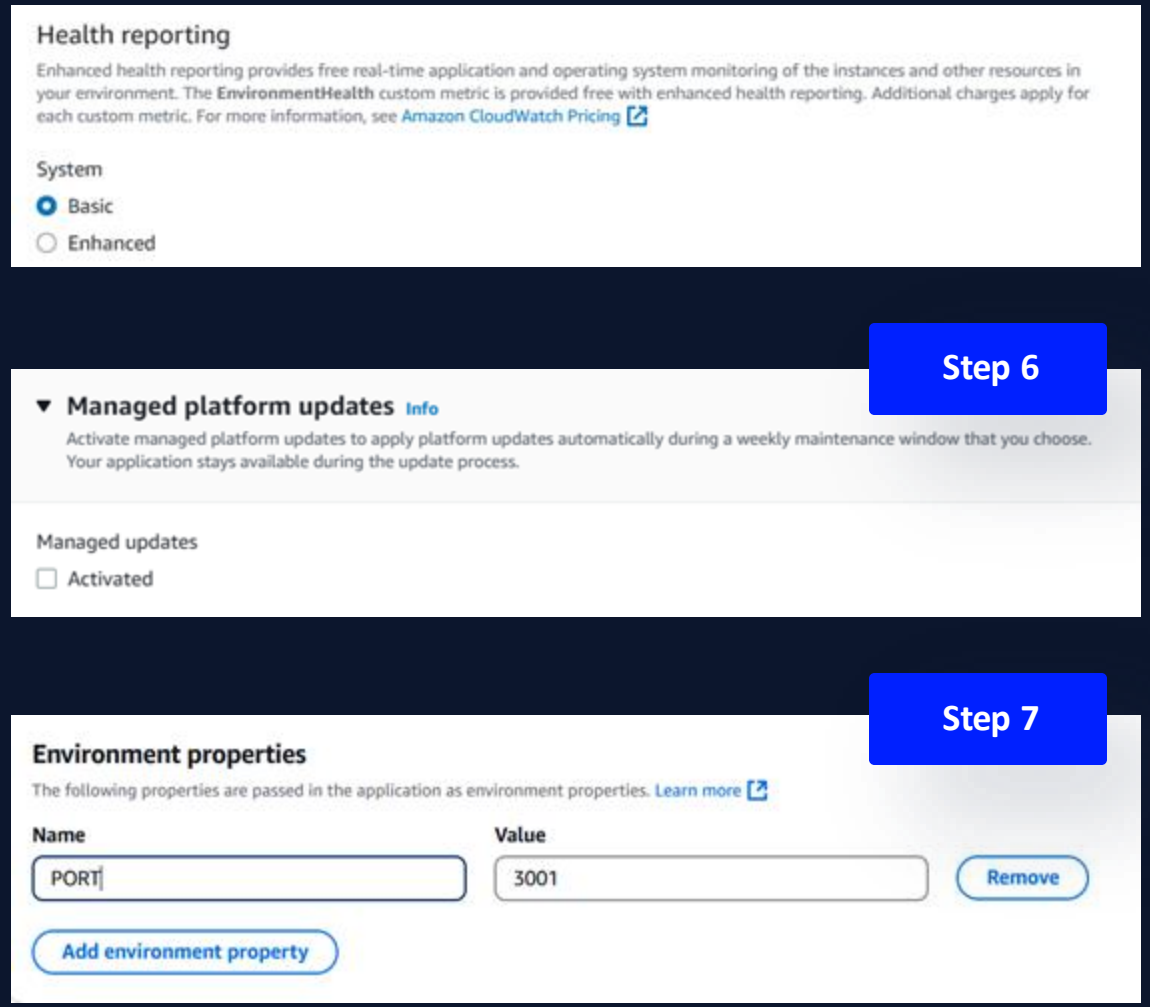


Setting up Elastic Beanstalk Server

6. In the ‘Configure updates, monitoring, and logging’ section, change Health Reporting from Enhanced to **Basic**, and **turn off** the ‘Managed platform updates’ so that the **Activated** checkbox is **unticked**. Click Next.

7. Add environment variables on the last section of the ‘Configure updates, monitoring, and logging’ page.

   Specify the PORT used internally by your application (check `index.js or server.js`).



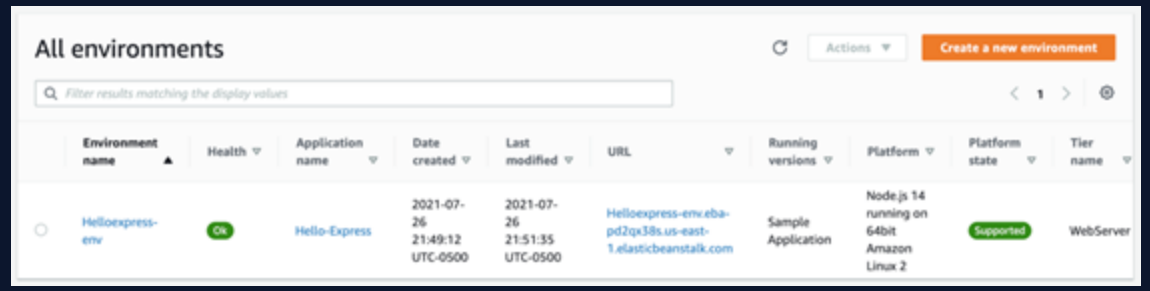
Setting up Elastic Beanstalk Server

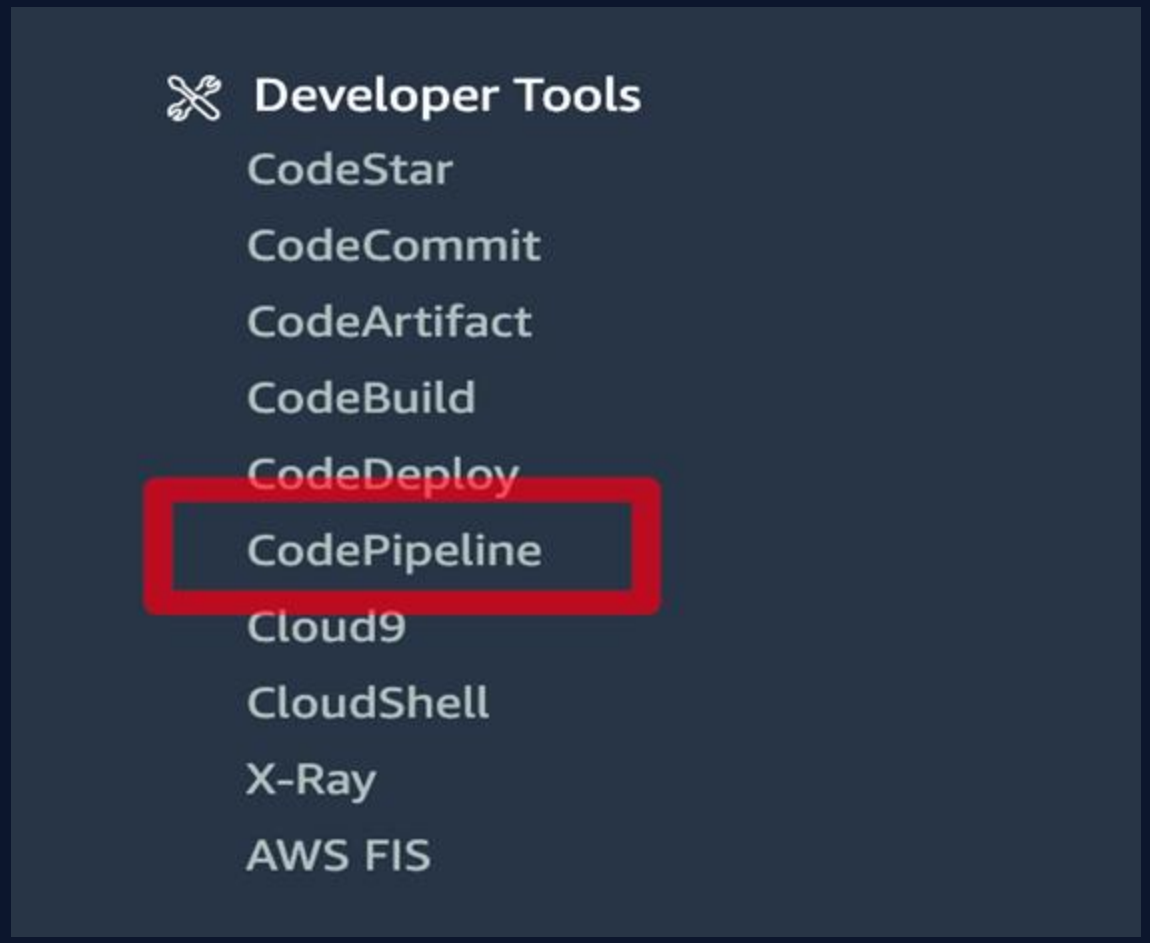
Once the create application setup is finished you should see a similar screen as shown in the screenshot.

The screenshot above indicates that the application “Hello-Express” has been created and it also contains a default environment “Helloexpress-env”.

Since we want our GitHub changes to propagate and auto-deploy to the AWS Elastic Beanstalk, we must set up a **Pipeline** on AWS management console.

Search for ‘CodePipeline’ in the top AWS search bar.





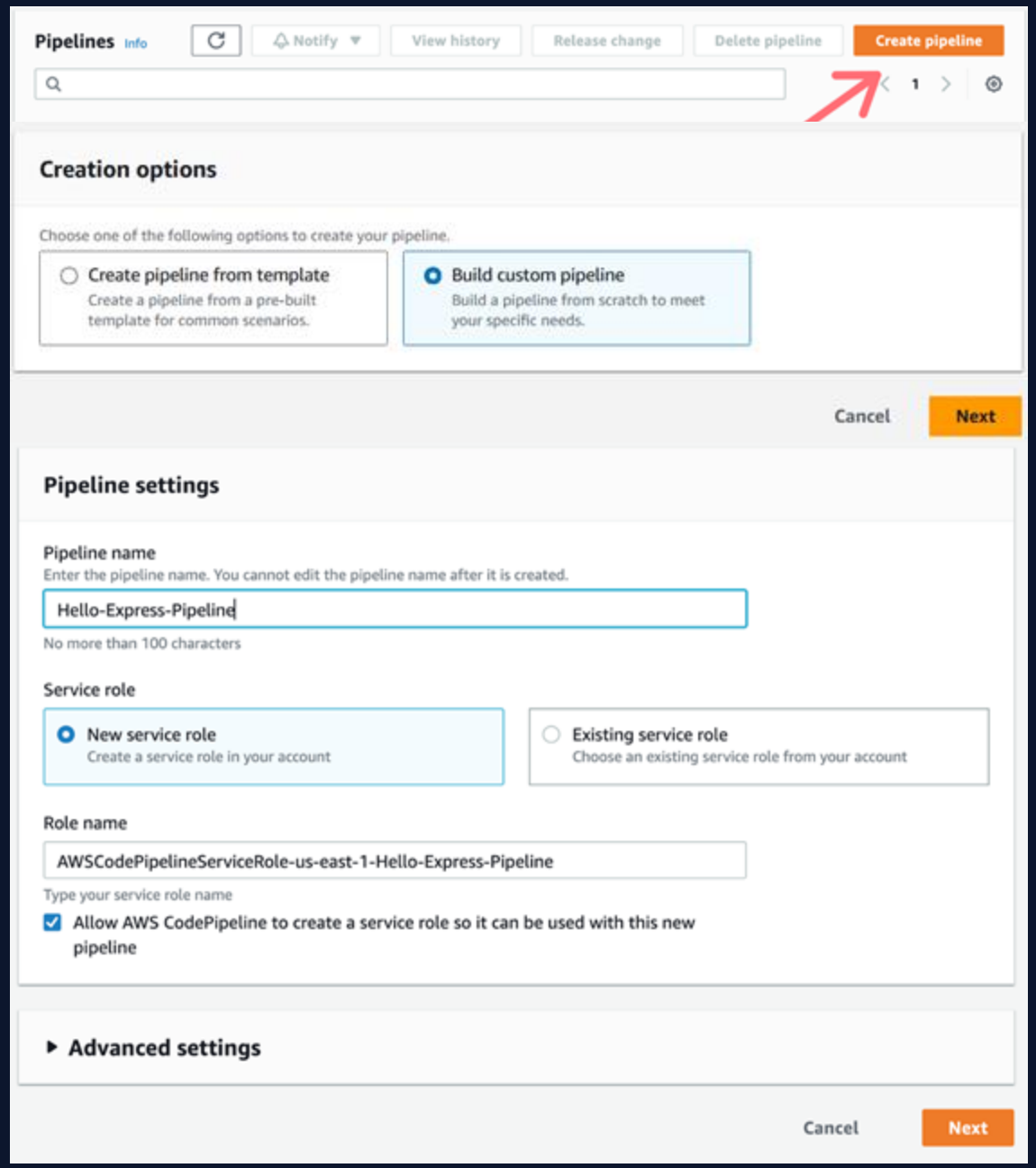
Setting up Elastic Beanstalk Server

This will open up a screen allowing you to create a new pipeline. Click the “**Create pipeline**” button.

After clicking on the “Create pipeline” button you will be taken to a page, where you can add details about the pipeline.

Choose the ‘**Build a custom pipeline**’ option and click Next.

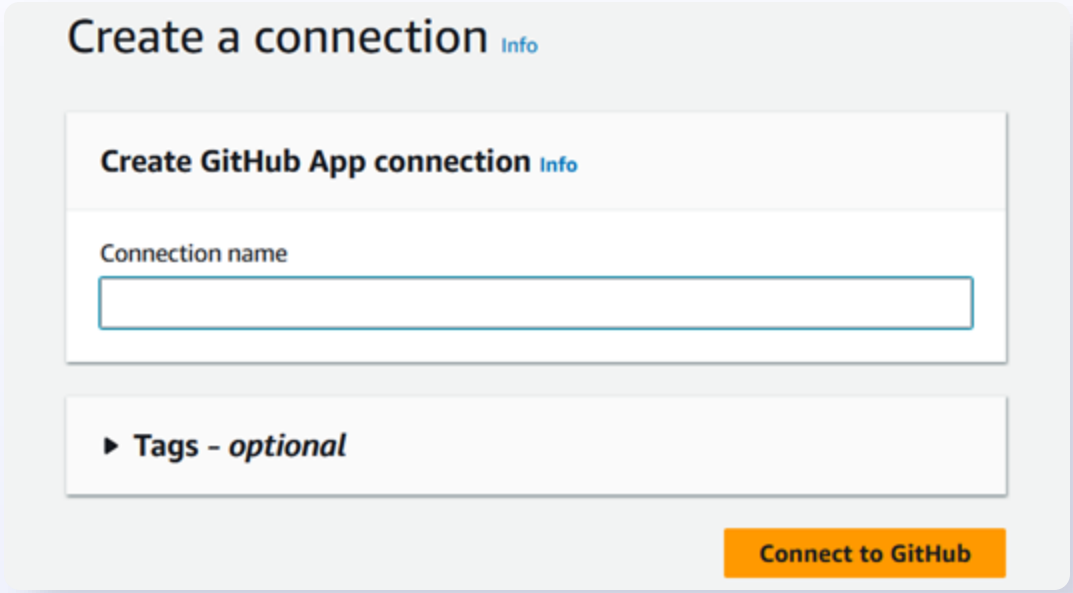
Once you fill out the “**Pipeline name**”, it will automatically fill out the Role name. Leave the **default** execution mode and click “Next” to continue.

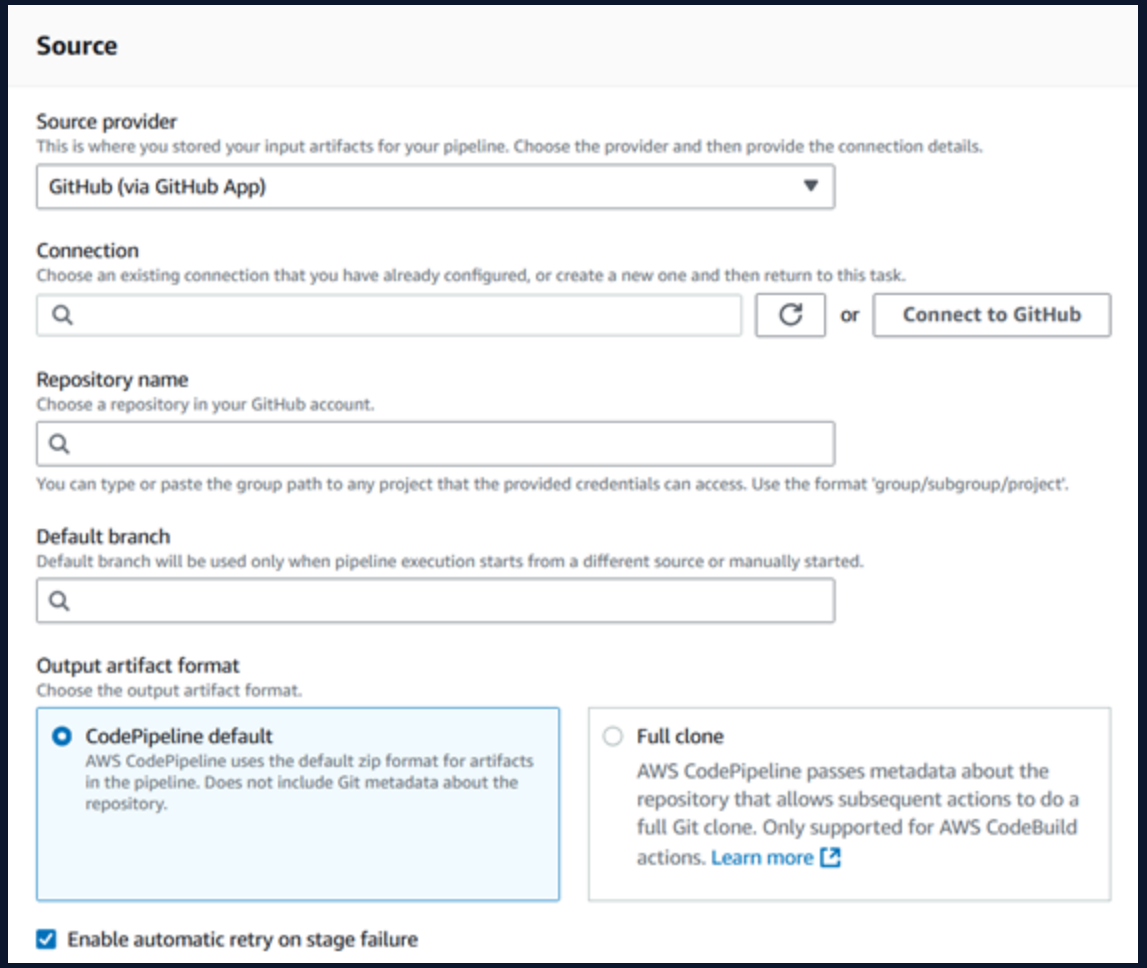


Setting up Elastic Beanstalk Server

In the next screen, you will choose the source stage. Since, we are using GitHub as our source repository we will choose **GitHub (via GitHub App)**.

Click on “Connect to GitHub” to start the process.

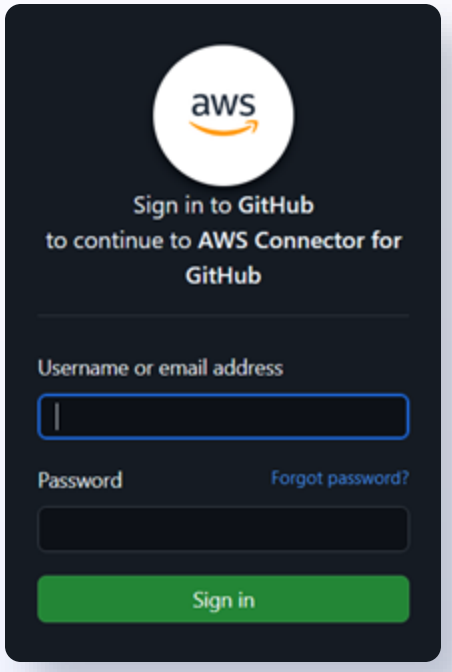




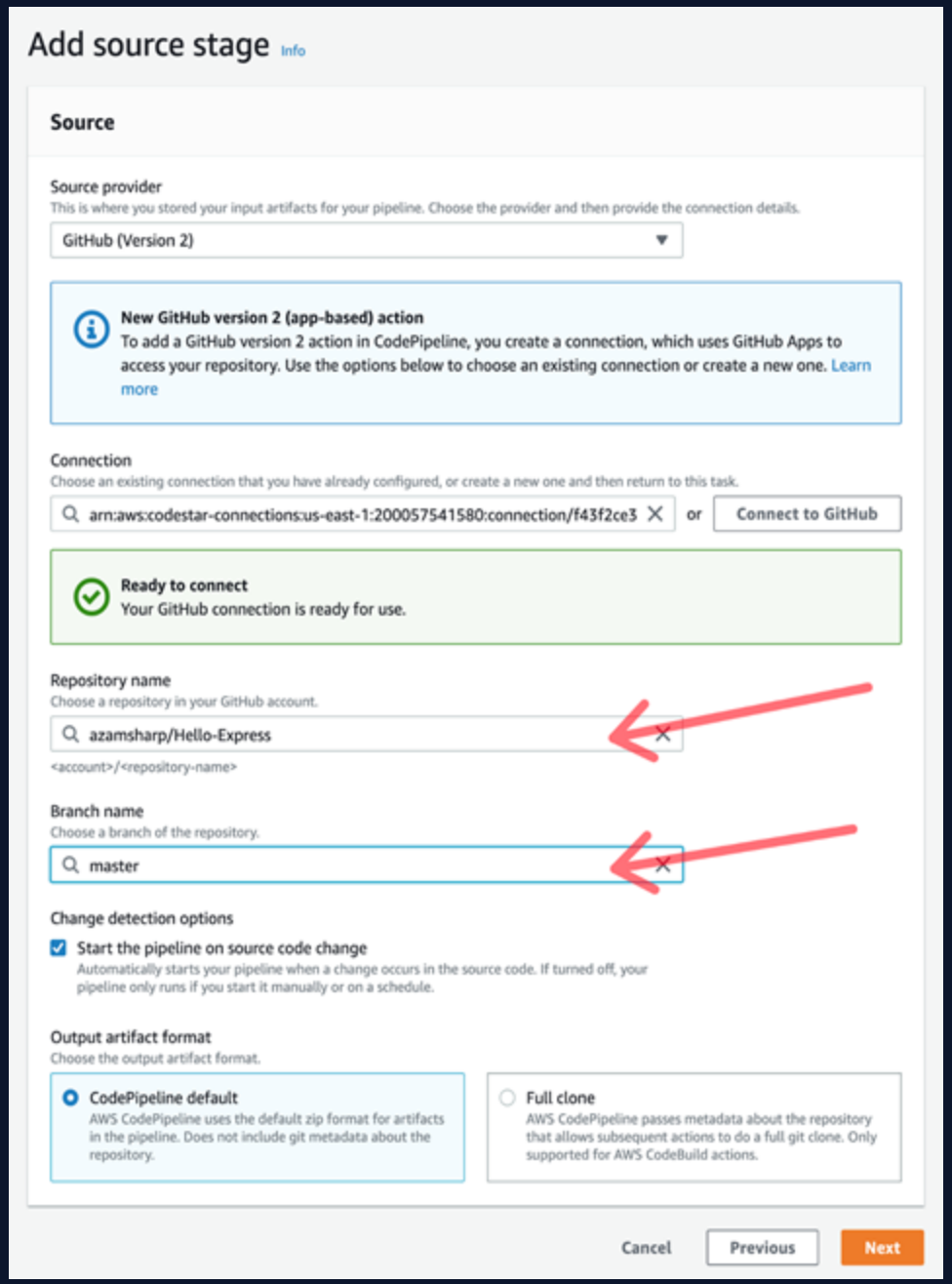
Setting up Elastic Beanstalk Server

When you click “Connect to GitHub” it will open a small popup, which will allow you to create a connection. Add a connection name and click on the “Connect to GitHub” button.

Enter your GitHub credentials to create a connection between GitHub and CodePipeline and install the app when prompted. You can then select your GitHub repository and the branch.



Select the same repository you’ve been working with (you can choose any standalone Node.js app) and the master (or main) branch of the repository, because we want to deploy the code in the master branch to the server. Click next.



Setting up Elastic Beanstalk Server

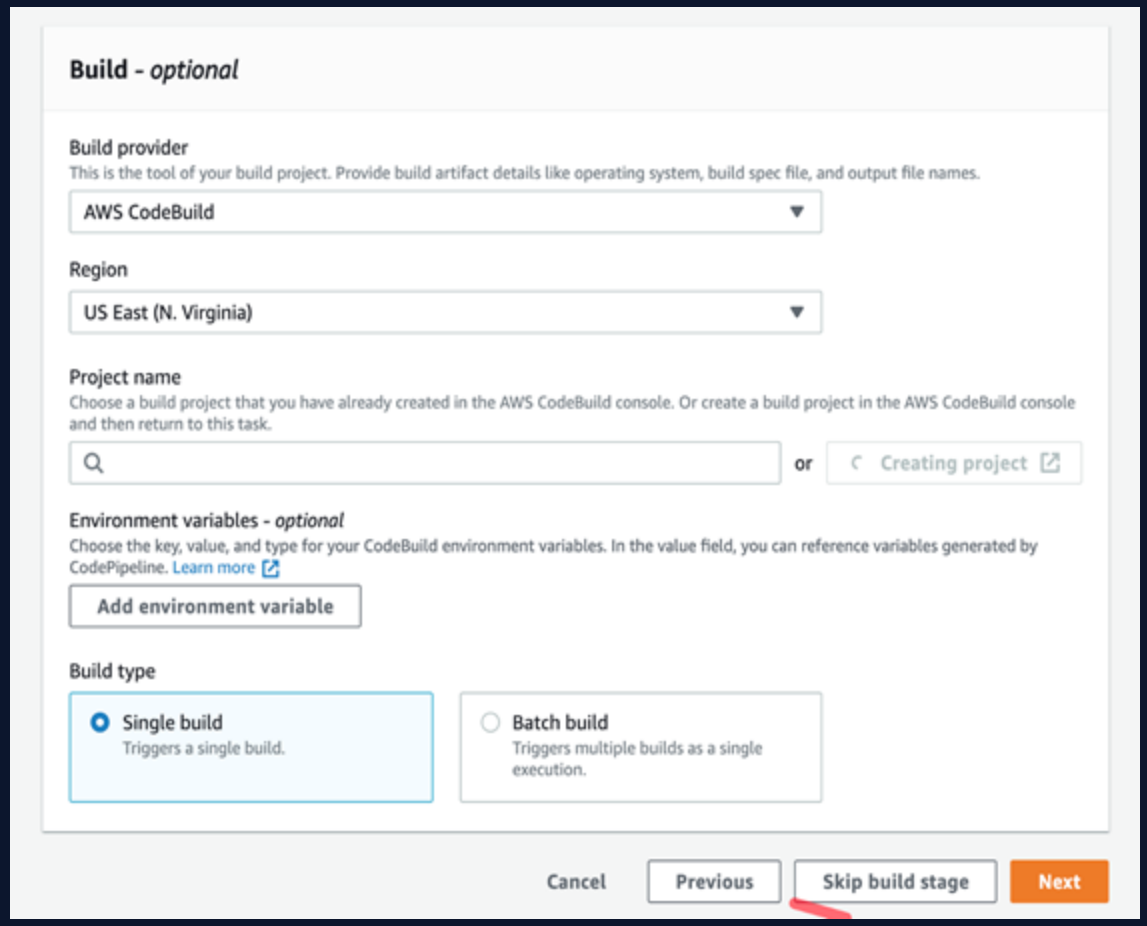
The next screen will allow you to add a **build** stage. We are going to skip this step, so click on the "**Skip build stage**" button at the bottom.

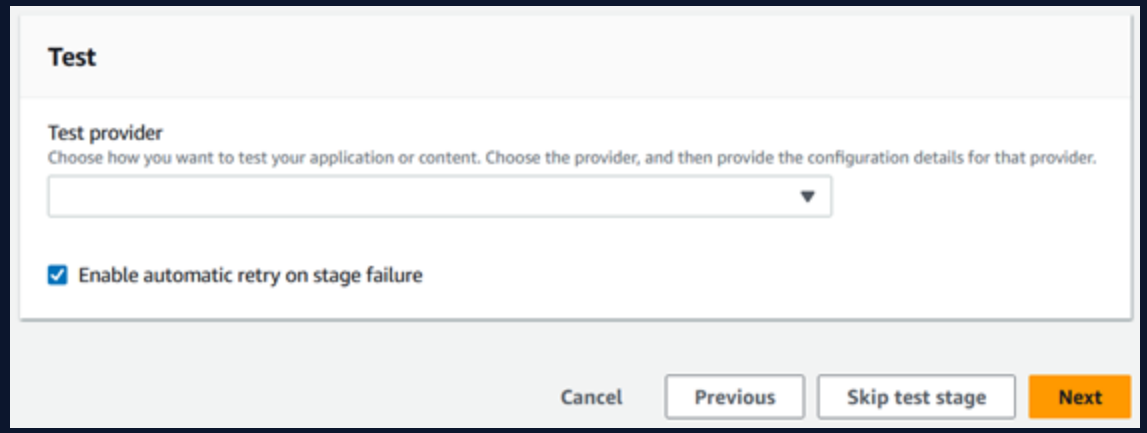
A confirmation dialog will popup, select "Skip".

The build stage allows you to configure how your app should be built and run - we don't need it for simple apps using common defaults.

After the **Build stage** comes the **Test stage**. This allows your app to be automatically tested before deployment, if you have unit tests set up using a tool like Jest.

We will skip this stage as well - click on "**Skip test stage**".





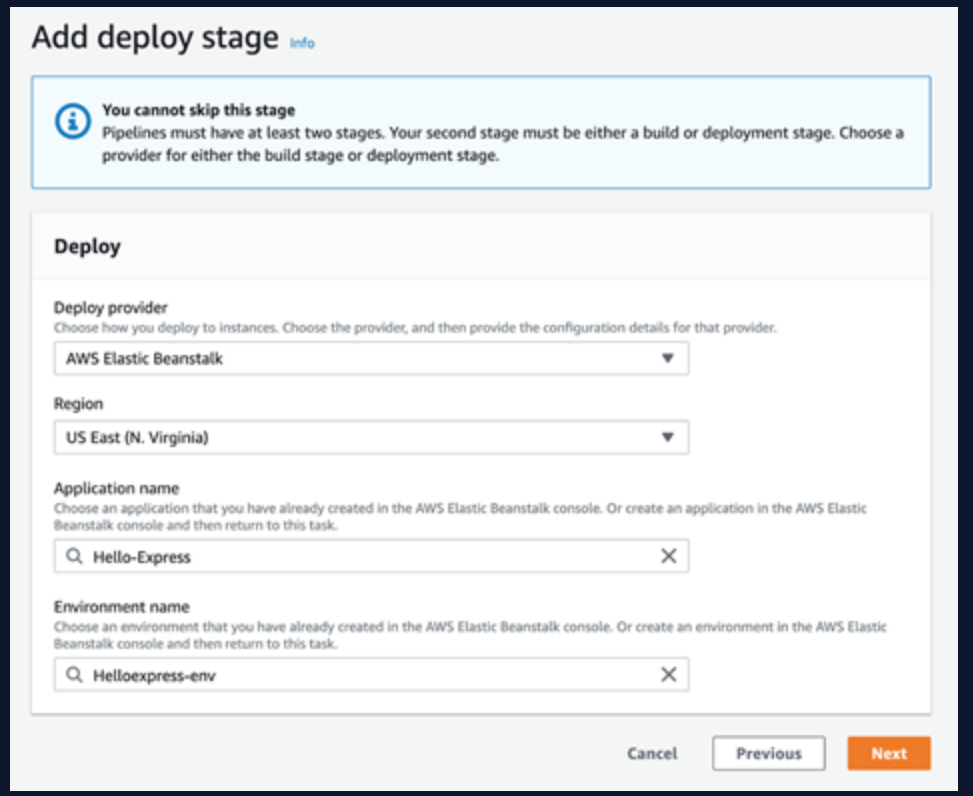
Setting up Elastic Beanstalk Server

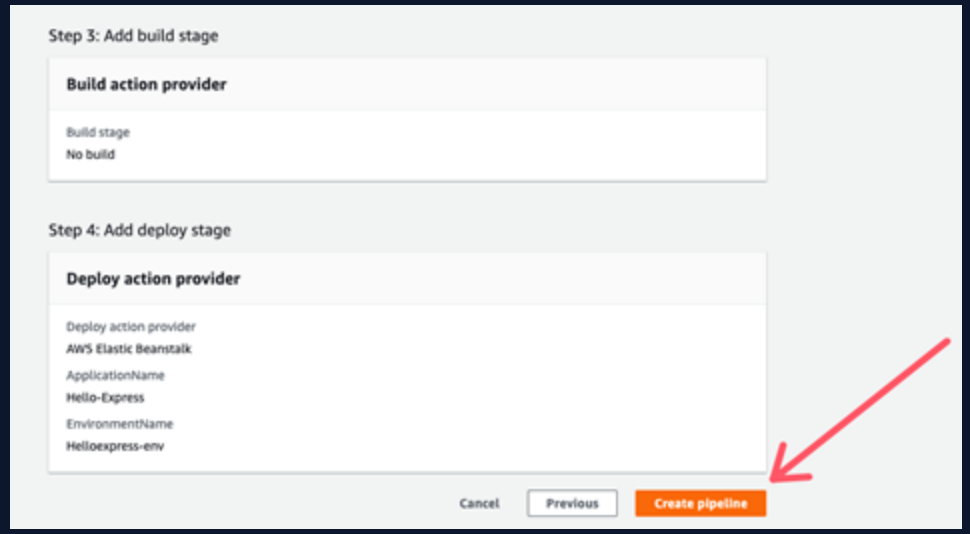
Next we will land on the “Add deploy stage” page. This is where we need to select our Elastic Beanstalk application and its environment.

Use the dropdowns to select **AWS Elastic Beanstalk** as the **Deploy provider**, the **same region** you used for your Beanstalk environment, and your previously created Beanstalk **Application name** and **Environment name**.

Click next.

The next screen will be the review screen. Make sure all settings and configurations are correct. Scroll down to the bottom and click the “Create pipeline” button.



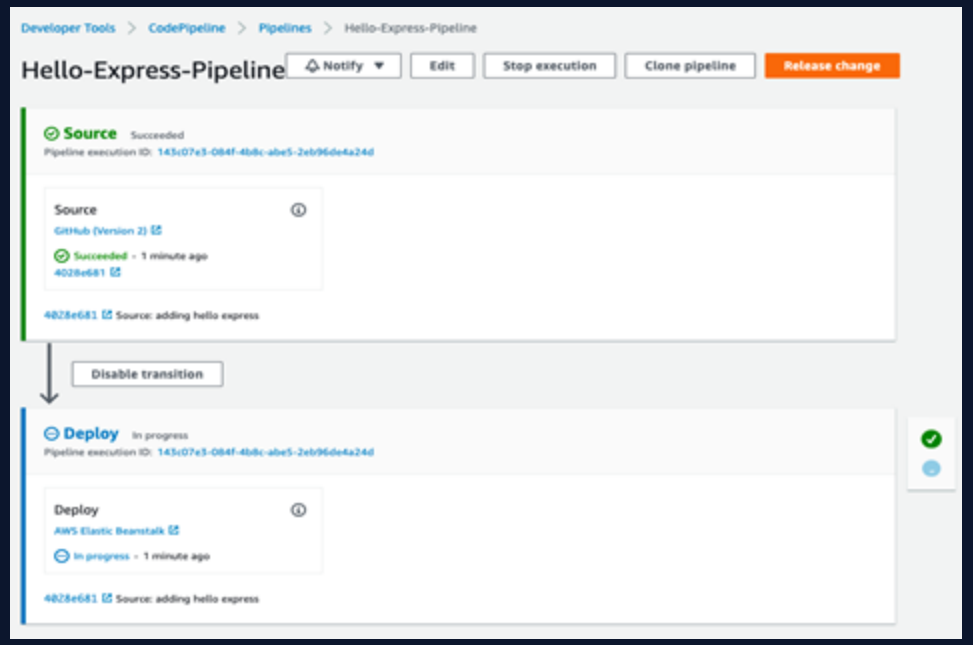


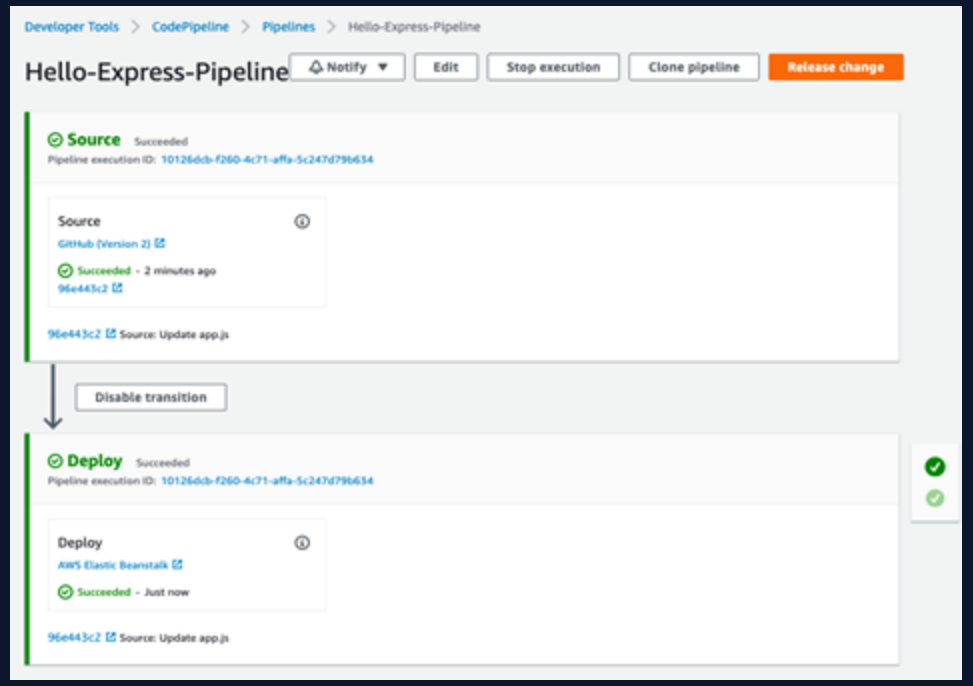
Setting up Elastic Beanstalk Server

As soon as the pipeline is set up, it will try to source your code from GitHub and automatically deploy the app to Amazon AWS Elastic Beanstalk.

This will take a few minutes.

After a while it should come back with deployment succeeded status.





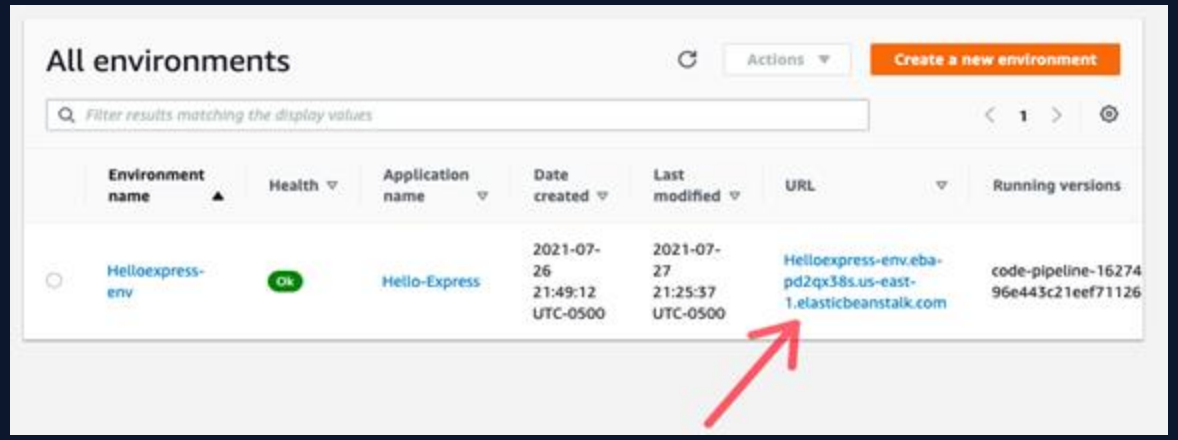
Setting up Elastic Beanstalk Server

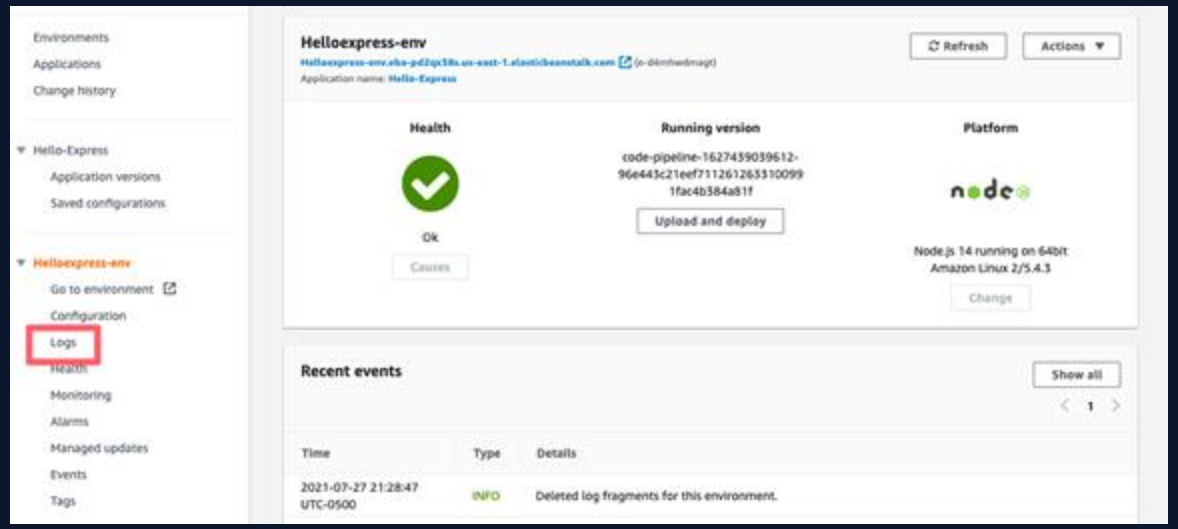
Now we need to see if we can access our application. Click on “All services” and then select Elastic Beanstalk.

Next click on the URL of your new application. This will launch the app in a browser window.

You can also click on the environment name and it will take you to a page that gives you all the details about the application and also lets you see the logs, monitor the application and many more.

Now you have successfully created an application on AWS Elastic Beanstalk.





Exercise 3

Try hosting the GitHub repo of a NodeJS application that you have already created, using AWS Elastic Beanstalk.

Choose a standalone repository with no dependencies on a database.

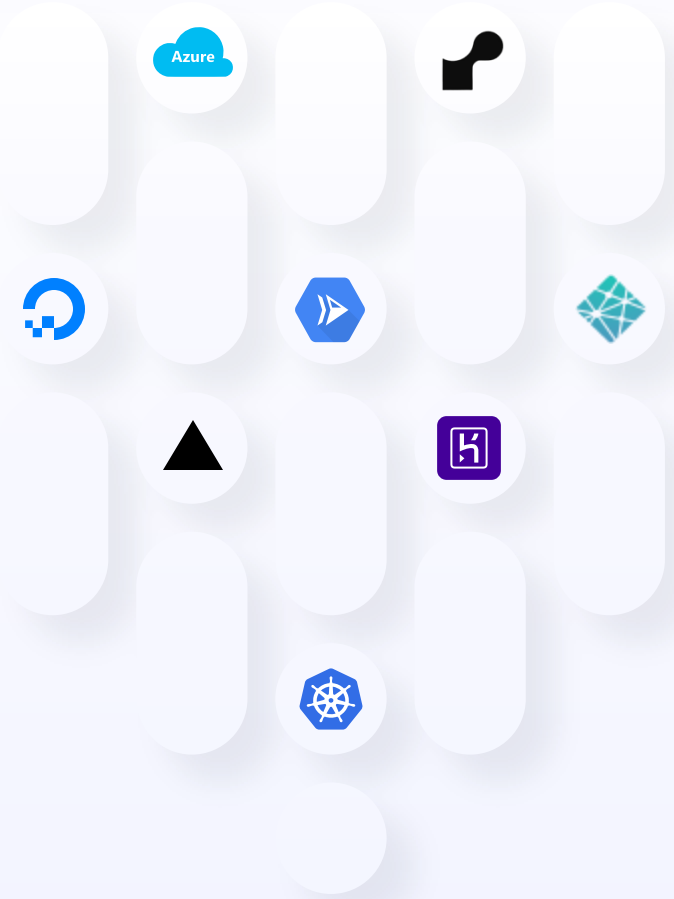
Share the link of your hosted application with your trainer.

Other Deployment options

There are many services available for **automating deployment in the cloud**. AWS is an excellent option because it provides everything from beginner-friendly, **entry-level** options on their free tier, to **enterprise-level**, scalable, secure, configurable platforms with reliable availability for varying traffic levels, including many different tools and services that can be added as needed.

AWS continually leads the market in terms of **innovation** and a focus on **security** and **performance**, focused on the most common needs of fullstack developers and devops specialists. Its focus on **scalability** ensures businesses can start small and grow seamlessly without re-architecting their applications.

Major companies like Netflix, Airbnb, and Samsung use AWS for their critical workloads, as do thousands of others. Familiarity with its major services adds credibility and employability to any developer.



Other Deployment options

Google Cloud Run
Simplified container deployment with automatic scaling and pay-per-use.

Heroku
Developer-friendly PaaS with Git-based deployment, ideal for small-to-medium apps.

Microsoft Azure App Service
Flexible PaaS with staging environments and scaling.

DigitalOcean App Platform
Affordable, easy-to-use platform for web apps and APIs.

Render
Managed platform with simple pricing and automatic scaling.

Vercel
Optimized for frontend frameworks (e.g., Next.js) with serverless functions.

Netlify
Focused on static and JAMstack apps, with serverless backend options.

Kubernetes
Complex, high-end container management services which can be automated with additional services from AWS, Google or Azure.