

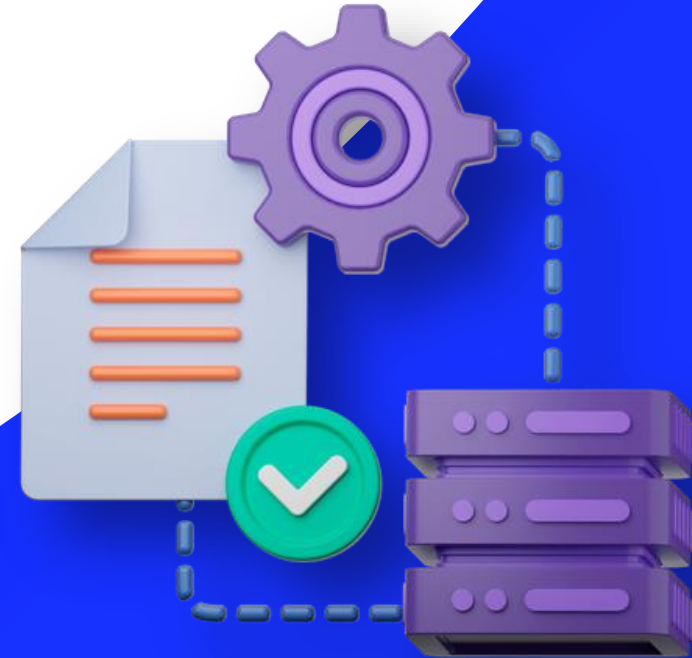


Institute of  
Data

# 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.

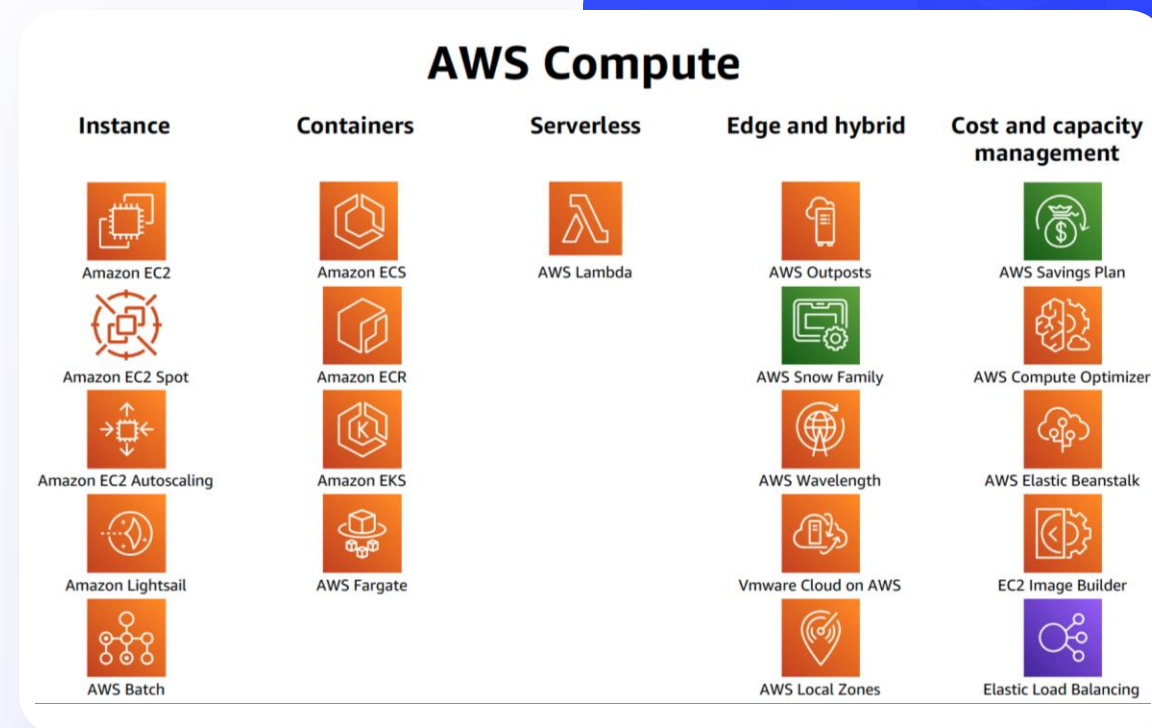
Amazon EC2 (elastic cloud computing) offers a range of instance types that are tailored to certain use cases. Instance types are different combinations of **CPU**, **memory**, **storage**, and **networking** capabilities that allow you to choose the best resource mix for your applications. Each instance 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

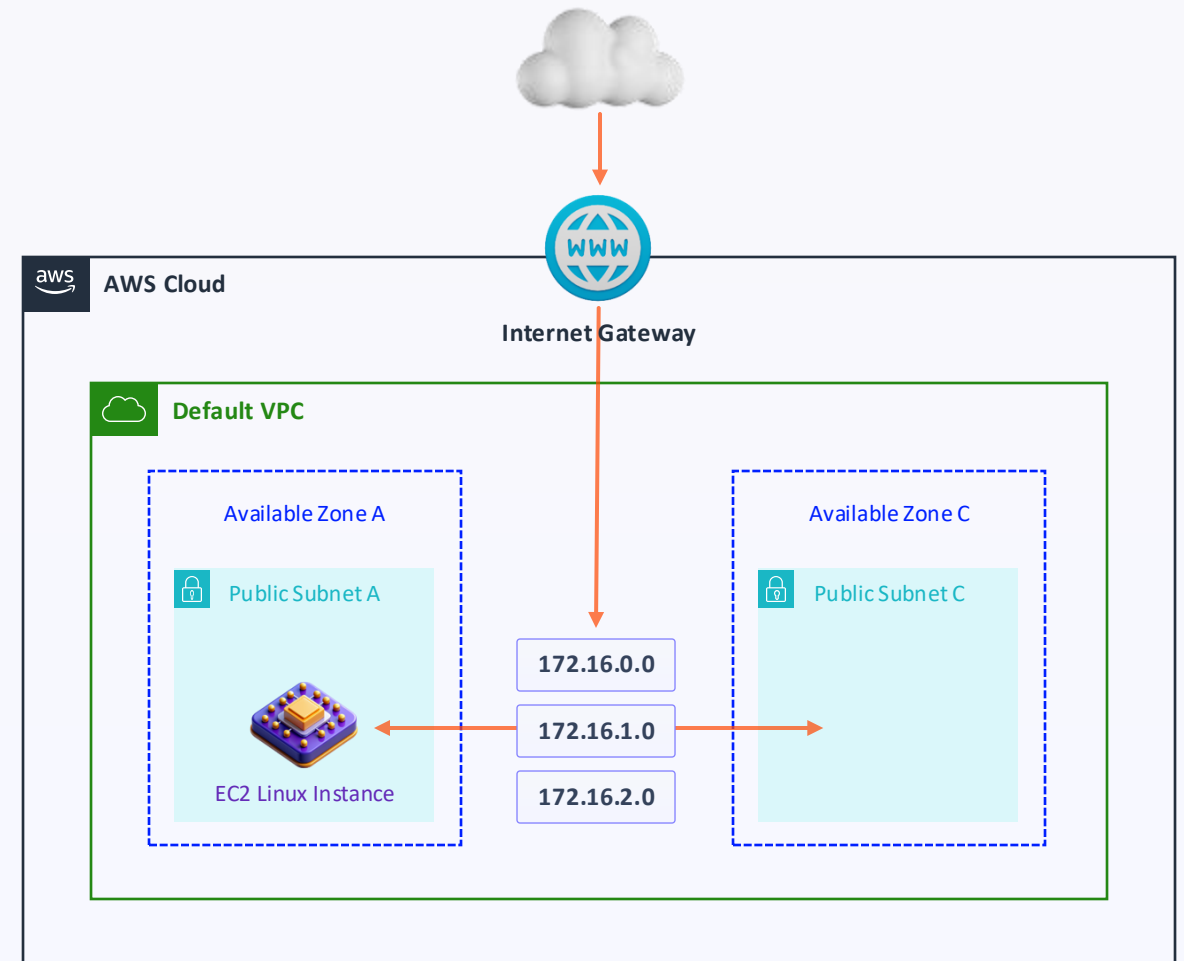


# EC2 Overview

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:

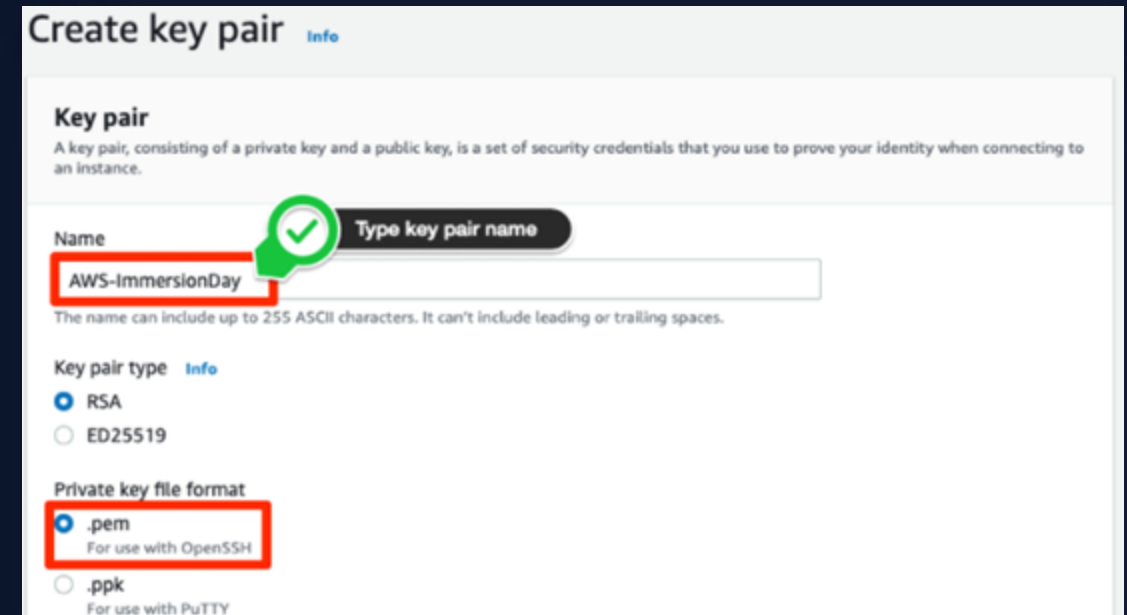
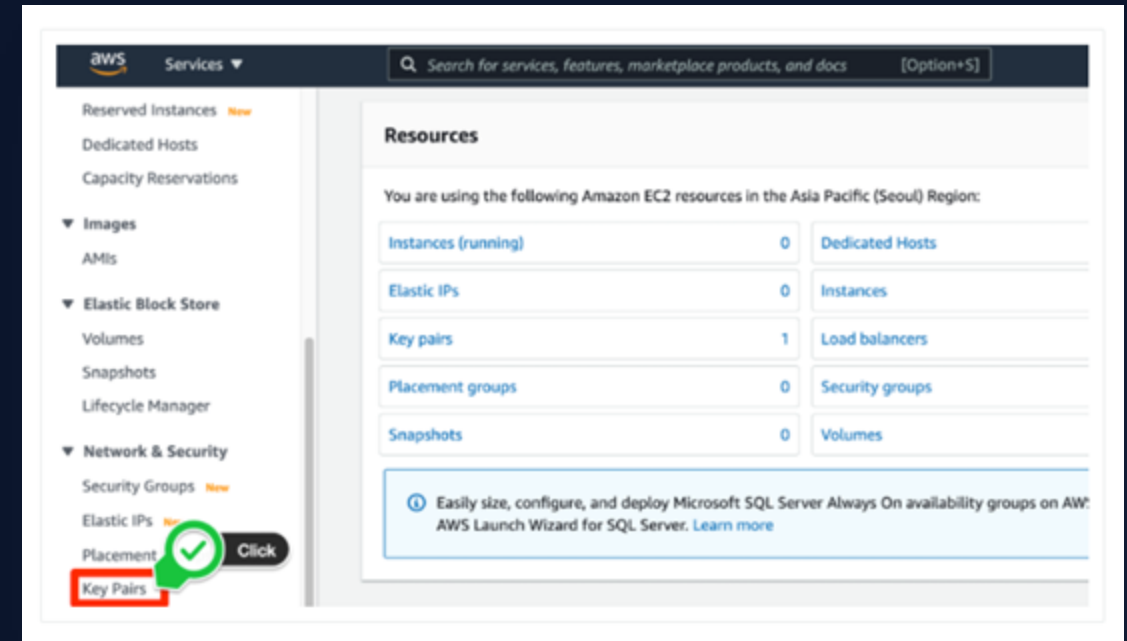
1. [Create a new key pair](#)
2. [Launch a Web Server Instance](#)
3. [Connect to your linux instance](#)
4. [Connect to EC2 Instance using PuTTY \(Optional\)](#)



# Create a key pair

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

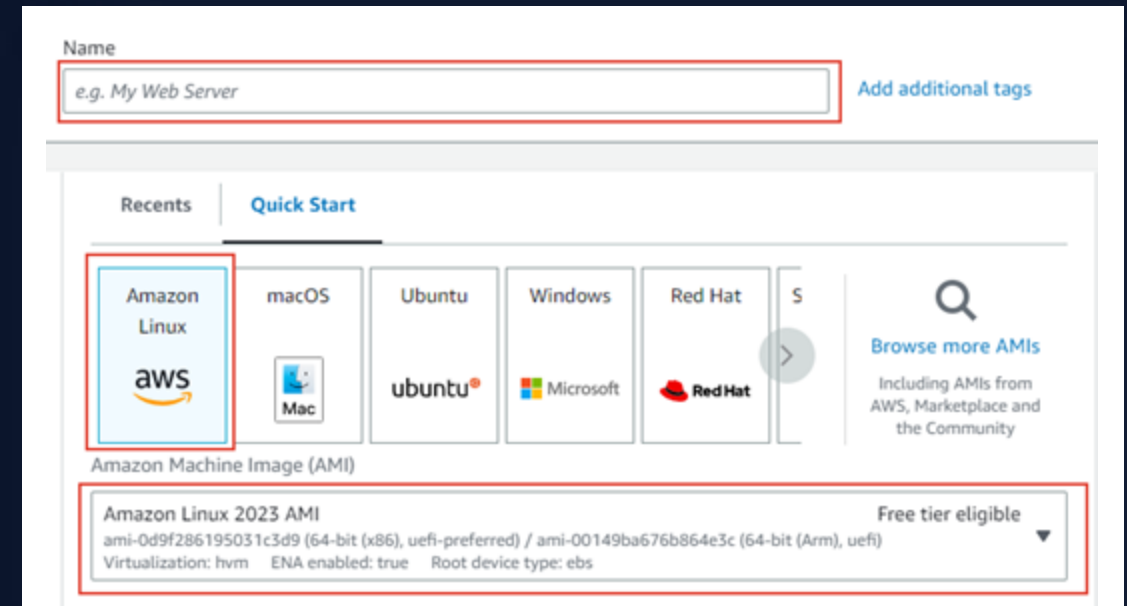
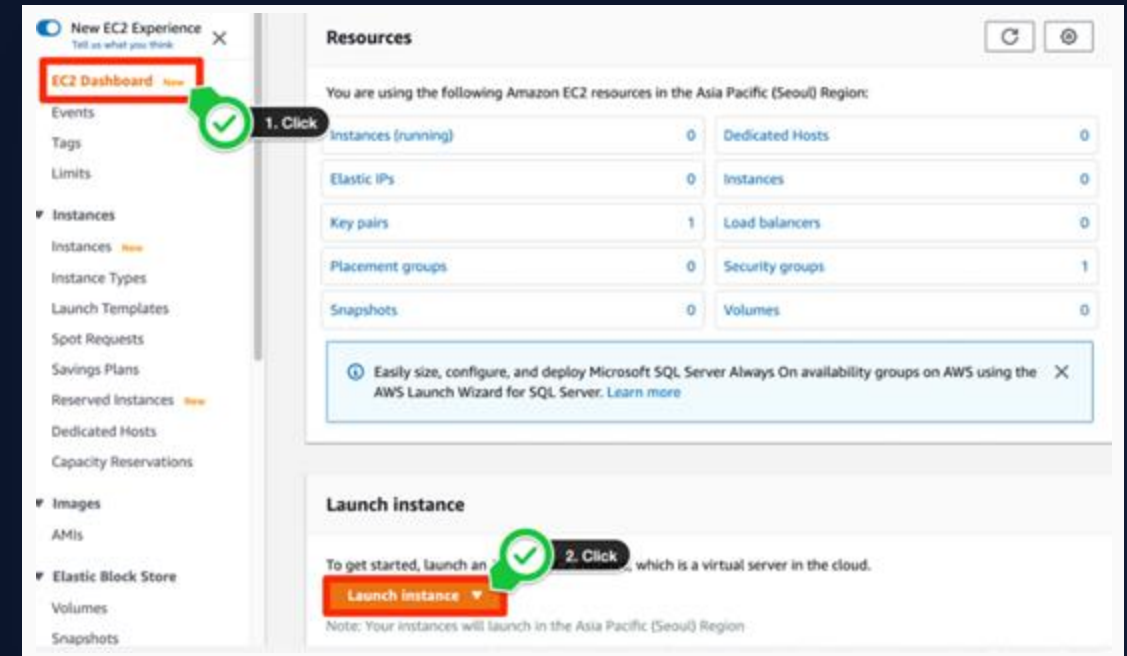
1. Sign into the AWS Management Console and open the [Amazon EC2 console](#). In the upper-right corner of the AWS Management Console, confirm you are in the closest AWS region.
2. Click on **Key Pairs** in the Network & Security section near the bottom of the leftmost menu. This will display a page 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

▼ Instance type [Info](#)

Instance type

<b>t2.micro</b>	Free tier eligible
Family: t2 1 vCPU 1 GiB Memory	
On-Demand Linux pricing: 0.0146 USD per Hour	
On-Demand Windows pricing: 0.0192 USD per Hour	

[Compare instance types](#)

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select

Q |

Proceed without a key pair (Not recommended) Default value

<b>AWS-ImmersionDay</b>	AWS-ImmersionDay
Type: rsa	
JB-ImmersionDay	
Type: rsa	

vpc-03ad75a11be4d032b

[Create new key pair](#)

[Edit](#)



# 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

The screenshot shows the 'Network settings' section of the AWS Management Console. It includes fields for Network (vpc-03ad75a11be4d032b), Subnet (No preference), and Auto-assign public IP (Enabled). The 'Firewall (security groups)' section is active, showing options to 'Create security group' or 'Select existing security group'. Below, it states a new security group 'launch-wizard-1' will be created with the following rules:

Rule	Source
<input checked="" type="checkbox"/> Allow SSH traffic from Helps you connect to your instance	My IP 49.177.92.48/32
<input type="checkbox"/> Allow HTTPs traffic from the internet To set up an endpoint, for example when c	Anywhere 0.0.0.0/0
<input type="checkbox"/> Allow HTTP traffic from the internet To set up an endpoint, for example when c	Custom My IP 49.177.92.48/32

# 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:

▼ Advanced details Info

Purchasing option Info

☐ Request Spot Instances

Request Spot Instances at the Spot price, capped at the On-Demand price

Domain join directory Info

Select

Create new directory

User data Info

☐ User data has already been base64 encoded

Cancel

Launch instance

✔

**Success**

Successfully initiated launch of instance (i-0ab2fd91be24faba3)

[▶ Launch log](#)

**Next Steps**

**Get notified of estimated charges**

[Create billing alerts](#) to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier)

**How to connect to your instance**

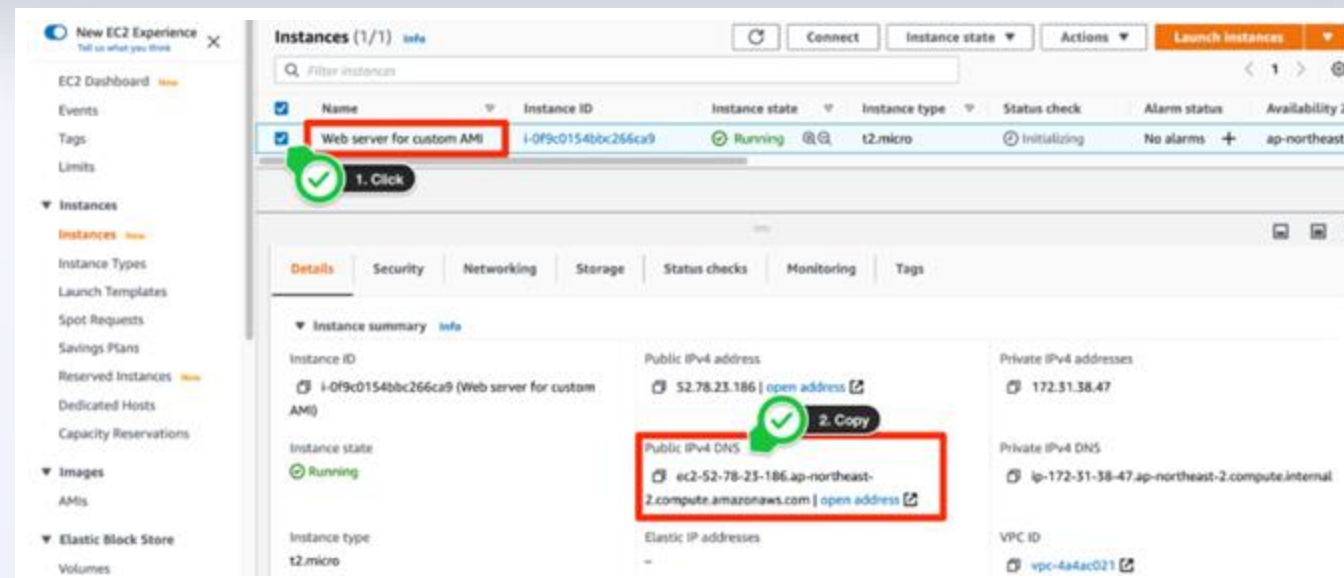
Your instance is launching and it might be a few minutes until it is in the running state, when it will be ready for you to use

Click [View Instances](#) to monitor your instance's status. Once your instance is in the 'running' state, you can connect to it from the Instances screen. Find out [how to connect to your instance](#)

[View more resources to get you started](#)

[View all instances](#)

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.



The screenshot shows the AWS Management Console interface for the EC2 Instances page. The instance 'Web server for custom AMI' is listed with the ID 'i-0f9c0154bbc266ca9' and is in the 'Running' state. The 'Details' tab is selected, showing the instance summary. The 'Public IPv4 DNS' field is highlighted with a red box and a green checkmark, indicating it is ready to be copied. The 'Public IPv4 address' is also visible as '52.78.23.186'.

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

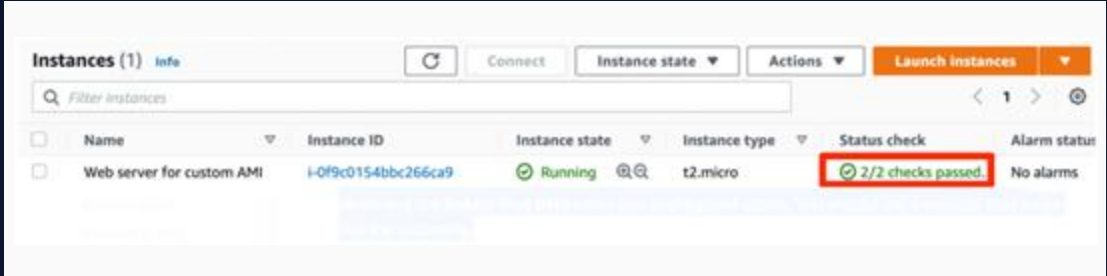
**Instance summary**

Field	Value
Instance ID	i-0f9c0154bbc266ca9 (Web server for custom AMI)
Instance state	Running
Instance type	t2.micro
Public IPv4 address	52.78.23.186   <a href="#">open address</a>
Public IPv4 DNS	ec2-52-78-23-186.ap-northeast-2.compute.amazonaws.com   <a href="#">open address</a>
Private IPv4 addresses	172.31.38.47
Private IPv4 DNS	ip-172-31-38-47.ap-northeast-2.compute.internal
Elastic IP addresses	-
VPC ID	vpc-4a4ac021

# 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.*

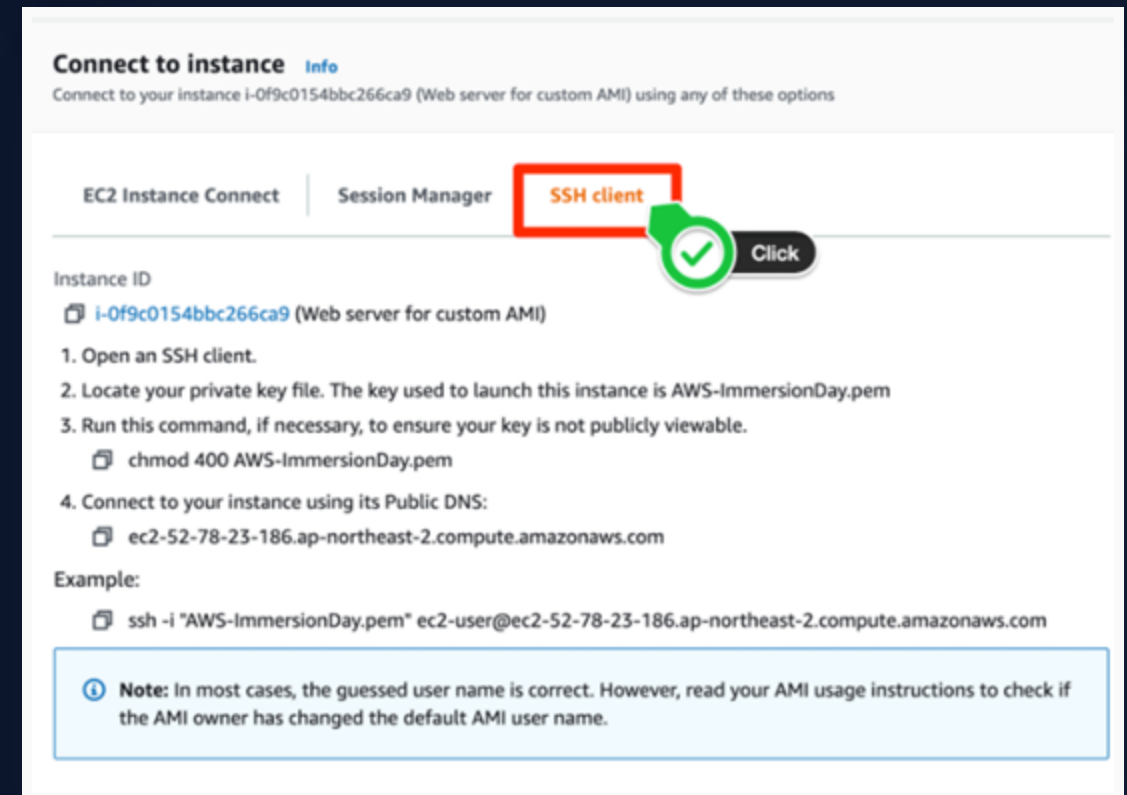
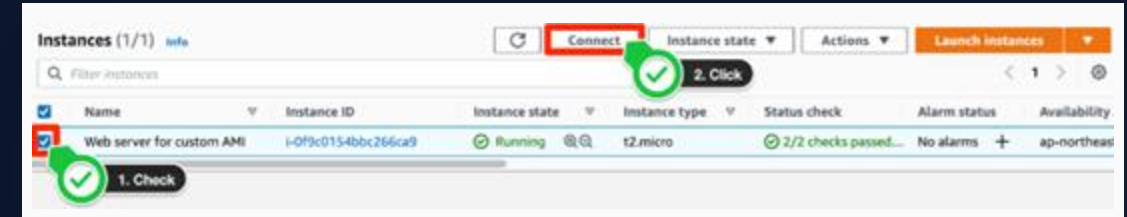


# Connect to your Linux instance

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).

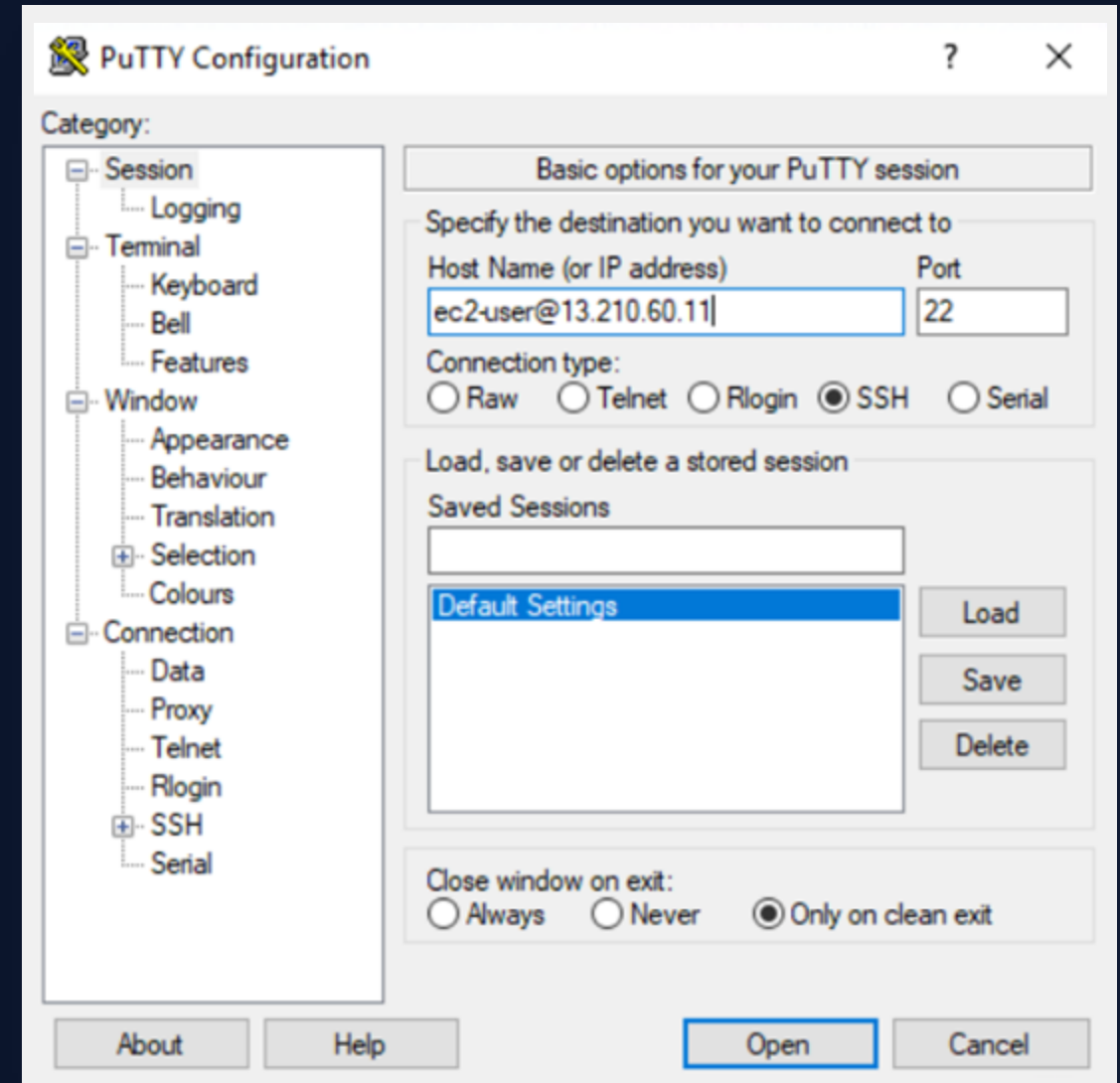
*You will need to run the ssh command from the same folder where you stored your .pem key file.*

*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.



**.ENV**

# 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
```

```
Last login: Tue Apr 19 00:06:32 on ttys003
Navit@alesios-Mini ~ % cd .ssh
Navit@alesios-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:~$
```

```
ec2-user@ip-172-31-5-180:~
Using username "ec2-user".
Authenticating with public key "tempkey"

#
#####
~\  #####\
~~ \#####|
~~  \##/
~~   V~' '->
~~~
~~~
~/m/' - - - - -
[ec2-user@ip-172-31-5-180 ~]$
```

Amazon Linux 2023

<https://aws.amazon.com/linux/amazon-linux-2023>

# 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
```

```
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
7da04ed7d1ef: Pull complete
a7c1bda96431: Pull complete
b7d42cf4838a: Pull complete
3c8f7b92220d: Pull complete
62ce17e40125: Pull complete
Digest: sha256:eb717e463f48bb92d50169a4e8ab496ecaf0b0a709c698d740d3c01a4bc476b1
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

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
```

```
~/.ssh — ubuntu@ip-172-31-41-169: ~ — ssh -l 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:~$
```

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:

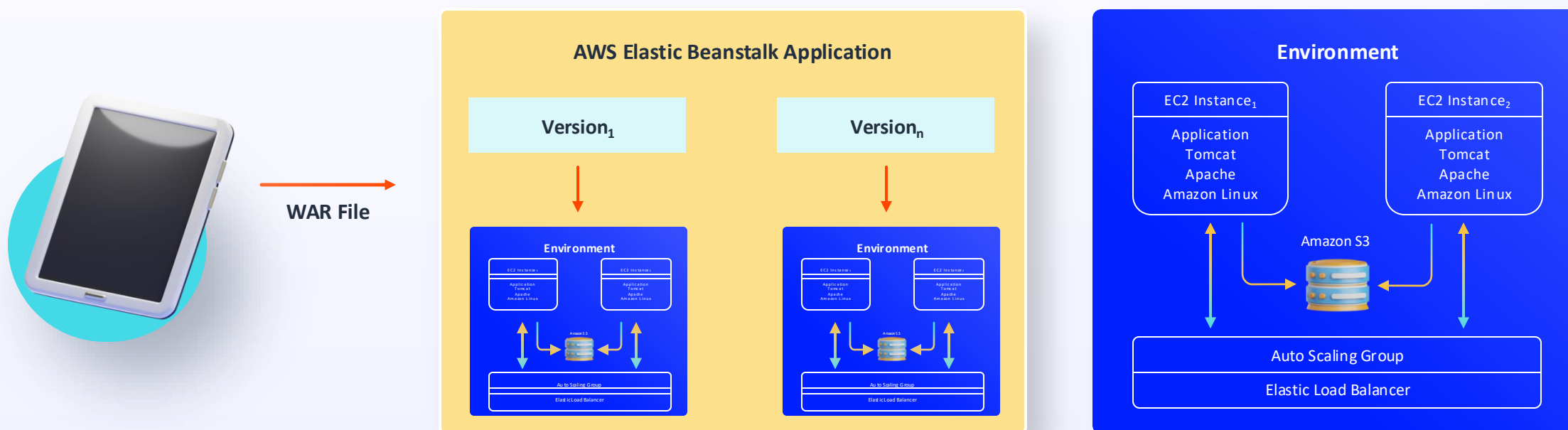
- ❖ The current version of the Elastic Beanstalk Application will be active in environments within the Elastic Beanstalk Application.
- ❖ At any one time, each environment only runs one application version. However, the same or different versions of an application can be operated in many settings at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

# Elastic Beanstalk Components

## Environment Tier:

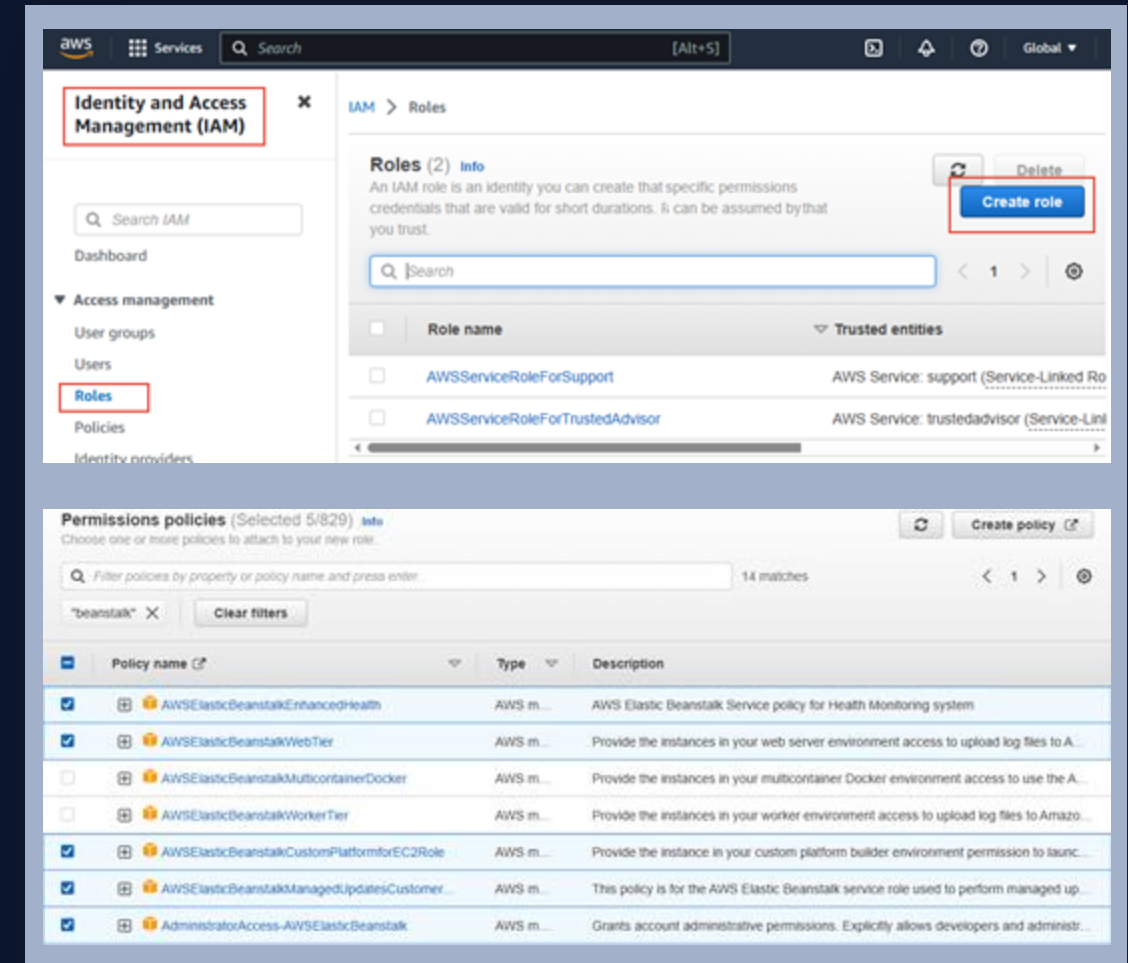
Based on requirements, Beanstalk offers two different Environment tiers: Web Server Environment, Worker Environment.

- ❖ 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.



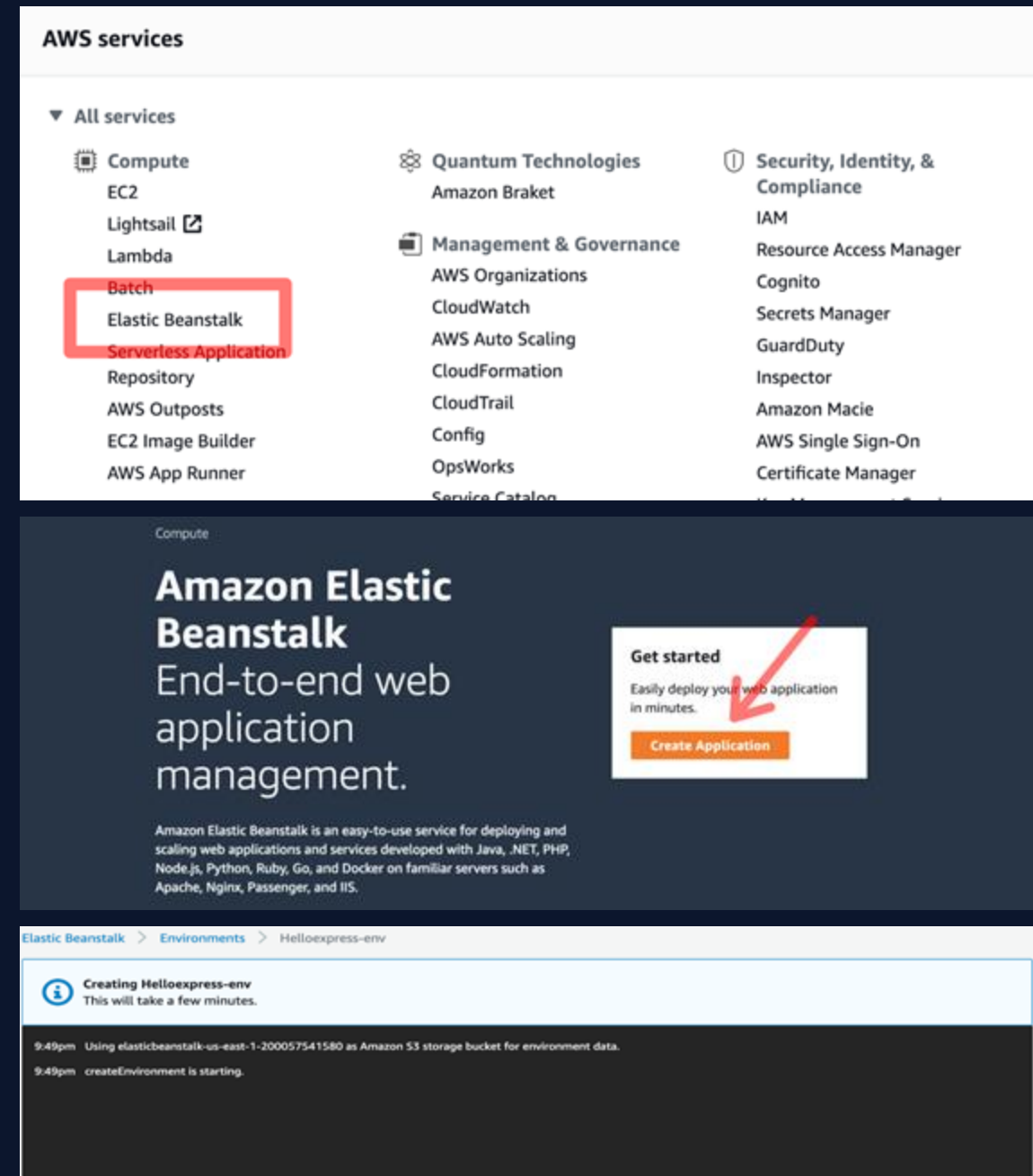
# Setting up Elastic Beanstalk Server

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

### Service role

☐ Create and use new service role

☒ Use an existing service role

**Existing service roles**  
Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

**EC2 key pair**  
Select an EC2 key pair to securely log in to your EC2 instances. [Learn more](#)

<NAME OF KEY PAIR>

**EC2 instance profile**  
Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-service-role

[View permission details](#)

## Configure instance traffic and scaling - optional [Info](#)

### ▼ Instances [Info](#)

Configure the Amazon EC2 instances that run your application.

#### Root volume (boot device)

##### Root volume type

General Purpos...

(Container default)

Magnetic

General Purpose (SSD)

General Purpose 3(SSD)✓

Provisioned IOPS (SSD)

3000

it volume attached to each instance.

GB

for a provisioned IOPS (SSD) volume.

IOPS

# 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`).

Health reporting

Enhanced health reporting provides free real-time application and operating system monitoring of the instances and other resources in your environment. The `EnvironmentHealth` custom metric is provided free with enhanced health reporting. Additional charges apply for each custom metric. For more information, see [Amazon CloudWatch Pricing](#)

System

☒ Basic

☐ Enhanced

Step 6

Managed platform updates [Info](#)

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

Managed updates

☐ Activated

Step 7

Environment properties

The following properties are passed in the application as environment properties. [Learn more](#)

Name

Value

PORT

3001

Remove

Add environment property

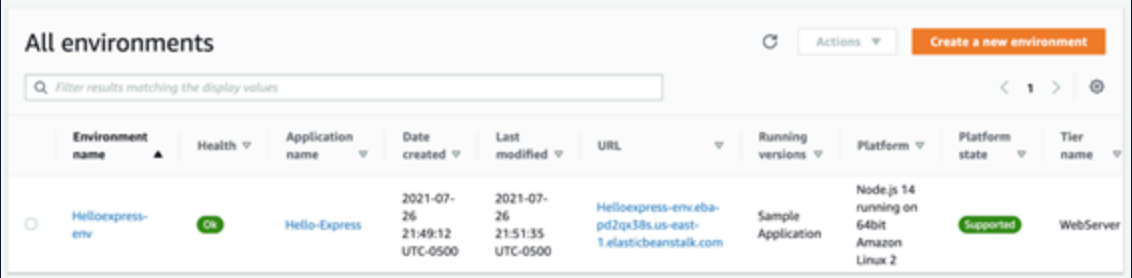
# Setting up Elastic Beanstalk Server

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.



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





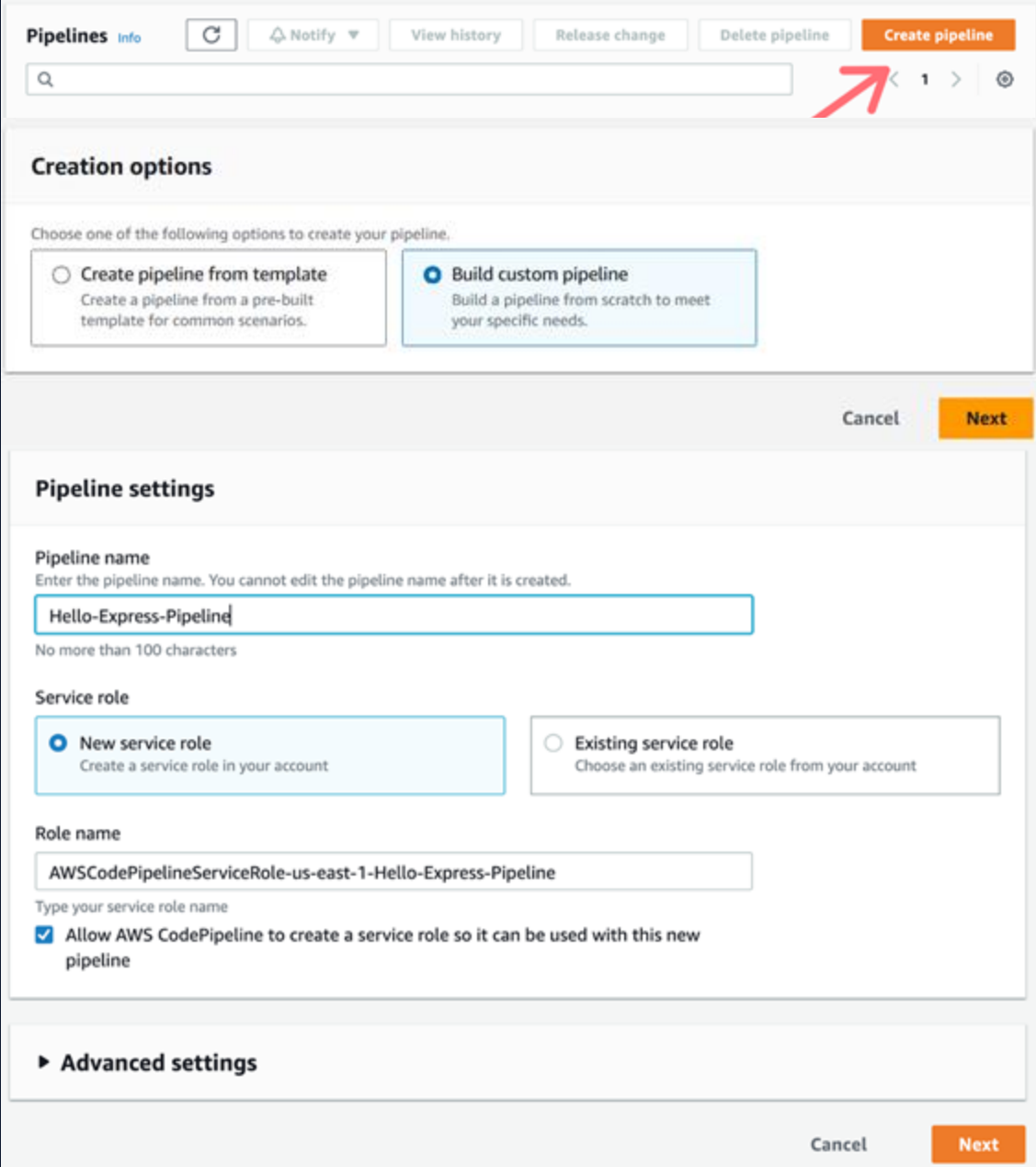
# Setting up Elastic Beanstalk Server

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.



The screenshot shows the 'Create pipeline' wizard in the AWS CodePipeline console. At the top, there are buttons for 'Pipelines Info', 'Notify', 'View history', 'Release change', 'Delete pipeline', and 'Create pipeline'. A red arrow points to the 'Create pipeline' button. Below this is a search bar and a pagination control showing '1' of 1 pages. The 'Creation options' section has two radio buttons: 'Create pipeline from template' (unselected) and 'Build custom pipeline' (selected). The 'Pipeline settings' section includes a 'Pipeline name' field with the value 'Hello-Express-Pipeline', a 'Service role' section with 'New service role' selected, and a 'Role name' field with the value 'AWSCodePipelineServiceRole-us-east-1-Hello-Express-Pipeline'. There is also a checkbox for 'Allow AWS CodePipeline to create a service role so it can be used with this new pipeline' which is checked. At the bottom, there is an 'Advanced settings' section and 'Cancel' and 'Next' buttons.

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

Search < 1 >

### Creation options

Choose one of the following options to create your pipeline.

☐ Create pipeline from template  
Create a pipeline from a pre-built template for common scenarios.

☒ Build custom pipeline  
Build a pipeline from scratch to meet your specific needs.

Cancel Next

### 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 (via GitHub App)**.

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

## Create a connection Info

### Create GitHub App connection Info

Connection name

► Tags - optional

Connect to GitHub

## Source

### Source provider

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

GitHub (via GitHub App) ▼

### Connection

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

 or

### Repository name

Choose a repository in your GitHub account.

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

### Default branch

Default branch will be used only when pipeline execution starts from a different source or manually started.

### 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. [Learn more](#)

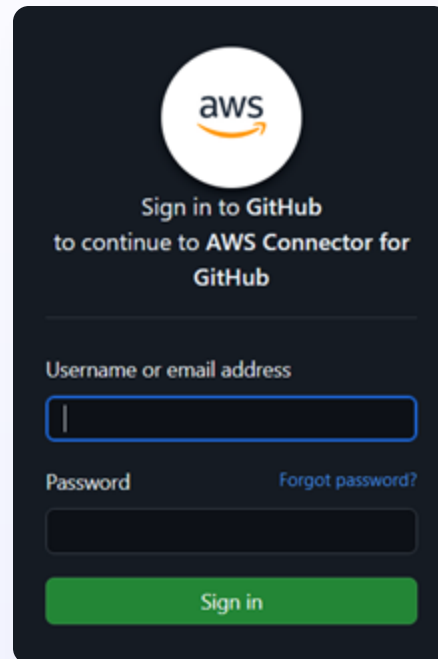
☒ Enable automatic retry on stage failure

# 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.



### 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.

arn:aws:codestar-connections:us-east-1:200057541580:connection/f43f2ce3 ✕ or [Connect to GitHub](#)

**Ready to connect**  
Your GitHub connection is ready for use.

**Repository name**  
Choose a repository in your GitHub account.

azamsharp/Hello-Express ✕  
<account>/<repository-name>

**Branch name**  
Choose a branch of the repository.

master ✕

**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

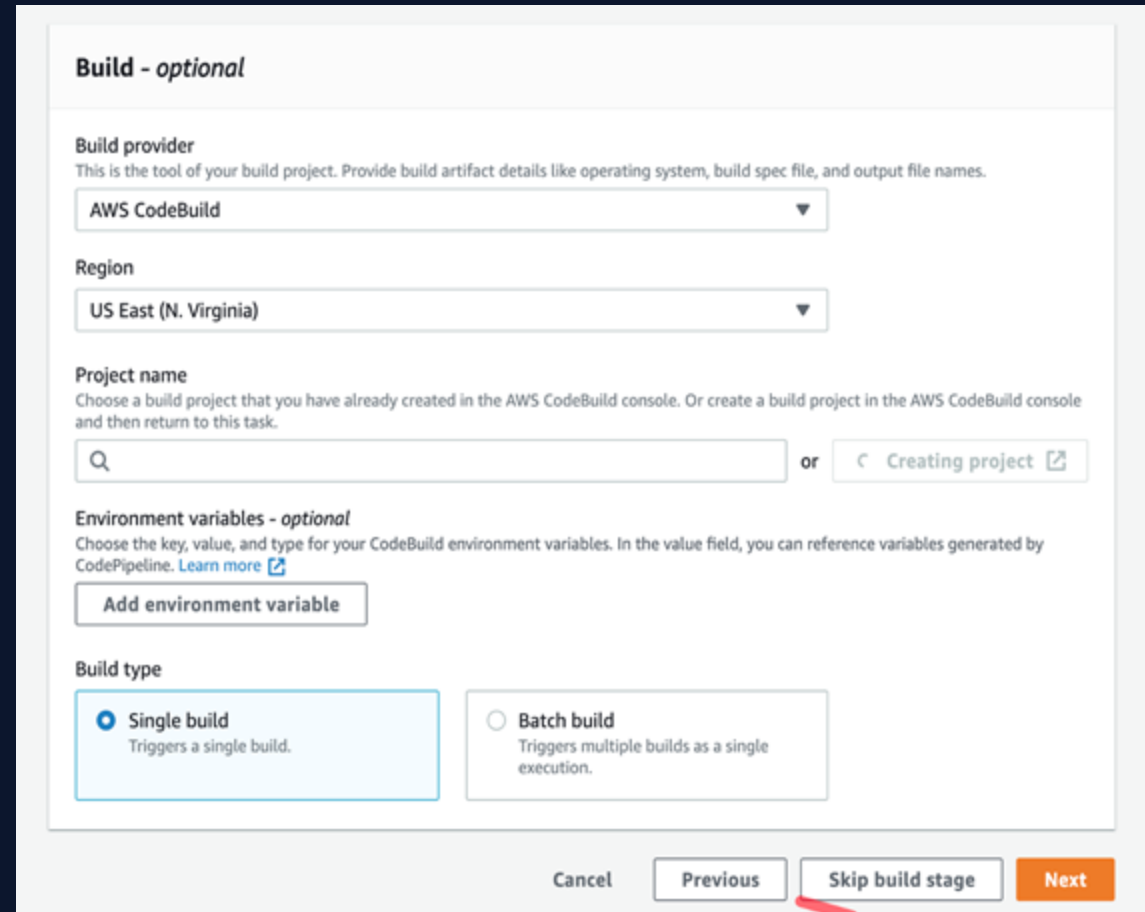
The next screen will allow you to add a **build** stage. We are going to skip this step, so click on the “**Skip build stage**” button at the bottom.

A confirmation dialog will popup, select “Skip”.

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**”.



**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](#)

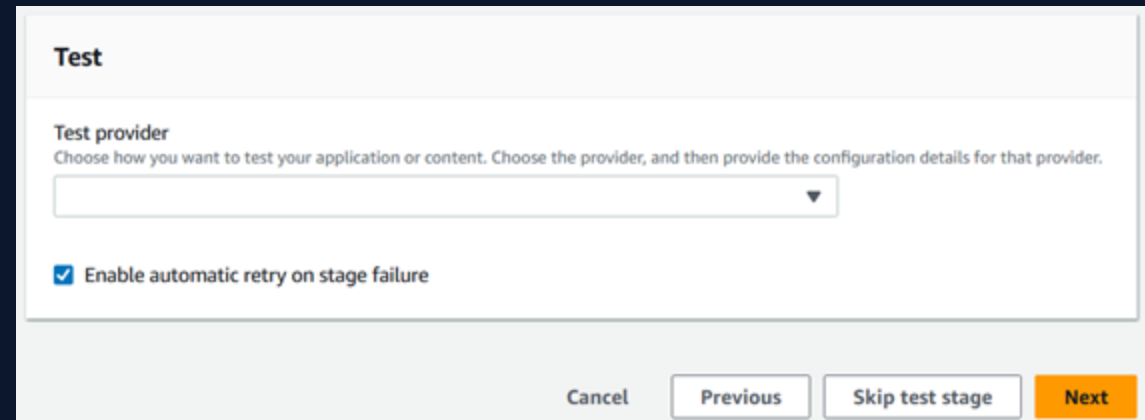
[Add environment variable](#)

**Build type**

☒ **Single build**  
Triggers a single build.

☐ **Batch build**  
Triggers multiple builds as a single execution.

Cancel Previous **Skip build stage** Next



**Test**

**Test provider**  
Choose how you want to test your application or content. Choose the provider, and then provide the configuration details for that provider.

▼

☒ **Enable automatic retry on stage failure**

Cancel Previous **Skip test 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 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.

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

 Institute of Data

www.institutedata.com | 37

# 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.

The screenshot shows the AWS CodePipeline console for a pipeline named 'Hello-Express-Pipeline'. The pipeline is in the 'In progress' state. The 'Source' stage, using 'GitHub (Version 2)', has completed successfully. The 'Deploy' stage, using 'AWS Elastic Beanstalk', is currently in progress. A 'Disable transition' button is visible between the stages. The pipeline execution ID is 143c07e3-084f-4bdc-abe5-2eb96de4a24d.

The screenshot shows the same 'Hello-Express-Pipeline' in the 'Succeeded' state. Both the 'Source' and 'Deploy' stages have completed successfully. The 'Source' stage used 'GitHub (Version 2)' and the 'Deploy' stage used 'AWS Elastic Beanstalk'. The pipeline execution ID is 10126ddb-f260-4c71-affa-5c247d798634.

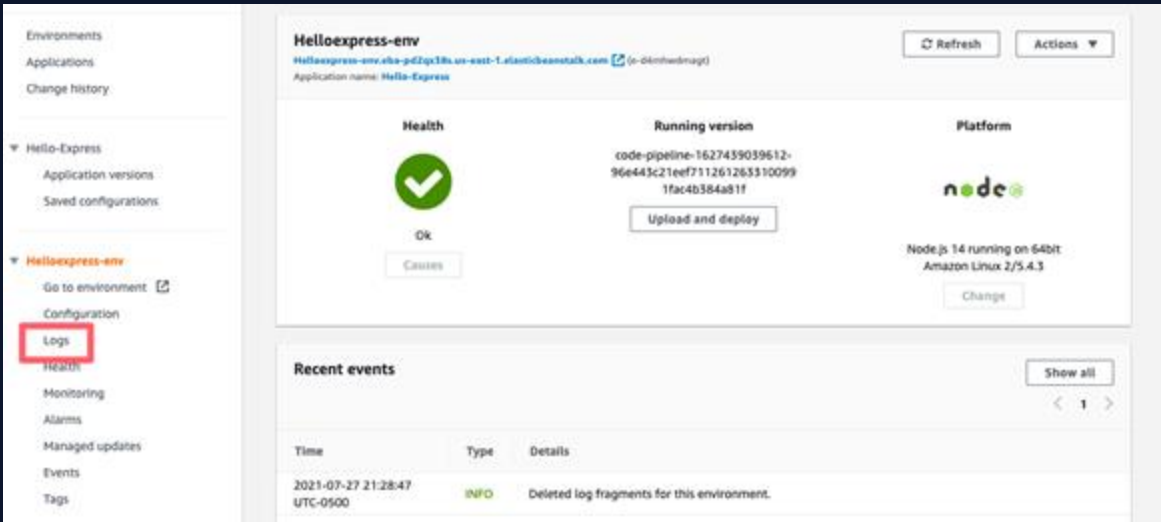
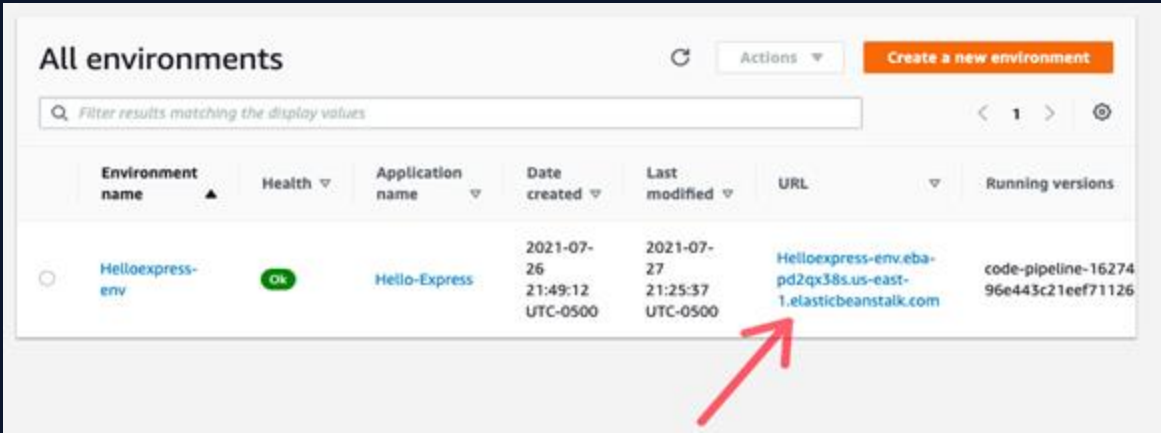
# Setting up Elastic Beanstalk Server

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.



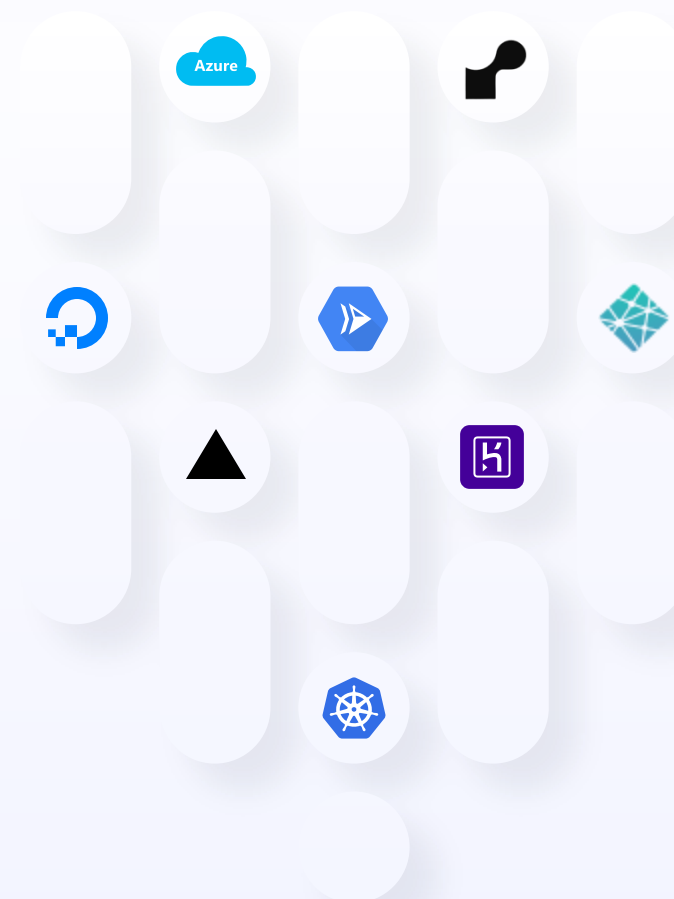


# Other Deployment options

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.