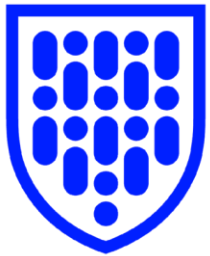


Institute of Data

2022

Software Engineering

Module 10 - Part 2

Deployment And Maintenance

Institute of Data

Agenda

Section 1 : Introduction to AWS EC2

Section 2 : Environment Variables

Section 3 : Deploying application in EC2

Section 4 : Beanstalk

Institute of Data

Section 1 : Introduction to AWS EC2

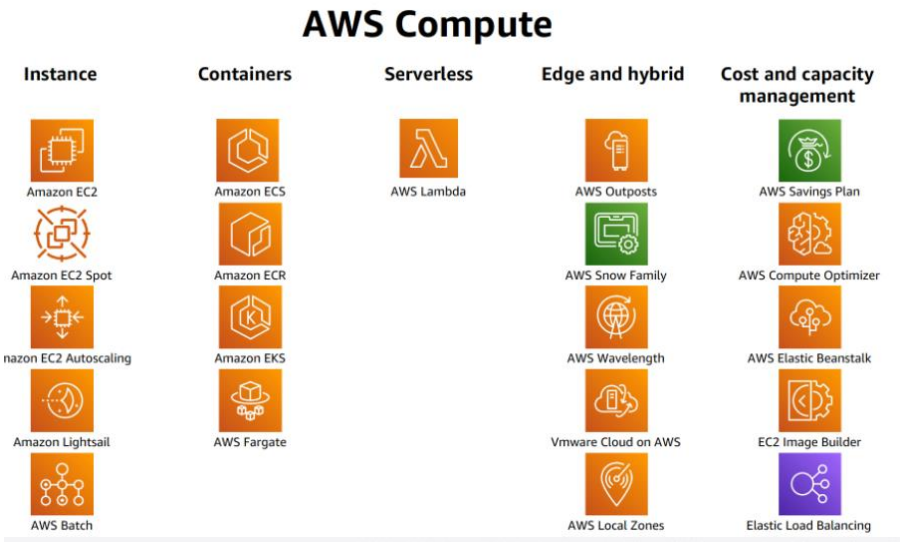
Amazon EC2 offers a range of instance types that are tailored to certain use cases. Instance types are different combinations of CPU, memory, storage, and networking capabilities that allow you to choose the best resource mix for your applications. Each instance type has one or more instance sizes, allowing 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 and workloads on the world's most powerful, secure, and innovative cloud.

AWS computing services have the following characteristics:

- Right compute for your workloads
- Accelerate from idea to market
- Offer built-in security
- Flexibility to optimise costs
- Provide compute resource where you need it

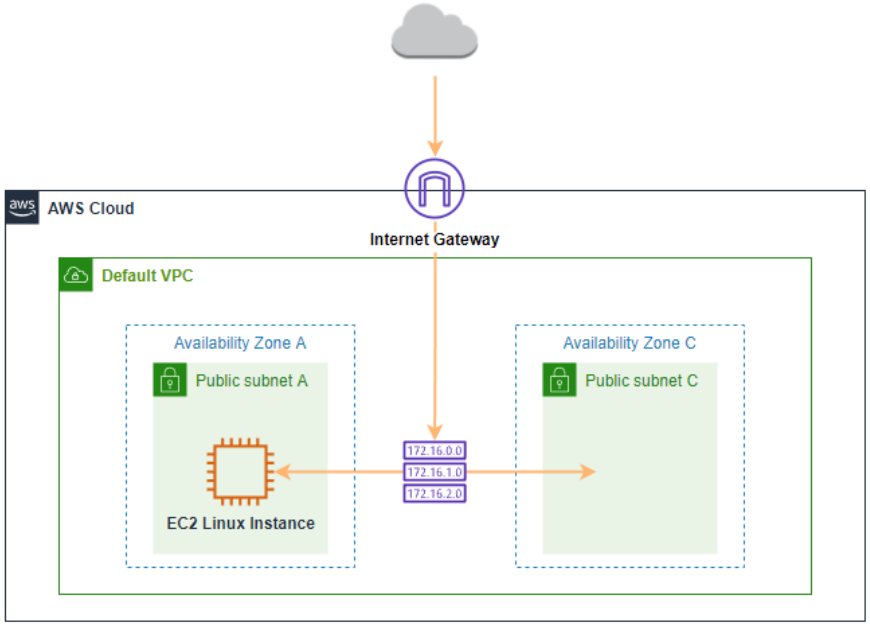


Overview

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. Amazon 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 Amazon EC2 to manage variations in demand or popularity spikes, decreasing the need to forecast traffic.

Create your own web server by going through the labs in the order below:

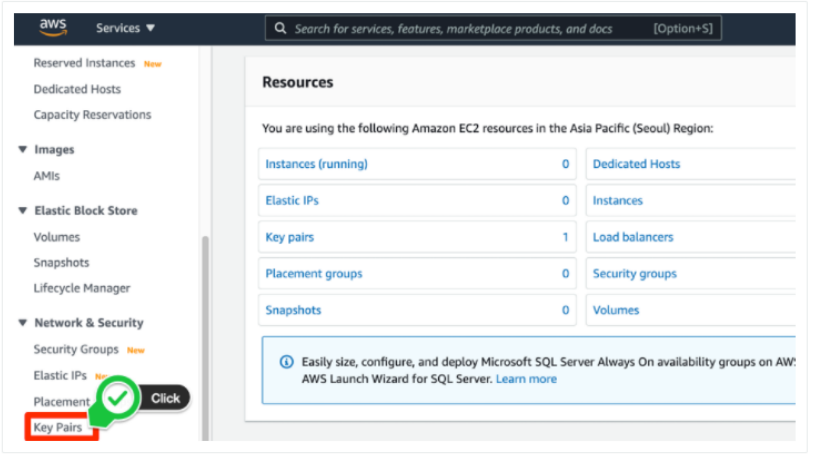
1. Create a new key pair
2. Launch a Web Server Instance
3. Connect to your linux instance
4. Connect to EC2 Instance using PuTTy (Optional)

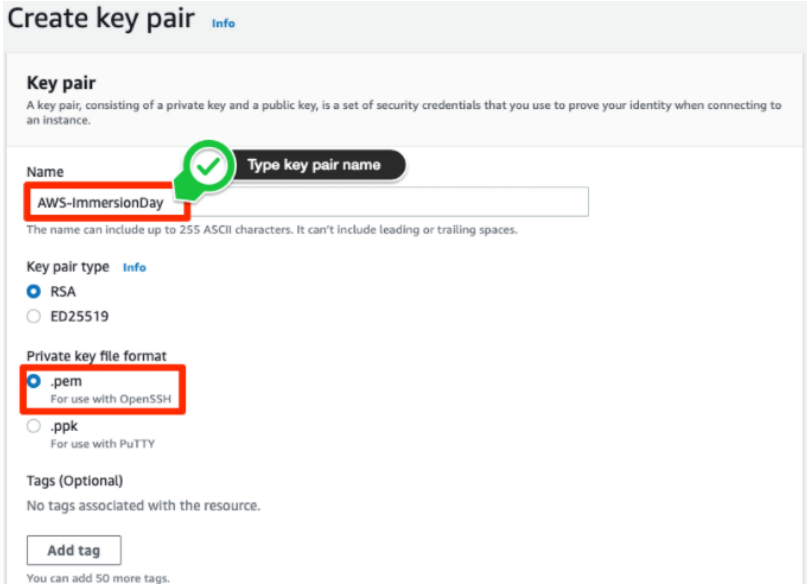


Create a key pair

In this lab, you will need to create an EC2 instance using an SSH keypair. The following steps outline creating a unique SSH keypair for you to use in this lab.

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 desired 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 users, please select **ppk** for file format.
5. The page will download the file **[Your Name]-ImmersionDay.pem** 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.

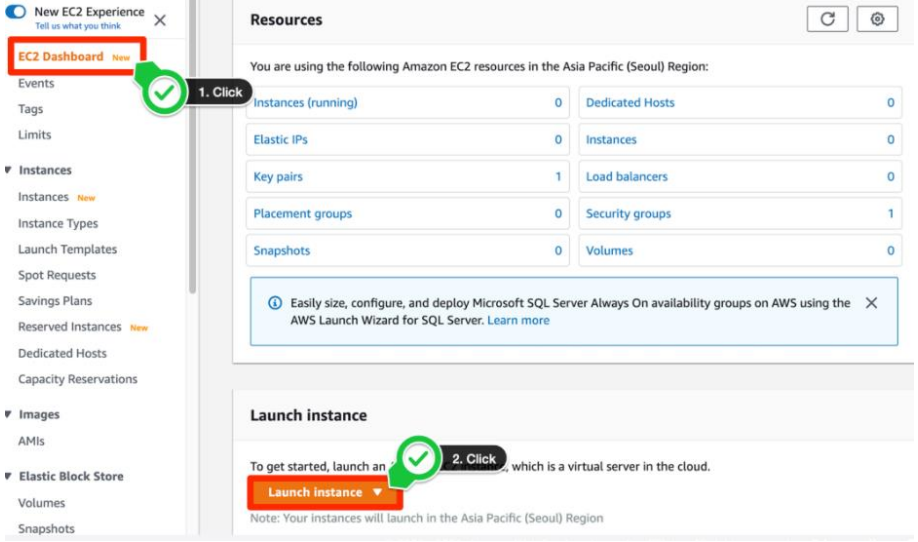




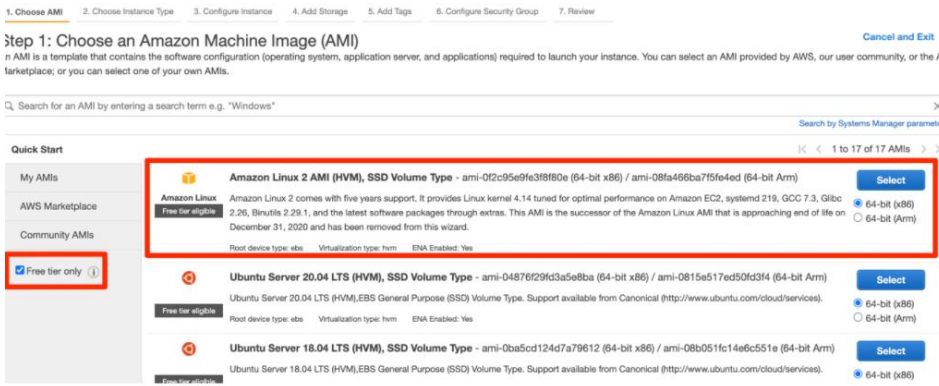
Launch a Web server

We will launch an Amazon Linux 2 instance

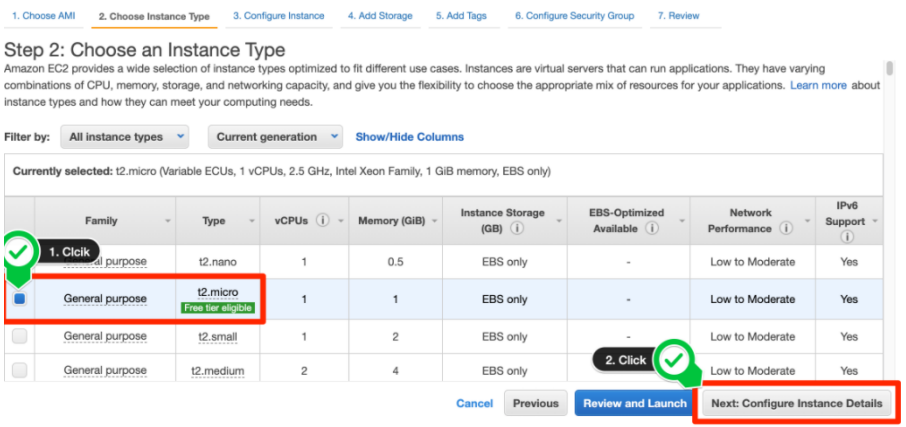
1. Click on **EC2 Dashboard** near the top of the leftmost menu. And Click on **Launch instances**.



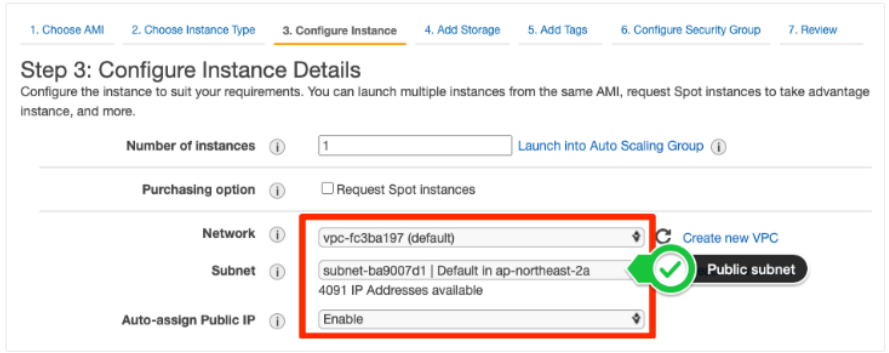
1. In the **Quick Start** section, click **Free tier only** and then select the first Amazon Linux 2 AMI for 64-bit (x86) architecture and click **Select**.



3. In Step 2, choose an *Instance Type*, select the **t2.micro** instance size and click **Next: Configure Instance Details**.

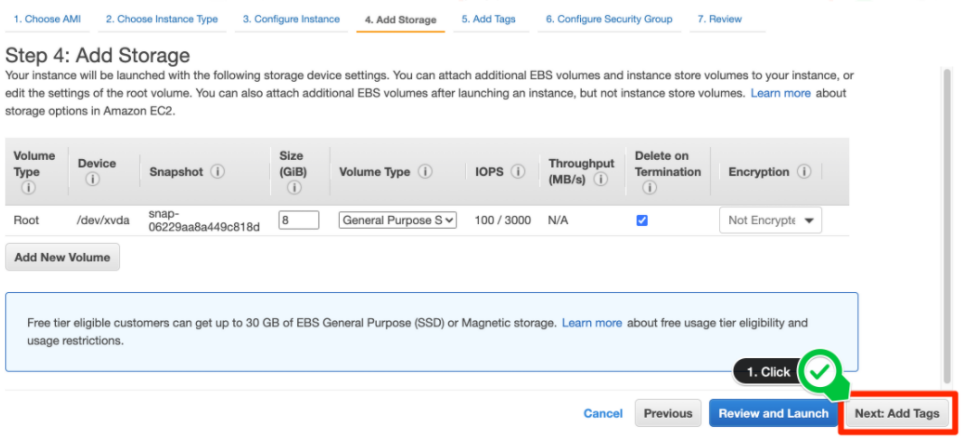


4. On Step 3, configure Instance Details page, We don't need to change anything in that as it is pre-set up for us. Click **Next: Add Storage**.

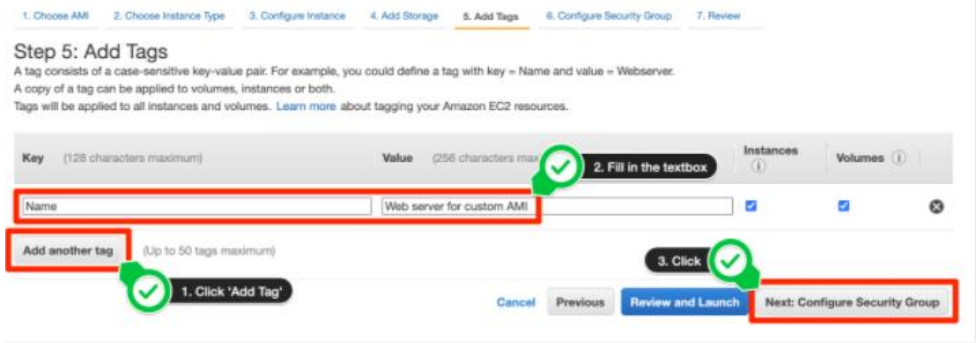


Storage and Tags

5. On this page you can modify or add storage and disk drives to the instance. For this lab, we will simply accept the storage defaults and click **Next: Add Tags**.

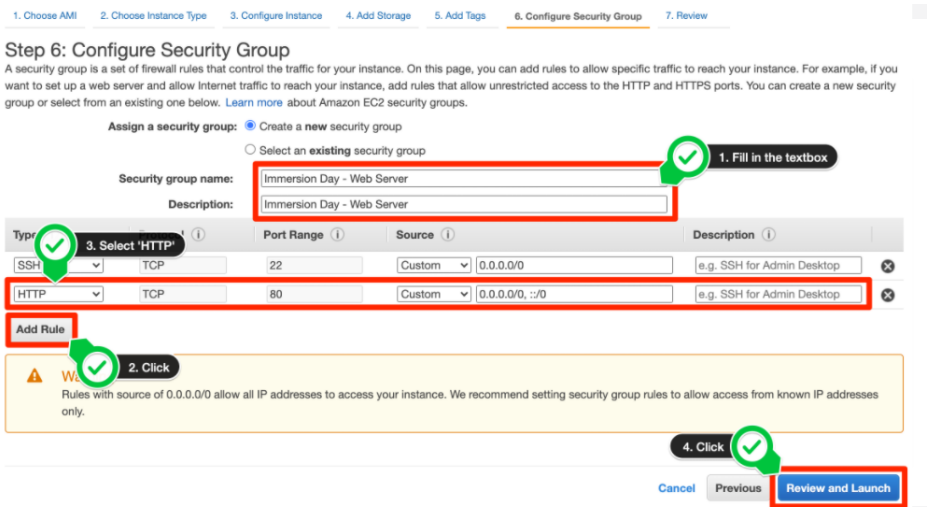


6. You can add a variety of information to identify instances. Tag information makes it easy for users to check the stage, purpose, and cost of the instance. Click Add Tag and type key, value shown as below. After finishing, click **Next: Configure Security Group**.

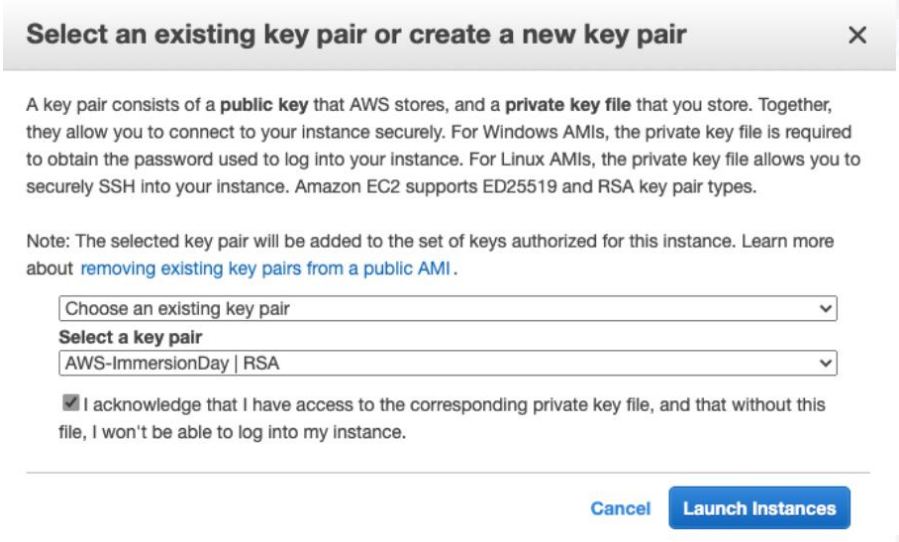


Security Groups

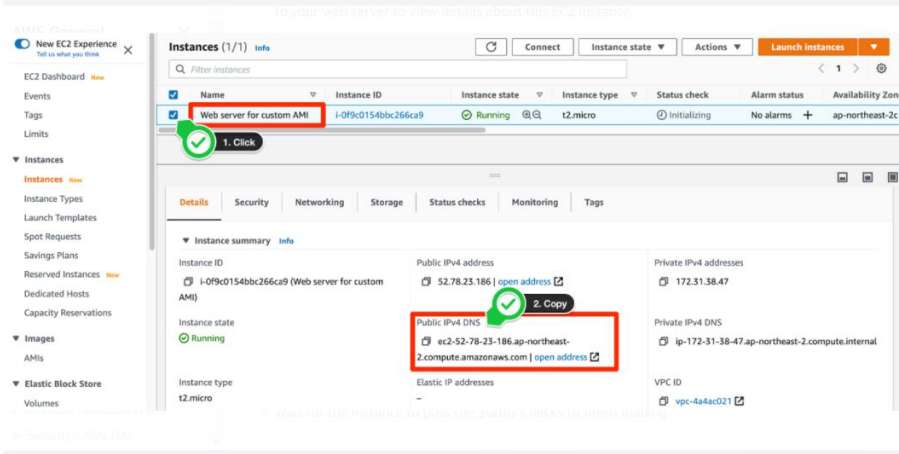
7. You can create a new security group or select a security group that already exists. Security group designates the protocols and addresses that you want to allow as firewall policies. Select **Create a new security group**. Enter `Immersion Day - Web Server` for the Security group name and Description, select the **Add Rule** button and specify **HTTP** under *Type* to allow TCP/80 for the Web Service. Under Source enter 0.0.0.0/0 to allow access from all networks. Click **Review and Launch** located in the lower right corner.
8. Review your configuration and choices, and then click **Launch**.



9. Select the key pair that was created in the beginning of this lab from the drop-down and check the **I acknowledge** checkbox. Then click the **Launch Instances** button. Your instance will now be starting, which may take a moment.

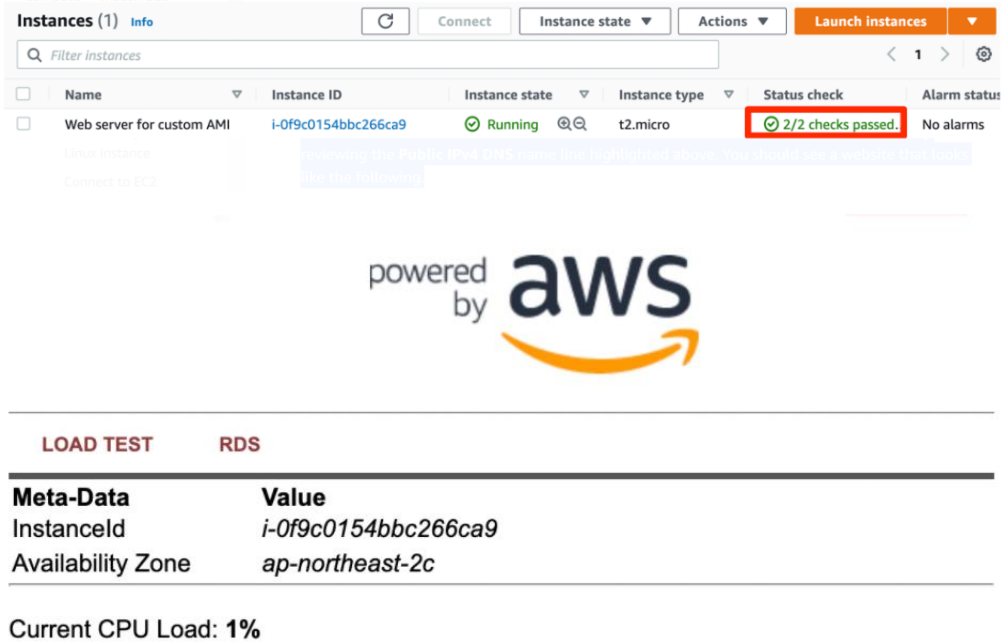


10. Click the **View Instances** button in the lower right hand portion of the screen to view the list of EC2 instances. Once your instance has launched, you will see your Web Server as well as the Availability Zone the instance is in, and the publicly routable **DNS name**. Click the checkbox next to your web server to view details about this EC2 instance.



View your website live

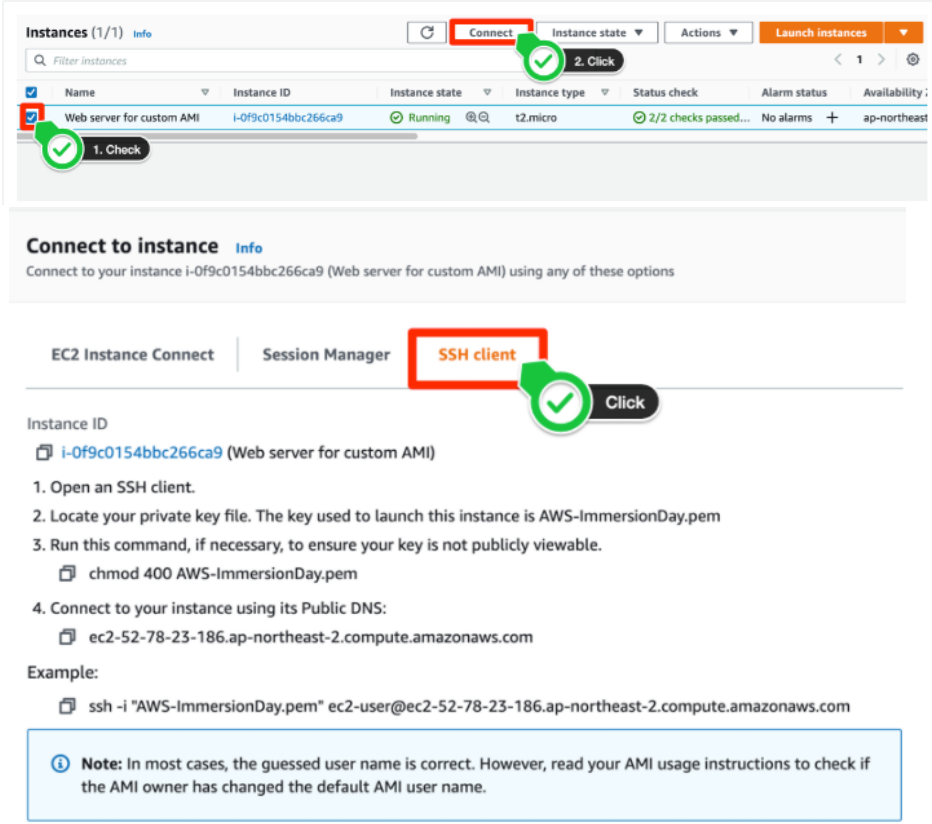
1. Wait for the instance to pass the Status Checks to finish loading.



2. 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.

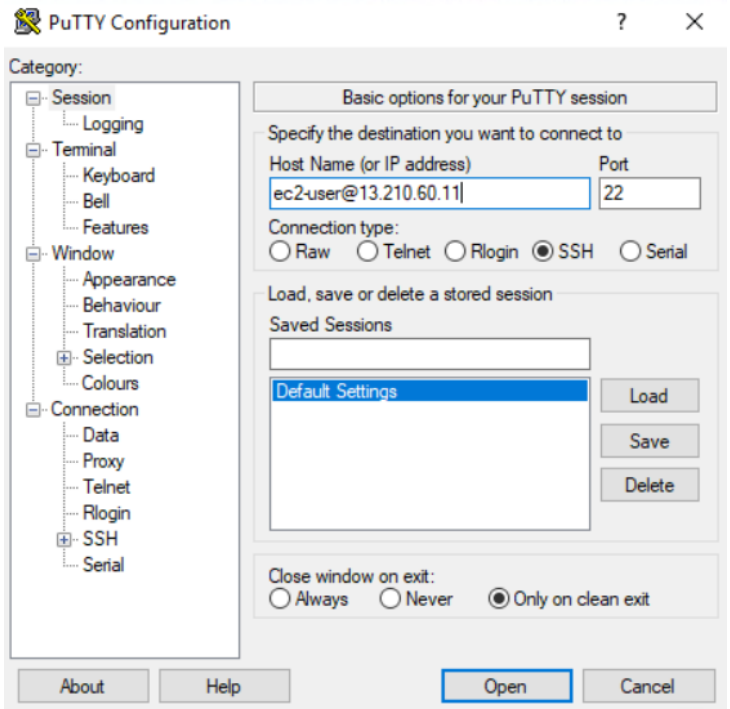
Connect to your Linux instance

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)



Connect using PuTTy (Windows)

1. Start PuTTy if you need to download PuTTY).
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 and choose **Open**.
7. 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. Remember to create only a free tier server so that AWS does not charge you anything for your instance.

Section 2: Environment Variables

The majority of programmers believe environment variables to be key-value pairs supplied to a specific programme.

Both keys and values are always character sequences for each pair. You can also refer to them as strings. These pairs would be passed to a server application in backend projects.

However, front-end projects are not as simple. Browsers do not allow environment variables because they execute code. Additional libraries (plugins) are typically used by developers to replace the use of these variables with predefined constants. This bond usually occurs during the development of a project.

Environment Variables

The purpose of using environment variables is to keep setup and code separate. Modern products use the Infrastructure-as-Code model to manage infrastructure and specify variables as inextricable parts of that model. Why is it preferable to storing project configuration in JSON files?

For starters, the JSON format supports nested items. We try to keep the settings as simple as possible; nested information is not necessary. Second, we don't want to save credentials in project repositories directly. Finally, we want complete control over the settings we send 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 package for reading environment variables.

Firstly we need to install the package. For doing so we run the command

```
$ npm install dotenv --save
```

In order to use dotenv, simply add this code to the top of your JS file and you'll be able to use environment variables.

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

Dotenv NPM

To set these environment variables just create a .env file in the root directory of your project and add your secret tokens like this.

You can see here we added two environment variables and we utilised them in the server.js file and the dbConnect.js file of our application by using the dotenv npm, and on running the application we were able to use them in our respective files.

.env

```
uri='mongodb+srv://admin:VzC4xlHgCC2Fy1D6@cloudbootcamp.bv4zn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority'
PORT=8000
```

server.js > ...

```
You, 11 seconds ago | 1 author (You)
require('dotenv').config()      You, 11 seconds ago • Uncommitted changes
let express = require("express");
let app = express();
let dbConnect = require("./dbConnect");

//dbConnect.dbConnect()
//var app = require('express')();
let http = require('http').createServer(app);
let io = require('socket.io')(http);
//const MongoClient = require('mongodb').MongoClient;

// routes
// let projectsRoute = require('./routes/projects')


var port = process.env.PORT || 8080;
app.use(express.json());
app.use(express.static(__dirname + '/public'));
let userRoute = require('./routes/userRoute')
app.use('/api/users',userRoute)
```

dbConnect.js > mongooseOptions

```
require('dotenv').config()
var Mongoose = require('mongoose');

const uri = process.env.uri || "mongodb://localhost/myFirstDatabase";

const mongooseOptions = {
  useNewUrlParser: true,      You, last month • Initial Commit …
  useUnifiedTopology: true
};
Mongoose.set('useCreateIndex', true);
Mongoose.set('useFindAndModify', false);


//Connect to MongoDB
Mongoose.connect(uri, mongooseOptions, function (err) {
    if (err) {
        console.log("DB Error: ", err);
        process.exit(1);
    } else {
        console.log('MongoDB Connected: ',process.env.uri);
    }
});
```

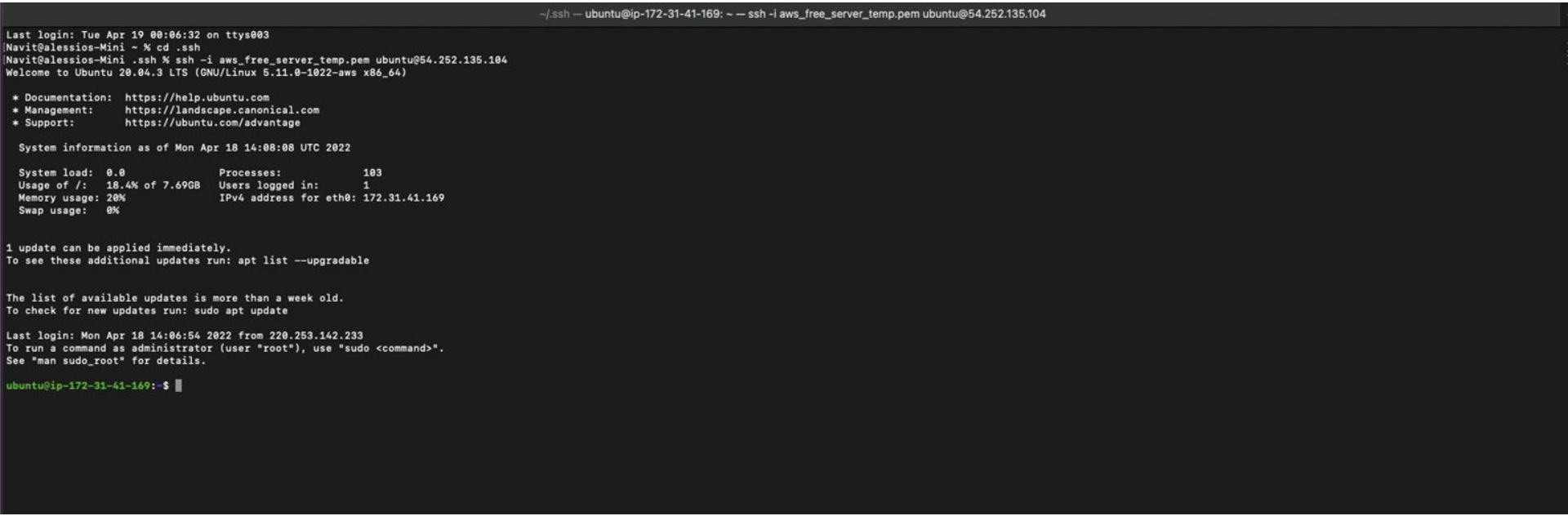
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS: VISUAL FILE HISTORY

```
Navit@alessios-Mini mvc-structure % npm start

> mvc-structure@0.0.0 start
> node server.js

Listening on port  8000
MongoDB Connected:  mongodb+srv://admin:VzC4xlHgCC2Fy1D6@cloudbootcamp.bv4zn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

Section 3: Deploying application in EC2



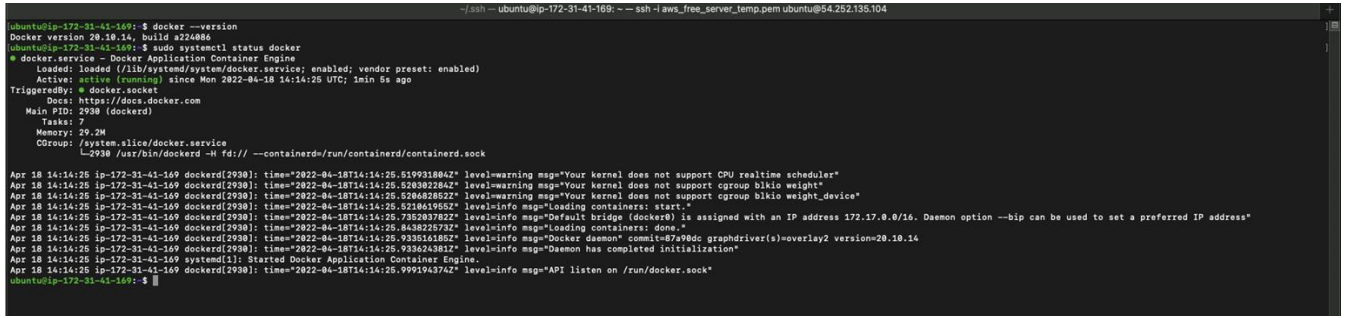
Deploying application in EC2

Next what we need to do is to install Docker on our EC2 instance, because we have an ubuntu server we just need to use the commands from here to install Docker to our server. Once we have Docker installed on our server, we can check if it was successfully installed by using the command

```
$ docker --version
```

And 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 all we need to do is to use the Docker image of our application that we created previously and hosted on Docker hub. First we need to pull the docker image on the EC2 instance.

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

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

Deploying application in EC2

So now we successfully cloned the image on our EC2 server, all we now need to do is to run the docker image

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

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

And now we have our application successfully running on our EC2 instance. You can also see the current application running here.

Exercise 2

Try hosting the docker image of your nodejs application that you must have created using the CI/CD pipeline that you created previously. 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 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 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:

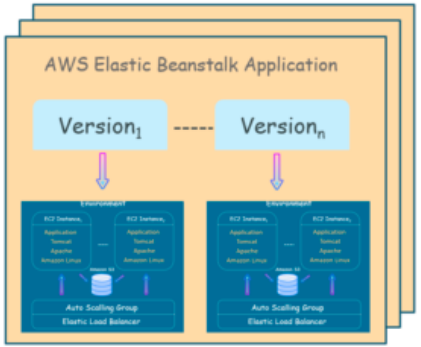
- The current version of the Elastic Beanstalk Application will be active in environments within the Elastic Beanstalk Application.
- At any one time, each environment only runs one application version. However, the same or different versions of an application can be operated in many settings at the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

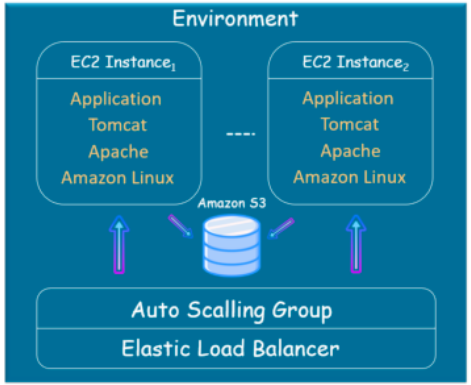
Elastic Beanstalk Components

Environment Tier:

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

- Web Server Environment: Handles HTTP requests from clients
- Worker Environment: Processes background tasks which are resource consuming and time intensive





Setting up Elastic Beanstalk Server

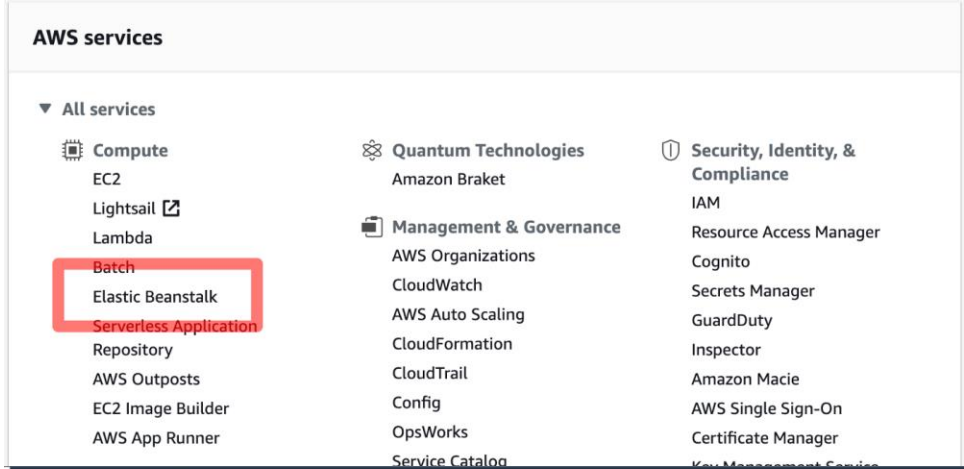
1. Our first step is to setup Elastic Beanstalk server on Amazon AWS. Login to your AWS Management Console and click on “Elastic Beanstalk” under services

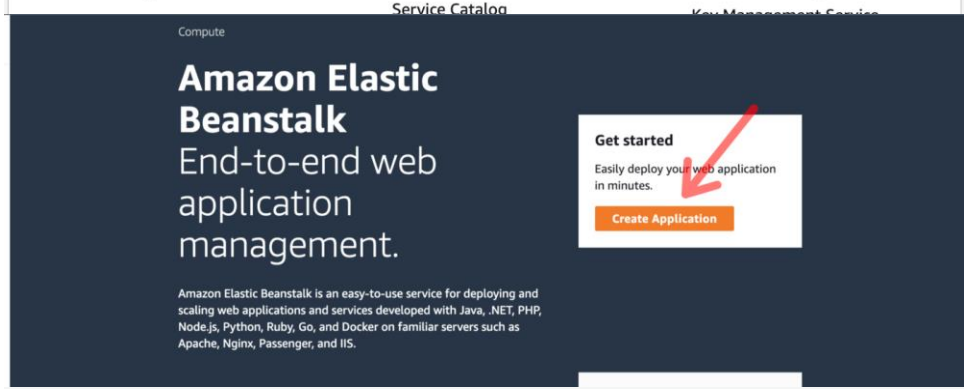
1. Next step is to create create an Amazon Elastic Beanstalk application. Click on the “Create Application” button

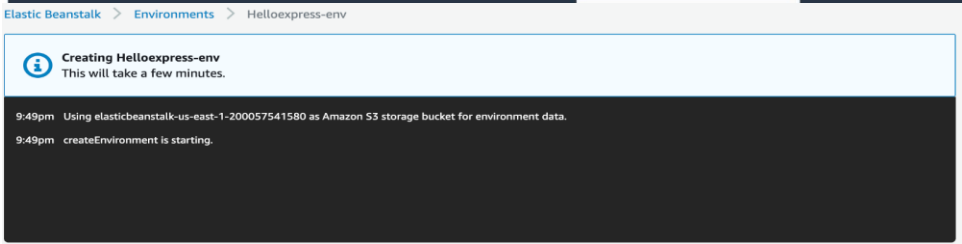
In the create application page fill out the details as shown below:

Application Name: Hello-Express
Platform: Node.js
Platform branch: Node.js 14 running on 64bit Amazon Linux 2
Platform version: 5.4.3 (Recommended)
Application Code: Sample Code

Finally, click on the “Create application” button. This will take few minutes to process and setup the application.





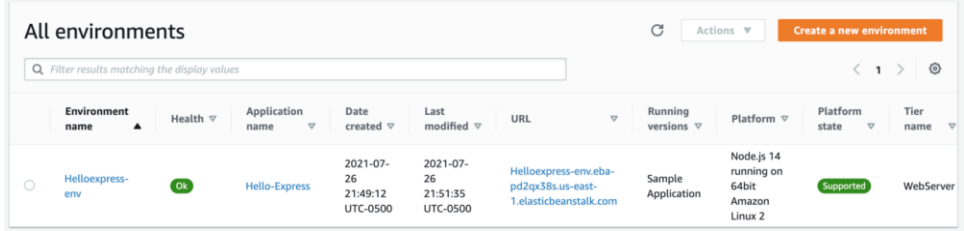


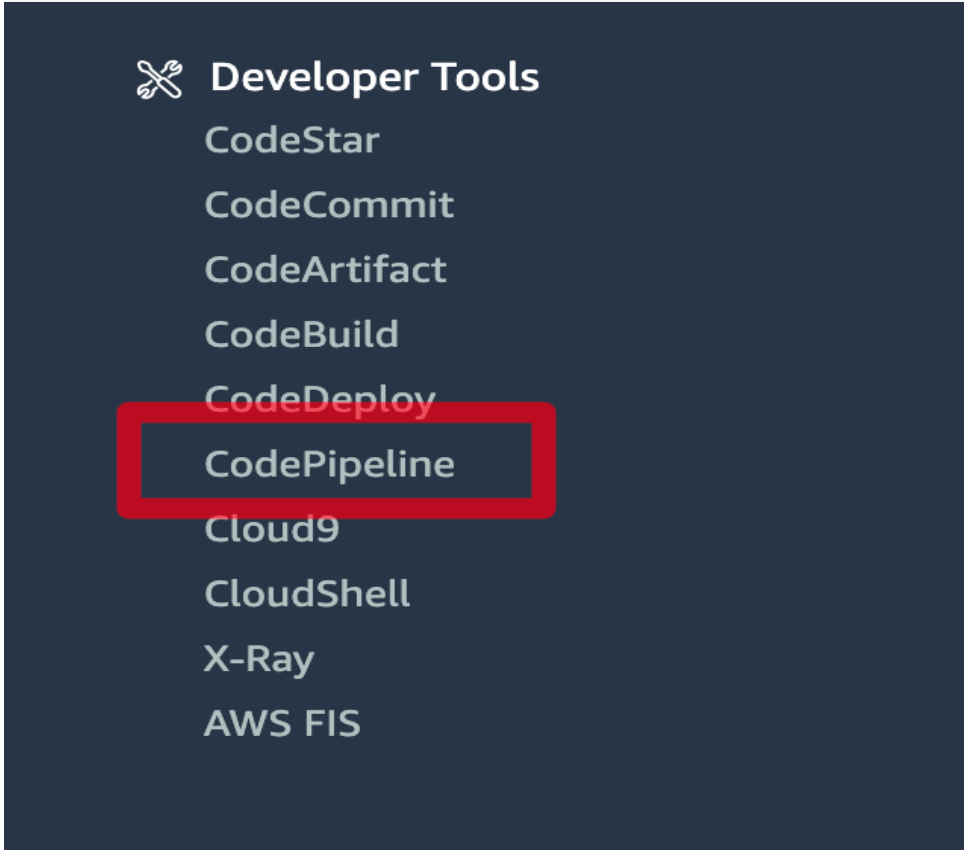
Setting up Elastic Beanstalk Server

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 deployed to the AWS Elastic Beanstalk, we must setup a Pipeline on AWS management console. Click on “All services” and then click on the “CodePipeline” under “Developer Tools”



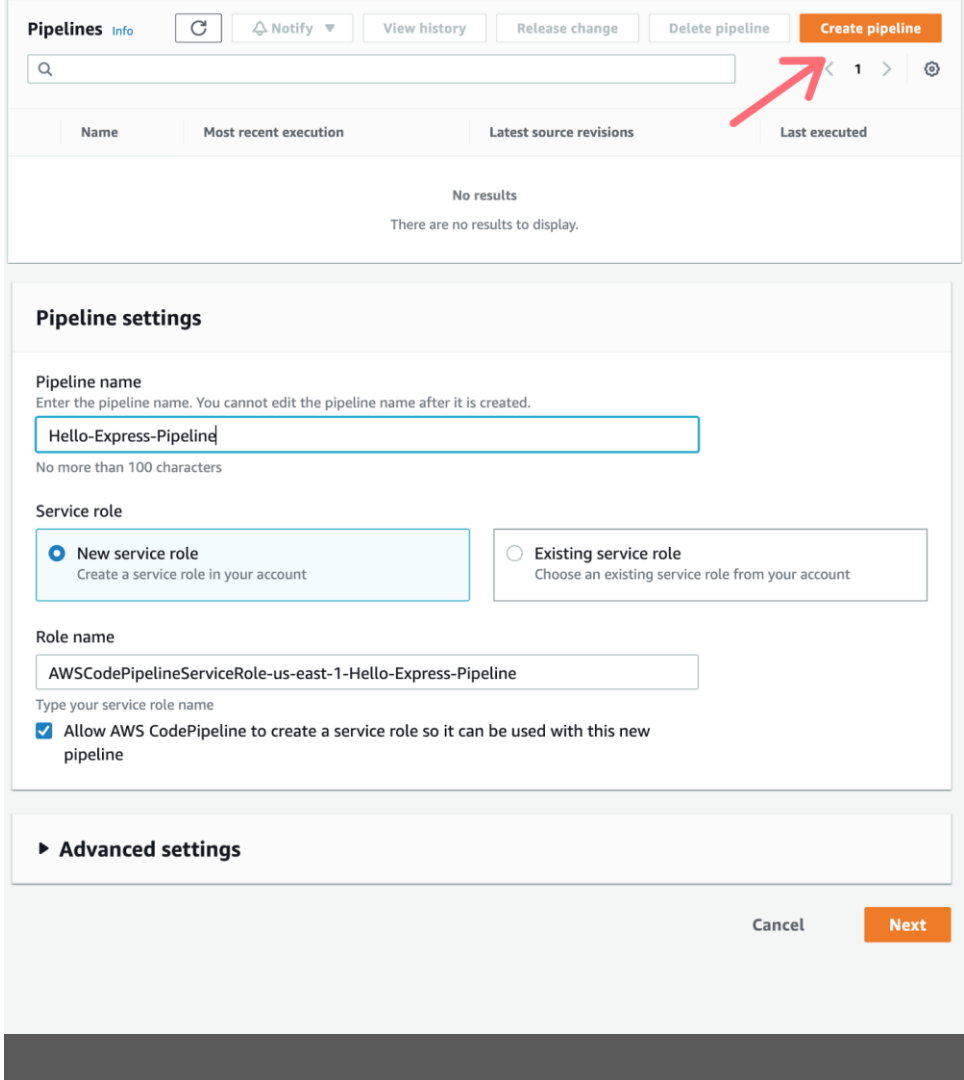


Setting up Elastic Beanstalk Server

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.

Once you fill out the “Pipeline name”, it will automatically fill out the Role name. Click “Next” to continue.

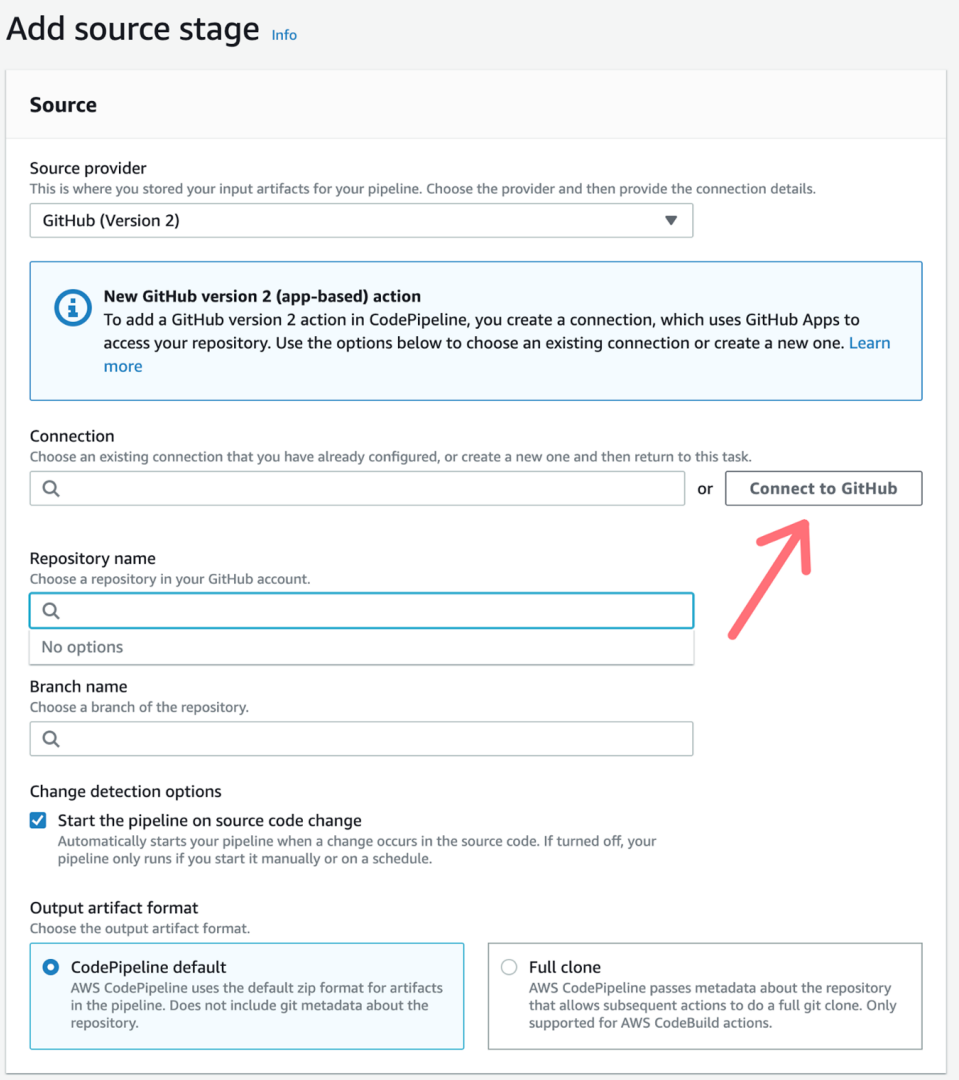


Setting up Elastic Beanstalk Server

In the next screen, you will choose the source stage. Since, we are using GitHub as our source repository we will choose GitHub (Version 2).

GitHub (Version 1) is no longer recommended as it does not use the updated authentication methods

When you select GitHub (Version 2) an additional form will show up which will allow you to integrate your GitHub repository to the pipeline. Click on “Connect to GitHub” to start the process.



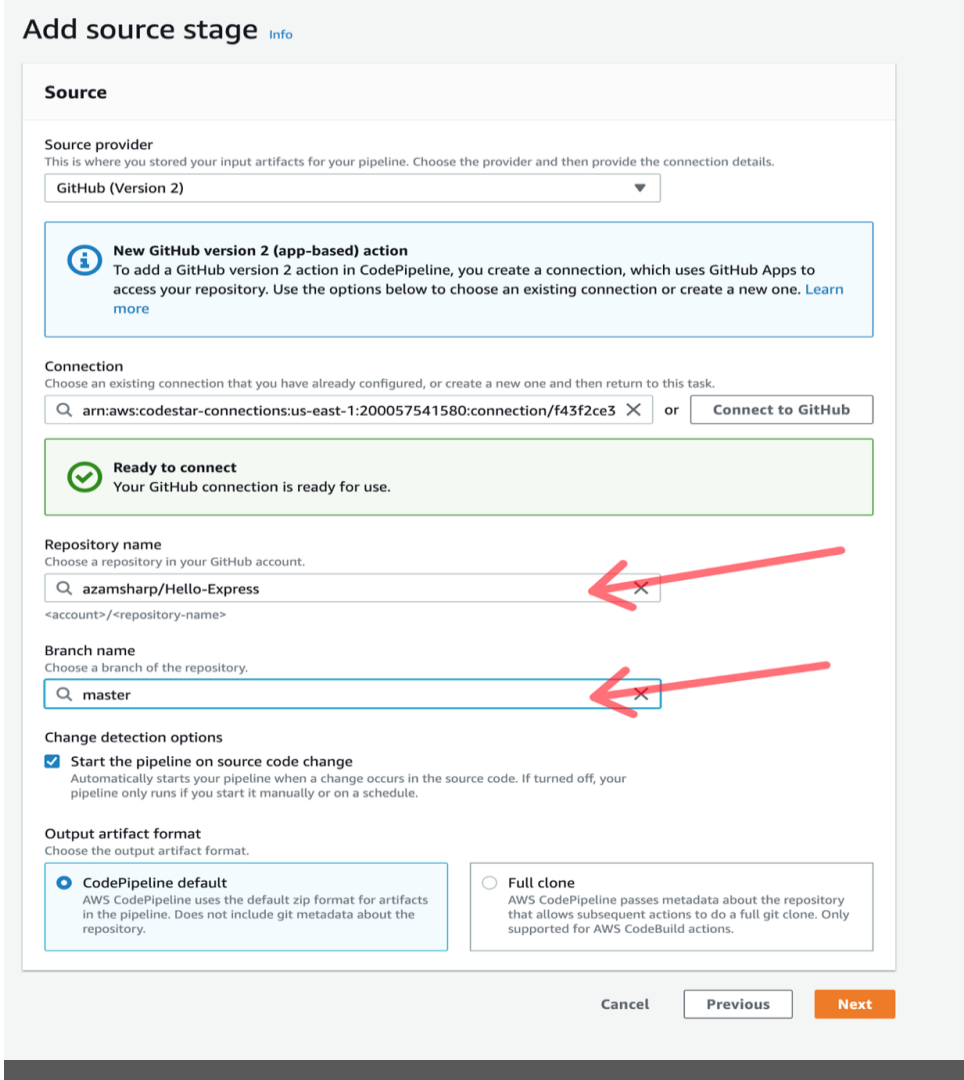
Setting up Elastic Beanstalk Server

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.

You will have to enter your GitHub credentials to create a connection between GitHub and Code pipeline. Once the connection has been made, it will allow you to pick your GitHub repository and the branch

We have selected “Hello-Express” repository, which was created earlier and the master branch of the repository because we want to deploy the code in the master branch to the server.

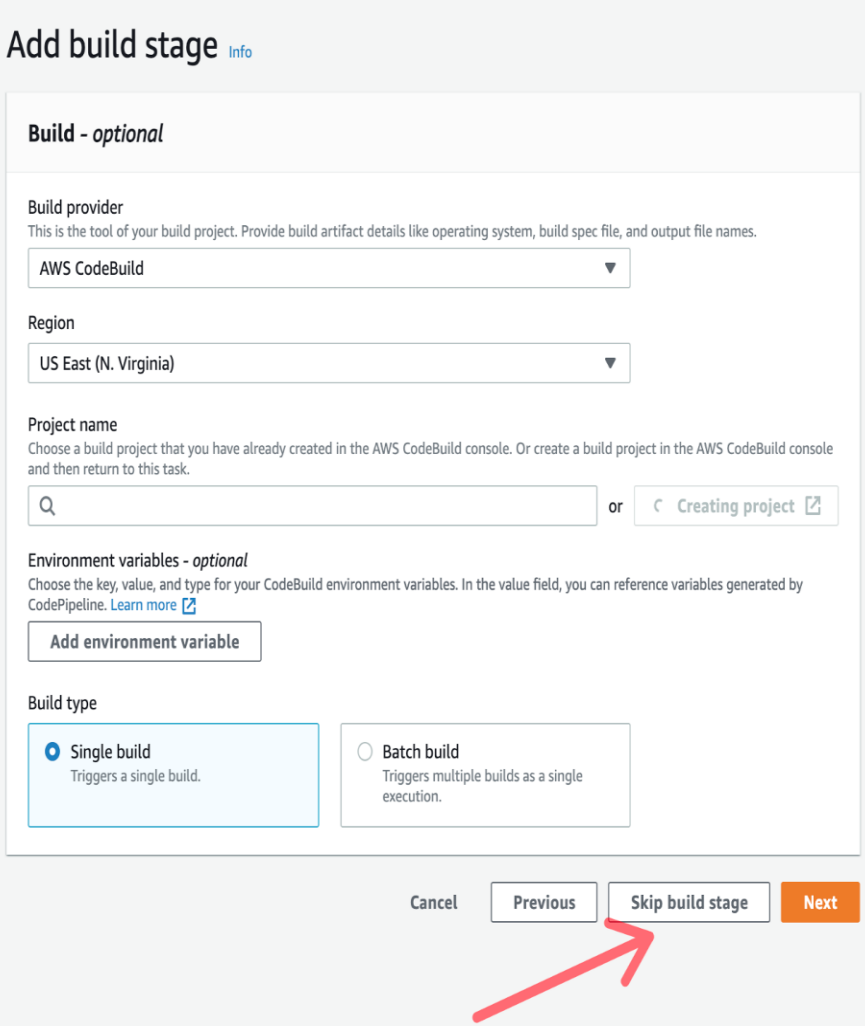
Click next.



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”

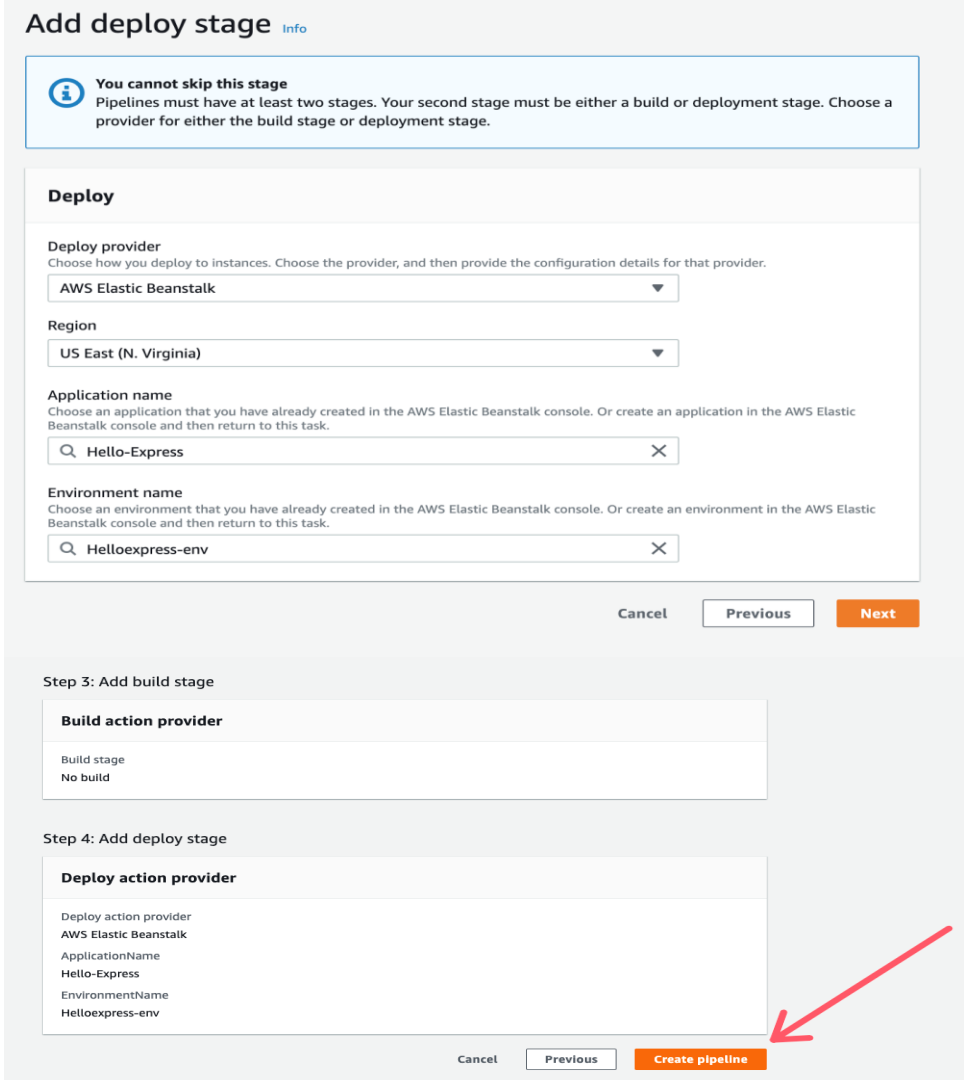


Setting up Elastic Beanstalk Server

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

Click next.

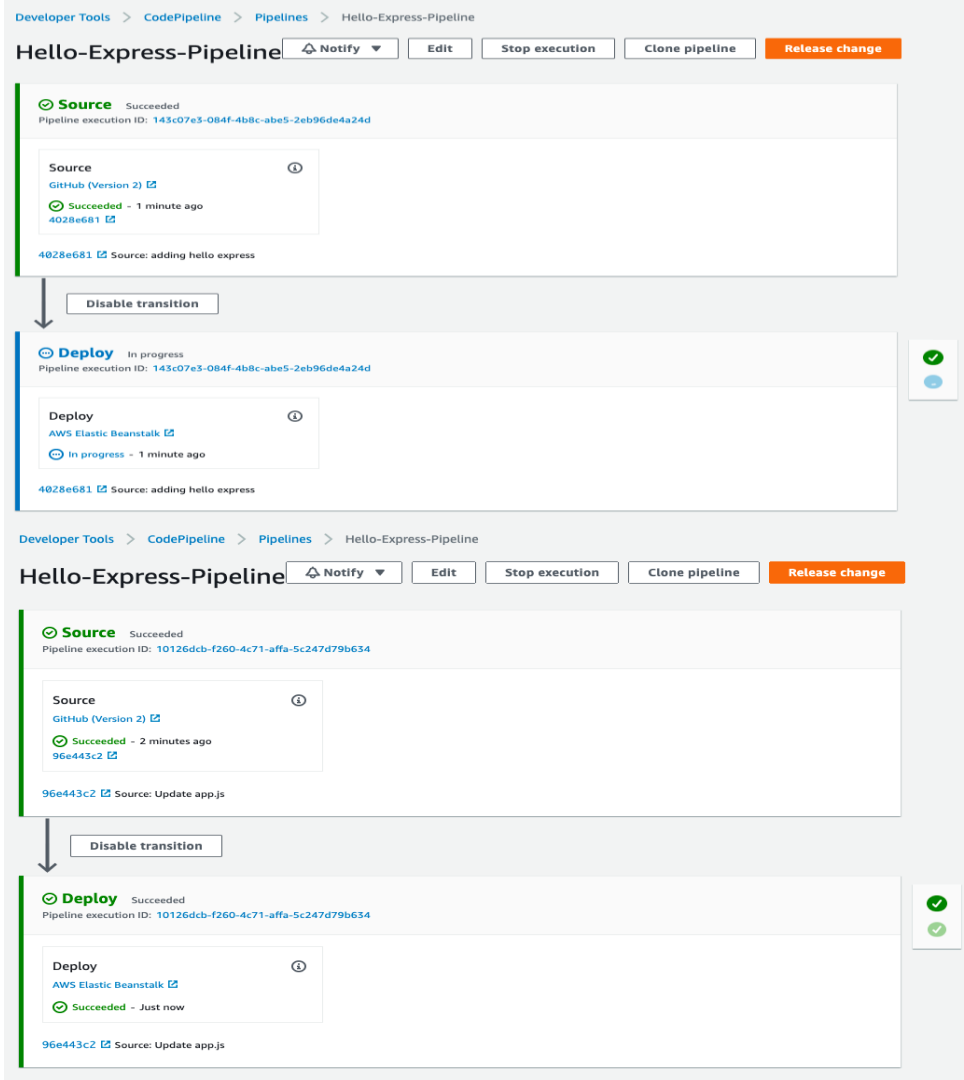
The next screen will be the review screen. Make sure all settings and configurations are correct. Scroll down at the bottom and click “Create pipeline” button.



Setting up Elastic Beanstalk Server

As soon as the pipeline is setup, it will try to deploy the app to Amazon AWS Elastic Beanstalk.

After a while it comes back with deployment succeeded status.



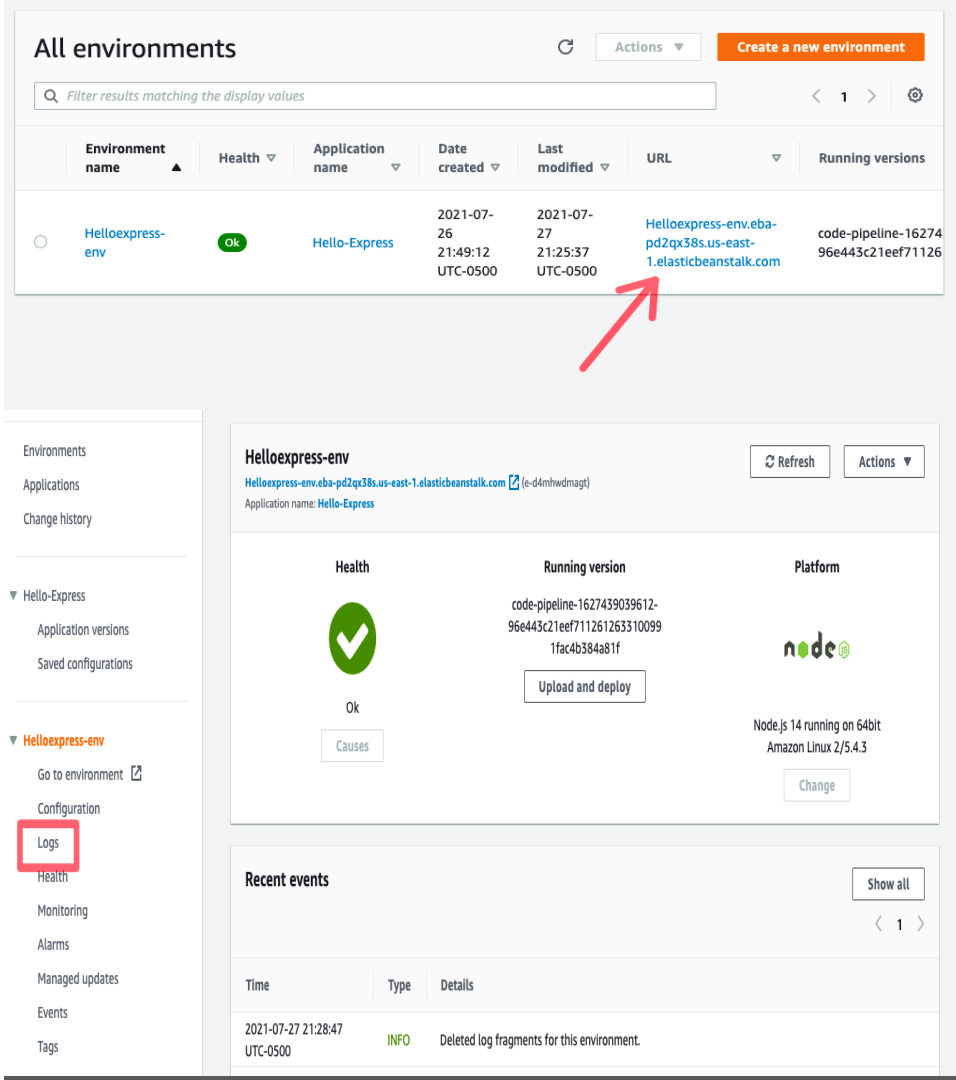
Setting up Elastic Beanstalk Server

Now we need to see if we can access our routes or not. Click on “All services” and then select Elastic Beanstalk

Next click on the URL of your Hello-Express application. This will launch the root endpoint.

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 the nodejs application that you must have created using AWS Elastic Beanstalk. Share the link of your hosted application with your trainer.

Other Deployment options

We can also look into a few other deployment options for deploying our application on the cloud. Some other popular options would be

- IBM Cloud Engine (Paid Service)
- Kubernetes (Paid and High difficulty level)

End of Presentation