



**Institute<sub>of</sub>  
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

## AWS Compute



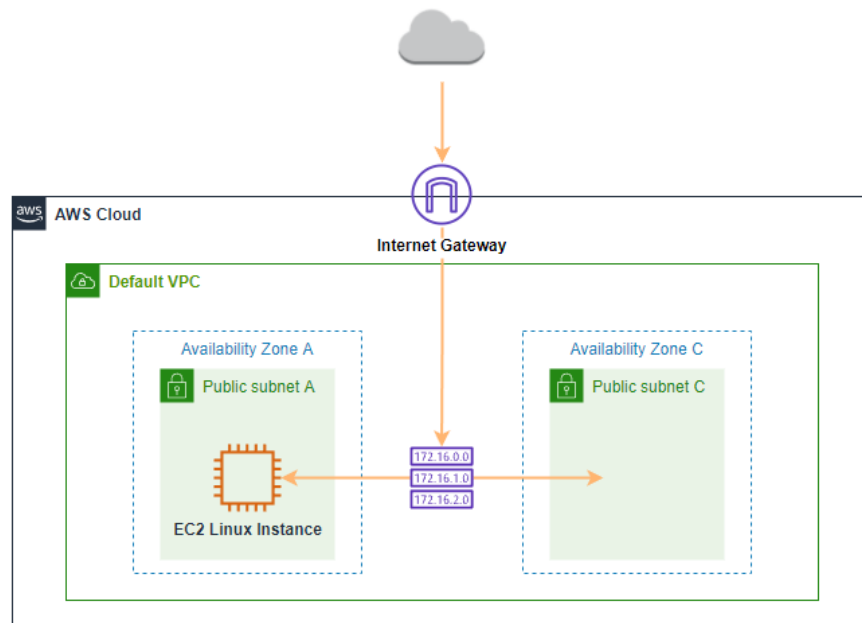


# Overview

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 \(Optional\)](#)

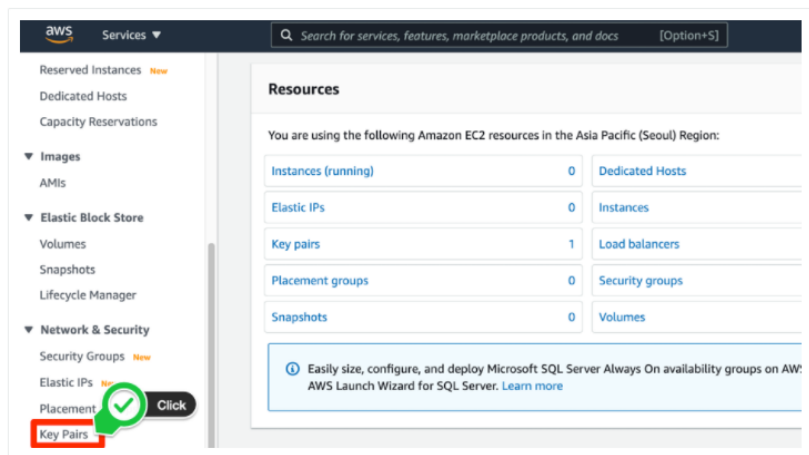




# Create a key pair

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.



## Create key pair

### Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

**AWS-ImmersionDay**

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA

☐ ED25519

Private key file format

☒ .pem

For use with OpenSSH

☐ .ppk

For use with PuTTY

Tags (Optional)

No tags associated with the resource.

Add tag

You can add 50 more tags.



# 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**.

The screenshot shows the AWS Management Console. On the left, the 'EC2 Dashboard' link is highlighted with a red box and a green checkmark, with a '1. Click' callout. Below it, the 'Launch instances' link is also highlighted with a red box and a green checkmark, with a '2. Click' callout. The main content area shows the 'Resources' section for the Asia Pacific (Seoul) Region, listing various EC2 resources like Instances, Elastic IPs, Key pairs, etc. Below this, the 'Launch instance' section is visible, with a 'Launch instance' button highlighted by a red box and a green checkmark.

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**.

The screenshot shows the 'Step 1: Choose an Amazon Machine Image (AMI)' screen. The 'Quick Start' section is visible, with the 'Free tier only' filter selected and highlighted by a red box. Below this, the 'Amazon Linux 2 AMI (HVM), SSD Volume Type' is highlighted with a red box and a green checkmark, with a 'Select' button next to it. The 'Ubuntu Server 20.04 LTS (HVM), SSD Volume Type' and 'Ubuntu Server 18.04 LTS (HVM), SSD Volume Type' are also visible, each with a 'Select' button.





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**.

## Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
1. Click	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
	General purpose	t2.micro	1	1	EBS only	-	Low to Moderate	Yes
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Security - 2023 IAM

1. If it isn't listed, you may need to choose

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

## Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage instance, and more.

Number of instances 1 Launch into Auto Scaling Group

Purchasing option Request Spot instances

Network vpc-fc3ba197 (default) Create new VPC

Subnet subnet-ba9007d1 | Default in ap-northeast-2a 4091 IP Addresses available Public subnet

Auto-assign Public IP Enable



# 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**.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

### Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type (1)	Device (1)	Snapshot (1)	Size (GiB) (1)	Volume Type (1)	IOPS (1)	Throughput (MB/s) (1)	Delete on Termination (1)	Encryption (1)
Root	/dev/xvda	snap-06229aa8a449c818d	8	General Purpose S	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

[Add New Volume](#)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

1. Click [Next: Add Tags](#)

Cancel Previous [Review and Launch](#)

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

### Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key (128 characters maximum)	Value (256 characters maximum)	Instances (1)	Volumes (1)
Name	Web server for custom AMI	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

[Add another tag](#) (Up to 50 tags maximum)

1. Click 'Add Tag' 2. Fill in the textbox 3. Click [Next: Configure Security Group](#)

Cancel Previous [Review and Launch](#)



# Security Groups

- 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.
- Review your configuration and choices, and then click **Launch**.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

### Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

1. Fill in the textbox

Security group name:

Description:

Type	Port Range	Source	Description	
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

Add Rule

2. Click

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

4. Click

Cancel Previous **Review and 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.

## Select an existing key pair or create a new key pair



A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance. Amazon EC2 supports ED25519 and RSA key pair types.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Choose an existing key pair

Select a key pair

AWS-ImmersionDay | RSA

☒ I acknowledge that I have access to the corresponding private key file, and that without this file, I won't be able to log into my instance.

Cancel

Launch Instances

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Web server for custom AMI	i-09c0154bcb266ca9	Running	t2.micro	Initializing	No alarms	ap-northeast-2

1. Click

Details

Instance summary

Instance ID: i-09c0154bcb266ca9 (Web server for custom AMI)

Instance state: Running

Instance type: t2.micro

Public IPv4 address: 52.78.23.186 | open address

Public IPv4 DNS: ec2-52-78-23-186.ap-northeast-2.compute.amazonaws.com | open address

Private IPv4 addresses: 172.31.38.47

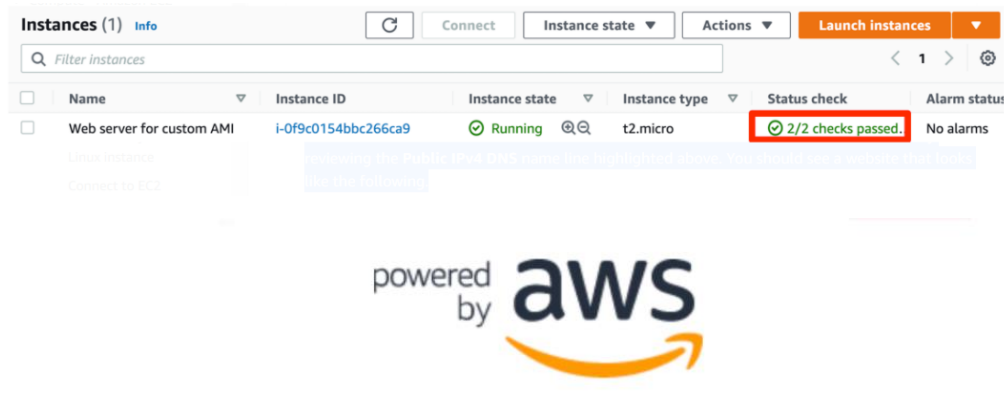
Private IPv4 DNS: ip-172-31-38-47.ap-northeast-2.compute.internal

VPC ID: vpc-4a4ec021



# 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.



The screenshot shows the AWS Management Console 'Instances' page. A table lists one instance: 'Web server for custom AMI' with ID 'i-0f9c0154bbc266ca9', state 'Running', and type 't2.micro'. The 'Status check' column shows '2/2 checks passed' with a green checkmark icon. Below the table, the 'Public IPv4 DNS' name is highlighted in blue. The AWS logo is visible below the console.

LOAD TEST	RDS
<b>Meta-Data</b>	<b>Value</b>
Instanceld	i-0f9c0154bbc266ca9
Availability Zone	ap-northeast-2c

Current CPU Load: 1%



# 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)

The screenshot shows the AWS Management Console interface for connecting to an EC2 instance. At the top, the 'Instances (1/1)' page is visible, with a table listing the instance 'Web server for custom AMI' (ID: i-0f9c0154bbc266ca9) in a 'Running' state. A red box highlights the 'Connect' button, and a green checkmark with the text '2. Click' indicates this step. Below, the 'Connect to instance' page is shown, with tabs for 'EC2 Instance Connect', 'Session Manager', and 'SSH client'. The 'SSH client' tab is selected and highlighted with a red box. A green checkmark with the text '1. Check' is next to it. The page provides instructions for connecting via SSH, including the instance ID, a list of steps (opening an SSH client, locating the private key, running a command to ensure the key is not publicly viewable, and connecting using the Public DNS), and an example SSH command. A 'Note' at the bottom states: 'In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.'

**Instances (1/1)** Info

Filter instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
Web server for custom AMI	i-0f9c0154bbc266ca9	Running	t2.micro	2/2 checks passed...	No alarms	ap-northeast-2

**Connect to instance** Info

Connect to your instance i-0f9c0154bbc266ca9 (Web server for custom AMI) using any of these options

EC2 Instance Connect | Session Manager | **SSH client**

Instance ID  
i-0f9c0154bbc266ca9 (Web server for custom AMI)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is AWS-ImmersionDay.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.  
chmod 400 AWS-ImmersionDay.pem
4. Connect to your instance using its Public DNS:  
ec2-52-78-23-186.ap-northeast-2.compute.amazonaws.com

Example:

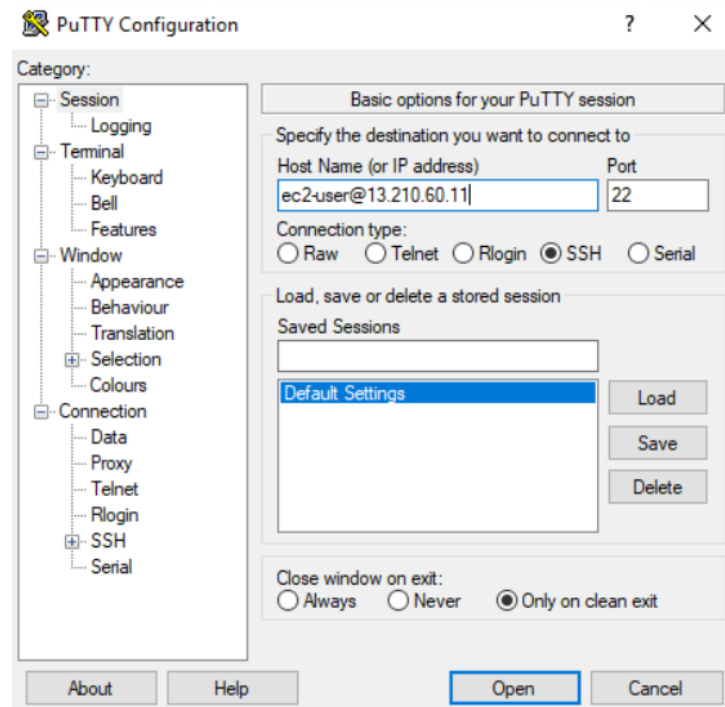
```
ssh -i "AWS-ImmersionDay.pem" ec2-user@ec2-52-78-23-186.ap-northeast-2.compute.amazonaws.com
```

**Note:** In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.



# 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
1 uri='mongodb+srv://admin:VzC4xHgCC2Fy1D6@cloudbootcamp.bv4zn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority'
2 PORT=8000
```

JS server.js > ...

```
You, 11 seconds ago | 1 author (You)
1 | require('dotenv').config() You, 11 seconds ago * Uncommitted changes
2 | let express = require("express");
3 | let app = express();
4 | let dbConnect = require("./dbConnect");
5 |
6 | //dbConnect.dbConnect()
7 | //var app = require('express')();
8 | let http = require('http').createServer(app);
9 | let io = require('socket.io')(http);
10 | //const MongoClient = require('mongodb').MongoClient;
11 |
12 | // routes
13 | // let projectsRoute = require('./routes/projects')
14 |
15 |
16 | var port = process.env.PORT || 8080;
17 | app.use(express.json());
18 | app.use(express.static(__dirname + '/public'));
19 | let userRoute = require('./routes/userRoute')
20 | app.use('/api/users',userRoute)
21 |
```

JS dbConnect.js > [e] mongooseOptions

```
6 | require('dotenv').config()
7 | var Mongoose = require('mongoose');
8 |
9 | const uri = process.env.uri || "mongodb://localhost/myFirstDatabase";
10 |
11 |
12 | const mongooseOptions = {
13 |   useNewUrlParser: true, You, last month * Initial Commit ...
14 |   useUnifiedTopology: true
15 | };
16 | Mongoose.set('useCreateIndex', true);
17 | Mongoose.set('useFindAndModify', false);
18 |
19 | //Connect to MongoDB
20 | Mongoose.connect(uri, mongooseOptions, function (err) {
21 |   if (err) {
22 |     console.log("DB Error: ", err);
23 |     process.exit(1);
24 |   } else {
25 |     console.log('MongoDB Connected: ',process.env.uri);
26 |   }
27 | });
28 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS: VISUAL FILE HISTORY

Navit@lessios-Mini mvc-structure % npm start

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

```
Listening on port 8000
MongoDB Connected: mongodb+srv://admin:VzC4xHgCC2Fy1D6@cloudbootcamp.bv4zn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```



## Section 3: Deploying application in EC2

```
~/ssh -- ubuntu@ip-172-31-41-169: ~ -- ssh -i aws_free_server_temp.pem ubuntu@54.252.135.104

Last login: Tue Apr 19 00:06:32 on ttys003
Navit@alessios-Mini ~ % cd .ssh
Navit@alessios-Mini .ssh % ssh -i aws_free_server_temp.pem ubuntu@54.252.135.104
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1022-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Apr 18 14:08:08 UTC 2022

System load:  0.0          Processes:            103
Usage of /:   18.4% of 7.69GB Users logged in:          1
Memory usage: 20%         IPv4 address for eth0: 172.31.41.169
Swap usage:   0%

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Mon Apr 18 14:06:54 2022 from 220.253.142.233
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-41-169:~$
```



# 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
```

```
~/.ssh -- ubuntu@ip-172-31-41-169: ~ -- ssh -i aws_free_server_temp.pem ubuntu@54.252.135.104
ubuntu@ip-172-31-41-169:~$ docker --version
Docker version 20.10.14, build a224866
ubuntu@ip-172-31-41-169:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-04-18 14:14:26 UTC; 1min 5s ago
     TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
   Main PID: 2938 (dockerd)
   Tasks: 7
   Memory: 25.2M
   CGroup: /system.slice/docker.service
           └─2938 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.519931884Z" level=warning msg="Your kernel does not support CPU realtime scheduler"
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.528382284Z" level=warning msg="Your kernel does not support cgroup blkio weight"
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.528643852Z" level=warning msg="Your kernel does not support cgroup blkio weight_device"
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.521861985Z" level=info msg="Loading containers: start."
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.735283782Z" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be used to set a preferred IP address"
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.843825379Z" level=info msg="Loading containers: done."
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.933516185Z" level=info msg="Docker daemon" commit=6790dc graphdriver(s)=overlay2 version=20.10.14
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.933624381Z" level=info msg="Daemon has completed initialization"
Apr 18 14:14:25 ip-172-31-41-169 systemd[1]: Started Docker Application Container Engine.
Apr 18 14:14:25 ip-172-31-41-169 dockerd[2938]: time="2022-04-18T14:14:25.999194374Z" level=info msg="API listen on /run/docker.sock"
ubuntu@ip-172-31-41-169:~$
```



# 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
```

```
ubuntu@ip-172-31-41-169:~$ sudo docker pull navitchoudhary22/mvc-structure
Using default tag: latest
latest: Pulling from navitchoudhary22/mvc-structure
df9b9388f04a: Pull complete
622e2b598d8a: Pull complete
f7c8a32a53f2: Pull complete
7da04ed7dief: Pull complete
a7c1bda96431: Pull complete
b7d42cf4838a: Pull complete
3c8f7b92220d: Pull complete
62ca17e40125: Pull complete
Digest: sha256:eb717e463f48bb92d50169a6e8ab496ecaf0b0a709c698d740d3c01a4bc476b1
Status: Downloaded newer image for navitchoudhary22/mvc-structure:latest
docker.io/navitchoudhary22/mvc-structure:latest
ubuntu@ip-172-31-41-169:~$
```



# 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
```

```
~/ssh - ubuntu@ip-172-31-41-169: ~ -- ssh -i aws_free_server_temp.pem ubuntu@54.252.135.104
ubuntu@ip-172-31-41-169:~$ sudo docker run -d -p 8080:8080 navitchoudhary22/mvc-structure
c3207469f6dab7b365597f84bd2cab3b9f2a54682891ece2747d5551b467dbf8
ubuntu@ip-172-31-41-169:~$ docker ps
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/json": dial unix /var/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-41-169:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c3207469f6da	navitchoudhary22/mvc-structure	"docker-entrypoint.s..."	13 seconds ago	Up 11 seconds	0.0.0.0:8080->8080/tcp, :::8080->8080/tcp	distracted_nightingale

```
ubuntu@ip-172-31-41-169:~$
```

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.

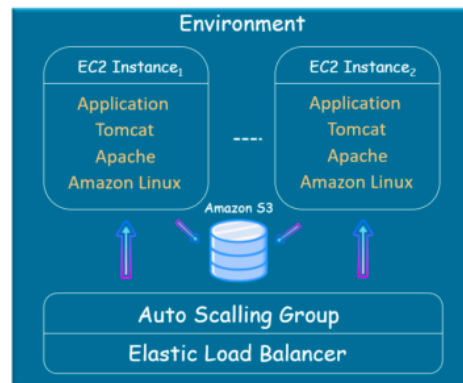
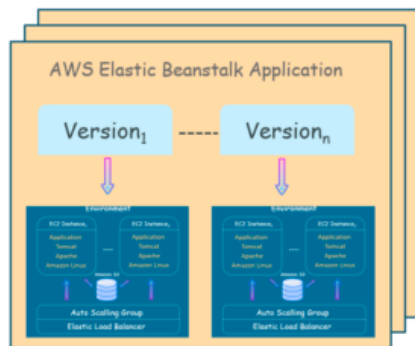


# Elastic Beanstalk Components

## 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.

## AWS services

### ▼ All services

#### Compute

EC2

Lightsail

Lambda

Batch

Elastic Beanstalk

Serverless Application

Repository

AWS Outposts

EC2 Image Builder

AWS App Runner

#### Quantum Technologies

Amazon Braket

#### Management & Governance

AWS Organizations

CloudWatch

AWS Auto Scaling

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

#### Security, Identity, & Compliance

IAM

Resource Access Manager

Cognito

Secrets Manager

GuardDuty

Inspector

Amazon Macie

AWS Single Sign-On

Certificate Manager

Key Management Service

Compute

## Amazon Elastic Beanstalk

### End-to-end web application management.

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.

#### Get started

Easily deploy your web application in minutes.

Create Application

Elastic Beanstalk > Environments > Helloexpress-env



#### Creating Helloexpress-env

This will take a few minutes.

9:49pm Using elasticbeanstalk-us-east-1-200057541580 as Amazon S3 storage bucket for environment data.

9:49pm createEnvironment is starting.

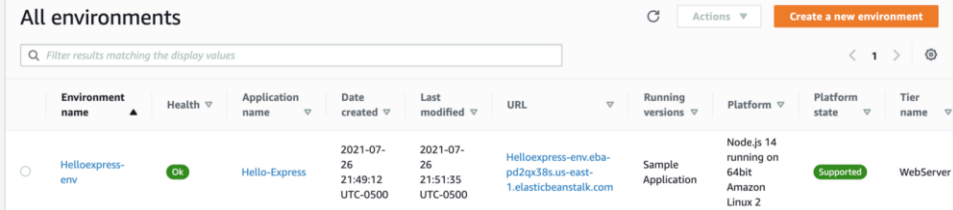


# Setting up Elastic Beanstalk Server

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”



Environment name ▲	Health ▾	Application name ▾	Date created ▾	Last modified ▾	URL ▾	Running versions ▾	Platform ▾	Platform state ▾	Tier name ▾
○ Helloexpress-env	OK	Hello-Express	2021-07-26 21:49:12 UTC-0500	2021-07-26 21:51:35 UTC-0500	Helloexpress-env.eba-pd2qx38s.us-east-1.elasticbeanstalk.com	Sample Application	Node.js 14 running on 64bit Amazon Linux 2	Supported	WebServer



## Developer Tools

CodeStar

CodeCommit

CodeArtifact

CodeBuild

CodeDeploy

CodePipeline

Cloud9

CloudShell

X-Ray

AWS FIS



# 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.

**Pipelines** info Refresh Notify View history Release change Delete pipeline Create pipeline

Search

Name	Most recent execution	Latest source revisions	Last executed
No results There are no results to display.			

### Pipeline settings

**Pipeline name**  
Enter the pipeline name. You cannot edit the pipeline name after it is created.  
  
No more than 100 characters

**Service role**

☒ **New service role**  
Create a service role in your account

☐ **Existing service role**  
Choose an existing service role from your account

**Role name**  
  
Type your service role name

☒ **Allow AWS CodePipeline to create a service role so it can be used with this new pipeline**

► **Advanced settings**

Cancel Next





# 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.

## Add source stage [Info](#)

### Source

#### Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (Version 2)



#### New GitHub version 2 (app-based) action

To add a GitHub version 2 action in CodePipeline, you create a connection, which uses GitHub Apps to access your repository. Use the options below to choose an existing connection or create a new one. [Learn more](#)

#### Connection

Choose an existing connection that you have already configured, or create a new one and then return to this task.



or

**Connect to GitHub**

#### Repository name

Choose a repository in your GitHub account.



No options

#### Branch name

Choose a branch of the repository.



#### Change detection options

##### ☒ Start the pipeline on source code change

Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

#### Output artifact format

Choose the output artifact format.



##### CodePipeline default

AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.



##### Full clone

AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.



# 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.

## Add source stage [Info](#)

### Source

#### Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (Version 2)



#### New GitHub version 2 (app-based) action

To add a GitHub version 2 action in CodePipeline, you create a connection, which uses GitHub Apps to access your repository. Use the options below to choose an existing connection or create a new one. [Learn more](#)

#### Connection

Choose an existing connection that you have already configured, or create a new one and then return to this task.

or [Connect to GitHub](#)



#### Ready to connect

Your GitHub connection is ready for use.

#### Repository name

Choose a repository in your GitHub account.

<account>/<repository-name>

#### Branch name

Choose a branch of the repository.

#### Change detection options

##### ☒ Start the pipeline on source code change

Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

#### Output artifact format

Choose the output artifact format.



##### CodePipeline default

AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.



##### Full clone

AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Cancel

Previous

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”

## Add build stage [Info](#)

### Build - optional

#### Build provider

This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.

AWS CodeBuild

#### Region

US East (N. Virginia)

#### Project name

Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

Q

or

[Creating project](#)

#### Environment variables - optional

Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. [Learn more](#)

Add environment variable

#### Build type



Single build

Triggers a single build.



Batch build

Triggers multiple builds as a single execution.

Cancel

Previous

Skip build stage

Next



# 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.

## Add deploy stage [Info](#)



### You cannot skip this stage

Pipelines must have at least two stages. Your second stage must be either a build or deployment stage. Choose a provider for either the build stage or deployment stage.

### Deploy

#### Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

AWS Elastic Beanstalk

#### Region

US East (N. Virginia)

#### Application name

Choose an application that you have already created in the AWS Elastic Beanstalk console. Or create an application in the AWS Elastic Beanstalk console and then return to this task.

Hello-Express

#### Environment name

Choose an environment that you have already created in the AWS Elastic Beanstalk console. Or create an environment in the AWS Elastic Beanstalk console and then return to this task.

Helloexpress-env

Cancel

Previous

Next

### Step 3: Add build stage

#### Build action provider

Build stage  
No build

### Step 4: Add deploy stage

#### Deploy action provider

Deploy action provider  
AWS Elastic Beanstalk  
ApplicationName  
Hello-Express  
EnvironmentName  
Helloexpress-env

Cancel

Previous

Create pipeline





# 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.

Developer Tools > CodePipeline > Pipelines > Hello-Express-Pipeline

Hello-Express-Pipeline [Notify](#) [Edit](#) [Stop execution](#) [Clone pipeline](#) [Release change](#)

**Source** Succeeded  
Pipeline execution ID: 143c07e3-084f-4b8c-abe5-2eb96de4a24d

**Source**  
GitHub (Version 2) [🔗](#)  
**Succeeded** - 1 minute ago  
4028e681 [🔗](#)

4028e681 [🔗](#) Source: adding hello express

[Disable transition](#)

**Deploy** In progress  
Pipeline execution ID: 143c07e3-084f-4b8c-abe5-2eb96de4a24d

**Deploy**  
AWS Elastic Beanstalk [🔗](#)  
**In progress** - 1 minute ago

4028e681 [🔗](#) Source: adding hello express

Developer Tools > CodePipeline > Pipelines > Hello-Express-Pipeline

Hello-Express-Pipeline [Notify](#) [Edit](#) [Stop execution](#) [Clone pipeline](#) [Release change](#)

**Source** Succeeded  
Pipeline execution ID: 10126dcb-f260-4c71-affa-5c247d79b634

**Source**  
GitHub (Version 2) [🔗](#)  
**Succeeded** - 2 minutes ago  
96e443c2 [🔗](#)

96e443c2 [🔗](#) Source: Update app.js

[Disable transition](#)

**Deploy** Succeeded  
Pipeline execution ID: 10126dcb-f260-4c71-affa-5c247d79b634

**Deploy**  
AWS Elastic Beanstalk [🔗](#)  
**Succeeded** - Just now

96e443c2 [🔗](#) Source: Update app.js



# 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.

All environments

Filter results matching the display values

Environment name	Health	Application name	Date created	Last modified	URL	Running versions
<a href="#">Helloexpress-env</a>	Ok	Hello-Express	2021-07-26 21:49:12 UTC-0500	2021-07-27 21:25:37 UTC-0500	<a href="#">Helloexpress-env.eba-pd2qx38s.us-east-1.elasticbeanstalk.com</a>	code-pipeline-1627496e443c21eef71126

Environments

Applications

Change history

▼ Hello-Express

Application versions

Saved configurations

▼ Helloexpress-env

Go to environment

Configuration

Logs

Health

Monitoring

Alarms

Managed updates

Events

Tags

Refresh

Actions

Helloexpress-env

[Helloexpress-env.eba-pd2qx38s.us-east-1.elasticbeanstalk.com](#) (e-d4mhwmdmagt)

Application name: Hello-Express

Health

Ok

Causes

Running version

code-pipeline-1627439039612-96e443c21eef7112612633100991fac4b384a81f

Upload and deploy

Platform

nodejs

Node.js 14 running on 64bit Amazon Linux 2/5.4.3

Change

Recent events

Show all

Time	Type	Details
2021-07-27 21:28:47 UTC-0500	INFO	Deleted log fragments for this environment.



## 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