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.

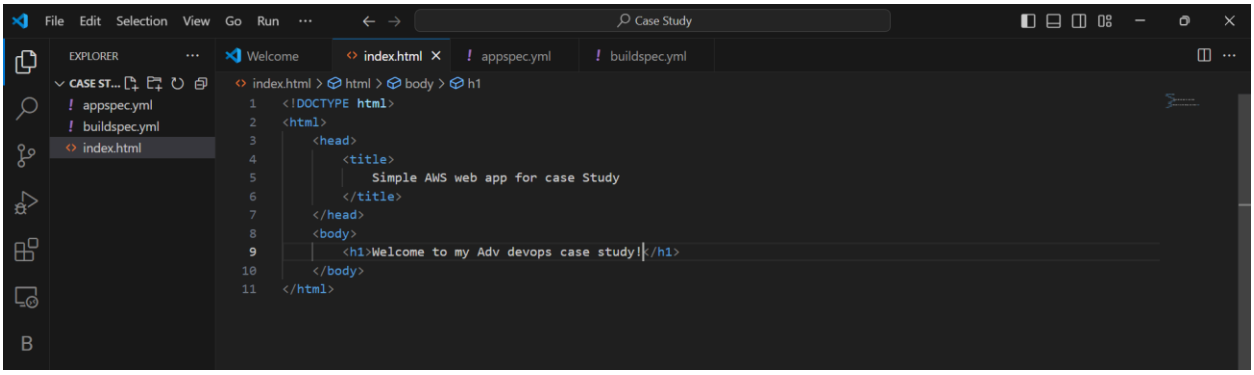
Pipeline Stages:

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.

<u>PROCEDURE & SCREENSHOTS:</u>

Create a Simple Web App

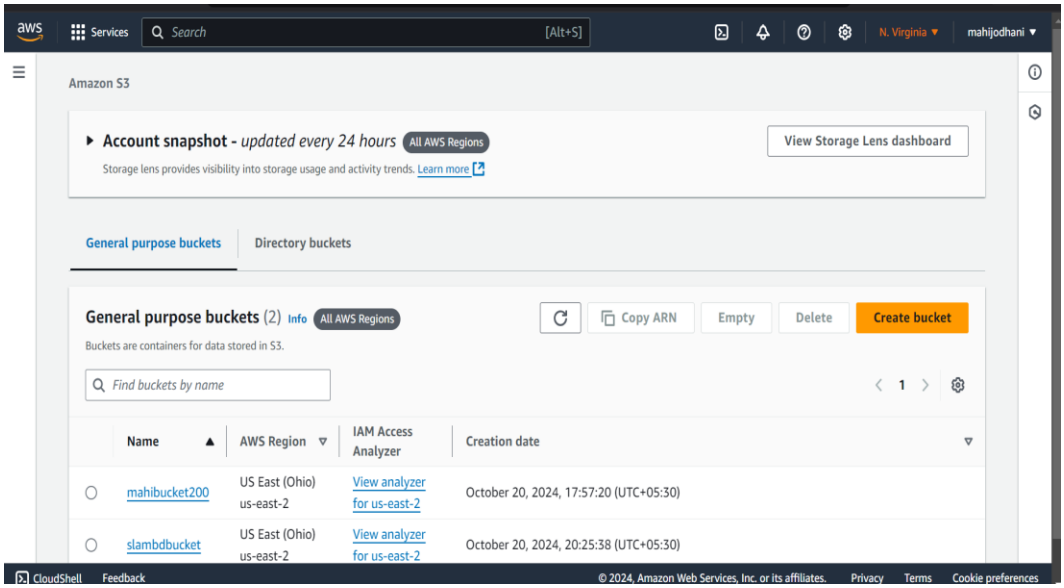
1. First, create a simple web app with an index.html file:



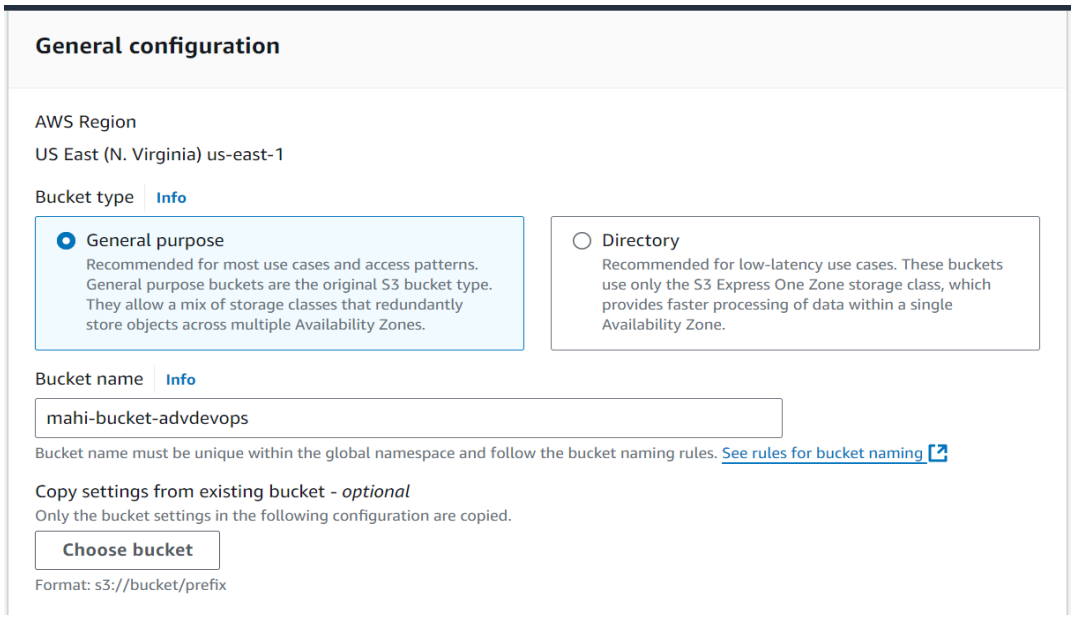
This file will serve as the web page deployed to your S3 bucket and later to the EC2 instance.

2. Set Up S3 Bucket for Web App Hosting

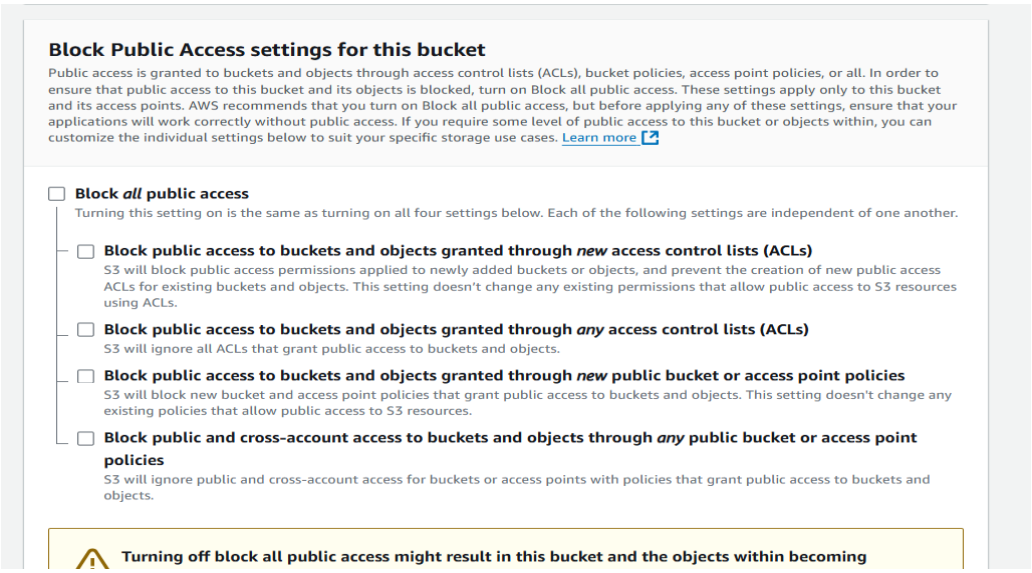
1. **Go to the AWS S3 Console**:
   - Open the S3 console.



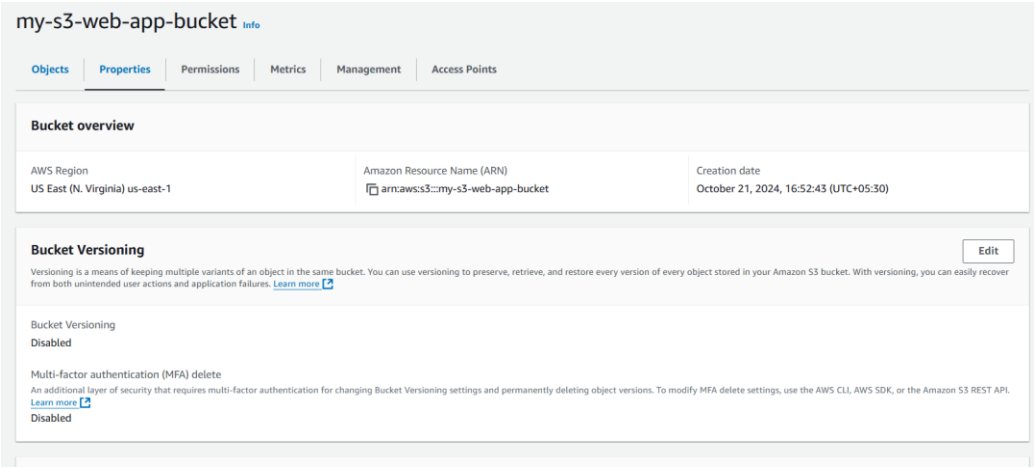
   - Create a new S3 bucket, giving it a unique name (e.g., my-s3-web-bucket).



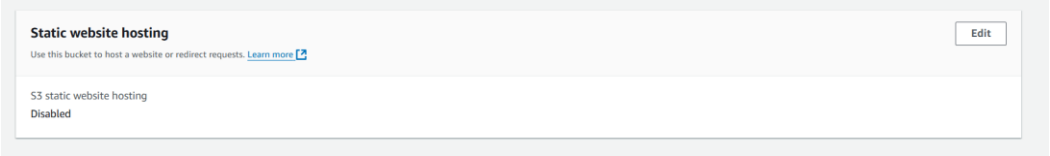
- Under **Permissions**, uncheck the "Block all public access" option, allowing public access for web hosting.



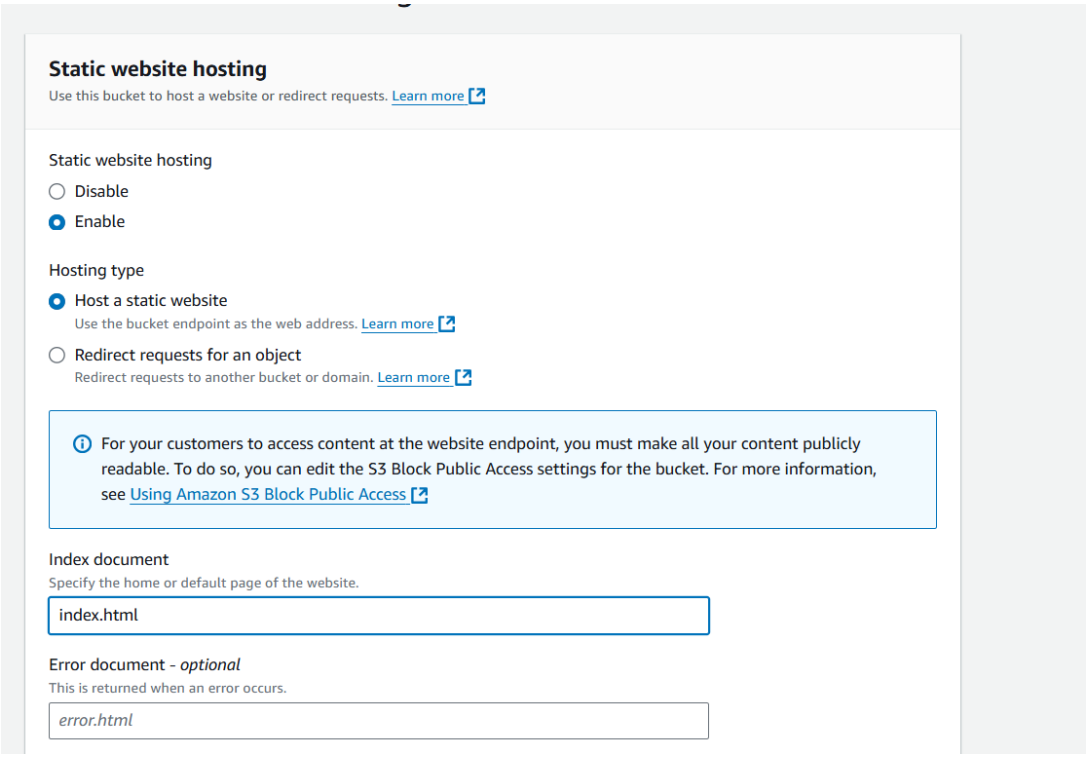
2. **Configure the Bucket for Website Hosting**:
   - Go to the **Properties** tab of your S3 bucket.



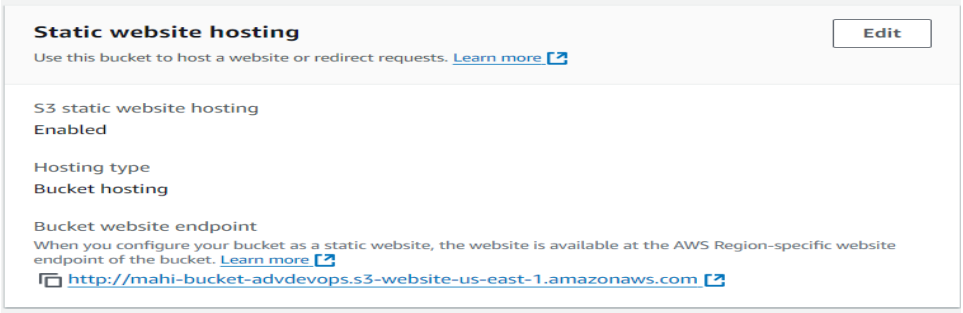
   - Scroll down to **Static website hosting**.



- 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

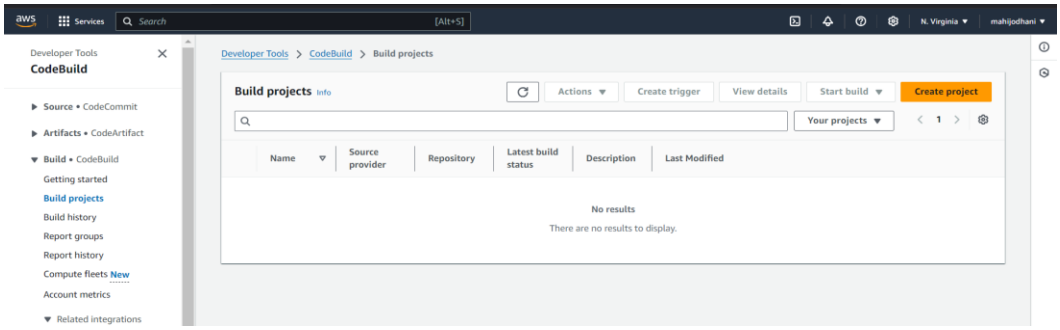
- if s3 website shows 403 forbidden, its a IAM permission issue.

3. Set Up CodeBuild for Your Web App

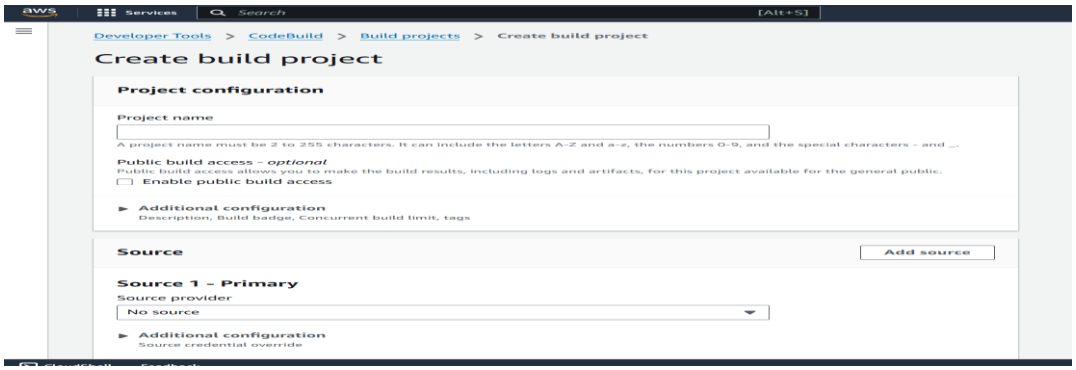
Create a Buildspec File: In your project directory (where index.html resides), create a buildspec.yml file. This file tells AWS CodeBuild what to do during the build.

Go to AWS CodeBuild:

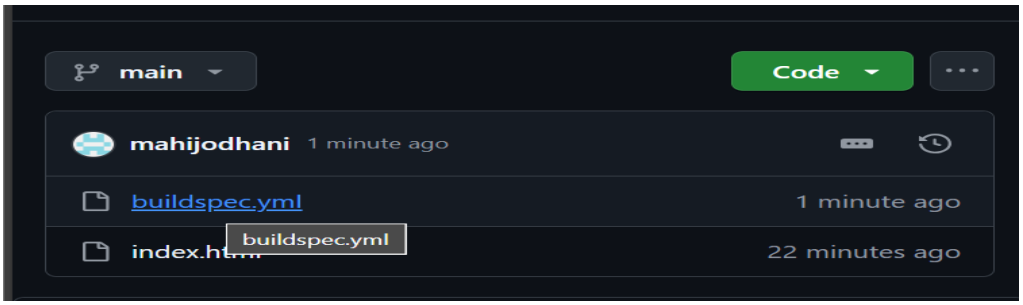
- Open the AWS CodeBuild console.

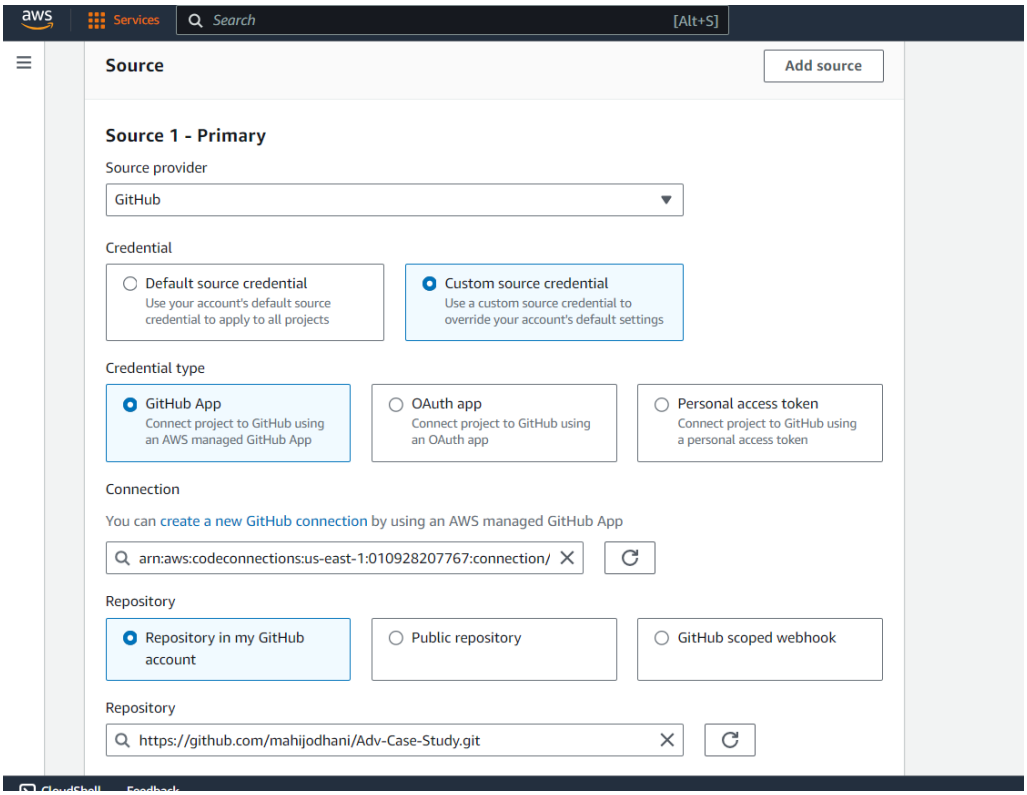


- Create a new build project.

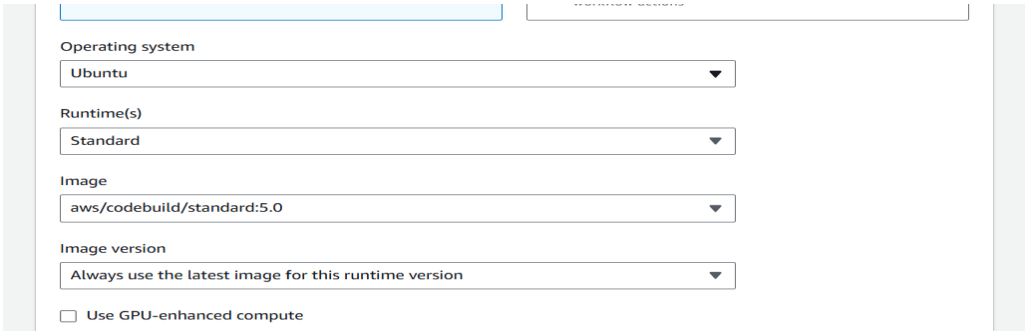


- For **Source**, choose your source repository (e.g., GitHub, Bitbucket, or S3).

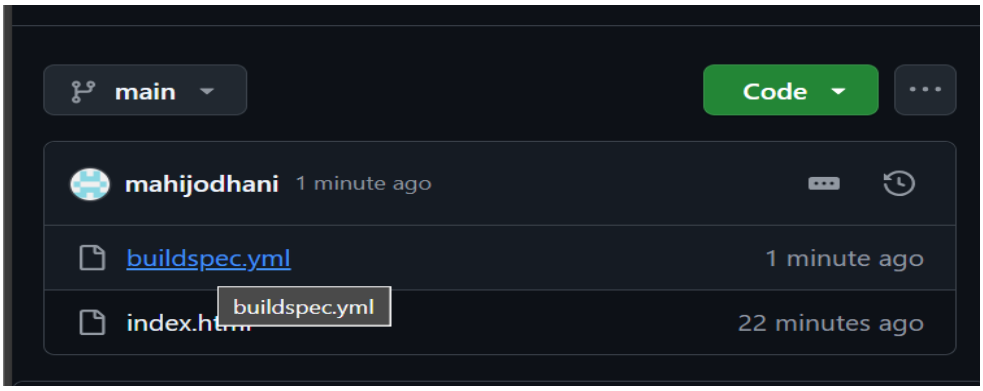


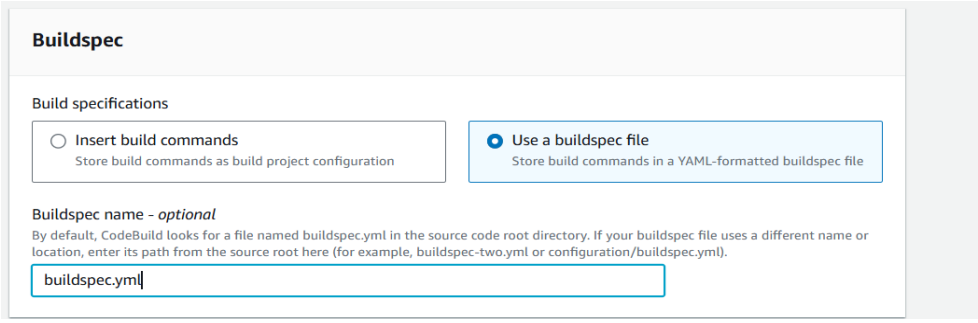


- Under **Environment**, choose **Managed Image** with **Ubuntu**, and select **Runtime: Standard**.

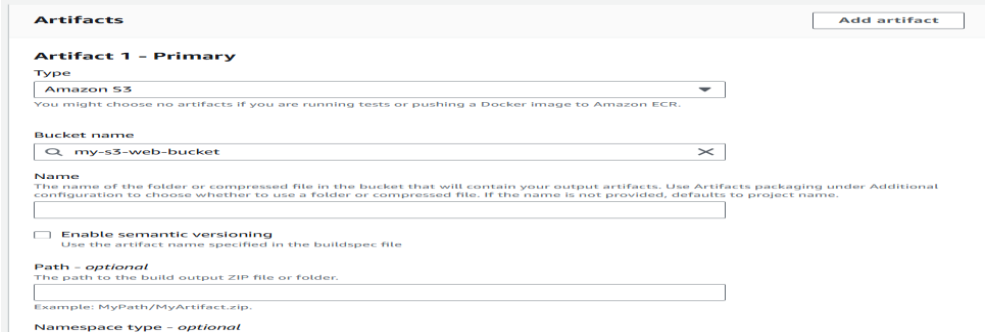


- specify the buildspec.yml file you created. before this add it to your git repo

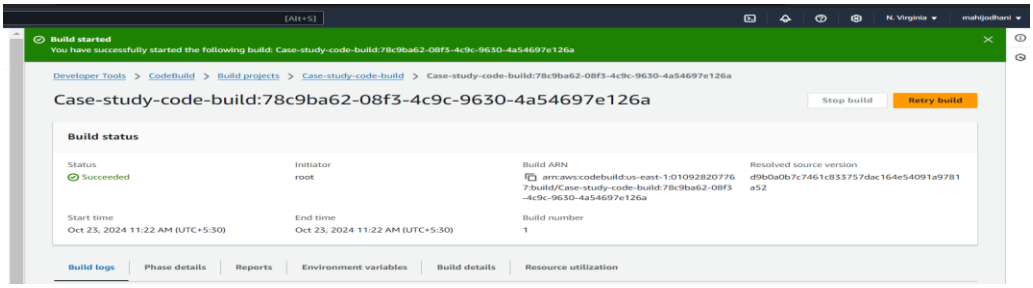


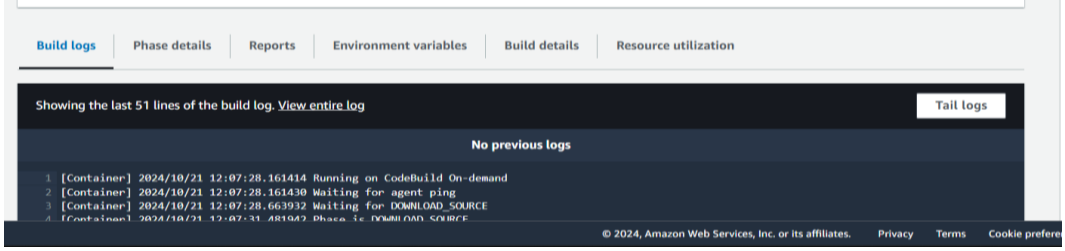


- Set **Artifacts** to "S3", and choose the bucket you created earlier.

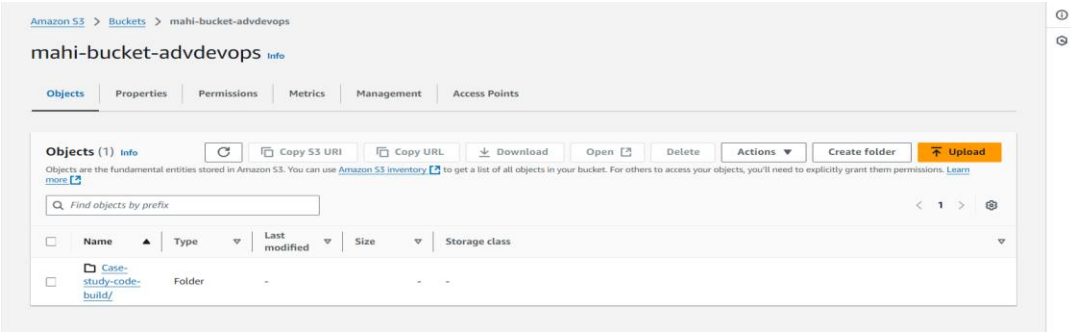


- Create the build project and start the build to ensure it uploads index.html to the S3 bucket.



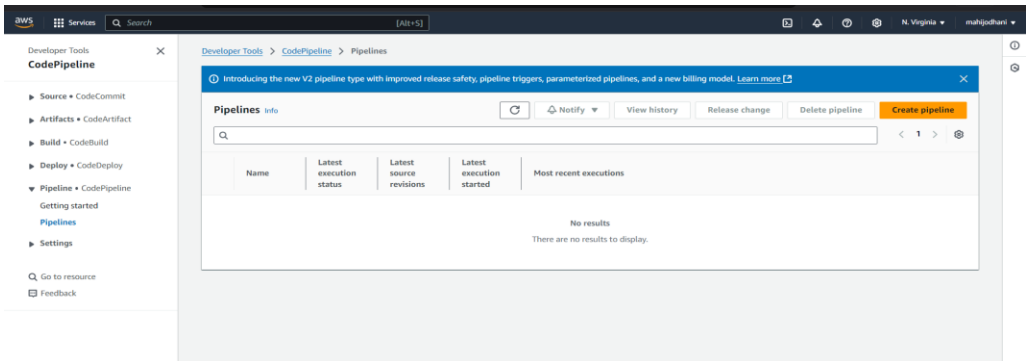


s3 bucket was updated:

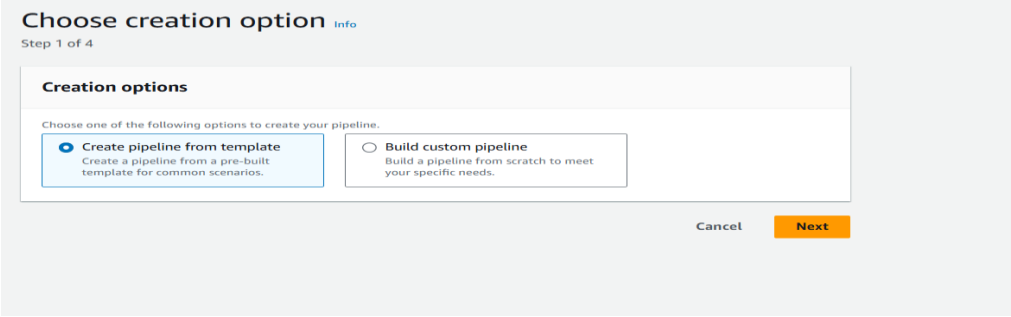


4. Set Up AWS CodePipeline

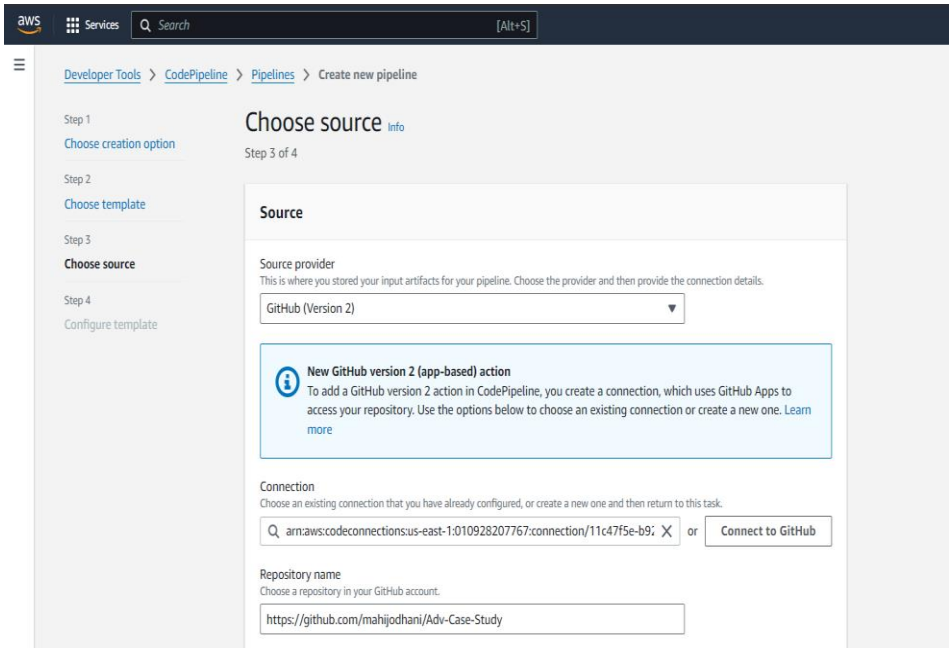
1. **Go to AWS CodePipeline**:
   - Open the CodePipeline console.



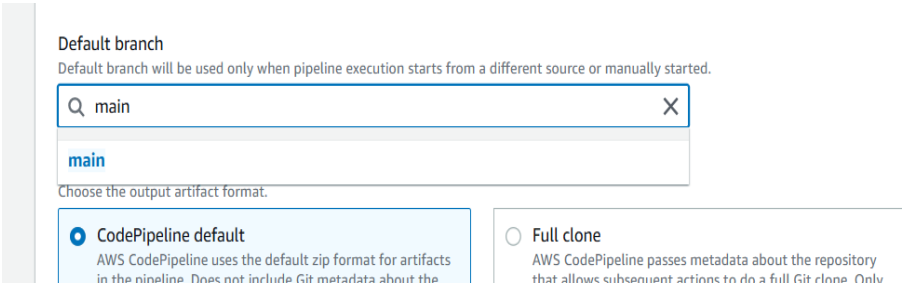
   - Create a new pipeline.
   - Choose build custom pipeline.



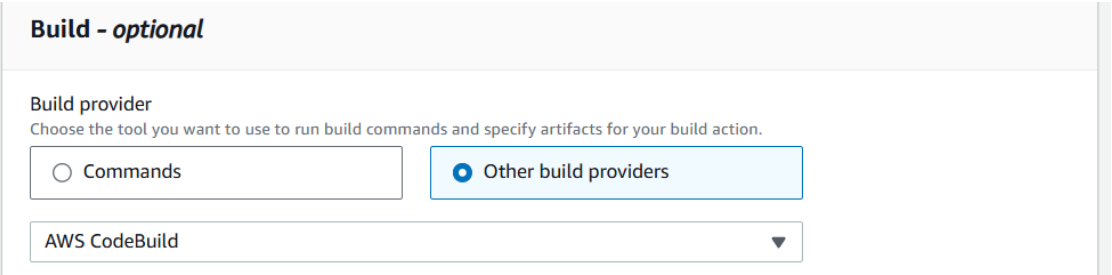
- For **Source**:
  - Select your repository (e.g., GitHub, Bitbucket).



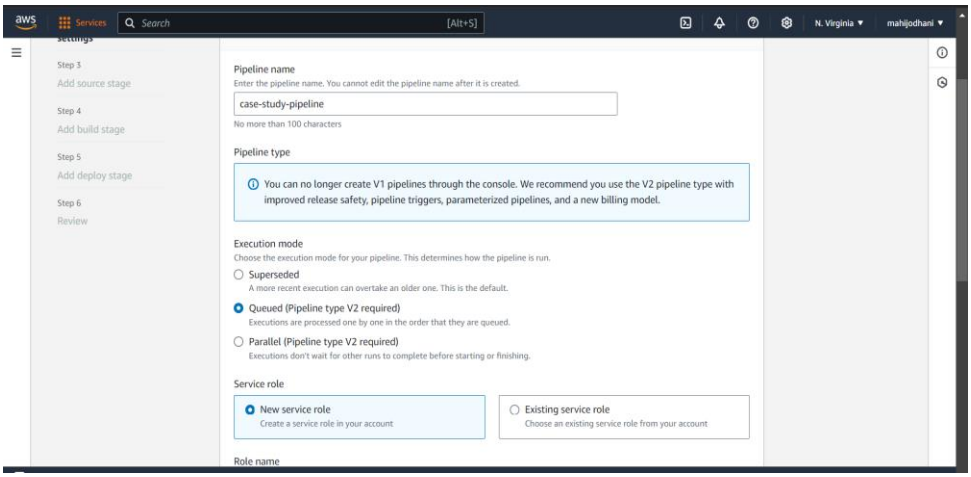
  - Connect and choose the appropriate branch where the index.html and buildspec.yml files are.



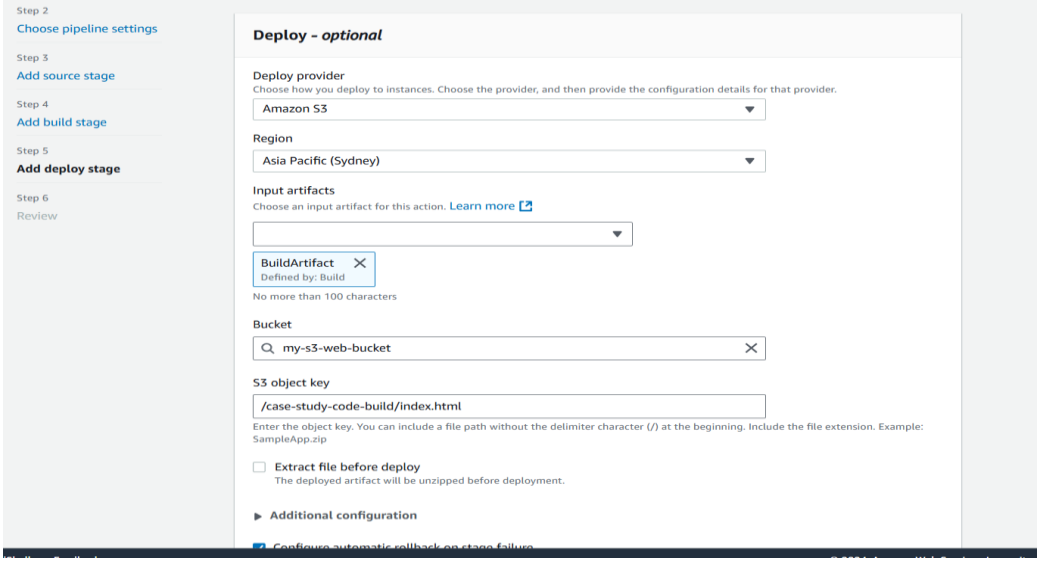
2. **Add Build Stage**:
   - In the **Build stage**, choose **AWS CodeBuild** as the build provider.



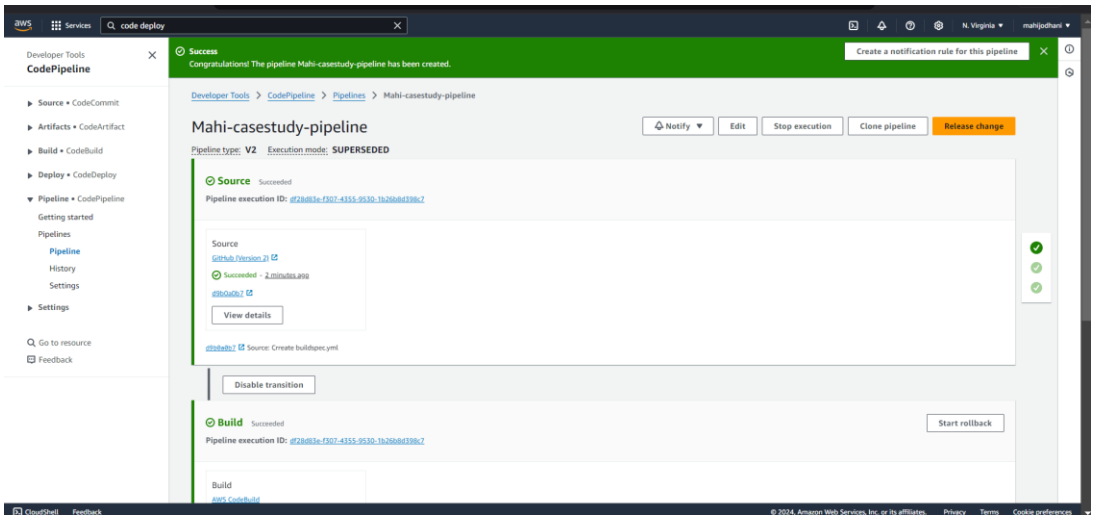
   - Select the CodeBuild project you created earlier.



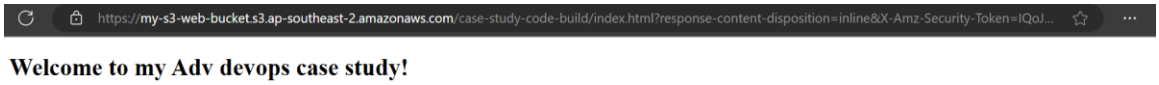
3. **Deploy to S3**:
   - In the next stage, choose **Deploy**. Select **Amazon S3**. Choose your S3 bucket (my-s3-bucket) where the index.html file will be deployed.



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.



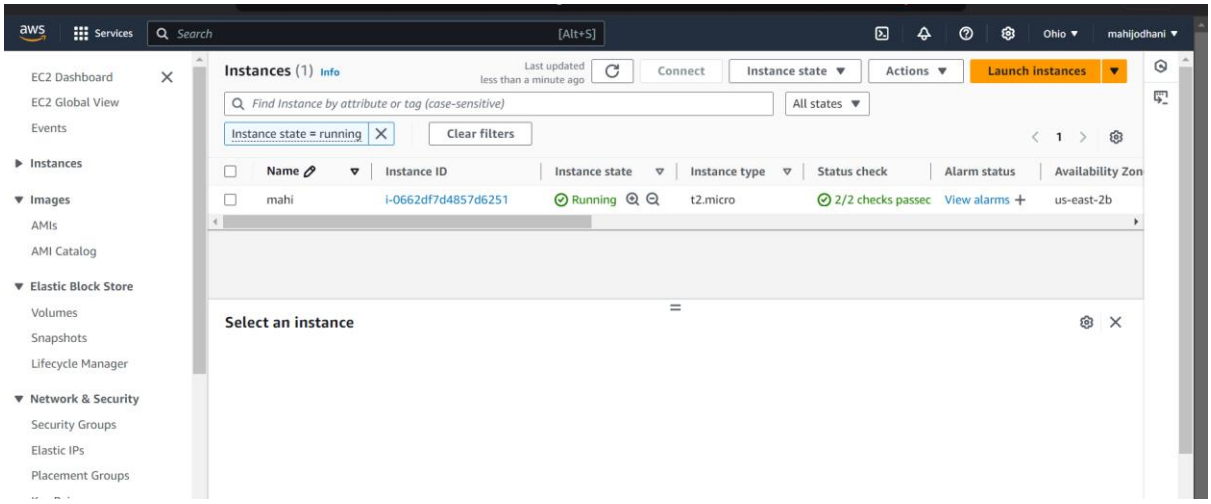
   - Visit the S3 bucket's website URL to verify that the index.html page is live.



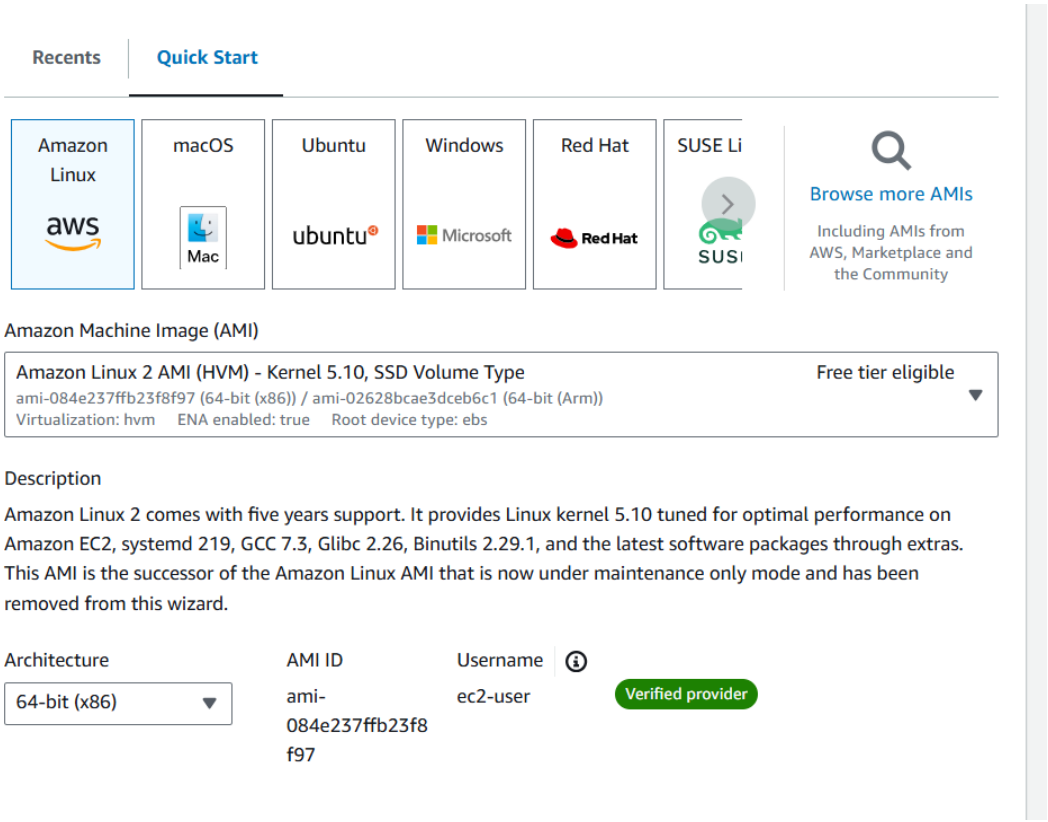
Welcome to my Adv devops case study!

5. Set Up EC2 Instance for Web Hosting

1. **Launch an EC2 Instance**:
   - Open the EC2 console.

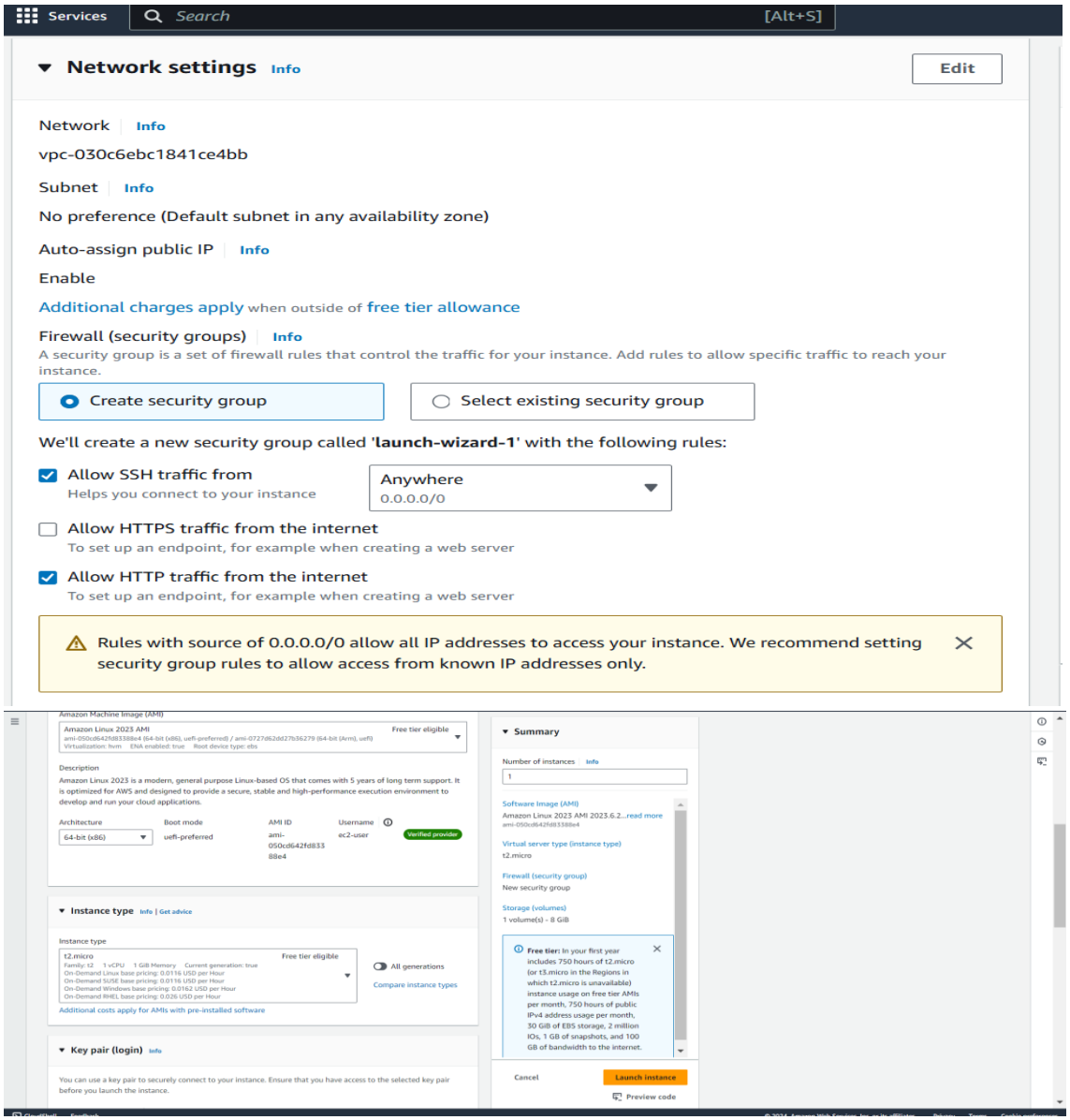


   - Launch a new instance, selecting an Amazon Linux 2 AMI.



   - Choose the default t2.micro instance type.

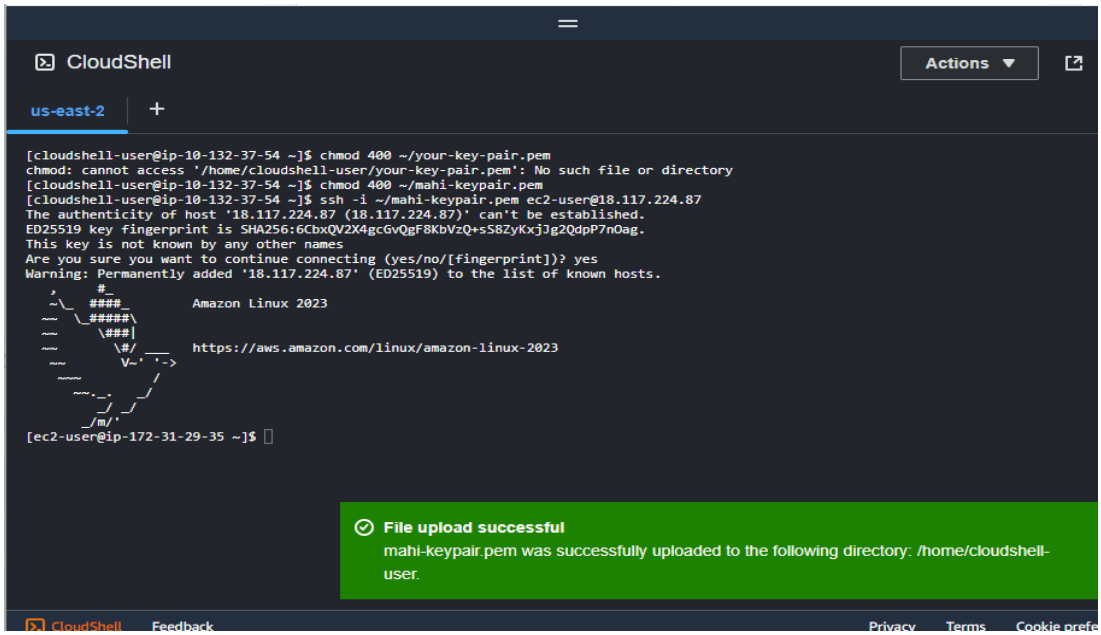
- Configure instance settings and storage (use defaults for now).
- In **Configure Security Group**, allow HTTP traffic by adding a rule to open port 80.



2. **Connect to the EC2 Instance**:
   - Once the instance is running, connect via SSH.

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

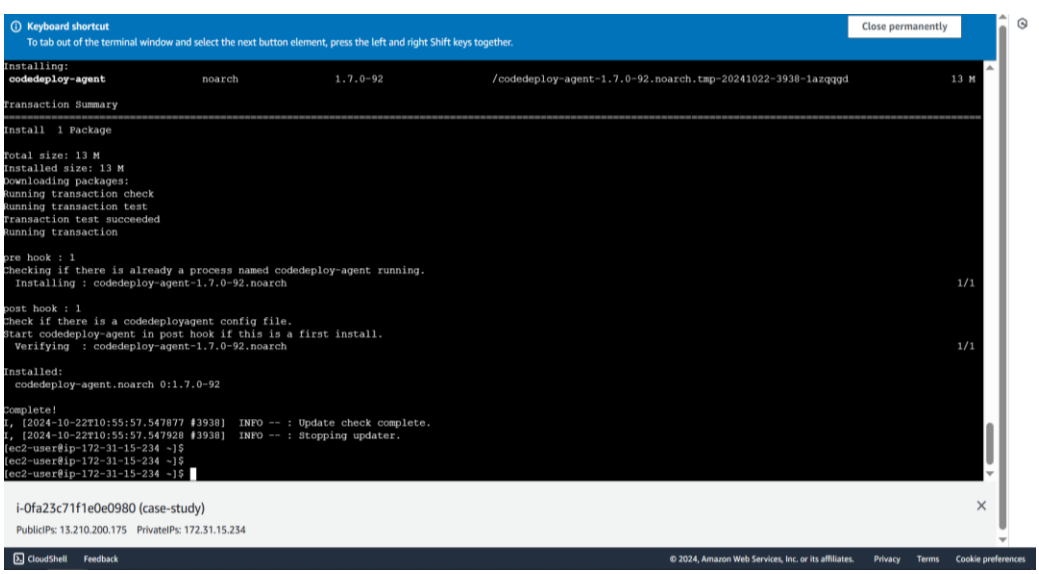
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd

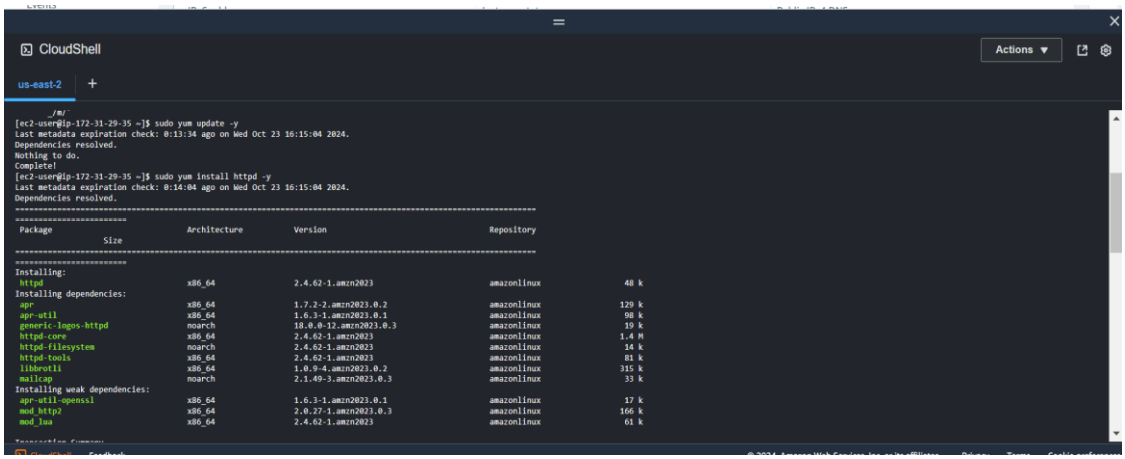


6. Set Up AWS CodeDeploy to Push Updates to EC2

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





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:

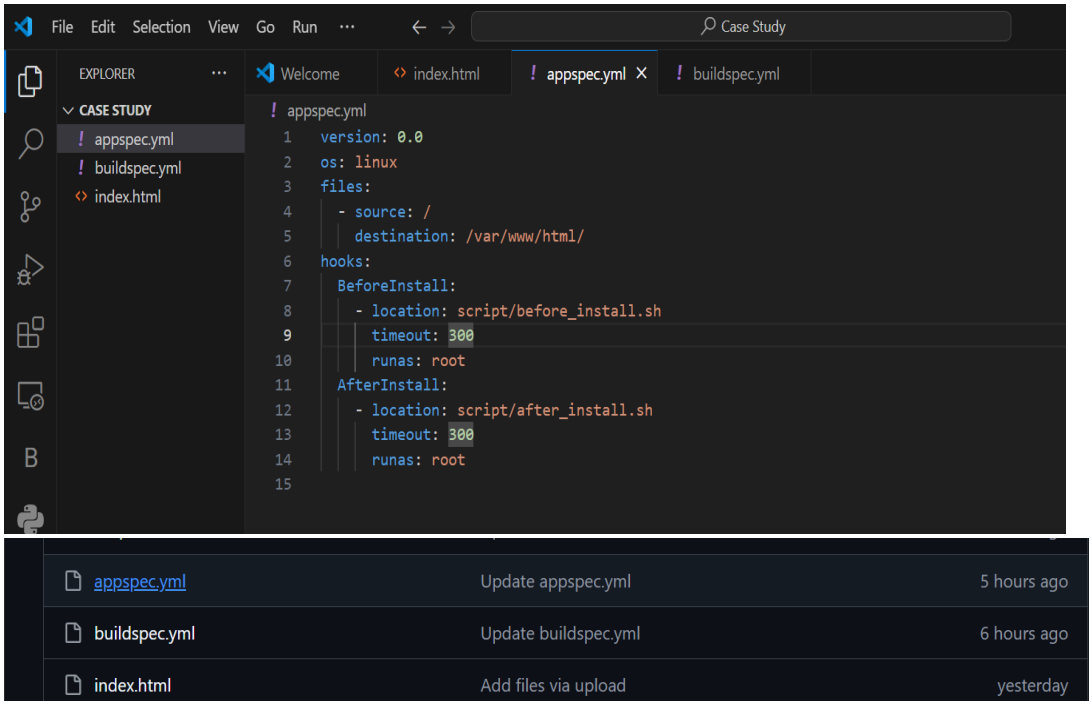
   aws deploy create-application --application-name my-webapp

3.2 Create Deployment Group

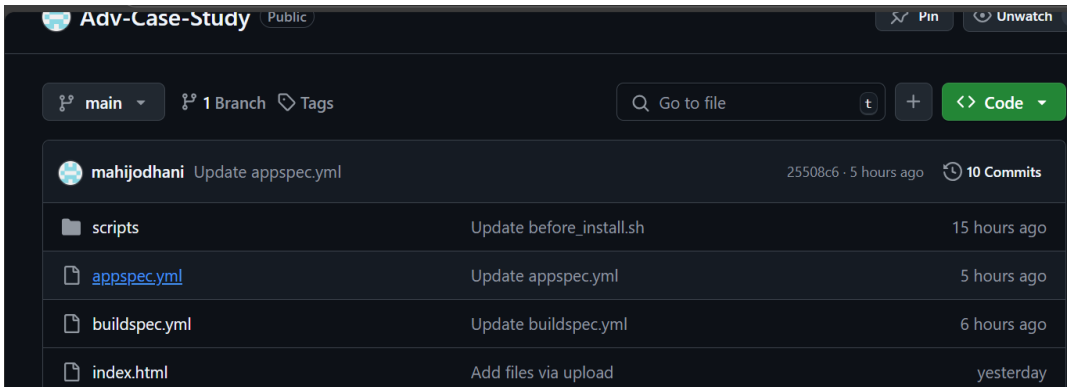
- Create deployment group:
  aws deploy create-deployment-group --application-name my-webapp --deployment-group-name my-webapp-group --service-role-arn

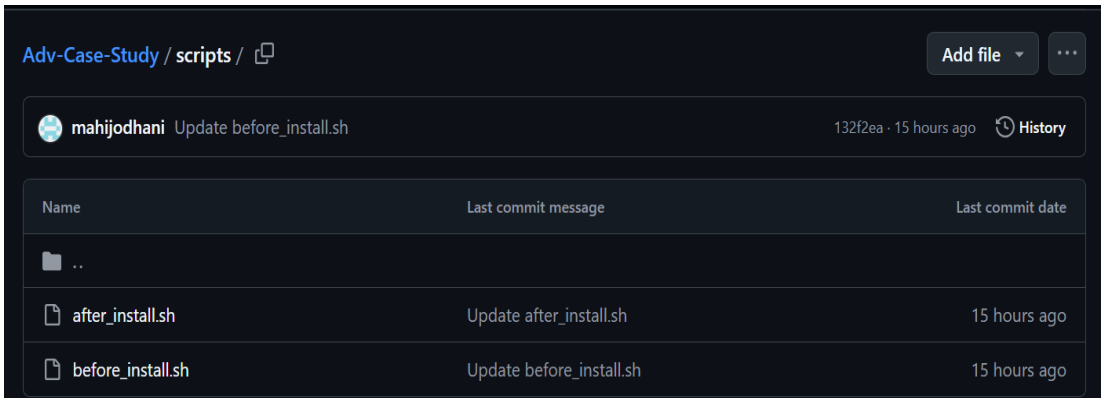
  arn:aws:iam::ACCOUNT_ID:role/CodeDeployServiceRole
- enter your ACCOUNT_ID above!!

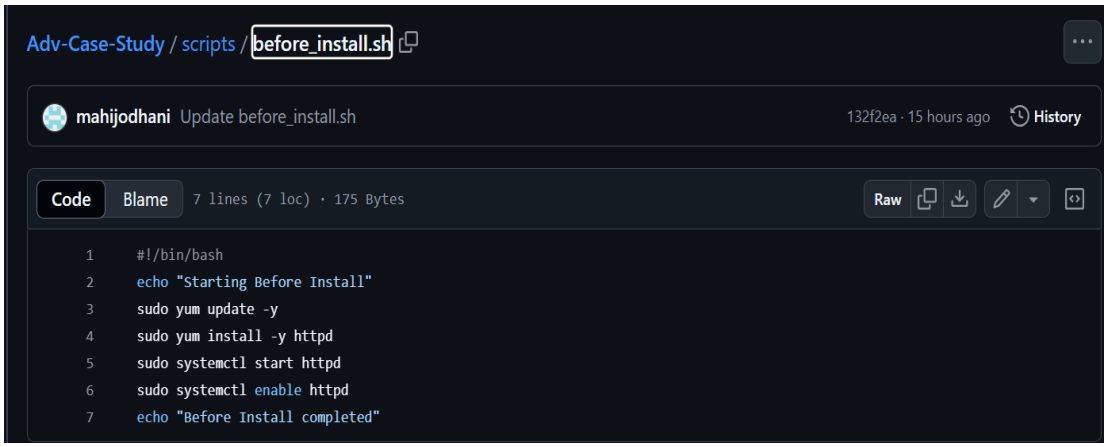
Create appspec.yml for CodeDeploy: In your project folder, create an appspec.yml file to specify how CodeDeploy should handle the deployment:



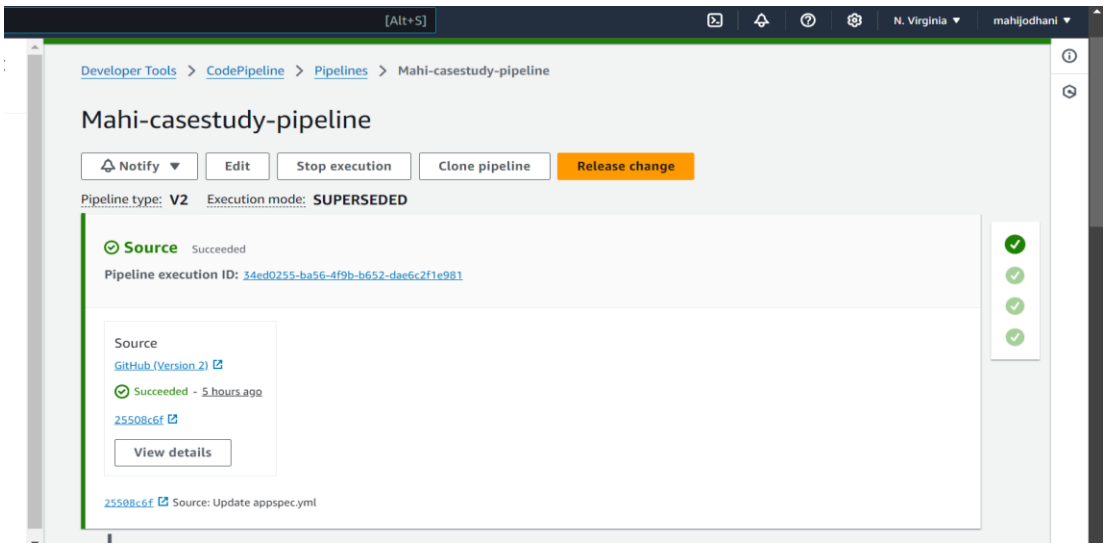
Now create a scripts folder inside your repo which will contain 2 files: **before_install.sh** and **after_install.sh**



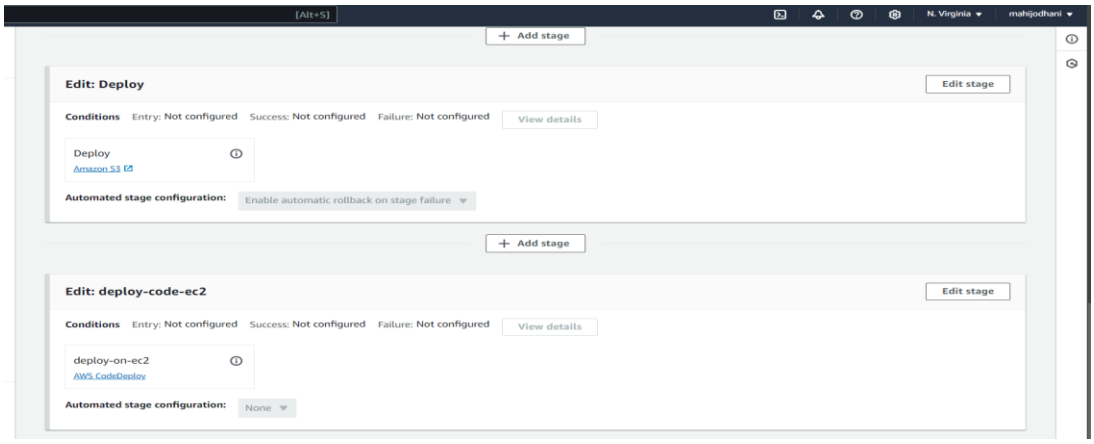




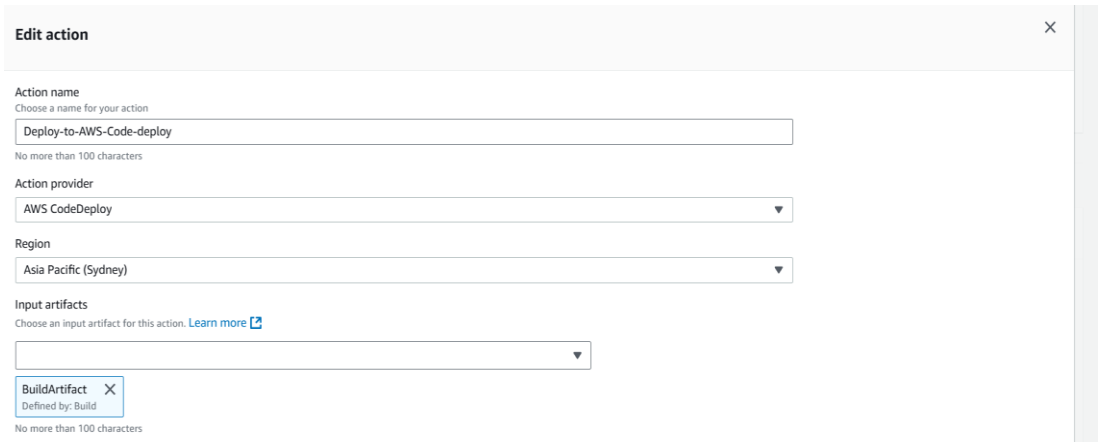
3. **Add Deployment Stage to CodePipeline**:
   - Go back to your CodePipeline.



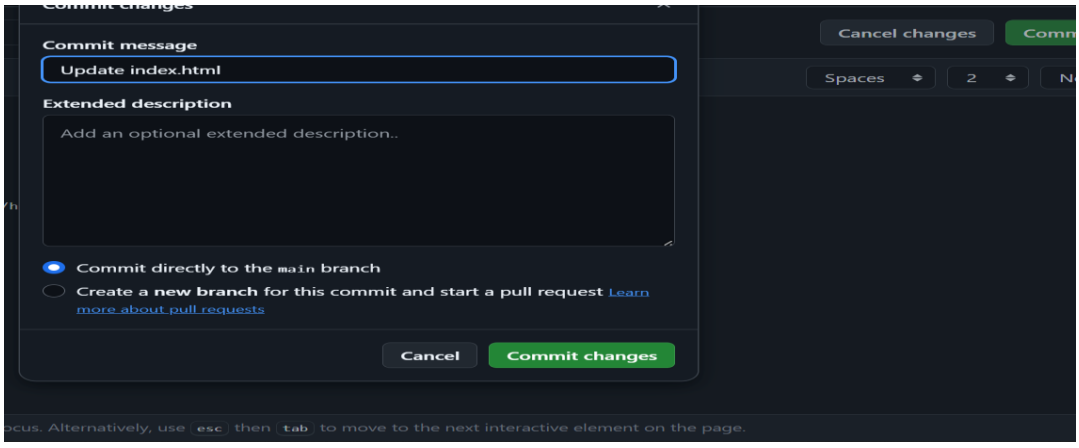
- Add a new stage for deployment.



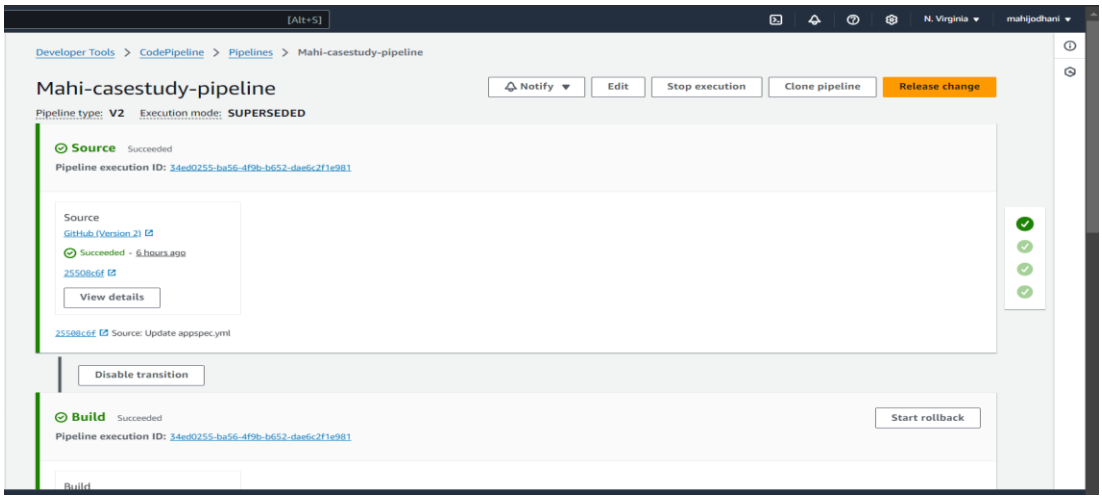
- Select **AWS CodeDeploy** and choose the application and deployment group you created earlier.

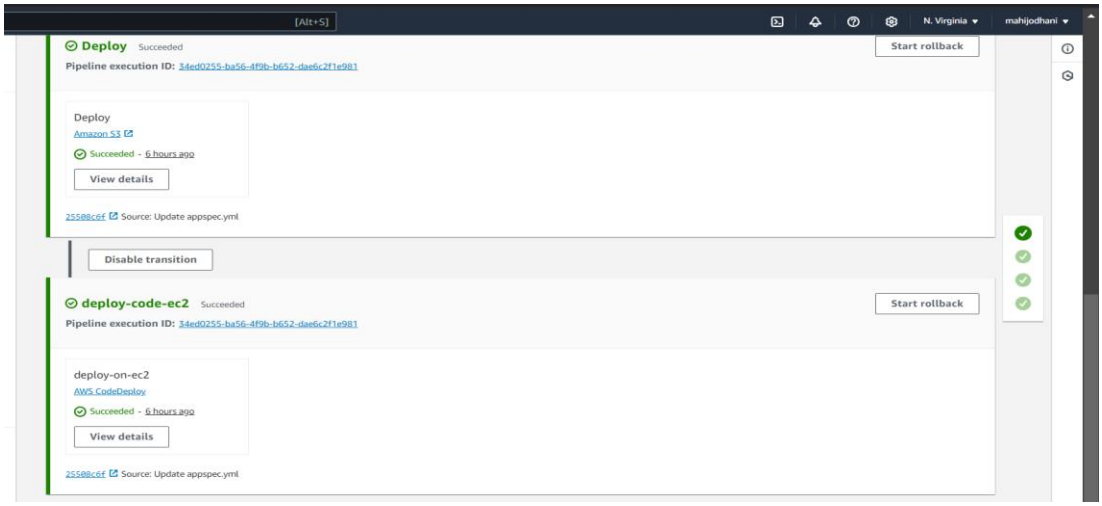


4. **Test Deployment**:
   - Make a change to index.html in your source repository and push it.

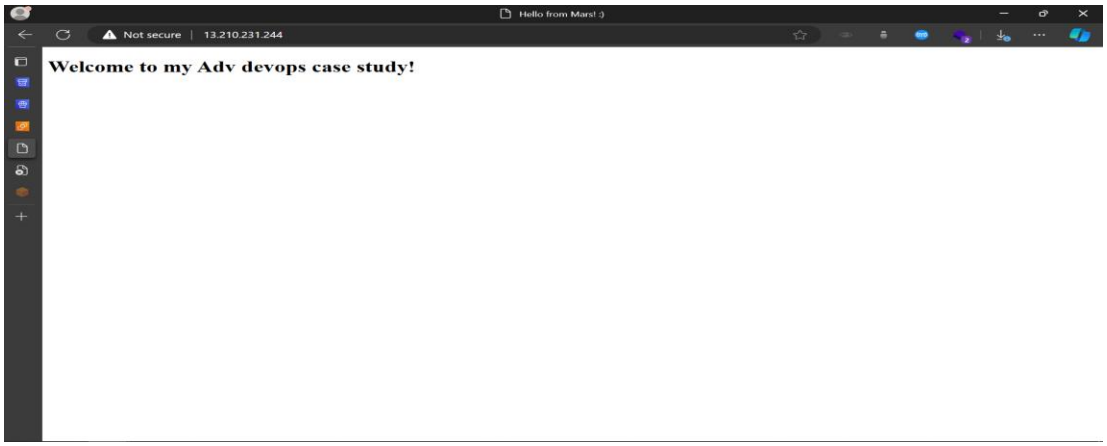


- This should trigger the pipeline, rebuild the app, push it to S3, and deploy it to the EC2 instance.



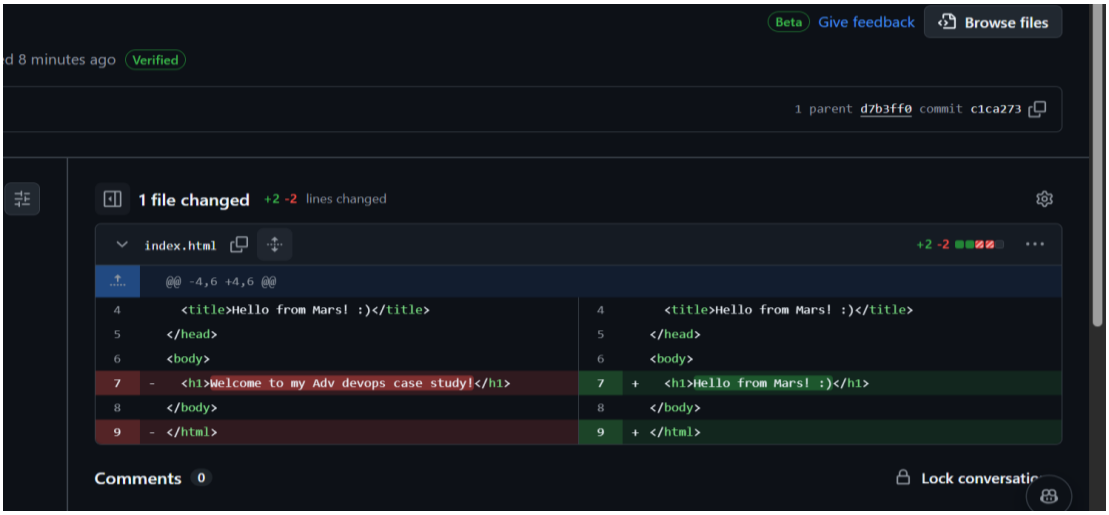


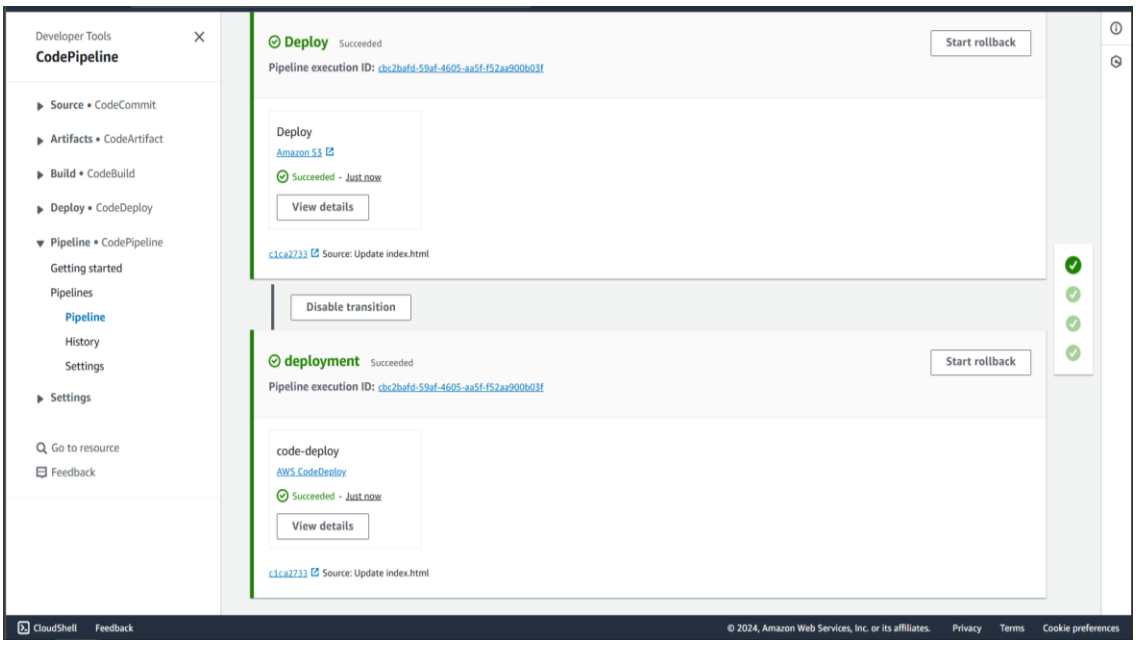
- You can access your EC2 instance via its public IP to view the updated web app.

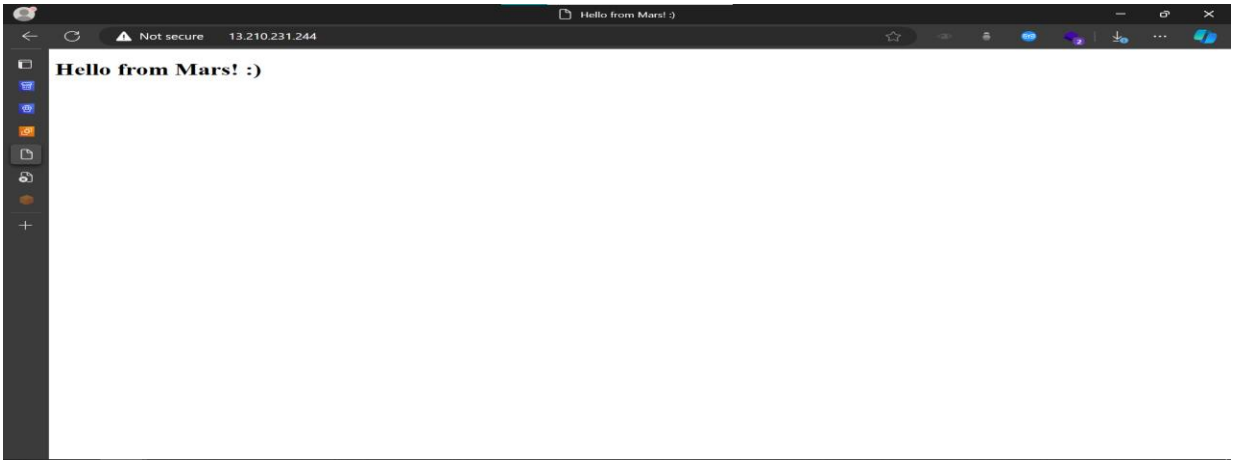


7. Verify Automation

- 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.