

# ADV-DEVOPS CASE STUDY

## AIM:

Cloud Deployment with Automation

- Concepts Used: AWS CodePipeline, EC2, and S3.
- Problem Statement: "Build a simple web application using AWS CodeBuild and deploy it to an S3 bucket. Then, automate the deployment process using AWS CodePipeline, ensuring the application is deployed on an EC2 instance. Use a sample index.html page for demonstration."
- Tasks:
  - Set up AWS CodeBuild for the web app.
  - Create a pipeline that deploys to an S3 bucket.
  - Use AWS CodeDeploy to push updates to an EC2 instance.

## THEORY:

### 1. Overview of AWS Services Involved

Before diving into the technical process, it is essential to understand the AWS services used in this experiment and how they interact with each other. The experiment will use the following AWS services:

- **Amazon EC2 (Elastic Compute Cloud):** EC2 is a cloud-based service that provides scalable compute capacity. It enables users to rent virtual machines (VMs) to host web applications and services. These virtual machines can be configured with various operating systems and software environments.
- **Amazon S3 (Simple Storage Service):** S3 is an object storage service that allows users to store and retrieve large amounts of data over the web. It can be used to store build artifacts, such as code, static files, and other resources required by the application.
- **AWS CodePipeline:** CodePipeline is a fully managed continuous integration and delivery service that helps automate the process of releasing applications. It builds, tests, and deploys code whenever there is a change in the source code repository (such as GitHub).
- **AWS CodeBuild:** CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software artifacts that are ready for deployment.
- **AWS CodeDeploy:** CodeDeploy automates the process of deploying applications to various compute platforms, such as EC2 instances, Lambda functions, or on-premises servers.

Each of these services plays a crucial role in the automation pipeline, and understanding their function and how they interact is key to building a successful deployment system.

## 2. Detailed Explanation of AWS CodePipeline

AWS CodePipeline acts as the backbone of this experiment, automating the movement of code from a version control repository (like GitHub or S3) through the stages of building, testing, and deployment. Each step in the pipeline is responsible for specific actions such as pulling the code from the repository, compiling it using CodeBuild, and deploying it using CodeDeploy.

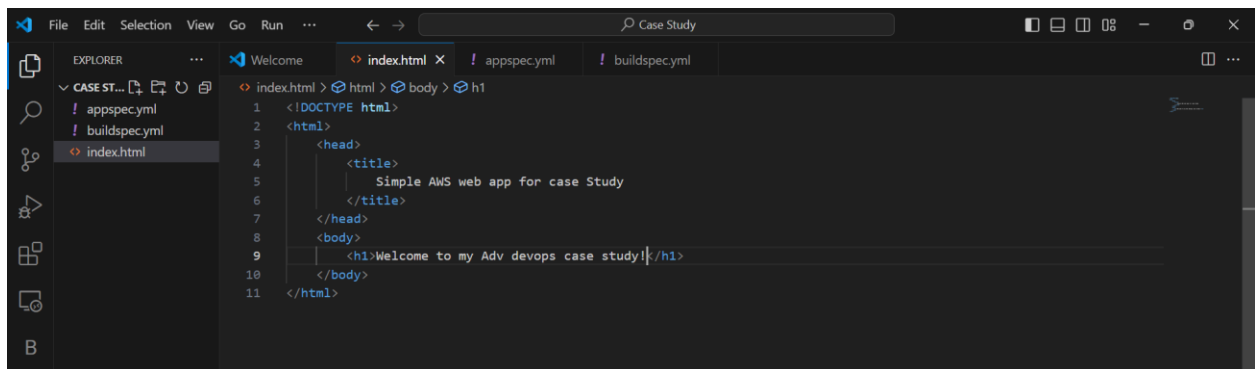
### Pipeline Stages:

1. **Source Stage:** The pipeline begins with the source stage, which pulls the latest version of the code from the specified repository (GitHub, Bitbucket, or S3). In this case, the source is a simple HTML file (`index.html`) that needs to be updated and deployed onto an EC2 instance.
2. **Build Stage:** This stage uses AWS CodeBuild to compile and package the application. For static websites or simple HTML files, this stage may not perform heavy compilation, but it will still produce an output artifact (such as an updated `index.html` file) that will be deployed to an EC2 instance.
3. **Deploy Stage:** In the final stage, AWS CodeDeploy pushes the build artifacts (e.g., `index.html`) to the EC2 instance. During this stage, the application is installed, and the EC2 instance is updated to reflect the latest changes in the code.

## PROCEDURE & SCREENSHOTS:

### Create a Simple Web App

1. First, create a simple web app with an `index.html` file:

A screenshot of a code editor window titled 'Case Study'. The Explorer panel on the left shows a project structure with files 'appspec.yml', 'buildspec.yml', and 'index.html'. The 'index.html' file is selected and its content is displayed in the main editor area. The code is a simple HTML document with a title 'Simple AWS web app for case Study' and a body containing a heading 'Welcome to my Adv devops case study!'.

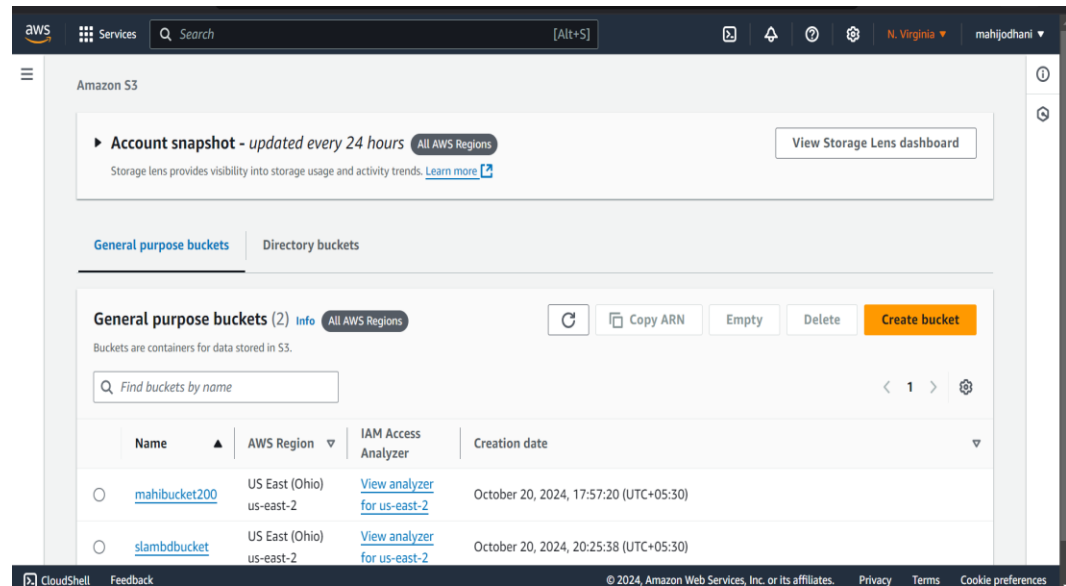
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>
5       Simple AWS web app for case Study
6     </title>
7   </head>
8   <body>
9     <h1>Welcome to my Adv devops case study!</h1>
10  </body>
11 </html>
```

This file will serve as the web page deployed to your S3 bucket and later to the EC2 instance.

## 2. Set Up S3 Bucket for Web App Hosting

### 1. Go to the AWS S3 Console:

- Open the [S3 console](#).



- Create a new S3 bucket, giving it a unique name (e.g., **my-s3-web-bucket**).

### General configuration

AWS Region  
US East (N. Virginia) us-east-1

Bucket type [Info](#)

☒ **General purpose**  
 Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ **Directory**  
 Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)

mahi-bucket-advdevops

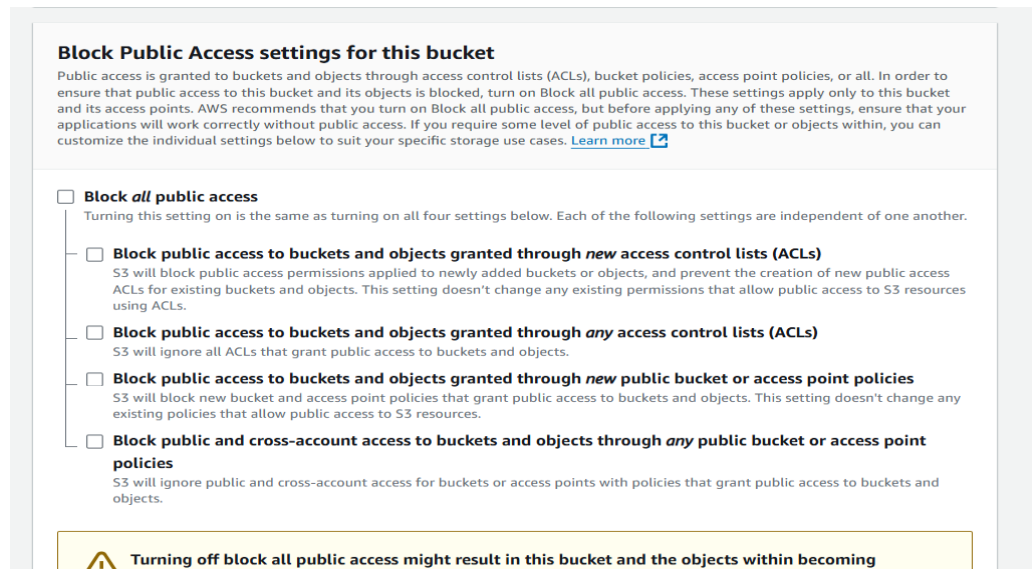
Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - *optional*  
 Only the bucket settings in the following configuration are copied.

**Choose bucket**

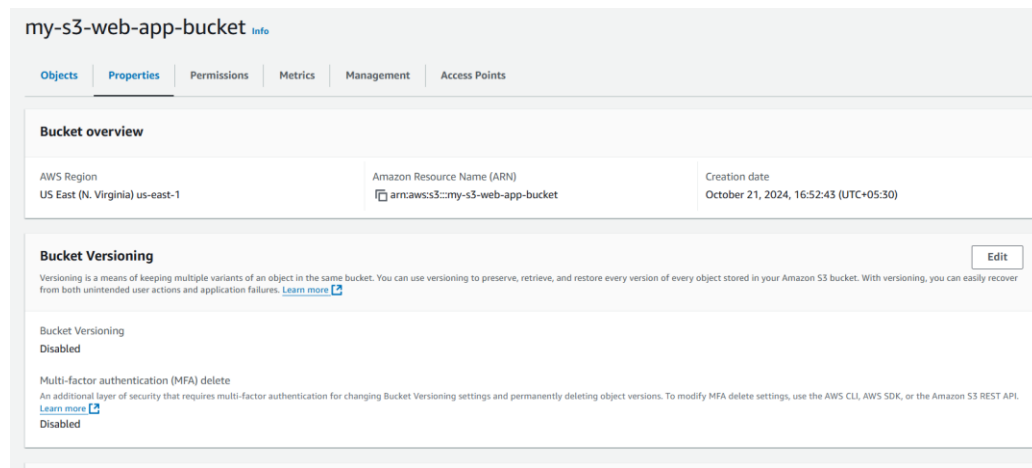
Format: s3://bucket/prefix

- Under **Permissions**, uncheck the "Block all public access" option, allowing public access for web hosting.

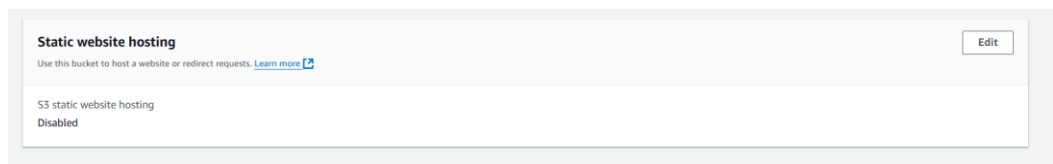


## 2. Configure the Bucket for Website Hosting:

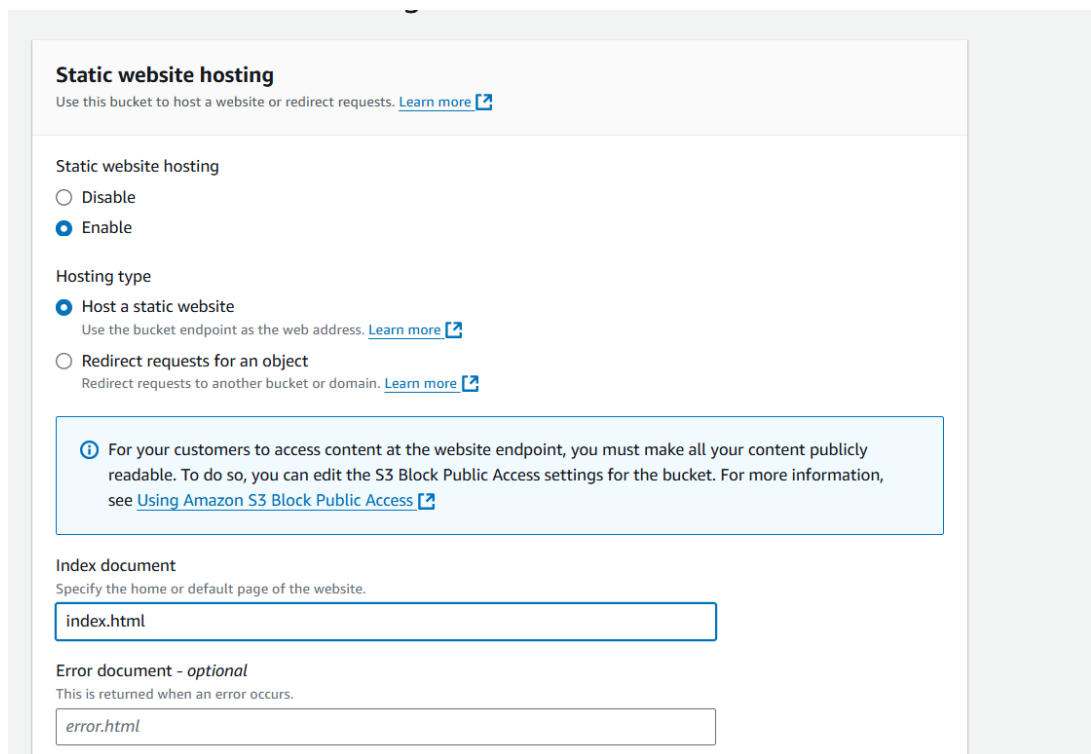
- Go to the **Properties** tab of your S3 bucket.



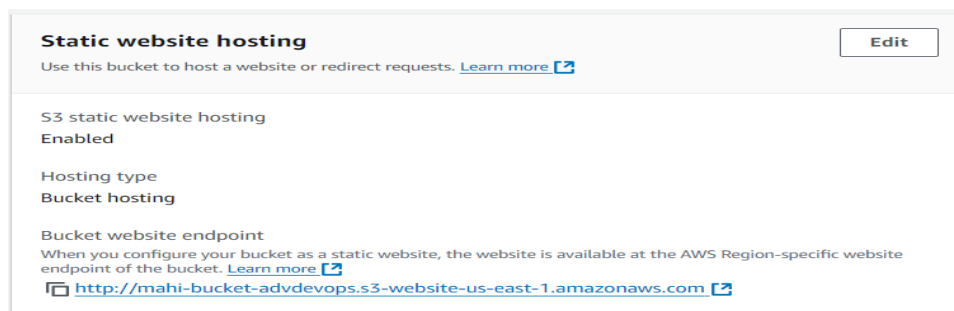
- Scroll down to **Static website hosting**.



- Enable it, and set the **Index document** as `index.html`.



- Copy the bucket website URL for testing the web app later.

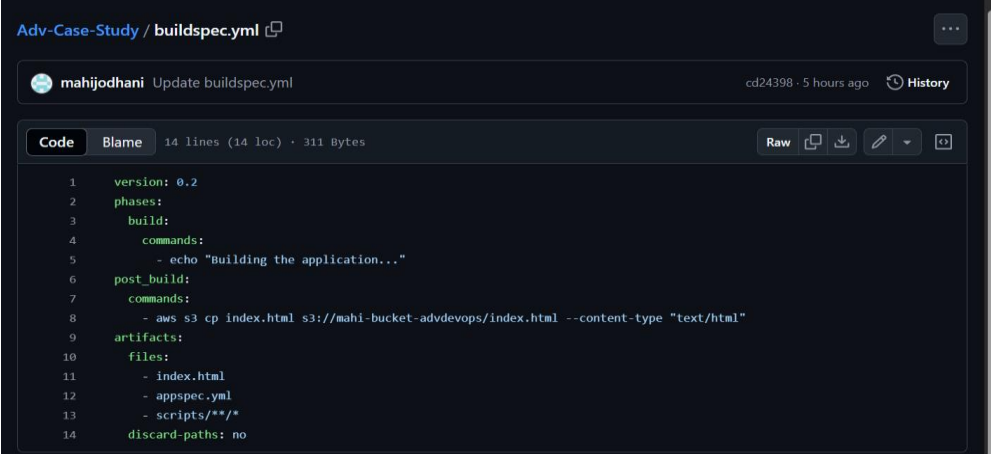


in my case it is: <http://mahi-casestudy-bucket.s3-website-us-east-1.amazonaws.com>

- if s3 website shows 403 forbidden, its a IAM permission issue.

### 3. Set Up CodeBuild for Your Web App

**Create a Buildspec File:** In your project directory (where `index.html` resides), create a `buildspec.yml` file. This file tells AWS CodeBuild what to do during the build.



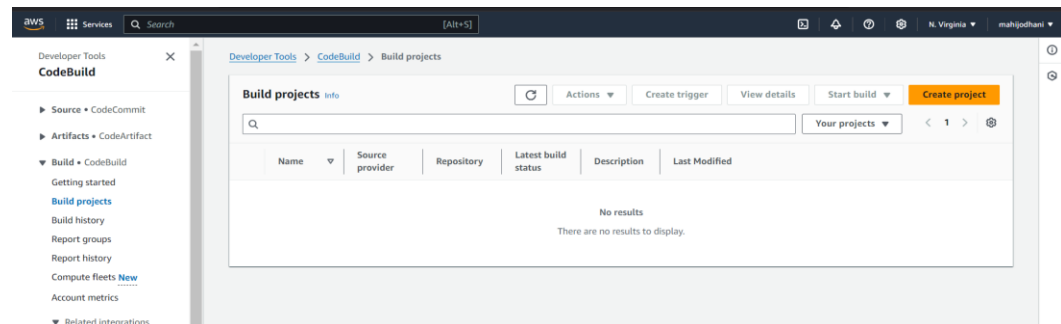
The screenshot shows the AWS CodeBuild console interface. At the top, it says "Adv-Case-Study / buildspec.yml". Below that, it shows the user "mahijodhani" and the action "Update buildspec.yml" from "cd24398 · 5 hours ago". The file content is as follows:

```

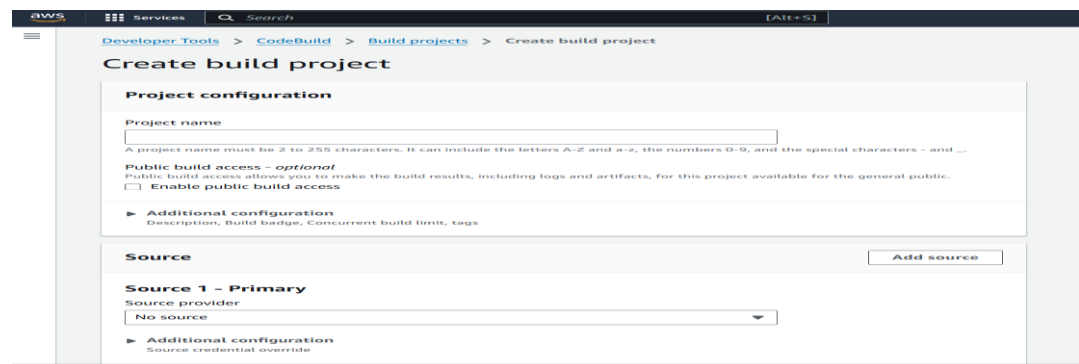
1 version: 0.2
2 phases:
3   build:
4     commands:
5       - echo "Building the application..."
6   post_build:
7     commands:
8       - aws s3 cp index.html s3://mahi-bucket-advdevops/index.html --content-type "text/html"
9   artifacts:
10    files:
11      - index.html
12      - appspec.yml
13      - scripts/**/*
14    discard-paths: no
  
```

**Go to AWS CodeBuild:**

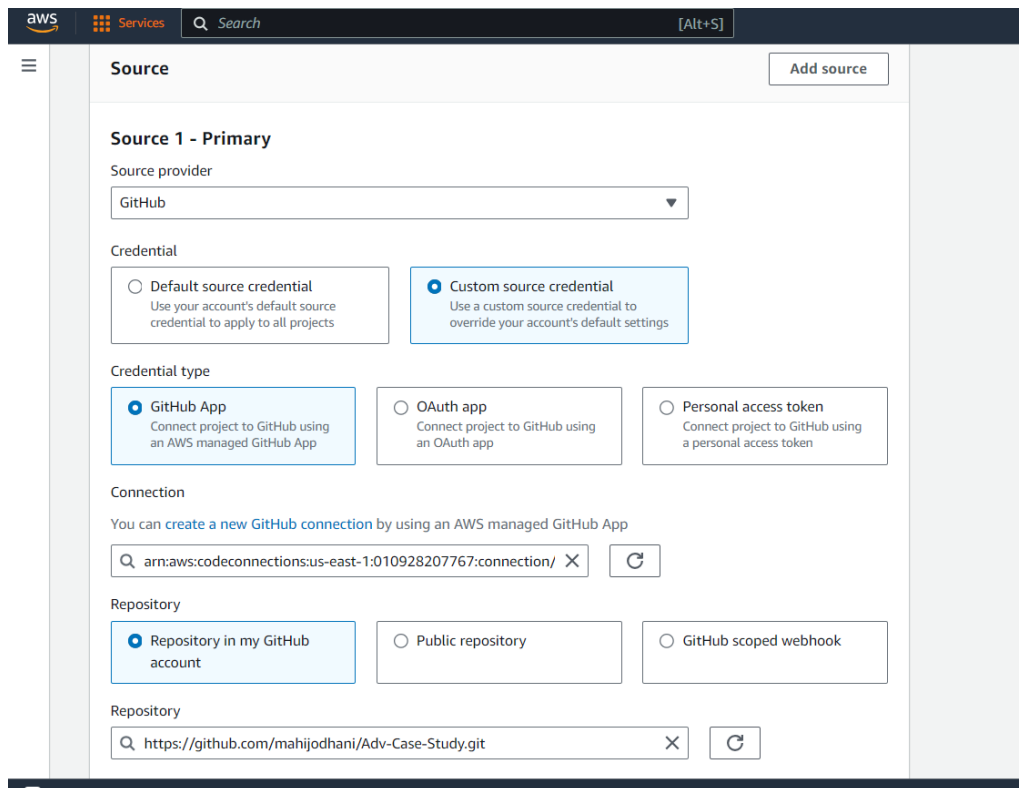
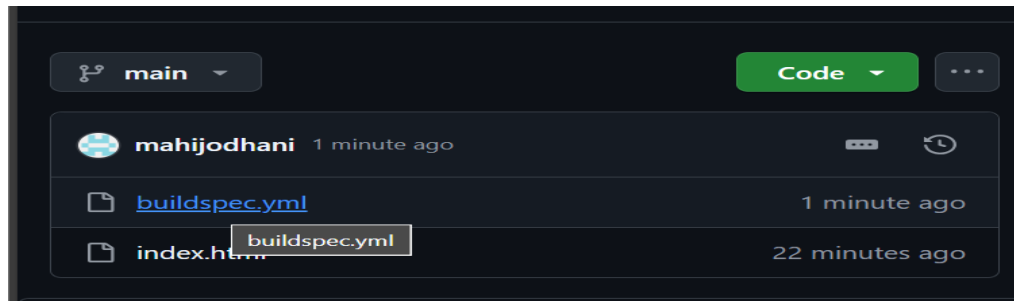
- Open the [AWS CodeBuild console](#).



- Create a new build project.



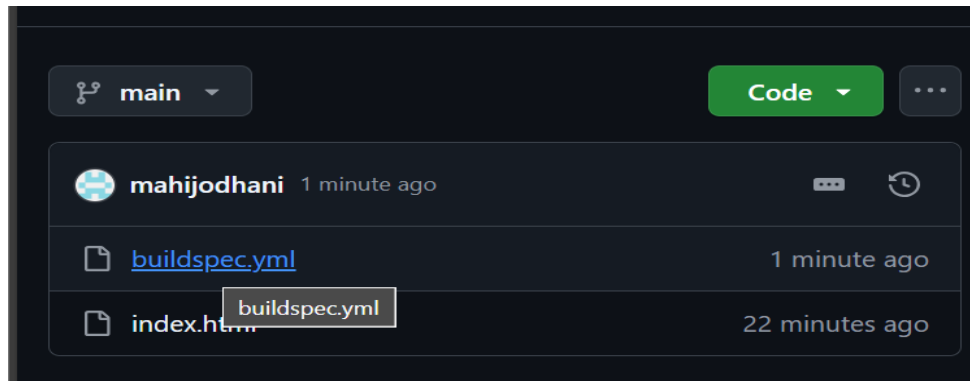
- For **Source**, choose your source repository (e.g., GitHub, Bitbucket, or S3).



- Under **Environment**, choose **Managed Image with Ubuntu**, and select **Runtime: Standard**.



- specify the `buildspec.yml` file you created. before this add it to your git repo



### Buildspec

**Build specifications**

☐ Insert build commands  
 Store build commands as build project configuration

☒ Use a buildspec file  
 Store build commands in a YAML-formatted buildspec file

**Buildspec name - optional**  
 By default, CodeBuild looks for a file named `buildspec.yml` in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, `buildspec-two.yml` or `configuration/buildspec.yml`).

`buildspec.yml`

- Set **Artifacts** to "S3", and choose the bucket you created earlier.

### Artifacts

[Add artifact](#)

**Artifact 1 - Primary**

Type  
  
You might choose no artifacts if you are running tests or pushing a Docker image to Amazon ECR.

Bucket name

Name  
The name of the folder or compressed file in the bucket that will contain your output artifacts. Use Artifacts packaging under Additional configuration to choose whether to use a folder or compressed file. If the name is not provided, defaults to project name.

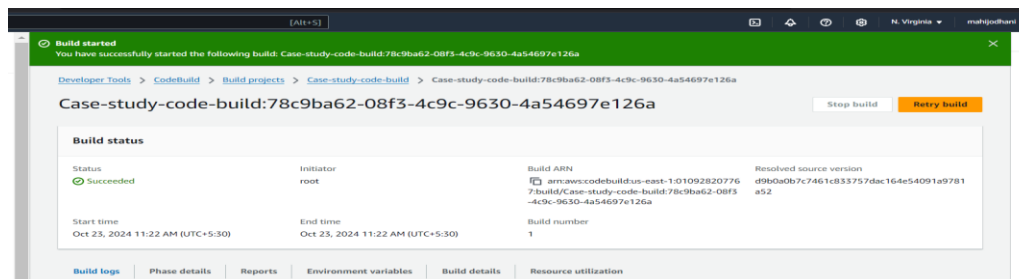
☐ Enable semantic versioning  
Use the artifact name specified in the buildspec file

Path - optional  
The path to the build output ZIP file or folder.

Example: MyPath/MyArtifact.zip.

Namespace type - optional

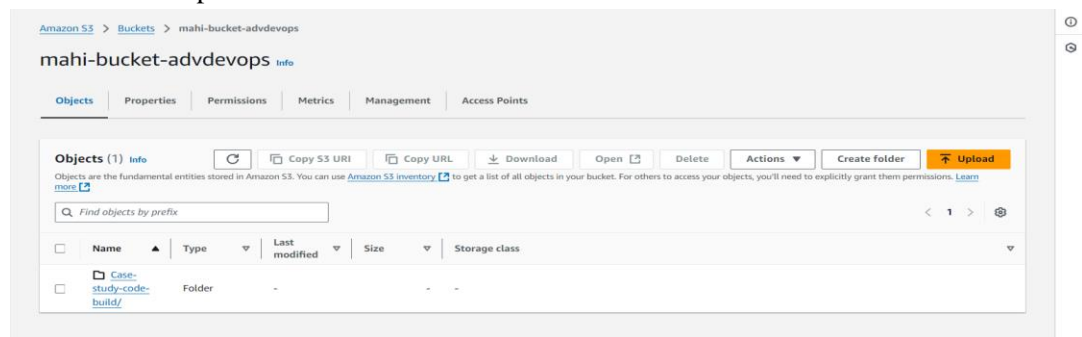
- Create the build project and start the build to ensure it uploads `index.html` to the S3 bucket.







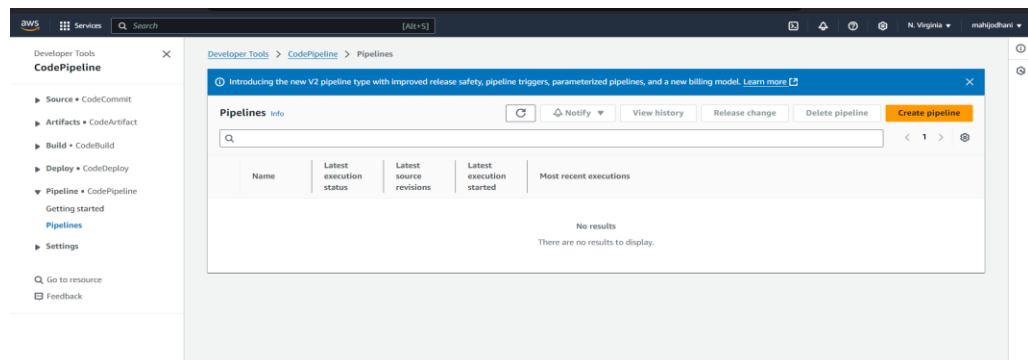
s3 bucket was updated:



## 4. Set Up AWS CodePipeline

### 1. Go to AWS CodePipeline:

- Open the [CodePipeline console](#).



- Create a new pipeline.
- Choose build custom pipeline.

### Choose creation option info

Step 1 of 4

**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**

○ For **Source**:

■ Select your repository (e.g., GitHub, Bitbucket).

The screenshot shows the AWS CodePipeline console interface. The breadcrumb trail is: Developer Tools > CodePipeline > Pipelines > Create new pipeline. The left sidebar shows the steps: Step 1: Choose creation option, Step 2: Choose template, Step 3: Choose source (active), and Step 4: Configure template. The main content area is titled 'Choose source' with a sub-header 'Step 3 of 4'. Under the 'Source' section, the 'Source provider' is set to 'GitHub (Version 2)'. A blue box highlights the 'New GitHub version 2 (app-based) action' with instructions to create a connection. The 'Connection' section shows an existing connection ID 'arn:aws:codeconnections:us-east-1:010928207767:connection/11c47f5e-b9...' and a 'Connect to GitHub' button. The 'Repository name' is set to 'https://github.com/mahijodhani/Adv-Case-Study'.

■ Connect and choose the appropriate branch where the **index.html** and **buildspec.yml** files are.

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

Q main X

main

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

☐ **Full clone**  
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only

## 2. Add Build Stage:

- In the **Build stage**, choose **AWS CodeBuild** as the build provider.

**Build - optional**

**Build provider**  
Choose the tool you want to use to run build commands and specify artifacts for your build action.

☐ Commands
 ☒ Other build providers

AWS CodeBuild ▼

- Select the CodeBuild project you created earlier.

aws Services [Search] [Alt+S] N. Virginia mahjodhani

Step 3: Add build stage

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

**Pipeline type**  
You can no longer create V1 pipelines through the console. We recommend you use the V2 pipeline type with improved release safety, pipeline triggers, parameterized pipelines, and a new billing model.

**Execution mode**  
Choose the execution mode for your pipeline. This determines how the pipeline is run.

☐ Superseded  
A more recent execution can overtake an older one. This is the default.
 ☒ Queued (Pipeline type V2 required)  
Executions are processed one by one in the order that they are queued.
 ☐ Parallel (Pipeline type V2 required)  
Executions don't wait for other runs to complete before starting or finishing.

**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

## 3. Deploy to S3:

- In the next stage, choose **Deploy**. Select **Amazon S3**. Choose your S3 bucket (**my-s3-bucket**) where the **index.html** file will be deployed.

Step 2: Choose pipeline settings

Step 3: Add source stage

Step 4: Add build stage

Step 5: Add deploy stage

Step 6: Review

**Deploy - optional**

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

Amazon S3 ▼

**Region**  
Asia Pacific (Sydney) ▼

**Input artifacts**  
Choose an input artifact for this action. [Learn more](#)

BuildArtifact x  
Defined by: Build

No more than 100 characters

**Bucket**  
my-s3-web-bucket x

**S3 object key**  
/case-study-code-build/index.html  
Enter the object key. You can include a file path without the delimiter character (/) at the beginning. Include the file extension. Example: SampleApp.zip

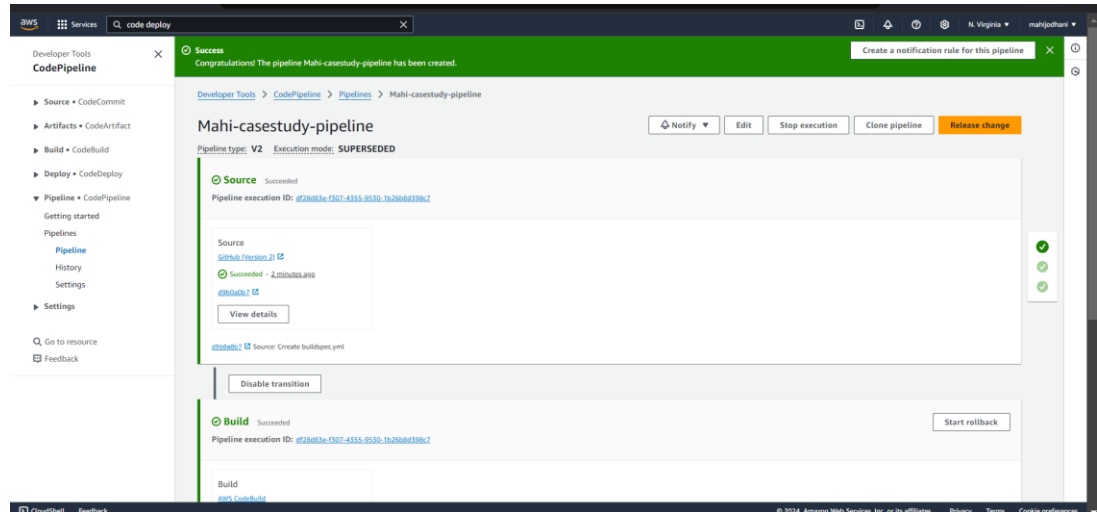
☐ Extract file before deploy  
The deployed artifact will be unzipped before deployment.

► Additional configuration

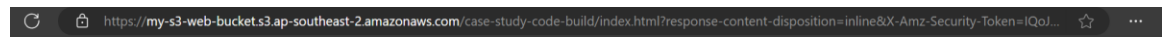
☒ Configure automatic rollback on stage failure

#### 4. Test the Pipeline:

- Once the pipeline is set up, click **Release Change** to start the pipeline. This should fetch the latest code, build it, and upload **index.html** to the S3 bucket.



- Visit the S3 bucket's website URL to verify that the **index.html** page is live.

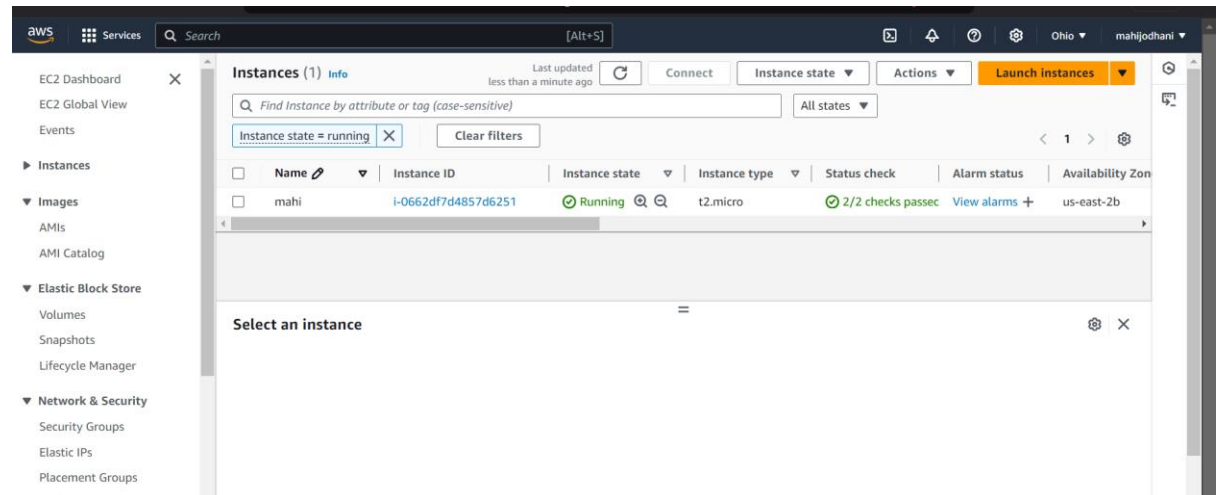


Welcome to my Adv devops case study!

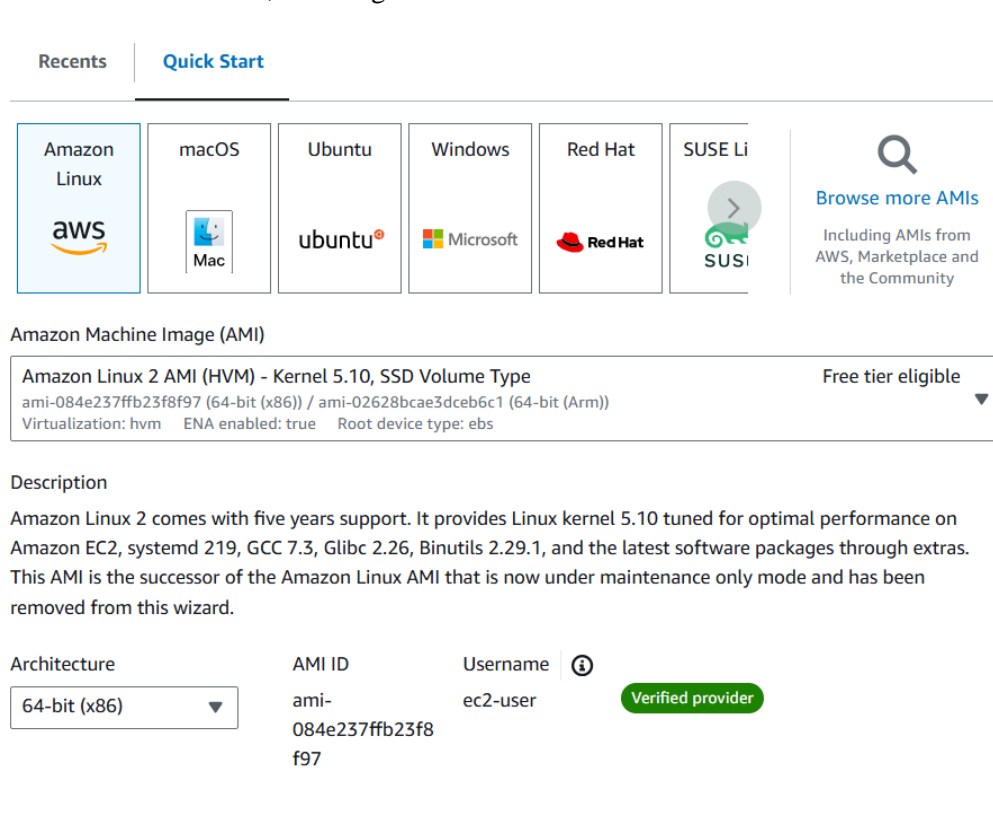
## 5. Set Up EC2 Instance for Web Hosting

### 1. Launch an EC2 Instance:

- Open the [EC2 console](#).



- Launch a new instance, selecting an Amazon Linux 2 AMI.



- Choose the default t2.micro instance type.

- Configure instance settings and storage (use defaults for now).
- In **Configure Security Group**, allow HTTP traffic by adding a rule to open port 80.

The screenshot shows the AWS Management Console interface for configuring an Amazon EC2 instance. The top navigation bar includes 'Services', a search bar, and a keyboard shortcut '[Alt+S]'. The main content area is titled 'Network settings' with an 'Info' link and an 'Edit' button.

**Network settings:**

- Network:** vpc-030c6ebc1841ce4bb
- Subnet:** No preference (Default subnet in any availability zone)
- Auto-assign public IP:** Enable
- Additional charges apply** when outside of **free tier allowance**
- Firewall (security groups):** Info. A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Below the firewall section, there are two radio buttons: 'Create security group' (selected) and 'Select existing security group'.

A message states: 'We'll create a new security group called 'launch-wizard-1' with the following rules:'

- ☒ **Allow SSH traffic from** (Helps you connect to your instance). Source: Anywhere (0.0.0.0/0).
- ☐ **Allow HTTPS traffic from the internet** (To set up an endpoint, for example when creating a web server).
- ☒ **Allow HTTP traffic from the internet** (To set up an endpoint, for example when creating a web server).

A yellow warning box at the bottom of the network settings section states: '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.'

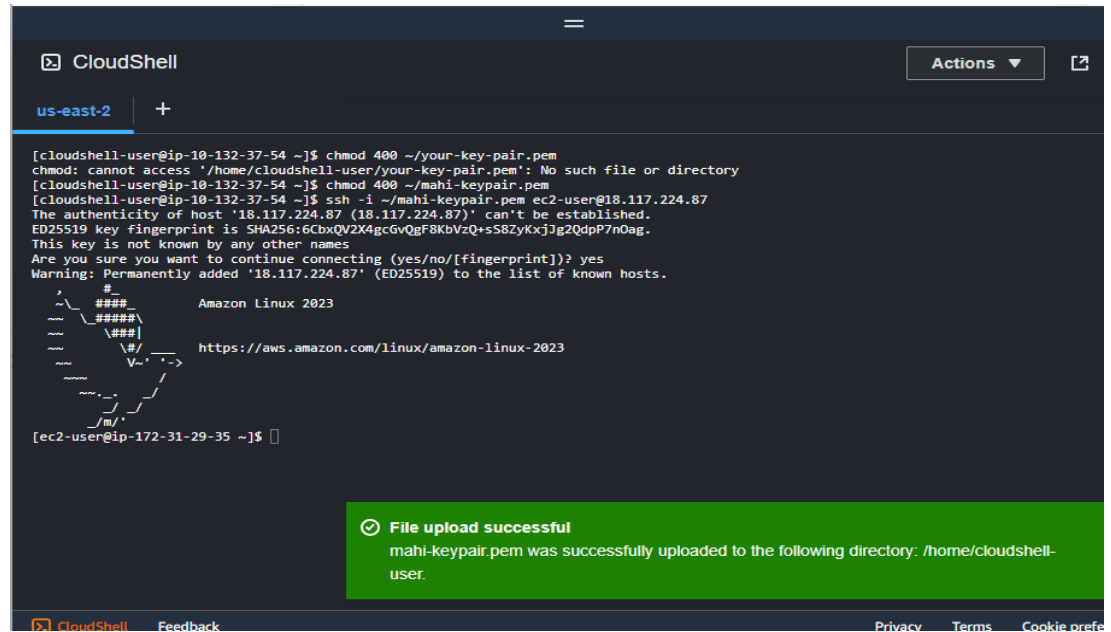
The bottom section of the console shows the 'Instance type' and 'Summary' tabs. The 'Instance type' tab is selected, showing details for the 't2.micro' instance type, which is 'Free tier eligible'. The 'Summary' tab shows the 'Number of instances' as 1, the 'Software Image (AMI)' as 'Amazon Linux 2023 AMI', and the 'Virtual server type (instance type)' as 't2.micro'. A 'Free tier' notice is also visible, stating that the first year includes 750 hours of t2.micro usage.

## 2. Connect to the EC2 Instance:

- Once the instance is running, connect via SSH.

Install the required web server (Apache) on your instance:

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
```



## 6. Set Up AWS CodeDeploy to Push Updates to EC2

### 1. Install CodeDeploy Agent on EC2:

- Connect to the EC2 instance and Install the CodeDeploy agent:

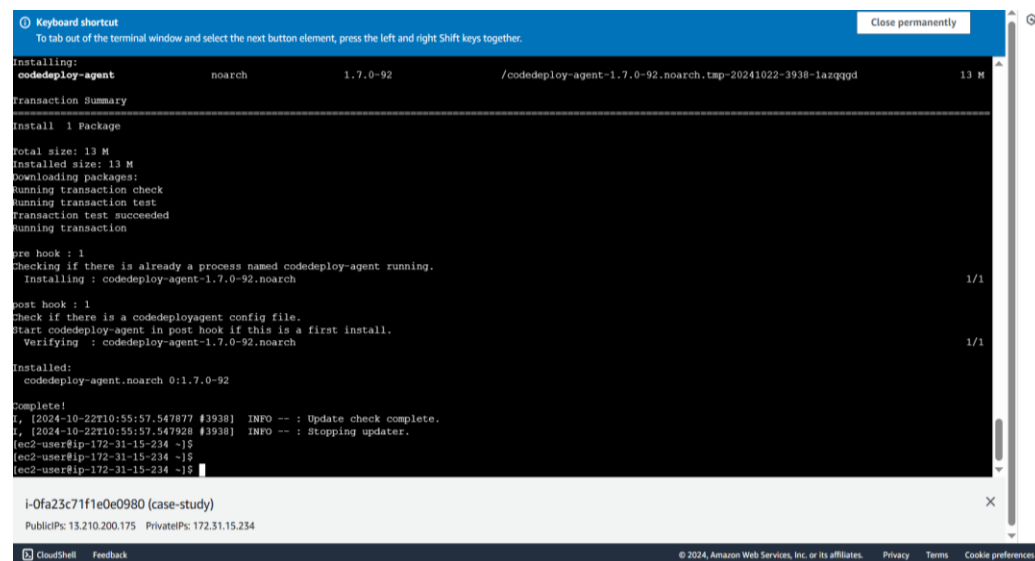
```
sudo yum update -y
```

```
sudo yum install -y ruby wget
```

```
wget https://aws-codedeploy-us-east-1.s3.amazonaws.com/latest/install
```

```
chmod +x ./install
```

```
sudo ./install auto
```



```

[ec2-user@ip-172-31-29-35 ~]$ sudo yum update -y
Last metadata expiration check: 0:13:34 ago on Wed Oct 23 16:15:04 2024.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-29-35 ~]$ sudo yum install httpd -y
Last metadata expiration check: 0:14:04 ago on Wed Oct 23 16:15:04 2024.
Dependencies resolved.
=====
Package              Size      Architecture Version      Repository
=====
Installing:
httpd                x86_64    2.4.62-1.amzn2023      amazonlinux 48 k
Installing dependencies:
apr                  x86_64    1.7.2-2.amzn2023.0.2    amazonlinux 129 k
apr-util             x86_64    1.6.3-1.amzn2023.0.1    amazonlinux 98 k
generic-iconv-httpd noarch    18.0-9.52.amzn2023.0.3  amazonlinux 20 k
httpd-core           x86_64    2.4.62-1.amzn2023      amazonlinux 1.4 M
httpdfilesystem      x86_64    2.4.62-1.amzn2023      amazonlinux 34 k
httpd-tools          x86_64    2.4.62-1.amzn2023      amazonlinux 81 k
libtool               x86_64    1.8.9-4.amzn2023.0.2    amazonlinux 315 k
mailcap              noarch    2.1.49-3.amzn2023.0.3   amazonlinux 33 k
Installing weak dependencies:
apr-util-openssl     x86_64    1.6.3-1.amzn2023.0.1    amazonlinux 17 k
mod_http2            x86_64    2.0.27-1.amzn2023.0.3   amazonlinux 166 k
mod_lua              x86_64    2.4.62-1.amzn2023      amazonlinux 61 k
=====
Transaction Summary
=====
Install      1 Package
Total size: 1.4 M

```

## 2. Set Up CodeDeploy Application:

### 3.1 Create Application

1. Using AWS CLI from Local Terminal on pc (not ec2, use your own pc's command line):

*# First ensure AWS CLI is installed*

**aws --version**

if not installed, install from <https://awscli.amazonaws.com/AWSCLIV2.msi>

**aws configure**

2. Create application:

**aws deploy create-application --application-name my-webapp**

### 3.2 Create Deployment Group

- Create deployment group:

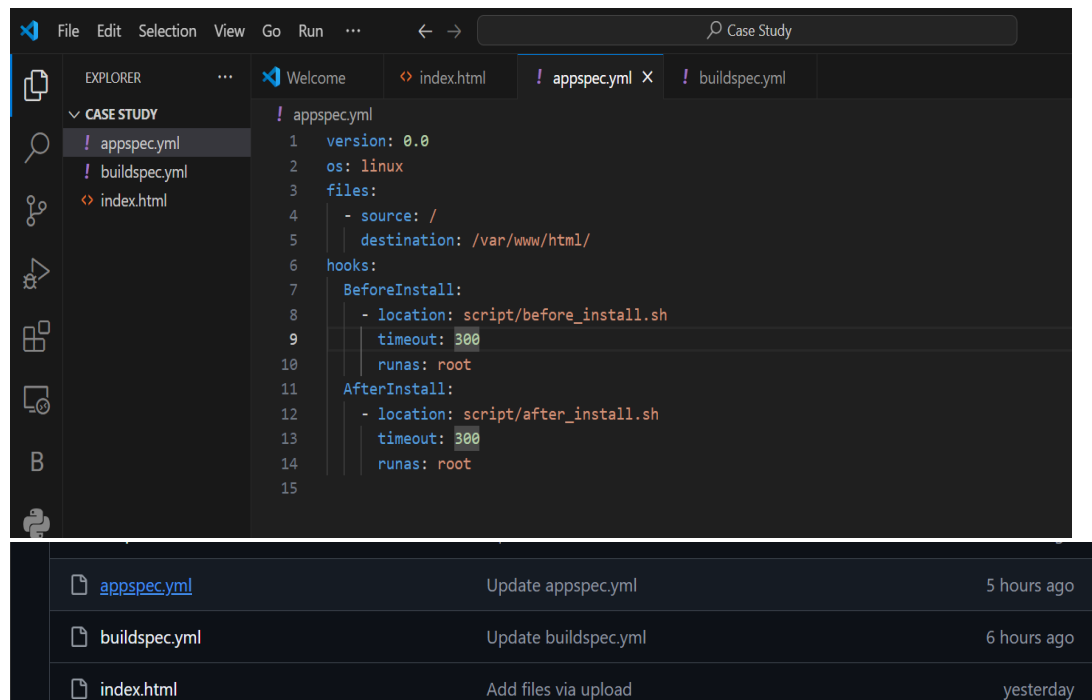
**aws deploy create-deployment-group --application-name my-webapp --deployment-group-name my-webapp-group --service-role-arn**

**arn:aws:iam::ACCOUNT\_ID:role/CodeDeployServiceRole**

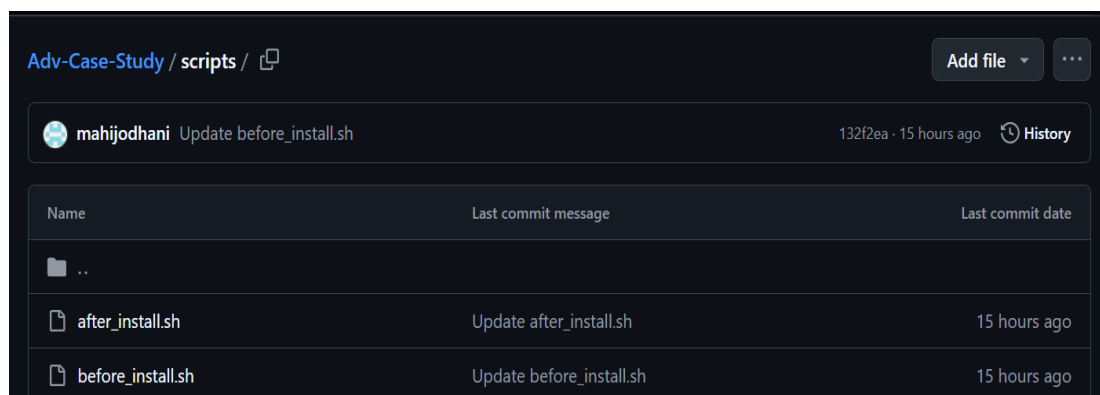
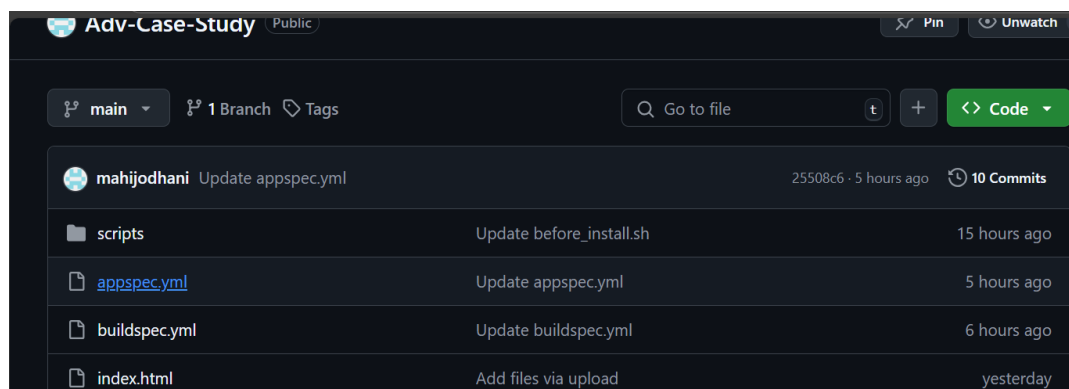
- enter your ACCOUNT\_ID above!!

**Create `appspec.yml` for CodeDeploy:** In your project folder, create an `appspec.yml` file to specify how CodeDeploy should handle the deployment:





Now create a scripts folder inside your repo which will contain 2 files: **before\_install.sh** and **after\_install.sh**



Adv-Case-Study / scripts / **after\_install.sh**

mahijodhani Update after\_install.sh 7d26b7b · 15 hours ago History

Code Blame 7 lines (7 loc) · 263 Bytes

```

1  #!/bin/bash
2  echo "Starting After Install"
3  sudo cp -r /opt/codedeploy-agent/deployment-root/* /deployment-archive/* /var/www/html/
4  sudo chmod -R 755 /var/www/html
5  sudo chown -R apache:apache /var/www/html
6  sudo systemctl restart httpd
7  echo "After Install completed"

```

Adv-Case-Study / scripts / **before\_install.sh**

mahijodhani Update before\_install.sh 132f2ea · 15 hours ago History

Code Blame 7 lines (7 loc) · 175 Bytes

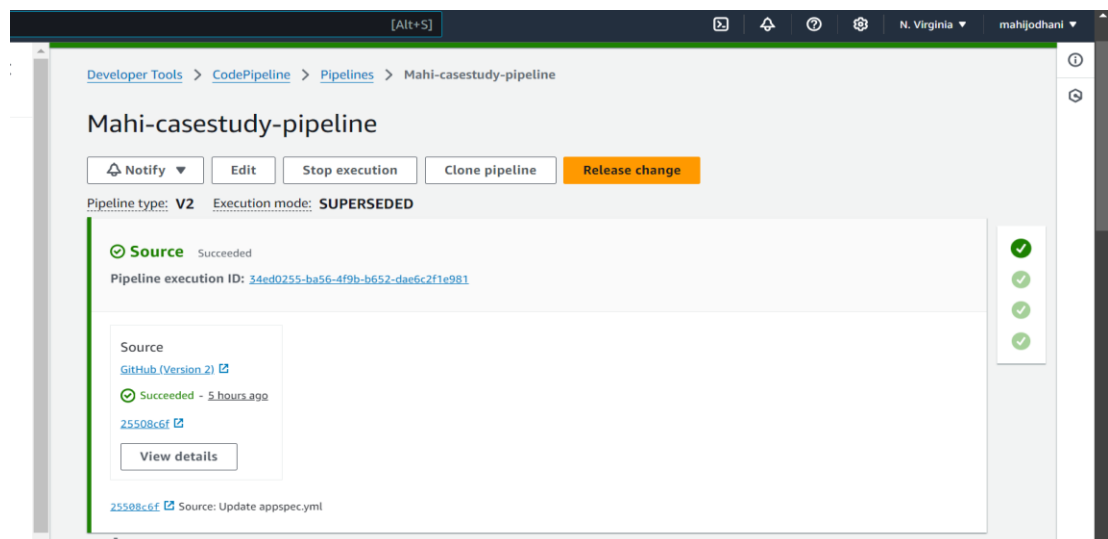
```

1  #!/bin/bash
2  echo "Starting Before Install"
3  sudo yum update -y
4  sudo yum install -y httpd
5  sudo systemctl start httpd
6  sudo systemctl enable httpd
7  echo "Before Install completed"

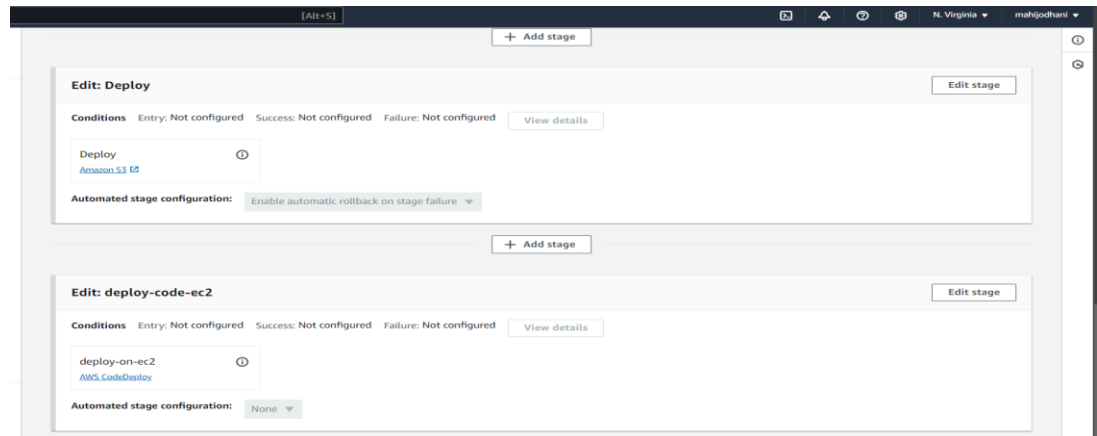
```

### 3. Add Deployment Stage to CodePipeline:

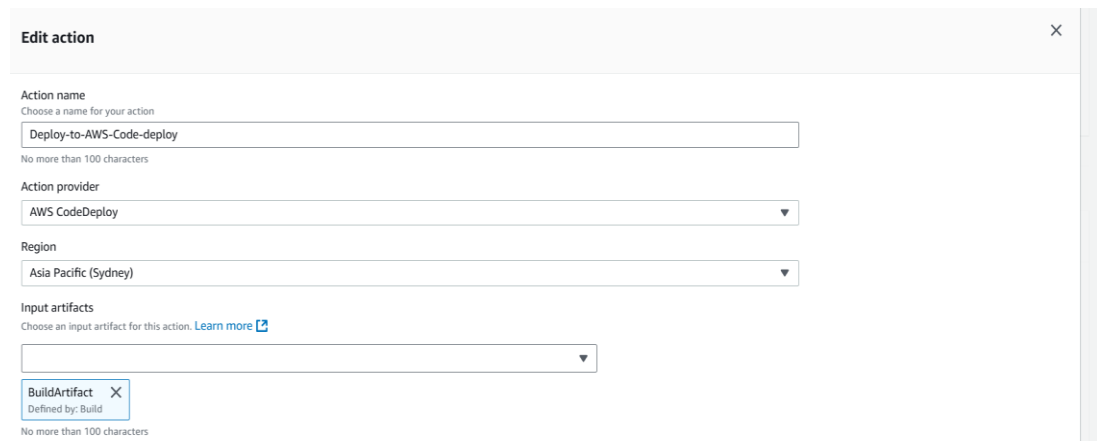
- Go back to your CodePipeline.



- Add a new stage for deployment.

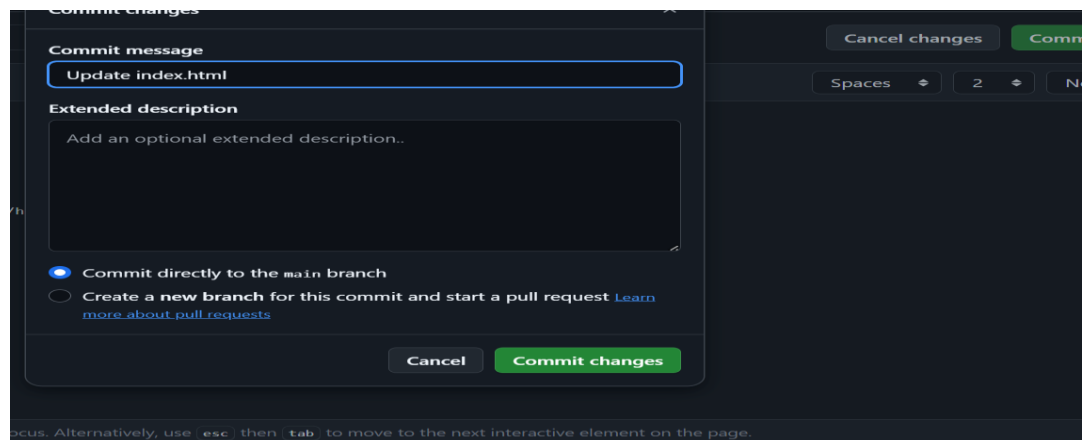


- Select **AWS CodeDeploy** and choose the application and deployment group you created earlier.

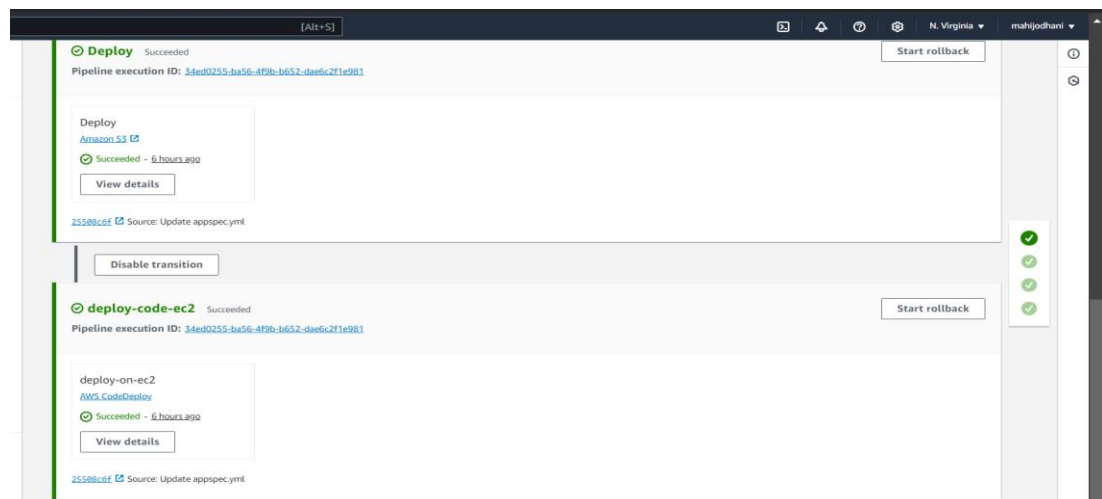
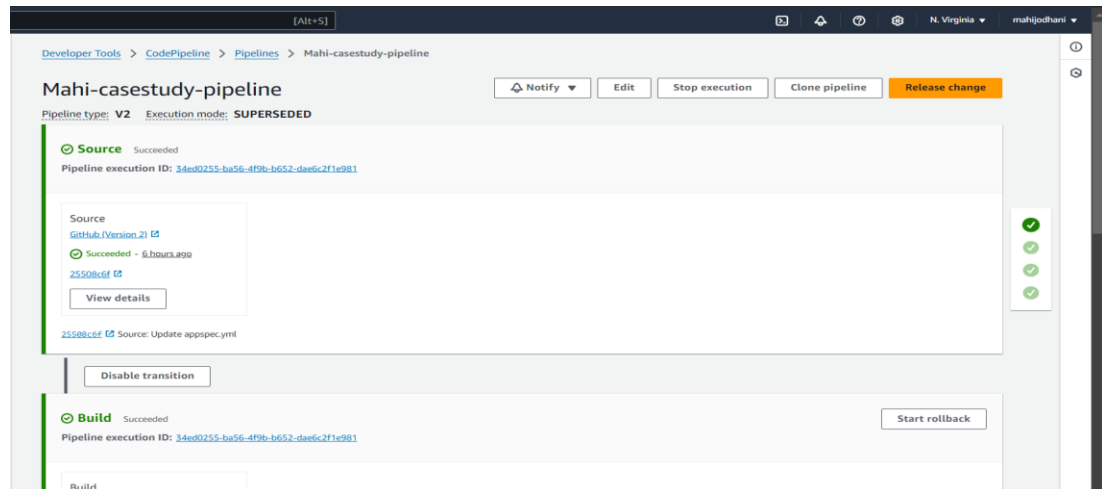


#### 4. Test Deployment:

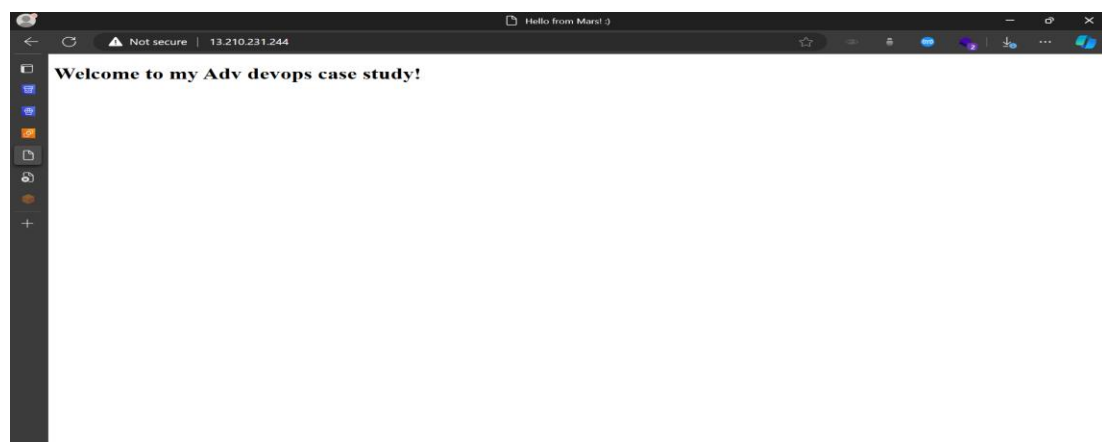
- Make a change to **index.html** in your source repository and push it.



- This should trigger the pipeline, rebuild the app, push it to S3, and deploy it to the EC2 instance.

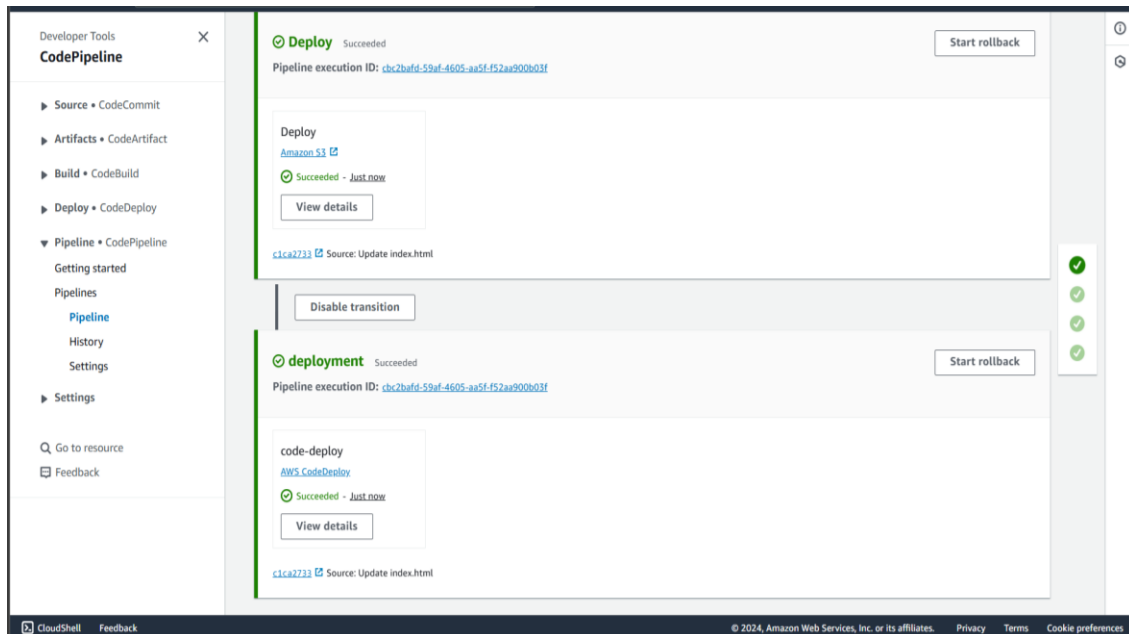
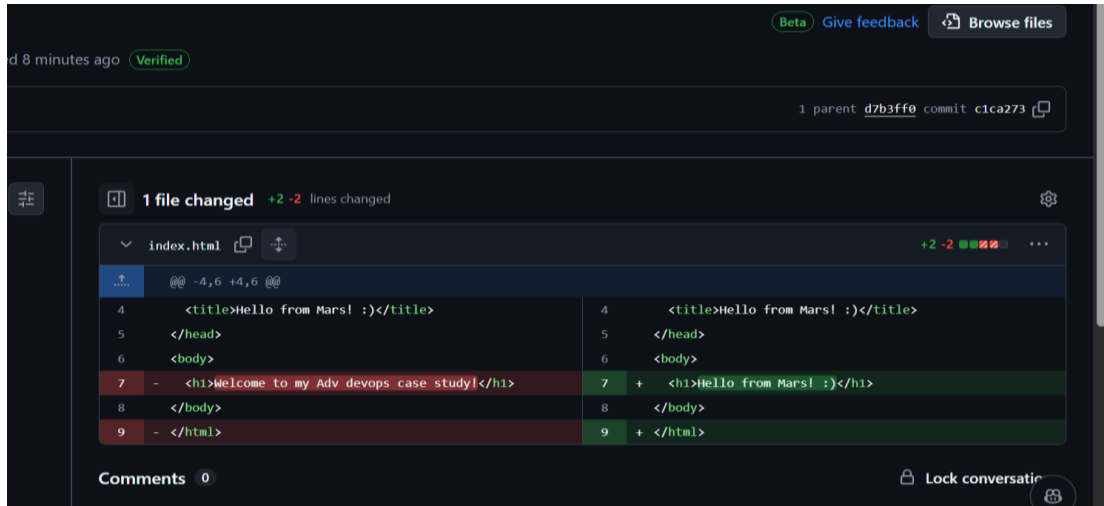


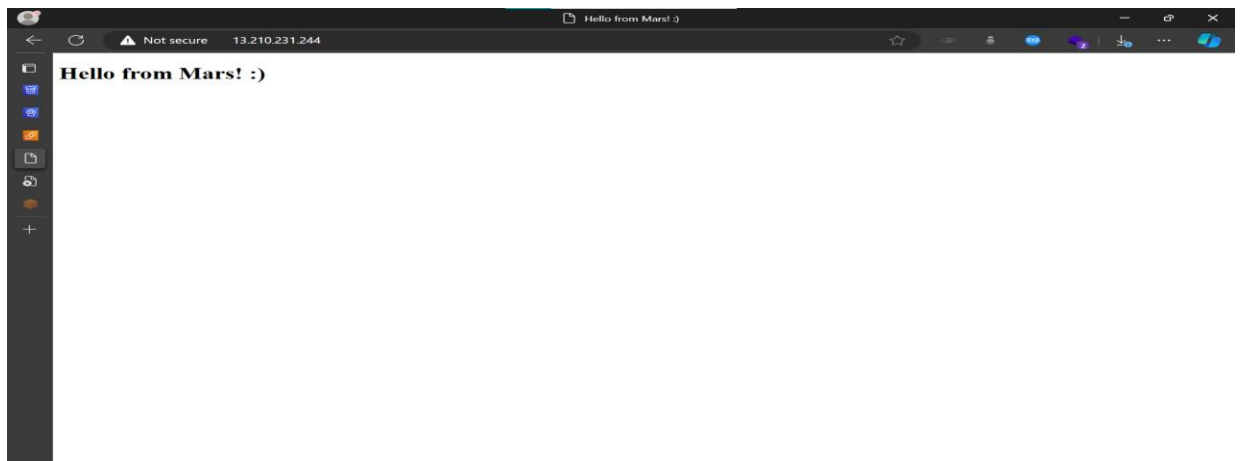
- You can access your EC2 instance via its public IP to view the updated web app.



## 7. Verify Automation

- Now that your pipeline is set up, any changes to your repository (e.g., modifying the `index.html` file) should automatically trigger the build, deploy it to S3, and push updates to your EC2 instance.





### **CHALLENGES FACED:**

1. **Permission Issues:** One of the main hurdles was setting the correct permissions in IAM roles and the S3 bucket policy. Errors like **403 Forbidden** when accessing S3 were resolved by adjusting the S3 bucket policy.
2. **Missing AppSpec File:** The CodeDeploy process failed initially because the **appspec.yml** file was not placed correctly in the root directory of the build artifacts.
3. **CodeDeploy Agent Issues:** The EC2 instance had issues with the CodeDeploy agent, such as failing to start or showing **Permission Denied** errors. These were resolved by restarting the agent and ensuring the instance had appropriate permissions to access S3.
4. **IAM:** Faced issues with insufficient permissions, such as not having the required policies attached to allow actions like **codedeploy:CreateApplication** and **iam:PassRole**.
5. **CodePipeline:** Encountered problems in configuring the deployment stage correctly, including selecting the right artifacts and troubleshooting failed pipeline executions.
6. **CodeDeploy:** Deployment failures occurred due to the deployment group not finding the tagged EC2 instances, requiring adjustments to the instance selection settings.
7. **EC2:** Issues with CodeDeploy agent not running or instances not being tagged correctly, causing deployments to fail.
8. **Security Credentials:** Issues related to missing or invalid AWS credentials (Access Key and Secret Key), which caused problems in accessing AWS services and executing CLI commands.
9. **Network Configuration:** Problems with network settings, such as security group rules or VPC configurations, that could have prevented successful communication between services like EC2 and CodeDeploy.

Despite these challenges, the overall deployment was successful, demonstrating the effectiveness of AWS's automation tools in managing continuous deployment workflows.

### **CONCLUSION:**

This experiment successfully demonstrated the process of building and deploying a simple web application using AWS CodePipeline, CodeBuild, CodeDeploy, and EC2. The pipeline automates the workflow from the moment the code is pushed to a repository until the application is deployed on an EC2 instance. In this case study, we focused on automating cloud deployment using several AWS services, mainly AWS CodePipeline, EC2, S3, and CodeDeploy. Our goal was to build a simple web application

with a sample index.html page and show how these services can work together for an easy deployment process. We started by creating an S3 bucket for hosting our static website. After setting it up, we uploaded our HTML file and made sure the permissions allowed public access so users could visit the website. Next, we used AWS CodeBuild to create a build project that builds our application code. This step is must because it automates the build process, making sure that only our code is deployed. Then, we set up AWS CodePipeline, which helps us manage the whole deployment flow. We connected the source stage to our S3 bucket and the build stage to our CodeBuild project, making the process easier. We then launched an EC2 instance to host our application. We created an IAM role to give the instance the right permissions to access S3 and CodeDeploy services. After installing the CodeDeploy agent on the EC2 instance, we were able to manage deployments easily. We created a CodeDeploy application and a deployment group linked to our EC2 instance. Finally, we started a deployment to push our application to the EC2 instance and checked that everything was working properly. Overall, this case study showed how effective AWS tools can be for automating deployment.