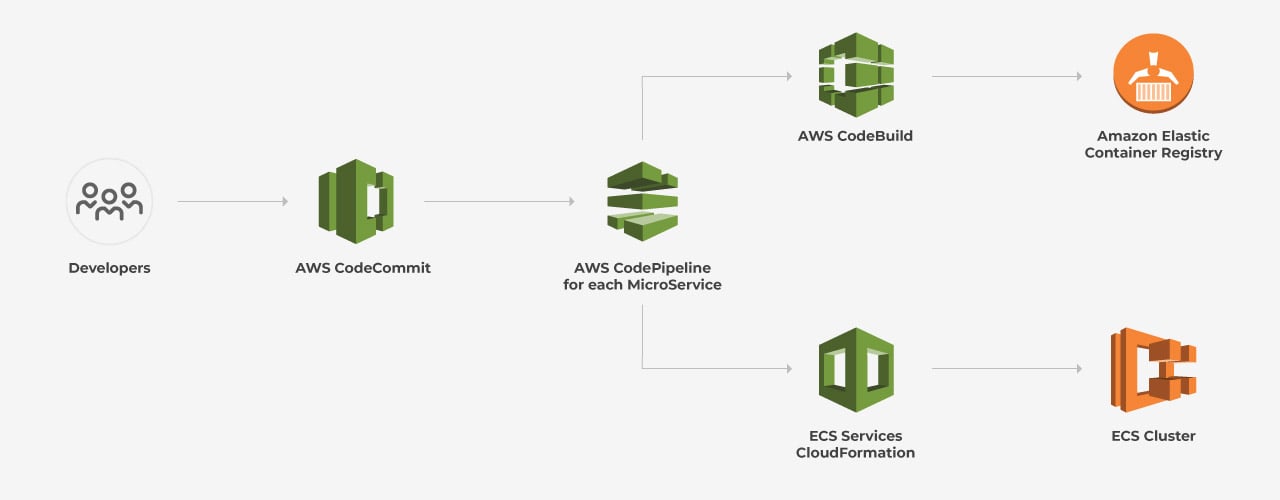
### ****Title****

**CI/CD Pipeline Implementation Using AWS Services for Python Application Deployment**



**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Title** | **Page No** |
| 1. | **Introduction** | 3 |
| 2. | **Technologies and Tools Overview**  GitHub  AWS CodeBuild  AWS CodeDeploy  AWS CodePipeline  AWS EC2 and ECR | 3 |
| 3. | **Step-by-Step Implementation**   * Step 1: Create Repository in Amazon ECR * Step 2: Store Parameters in AWS Systems Manager * Step 3: Set Up AWS CodeBuild Project * Step 4: Set Up AWS CodePipeline * Step 5: Create CodeDeploy Application and Deployment Group * Step 6: Set Up EC2 Instance for Deployment * Step 7: Configure CodeDeploy Deployment Group * Step 8: Create a Deployment Using CodeDeploy * Step 9: Add CodeDeploy Stage in CodePipeline * Step 10: Test the Pipeline | 4-23 |
| 4. | **Conclusion** | 23 |

### ****1. Introduction****

This document describes how to create a CI/CD (Continuous Integration and Continuous Deployment) pipeline using AWS services. CI/CD pipelines automate the process of building, testing, and deploying software, allowing developers to quickly and safely release changes.

In this task, we built an end-to-end pipeline for deploying a Python application. The pipeline automatically pulls code from GitHub, builds a Docker image, pushes it to Amazon Elastic Container Registry (ECR), and deploys it to an EC2 instance using CodeDeploy. All of this was orchestrated using AWS CodePipeline.

This guide is suitable for beginners and provides step-by-step instructions to set up this pipeline using various AWS tools.

### ****2. Technologies and Tools Overview****

#### ****GitHub****

GitHub is a platform for version control that hosts code repositories. Developers use GitHub to store their project code and collaborate with others. In this case, we used GitHub to store the Python application source code.

#### ****AWS CodeBuild****

AWS CodeBuild is a fully managed build service that automates the process of compiling source code, running tests, and generating software packages. It simplifies the process of building Docker images for applications. In our case, CodeBuild was used to build a Docker image of the Python application and push it to Amazon ECR.

#### ****AWS CodeDeploy****

AWS CodeDeploy is a deployment service that automates the process of delivering applications to compute instances like EC2 or Lambda. Here, it is used to deploy the Docker image from ECR to an EC2 instance, ensuring the latest version of the application is running.

#### ****AWS CodePipeline****

AWS CodePipeline is a CI/CD service that automates the workflow for continuous integration and delivery. It connects various AWS services (like GitHub, CodeBuild, and CodeDeploy) into a sequence that automatically triggers the next action, like building the application or deploying it, after each change in the code repository.

#### ****AWS EC2 and ECR****

* **EC2 (Elastic Compute Cloud)**: EC2 is a service that provides virtual machines (instances) on which you can run applications. We used an EC2 instance to run the Python application.
* **ECR (Elastic Container Registry)**: ECR is a managed Docker image storage service. We used ECR to store Docker images built by CodeBuild.

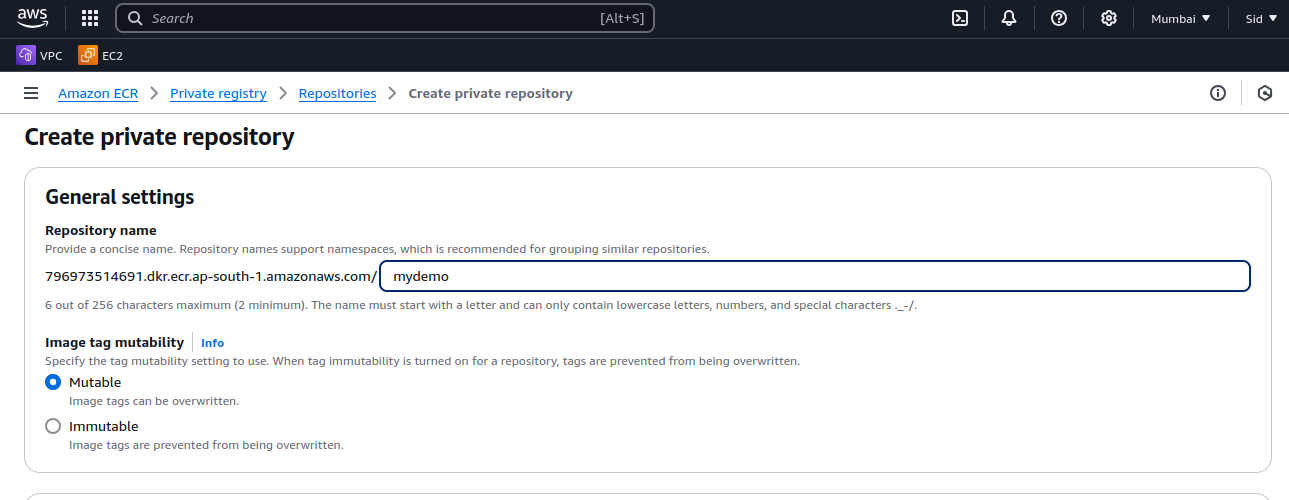
**3. Step-by-Step Implementation**

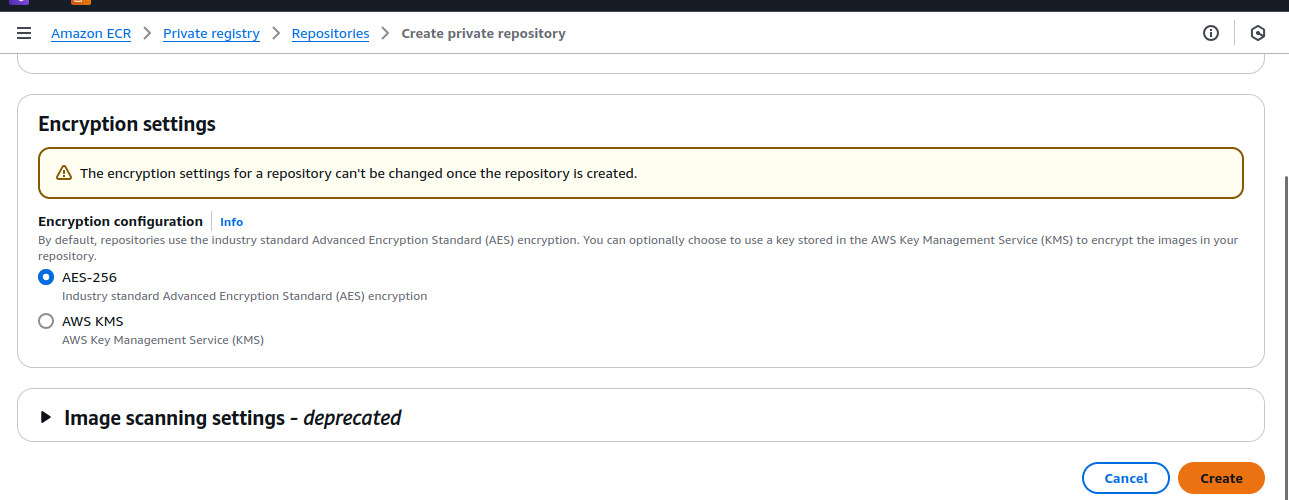
1. **Create Repository in Amazon ECR**

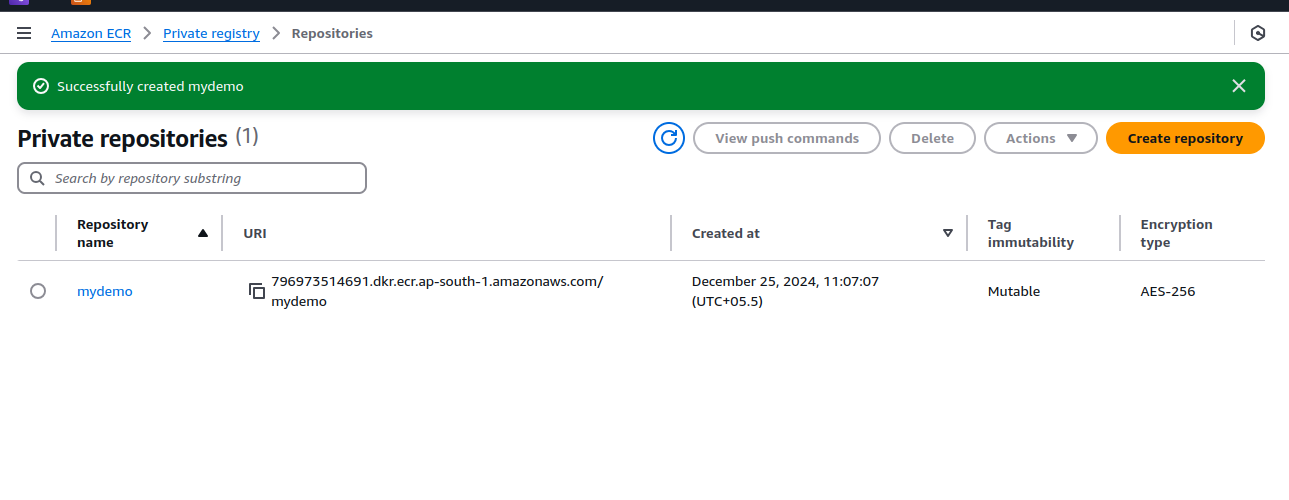
**What is Amazon ECR?** Amazon ECR is where you can store your Docker images. When you build an application in Docker, you can push that Docker image to ECR so it can be accessed by other services like CodeDeploy.

**How to do it:**

1. Go to the **Amazon ECR Console**.
2. Click **Create repository**.
3. Name the repository (e.g., mydemo).
4. Use the default settings for the repository and create it.





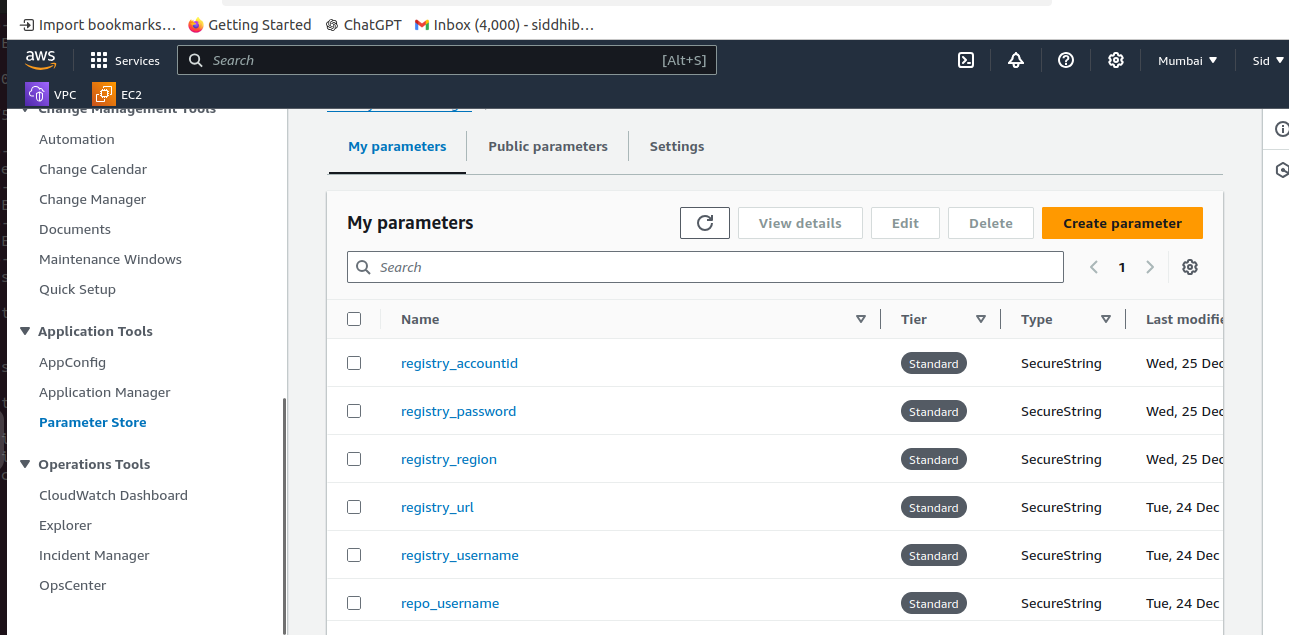


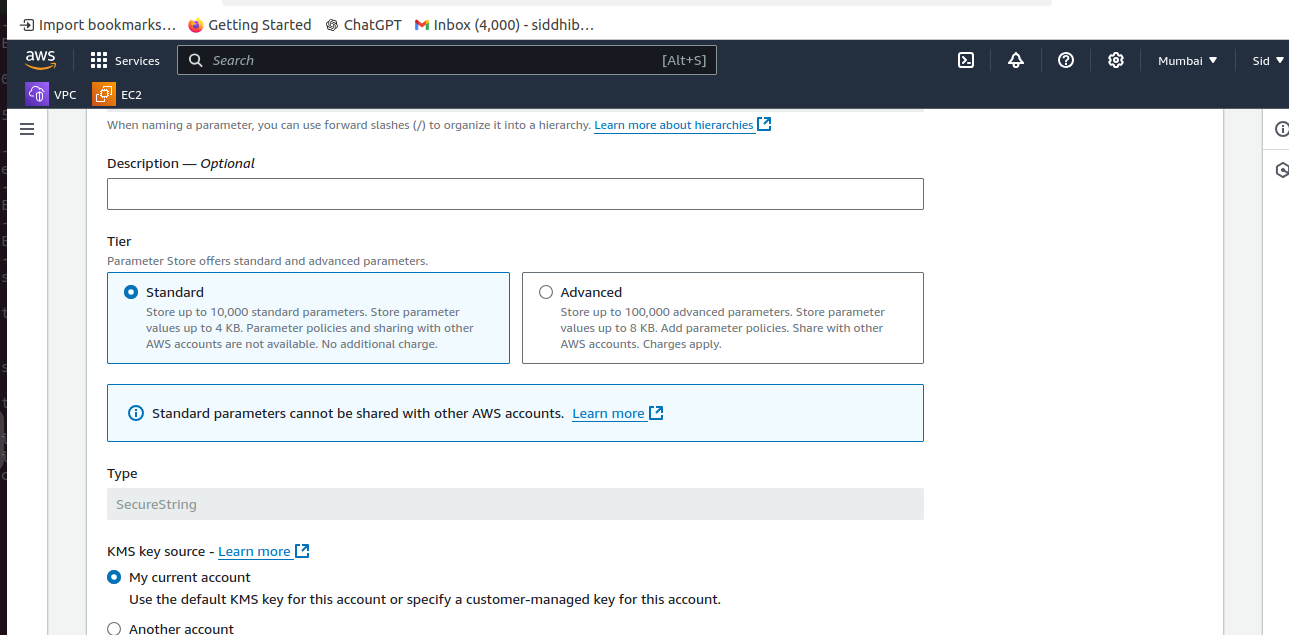
1. **Store Parameters in AWS Systems Manager**

**What is AWS Systems Manager Parameter Store?** AWS Systems Manager Parameter Store is a service to store configuration data, such as credentials, securely. Here, we store sensitive information like Docker registry credentials, so the build process can access them securely.

· **How to do it:**

1. Go to **AWS Systems Manager**.
2. Choose **Parameter Store** and click **Create parameter**.
3. Store parameters like registry\_url, registry\_username, and registry\_password as **SecureString** to keep them encrypted.





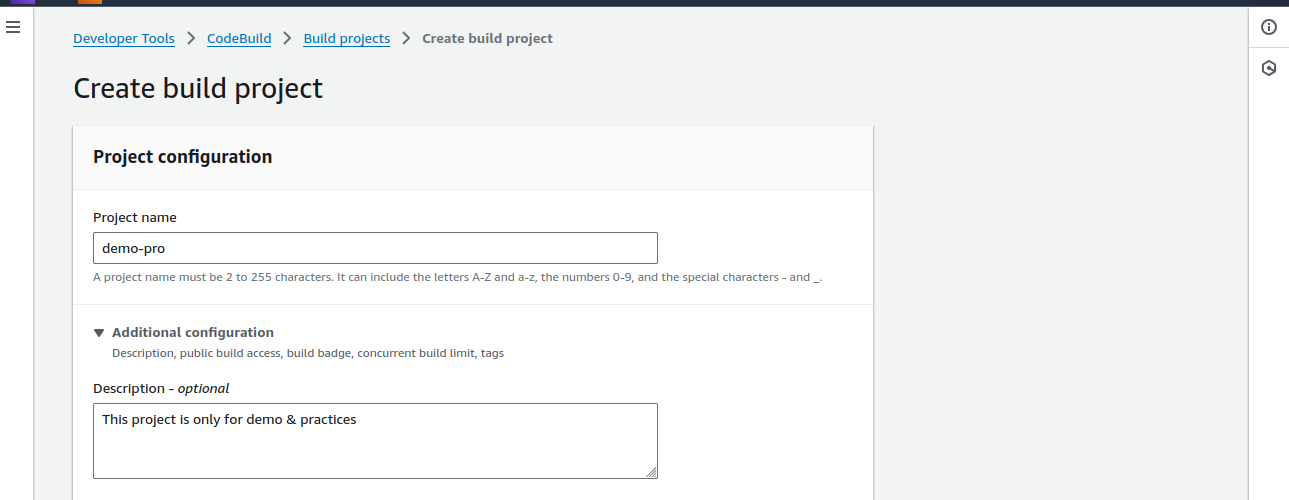
1. **Set Up AWS CodeBuild Project**

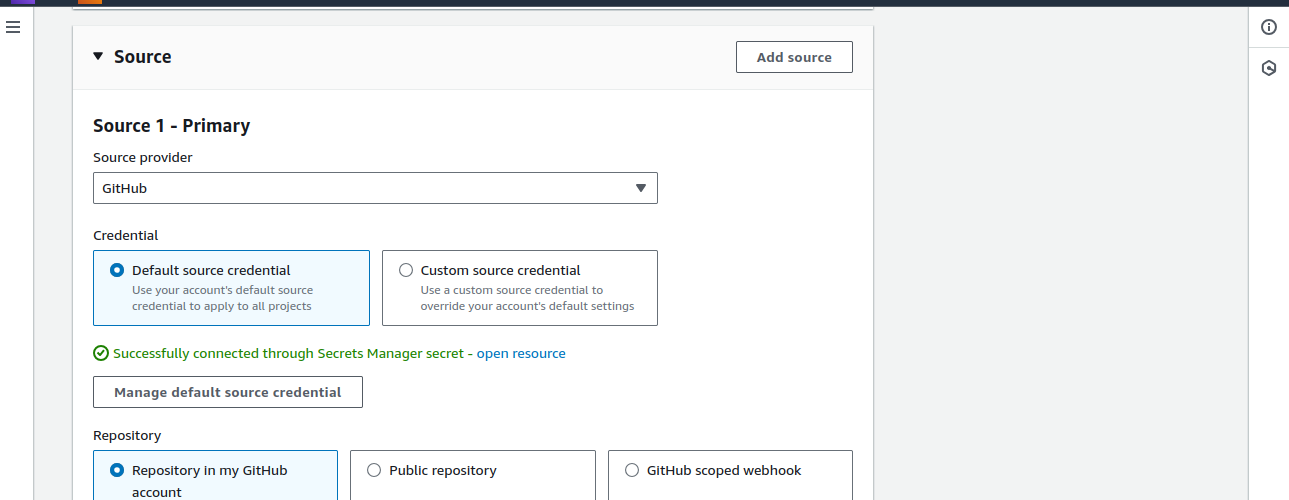
**What is AWS CodeBuild?** AWS CodeBuild is used to automatically build and test your application whenever there’s a code change.

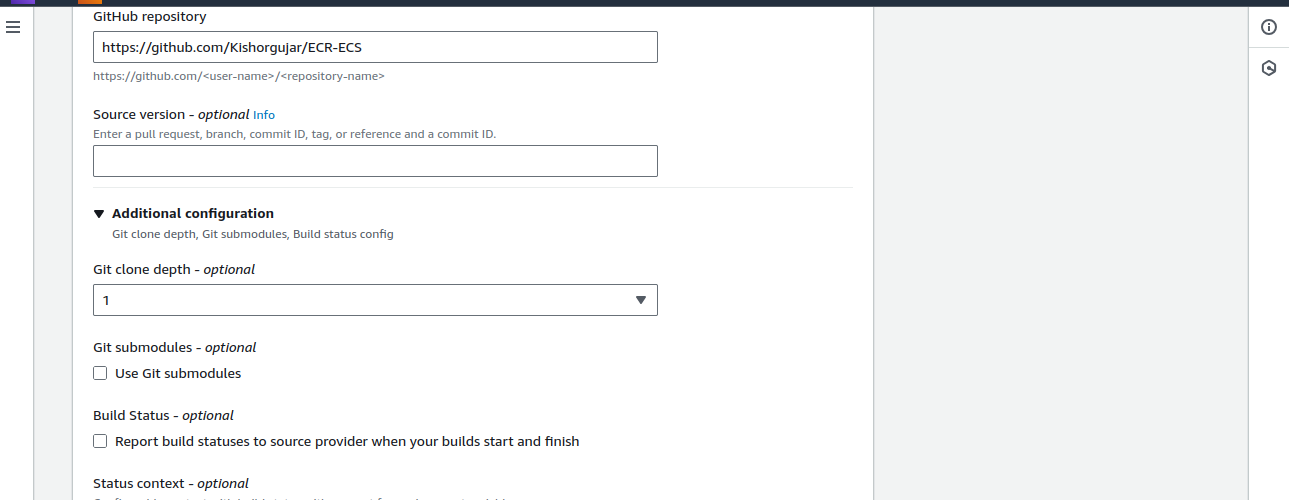
· **How to do it:**

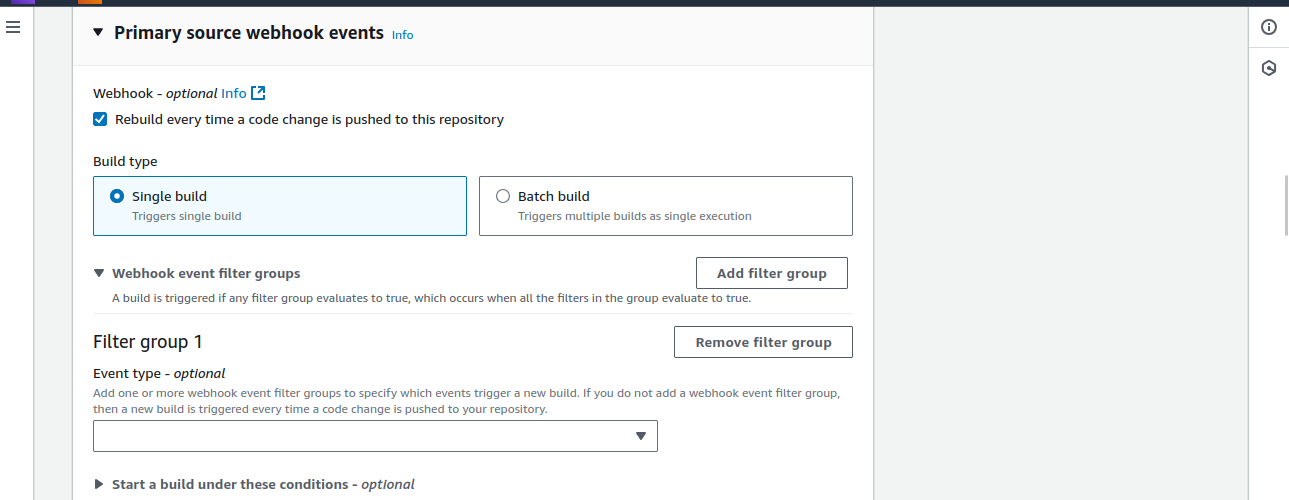
1. Go to **AWS CodeBuild Console** and create a new project.
2. Choose **GitHub** as the source provider, so CodeBuild pulls code from your GitHub repository.
3. Select **Ubuntu** as the operating system.
4. In the buildspec file (a file that tells CodeBuild how to build your application), use commands to:
   1. Install dependencies for your Python application.
   2. Log in to the Docker registry (ECR).
   3. Build and push the Docker image to ECR.

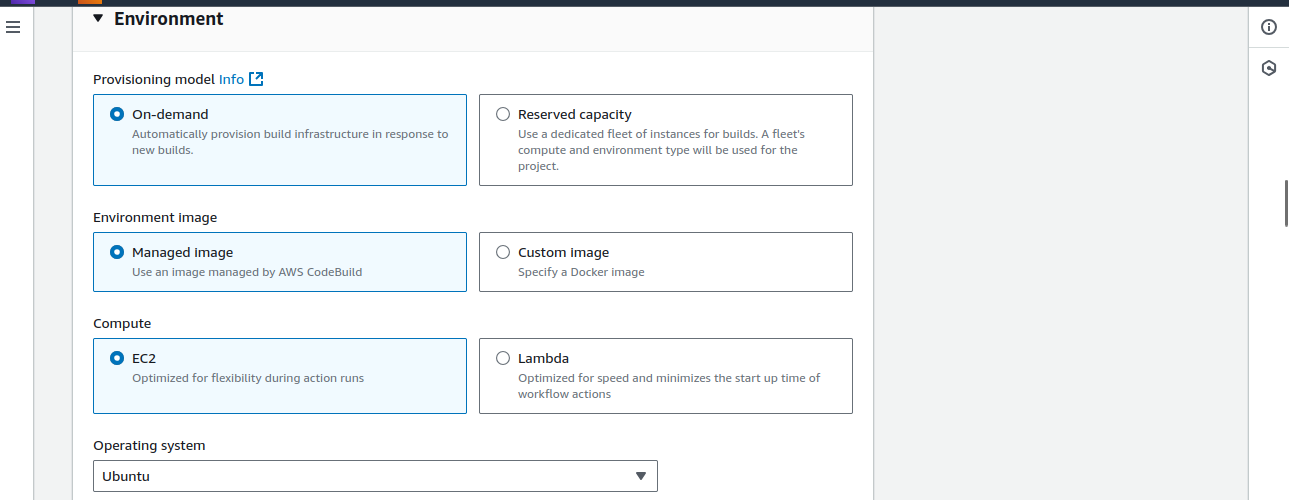
Example buildspec.yml: <https://github.com/Kishorgujar/ECR-ECS/blob/master/simple-python-app/buildspec.yml>

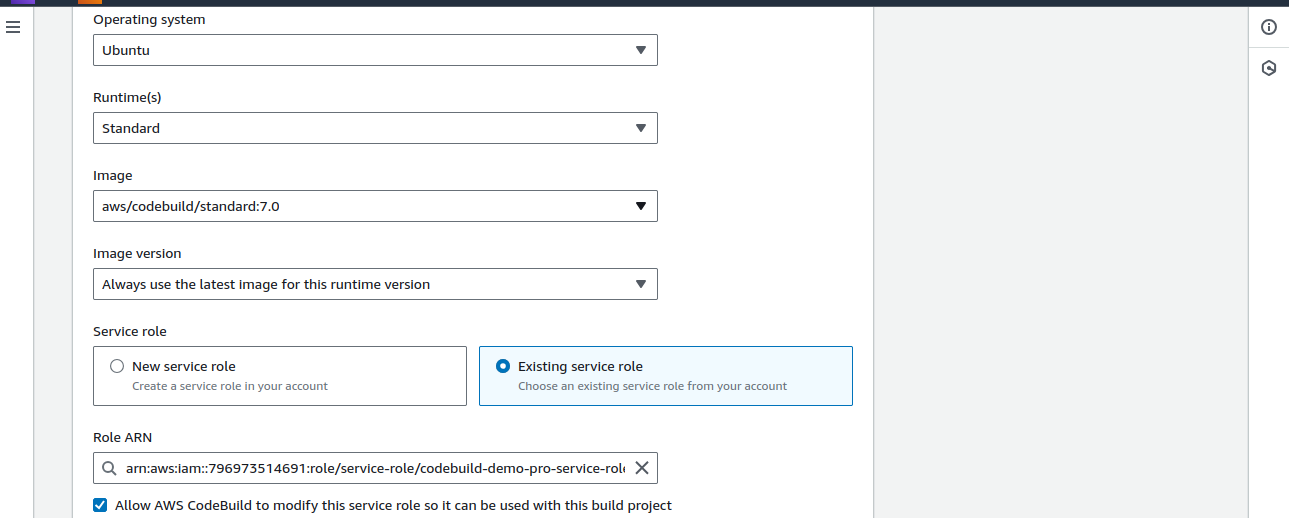


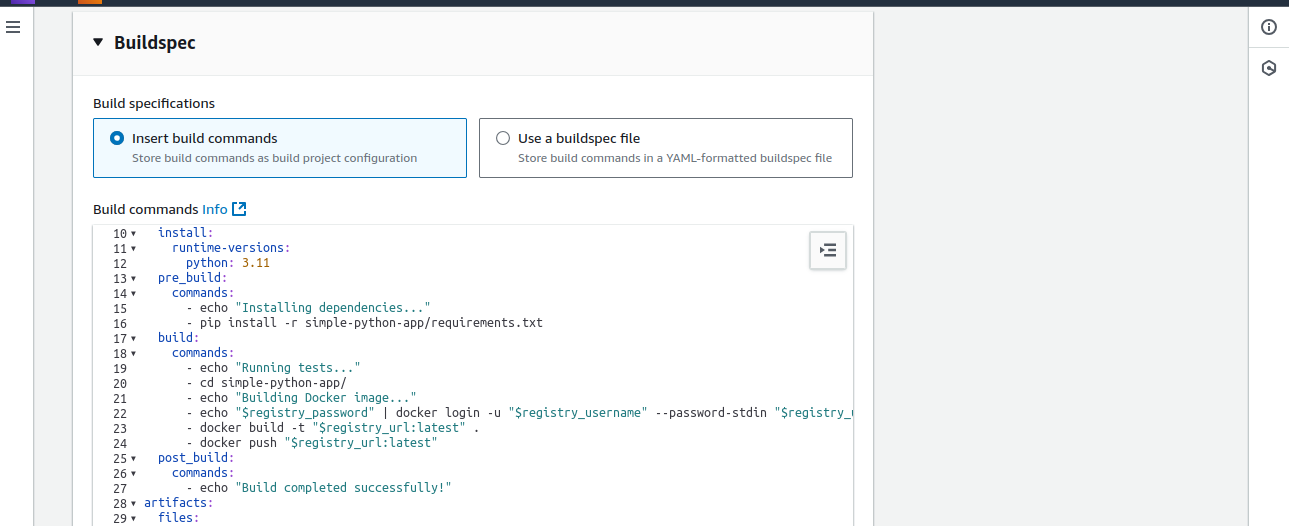


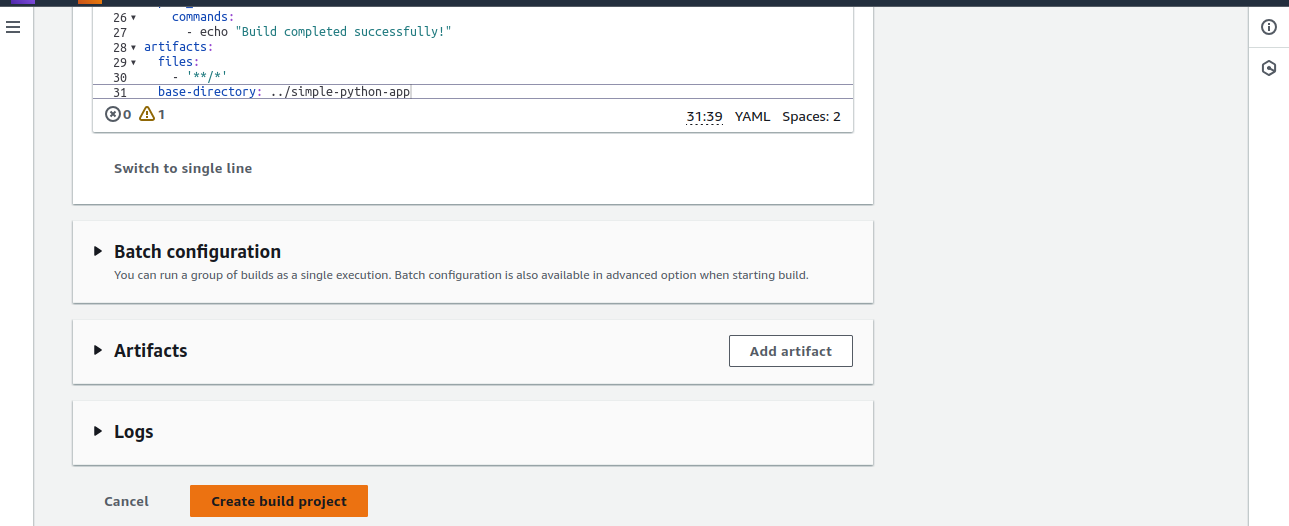


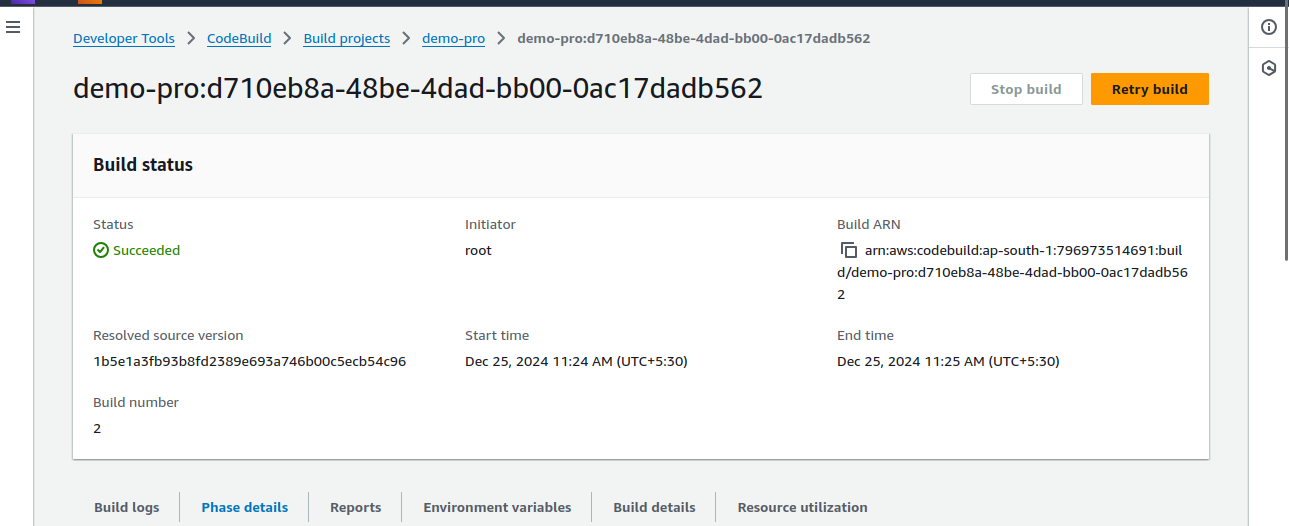










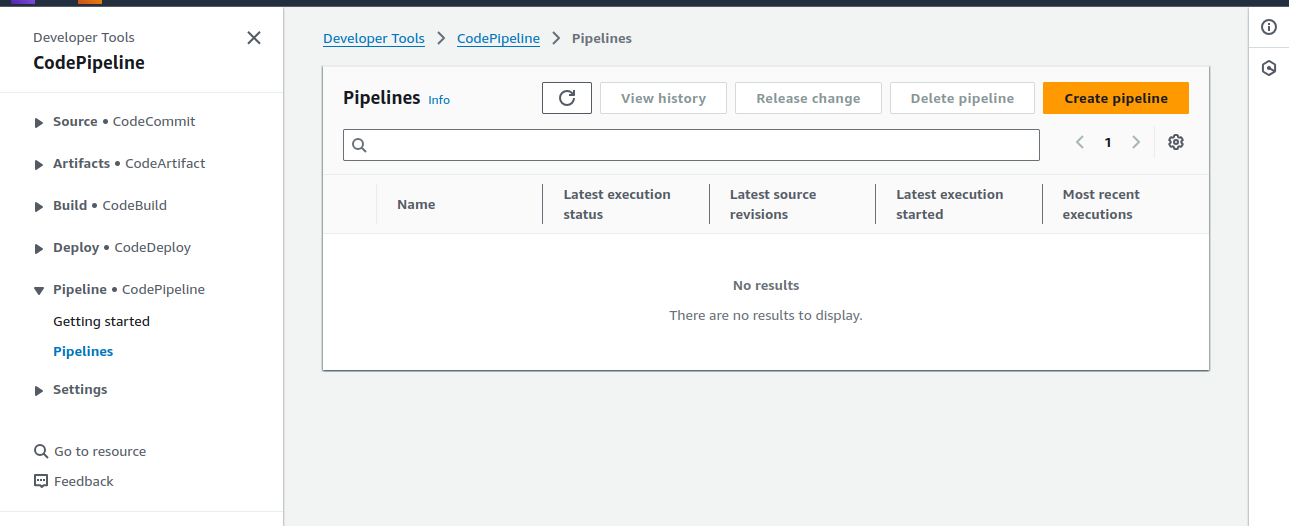


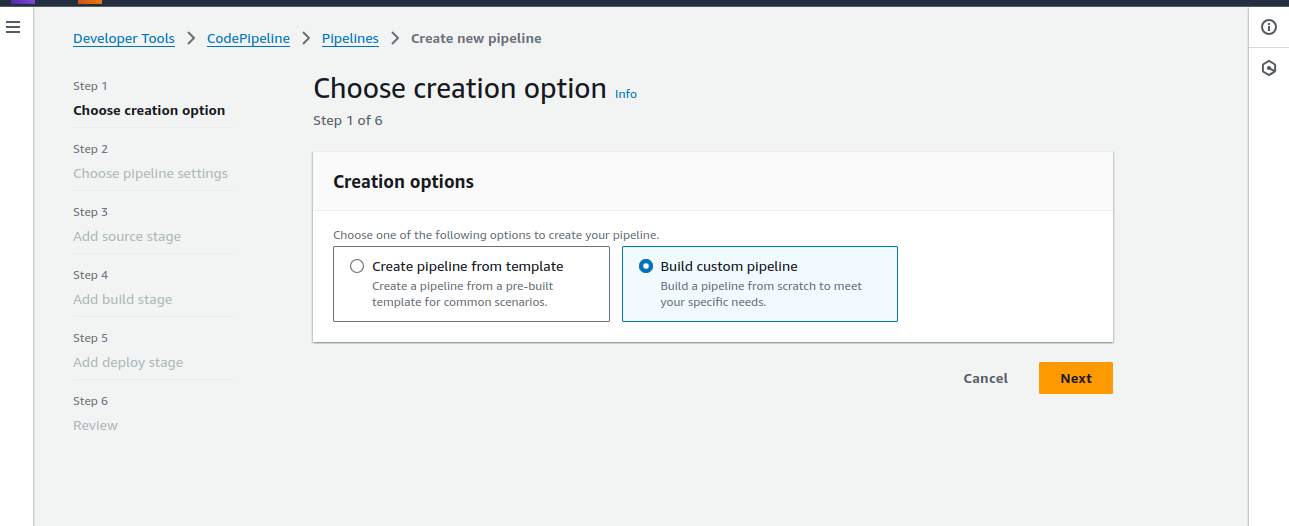
1. **Set Up AWS CodePipeline**

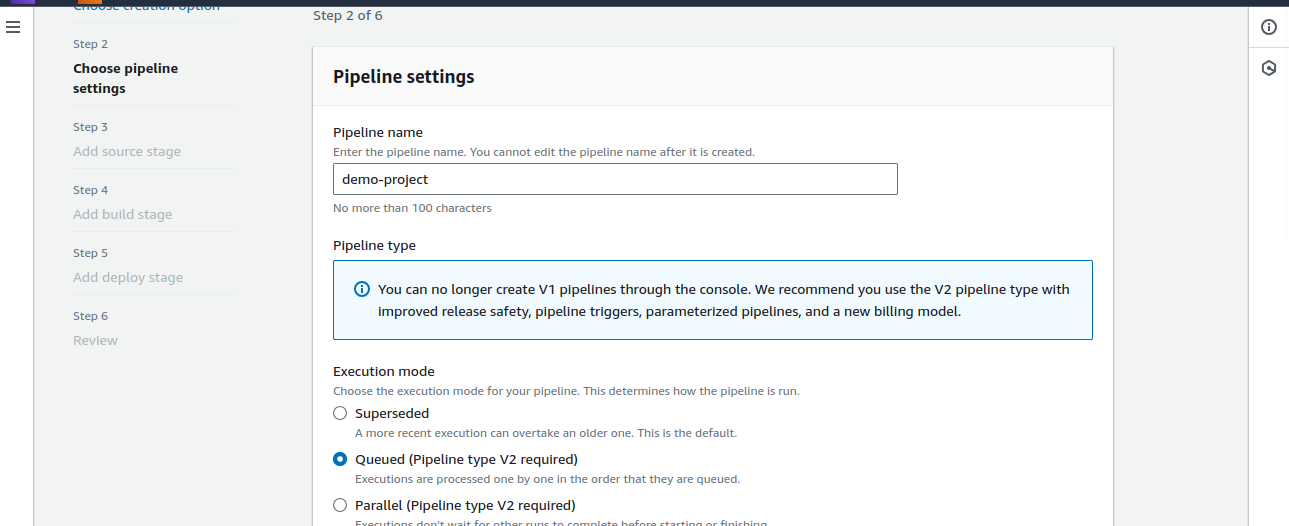
**What is AWS CodePipeline?** AWS CodePipeline is used to automate the CI/CD workflow. It connects the different steps of your pipeline (like CodeBuild for building and CodeDeploy for deployment).

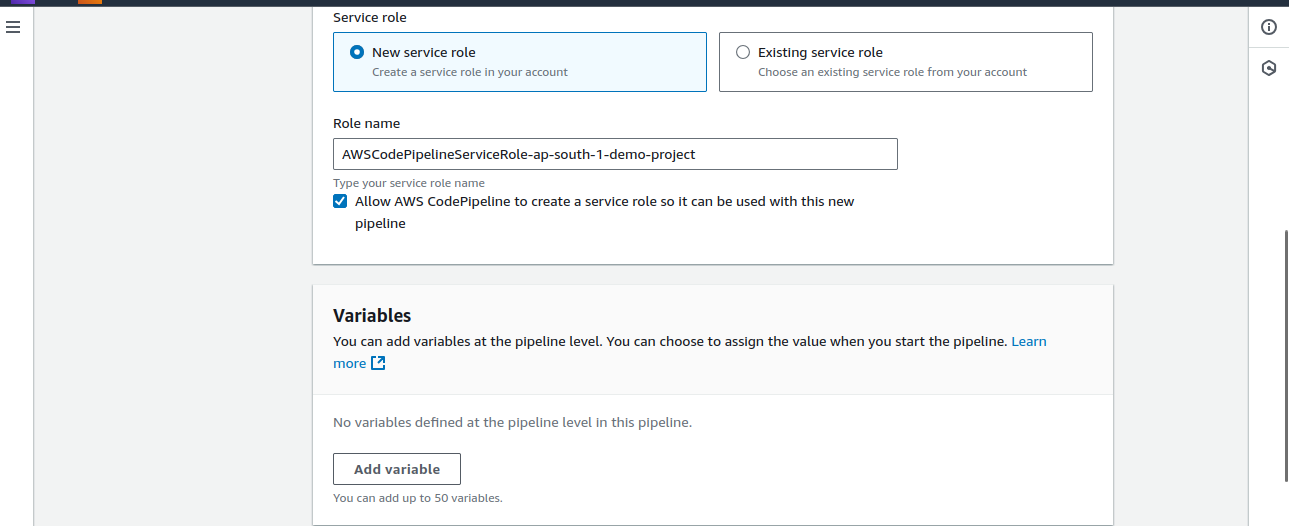
· **How to do it:**

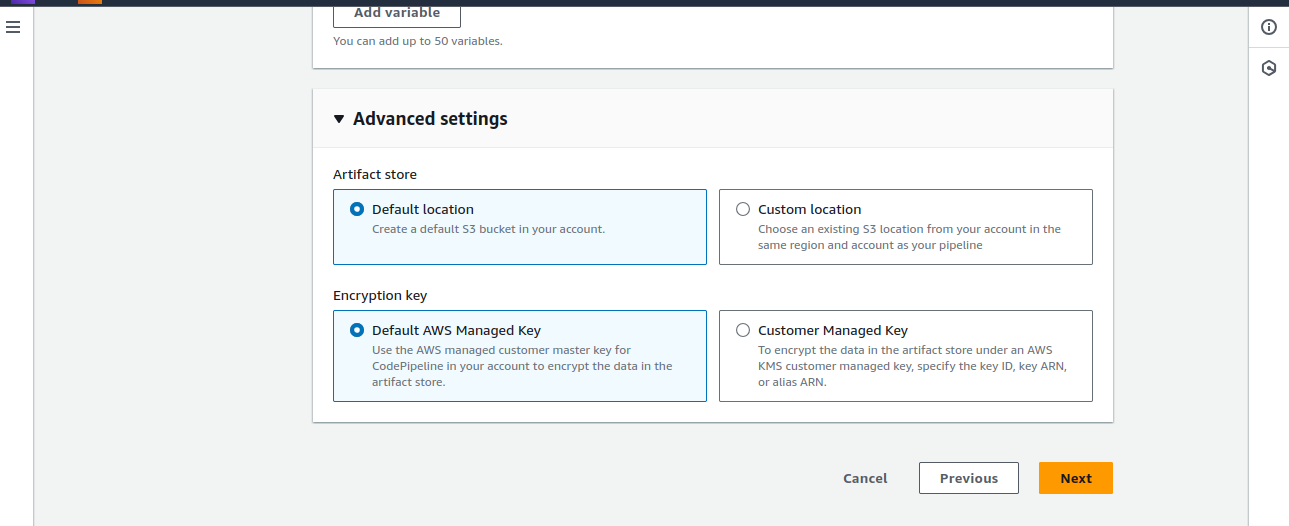
1. Go to **AWS CodePipeline Console**.
2. Create a new pipeline named demo-project.
3. Select **GitHub** as the source.
4. Add **AWS CodeBuild** as the build provider.
5. Skip the deploy stage for now (we will add this later)

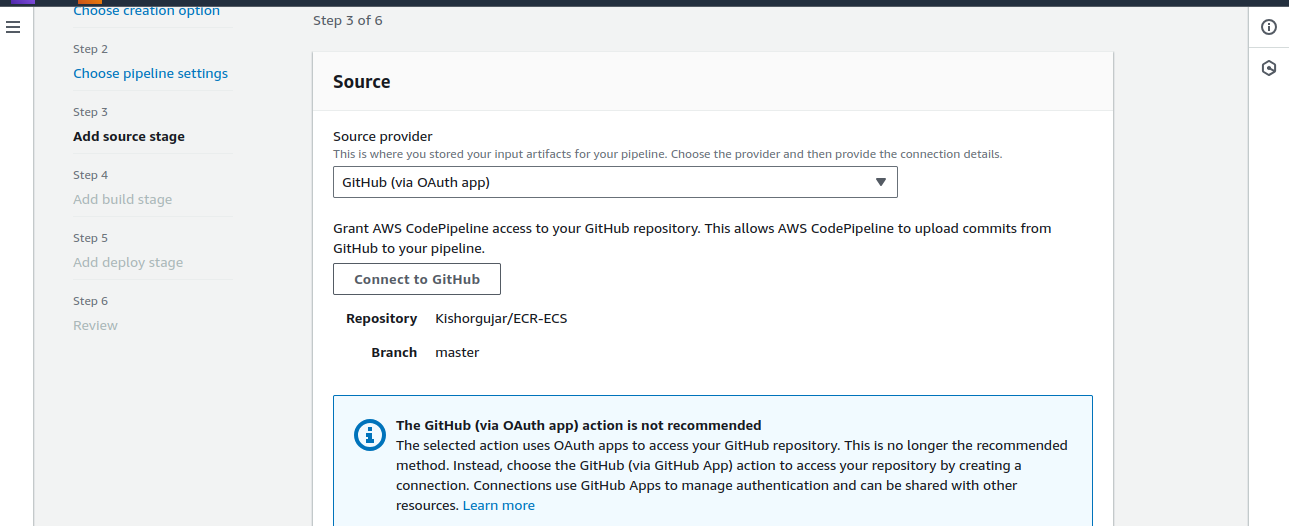


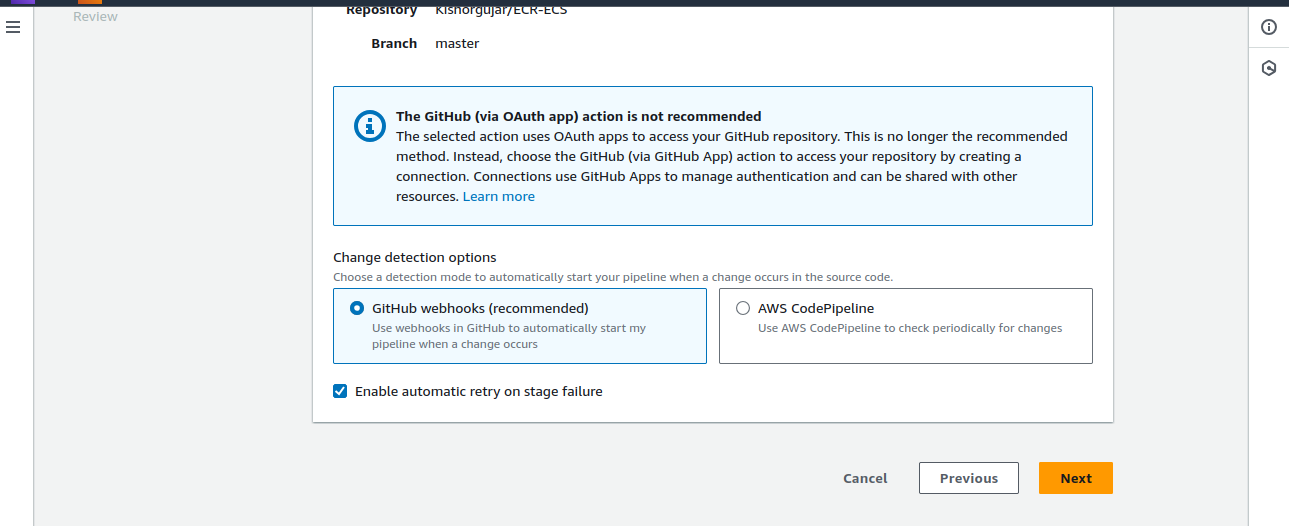


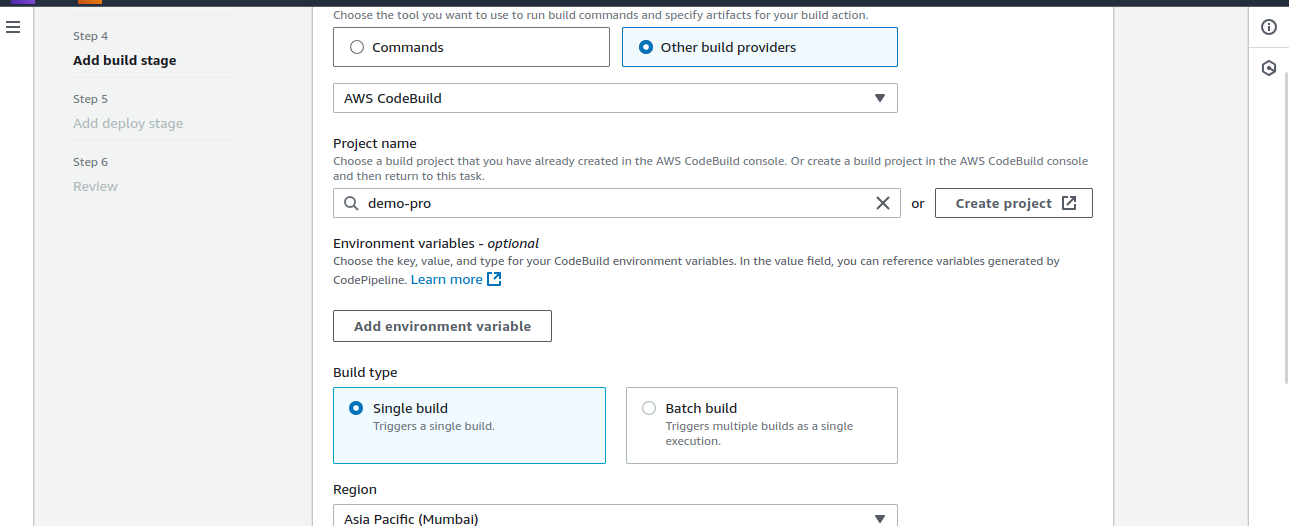


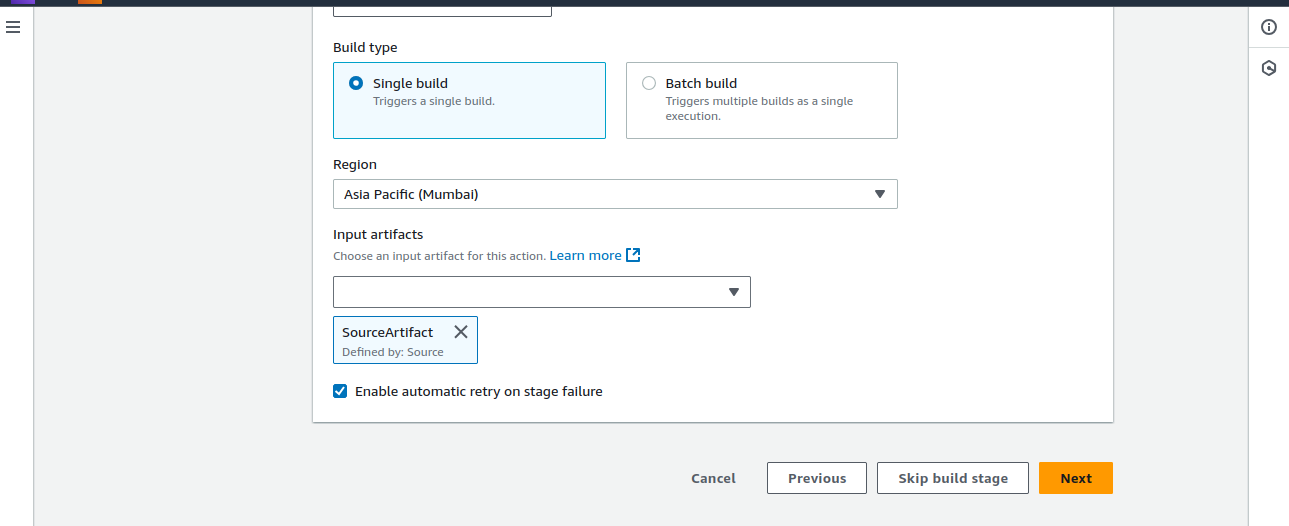


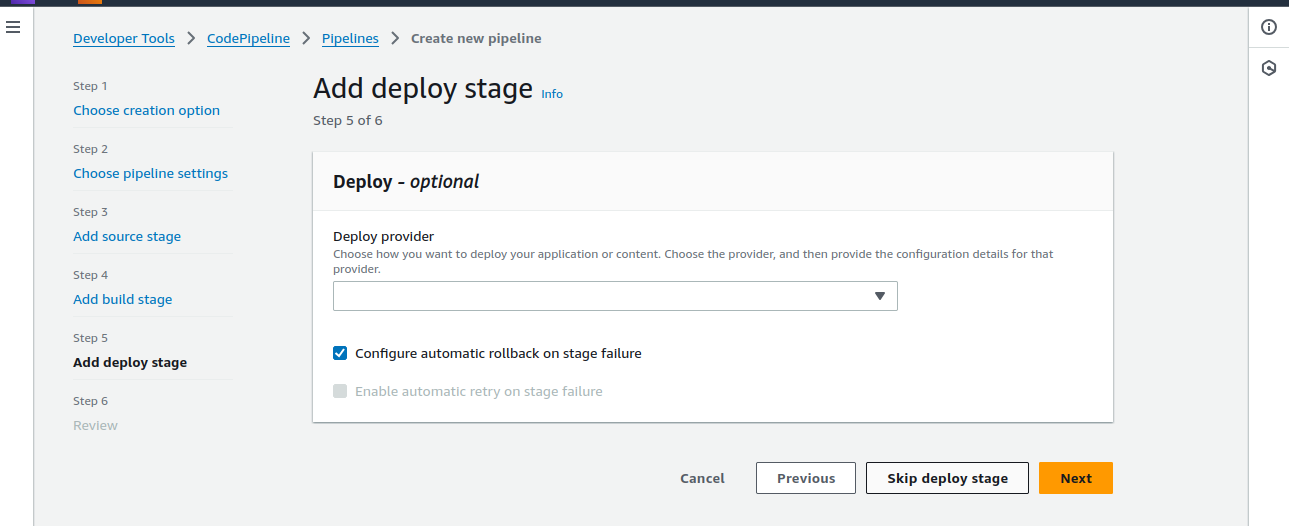


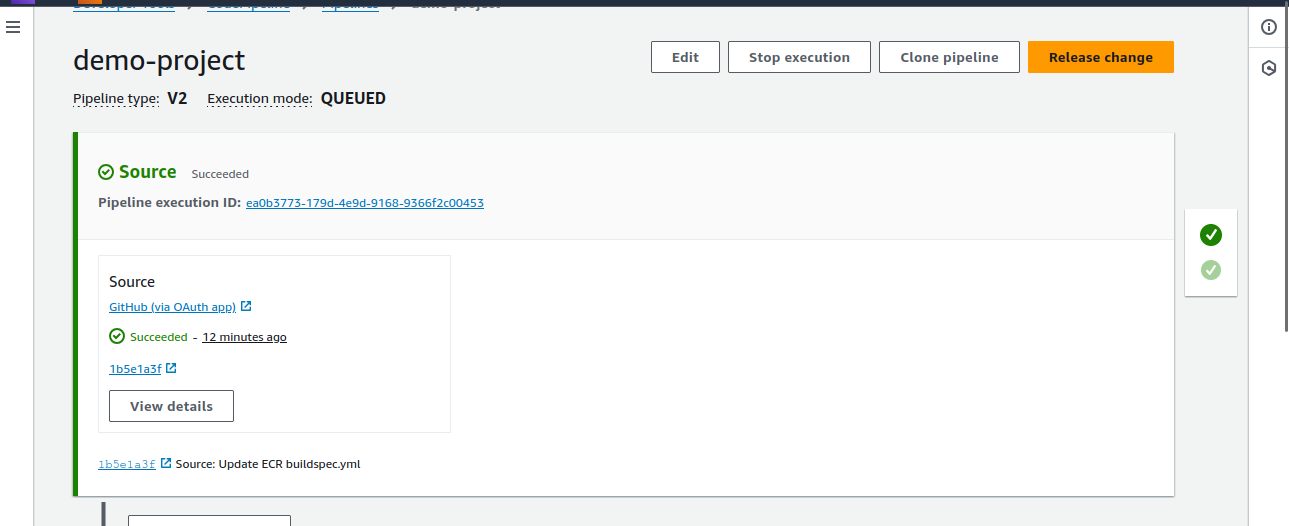


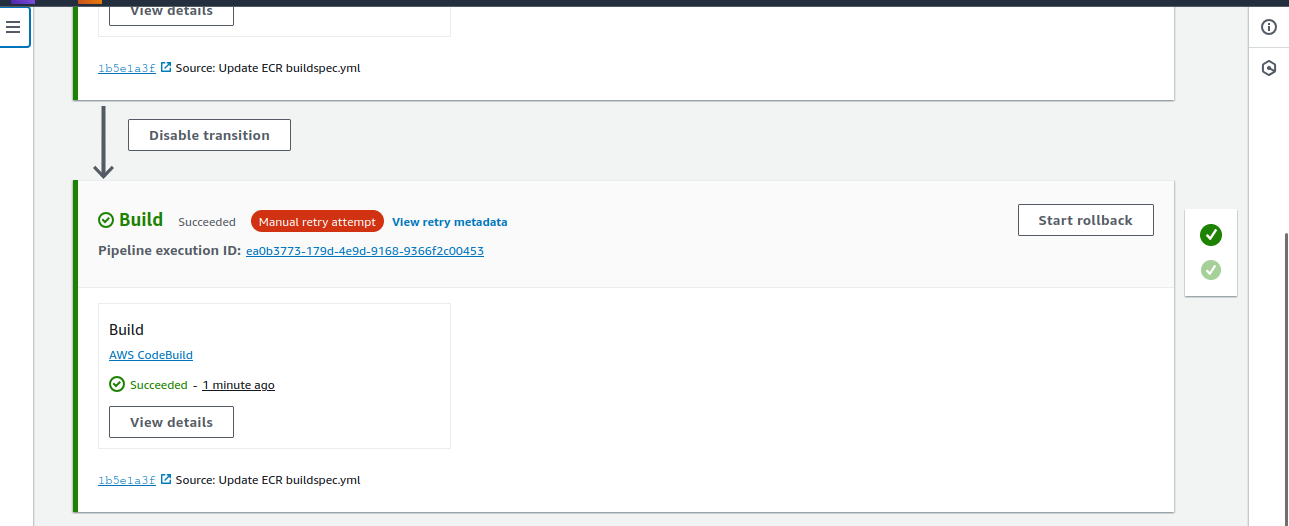










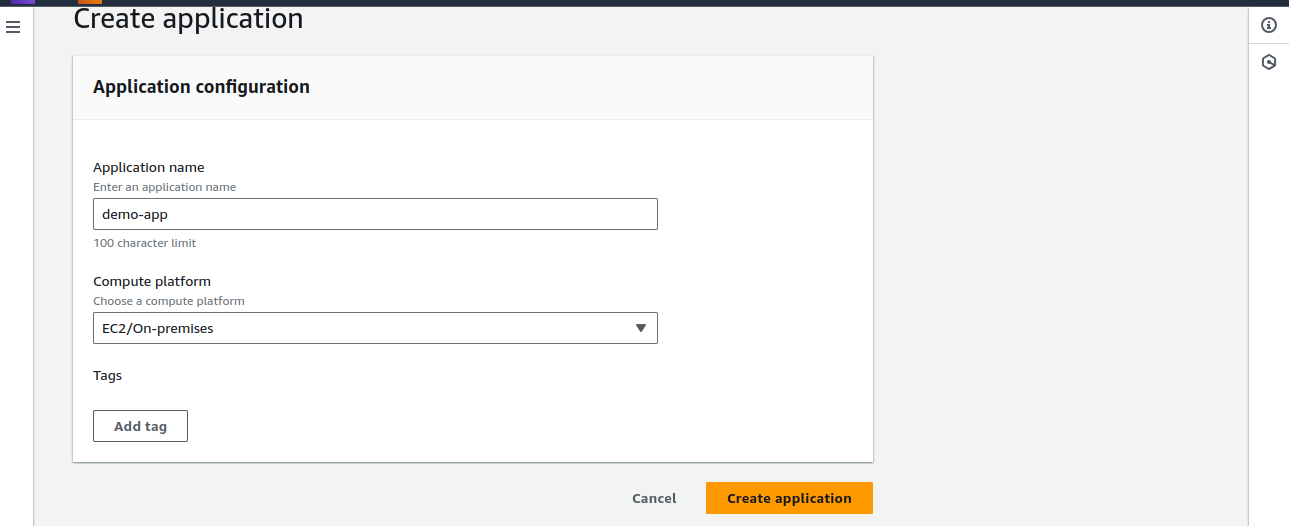


1. **Create CodeDeploy Application and Deployment Group**

**What is CodeDeploy?** CodeDeploy is used to deploy your application to EC2 or Lambda. In this case, we deploy the Docker image to an EC2 instance.

· **How to do it:**

1. Go to **AWS CodeDeploy Console**.
2. Create an application (e.g., demo-app).
3. Create a deployment group (e.g., demo-deploy-group) and choose **EC2** as the compute platform.

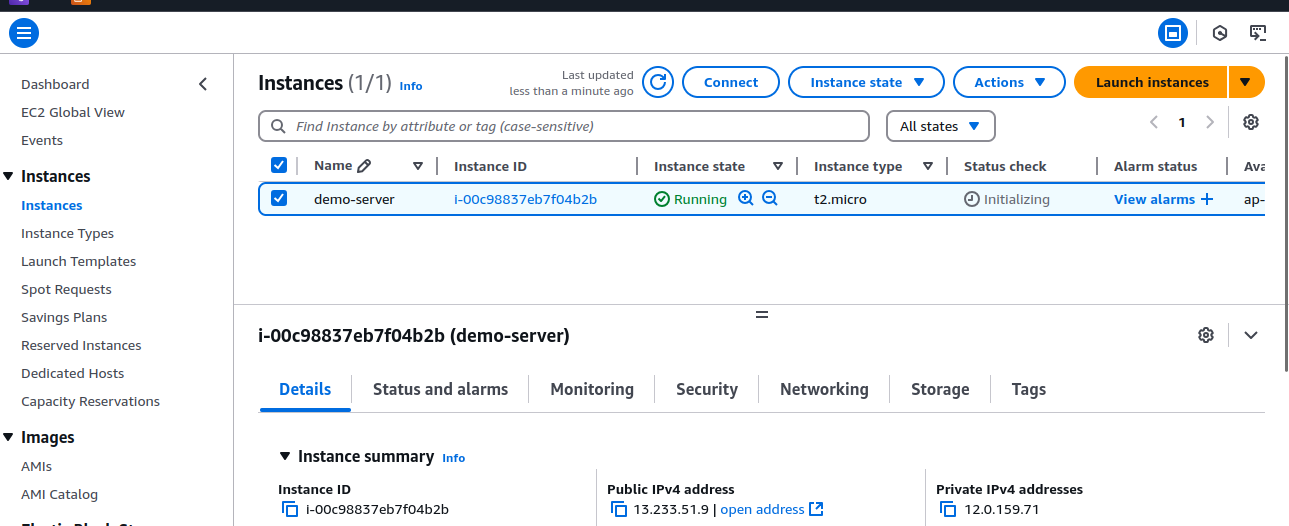


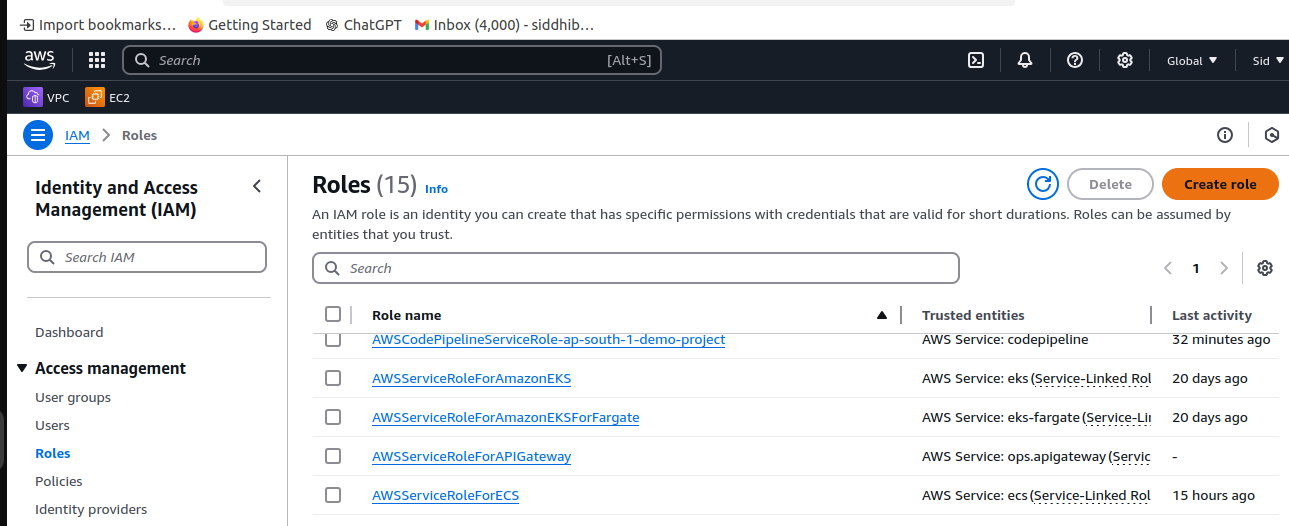
1. **Set Up EC2 Instance for Deployment**

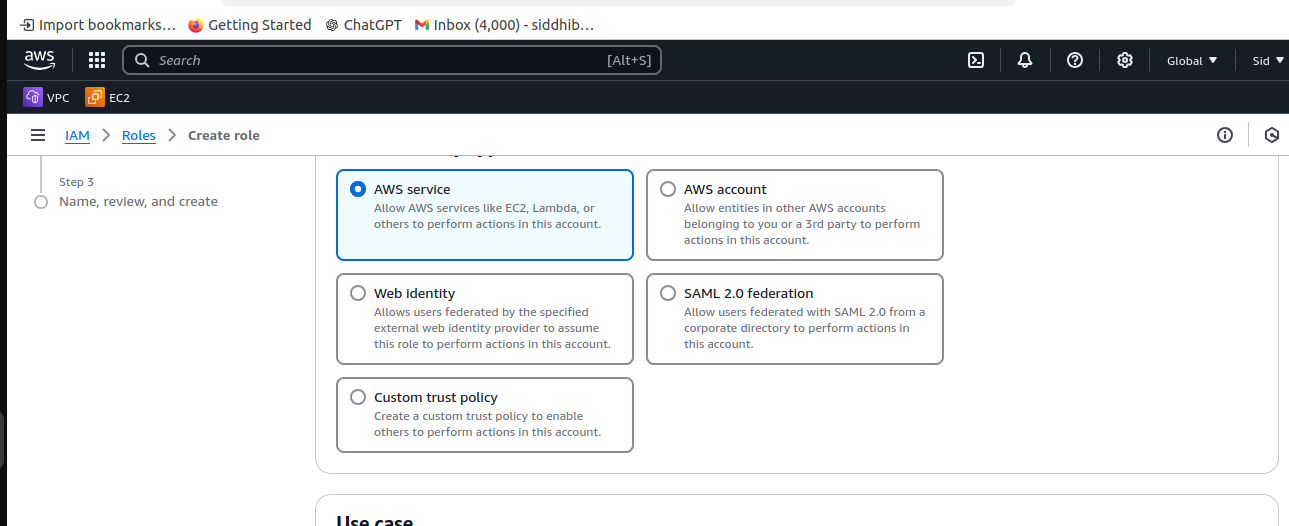
**What is EC2?** EC2 provides virtual machines that can run your applications.

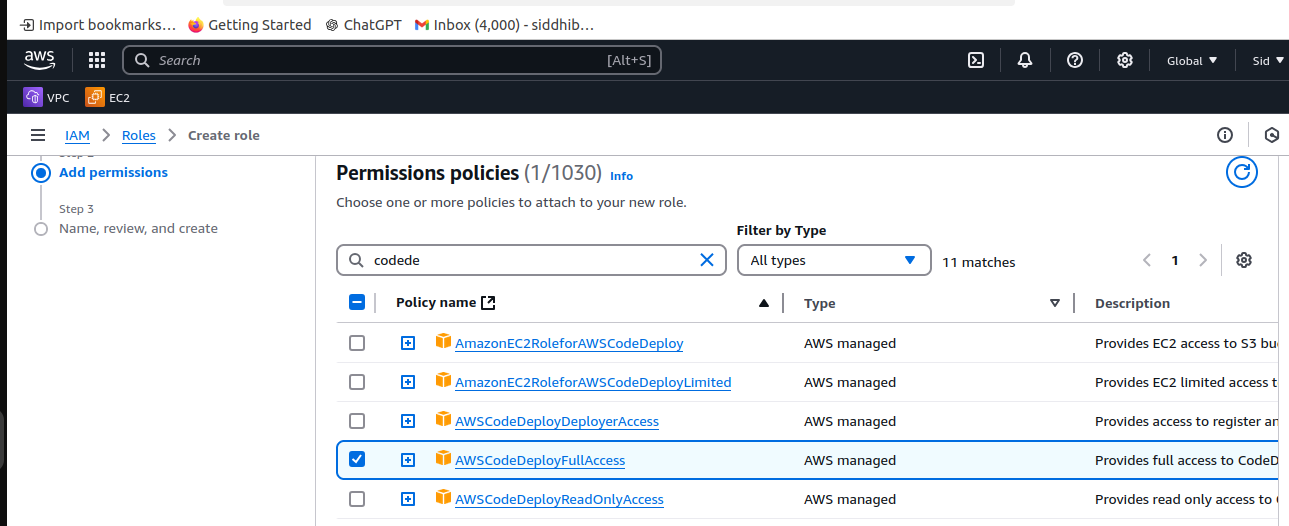
· **How to do it:**

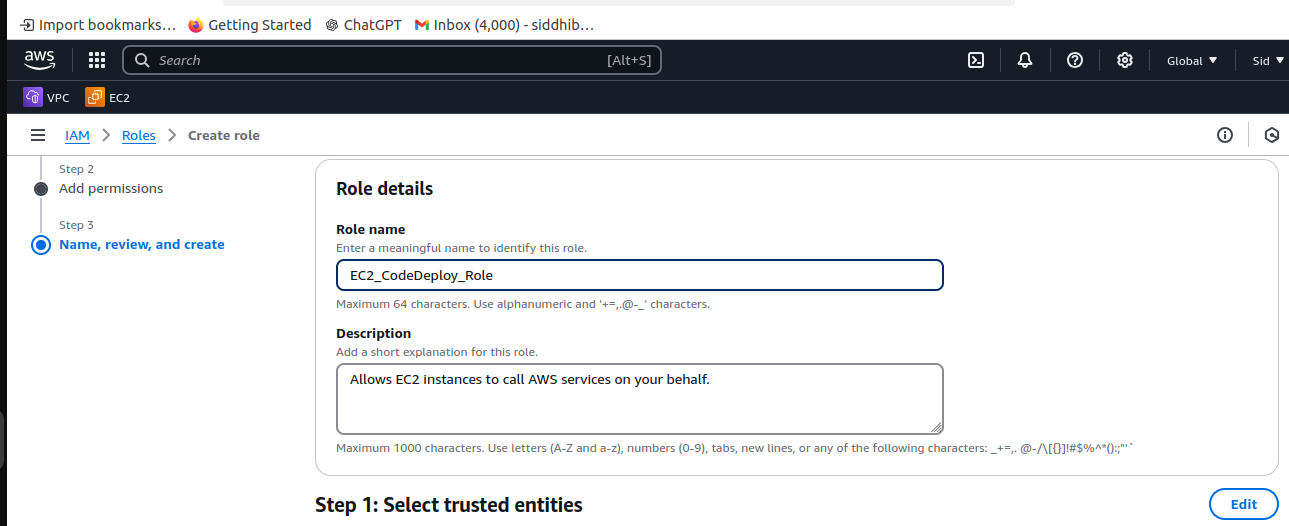
1. Launch an EC2 instance (Ubuntu) from the EC2 console.
2. Install the **CodeDeploy agent** on the EC2 instance using the following documentation: <https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install-ubuntu.html>
3. Attach necessary IAM roles to the EC2 instance and ensure they have permissions to access CodeDeploy, ECR, and other resources.

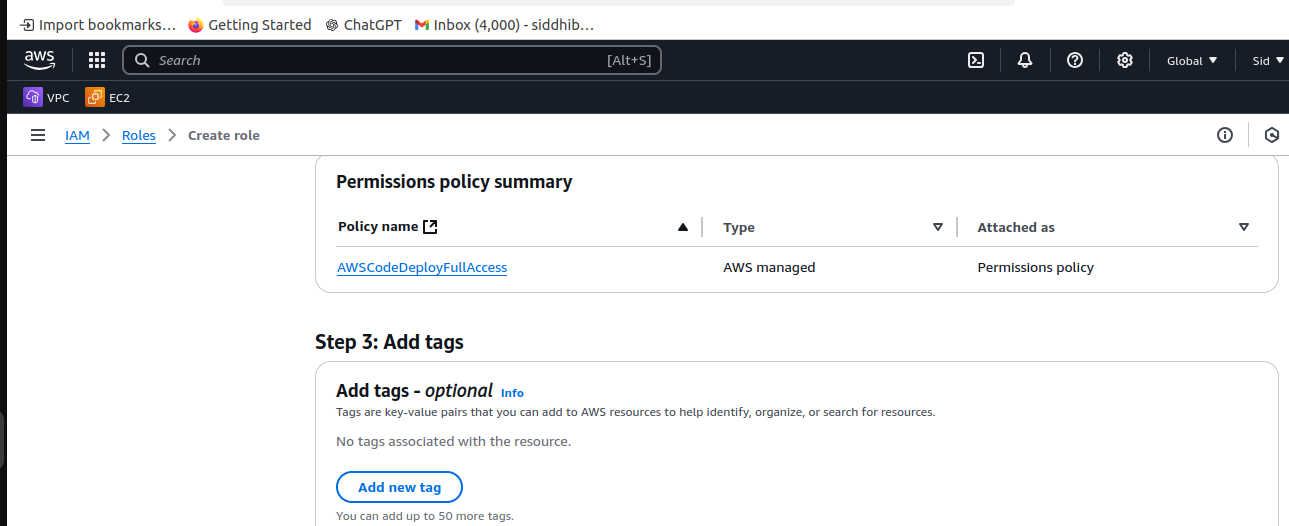




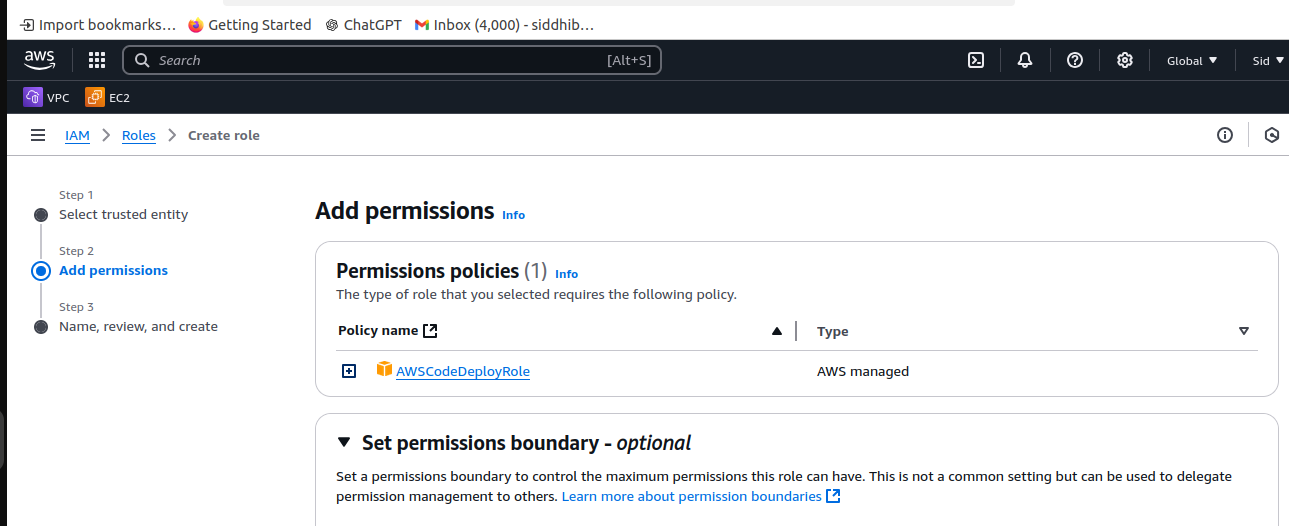


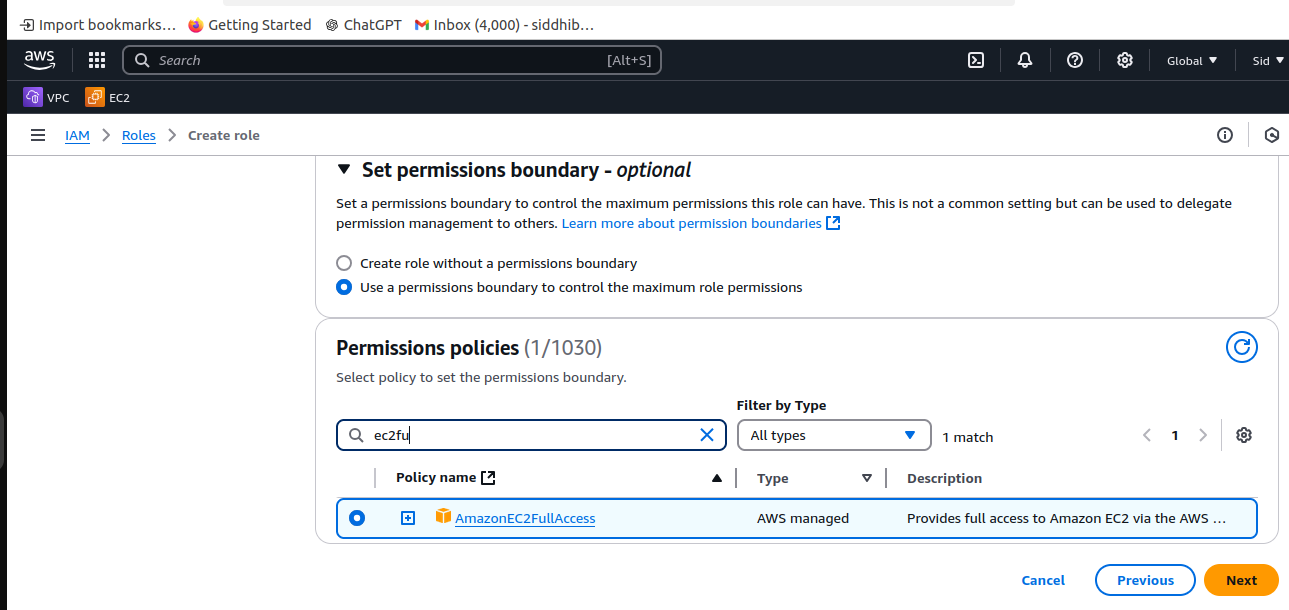


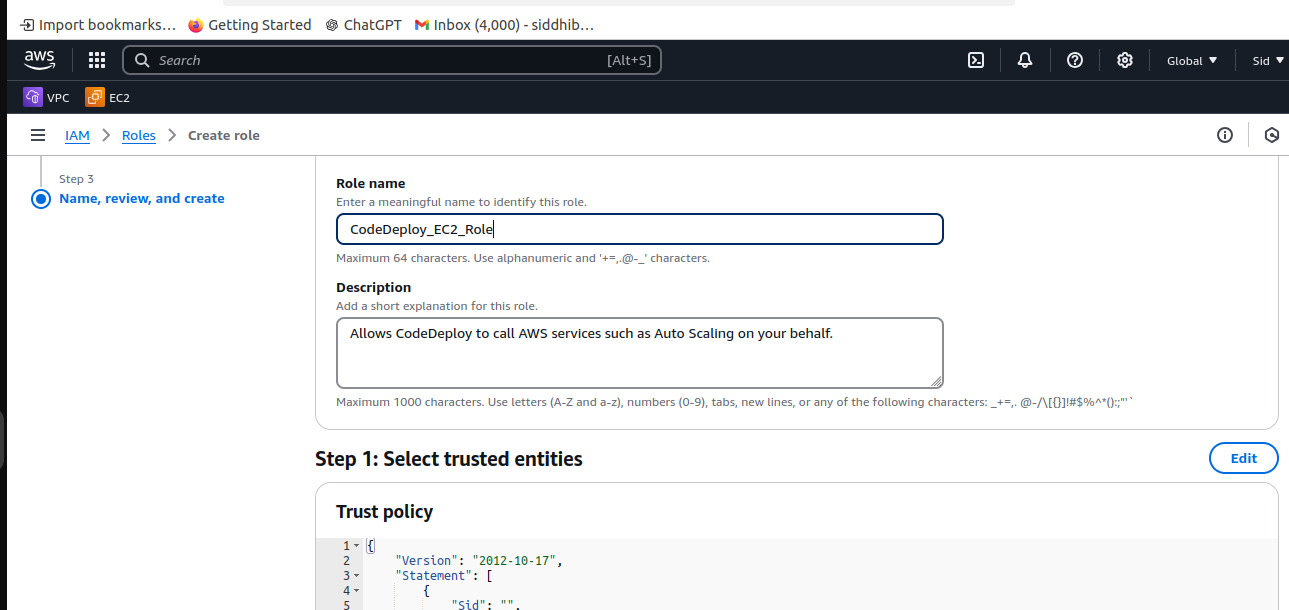


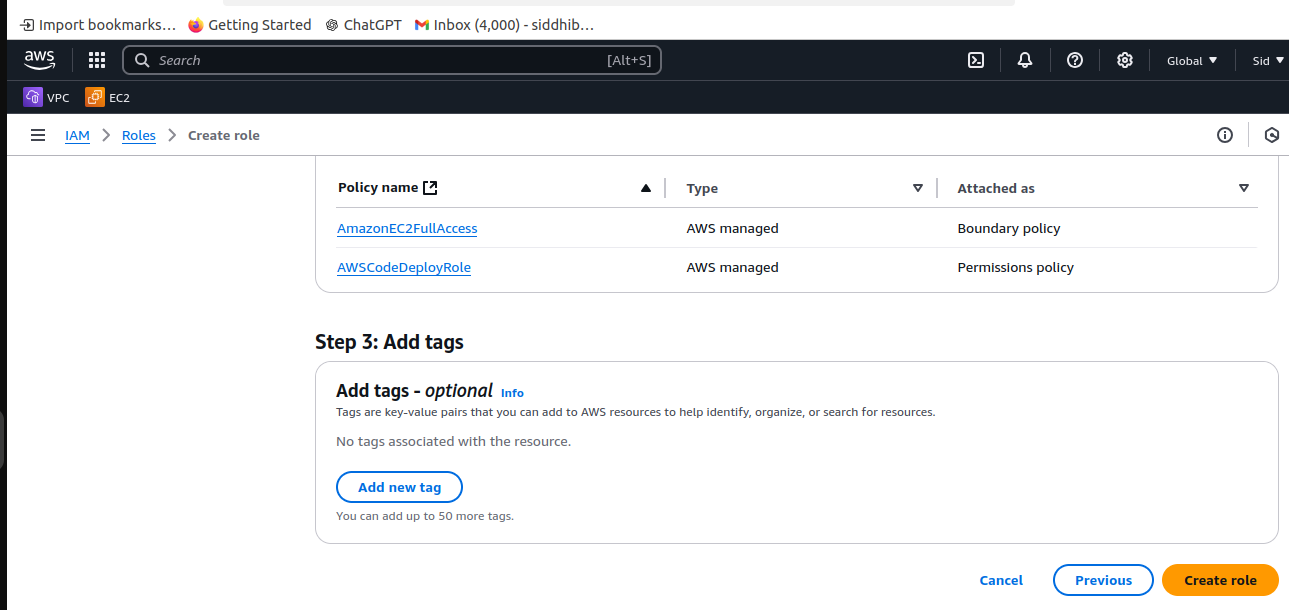


- Role For CodeDeploy

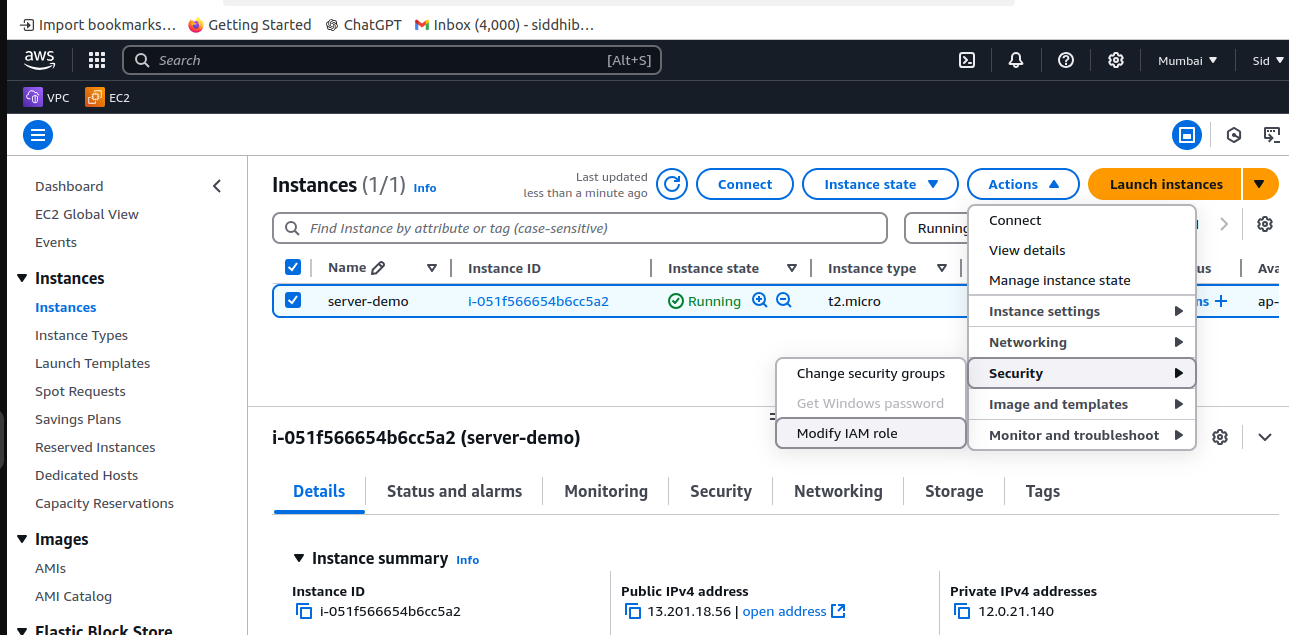


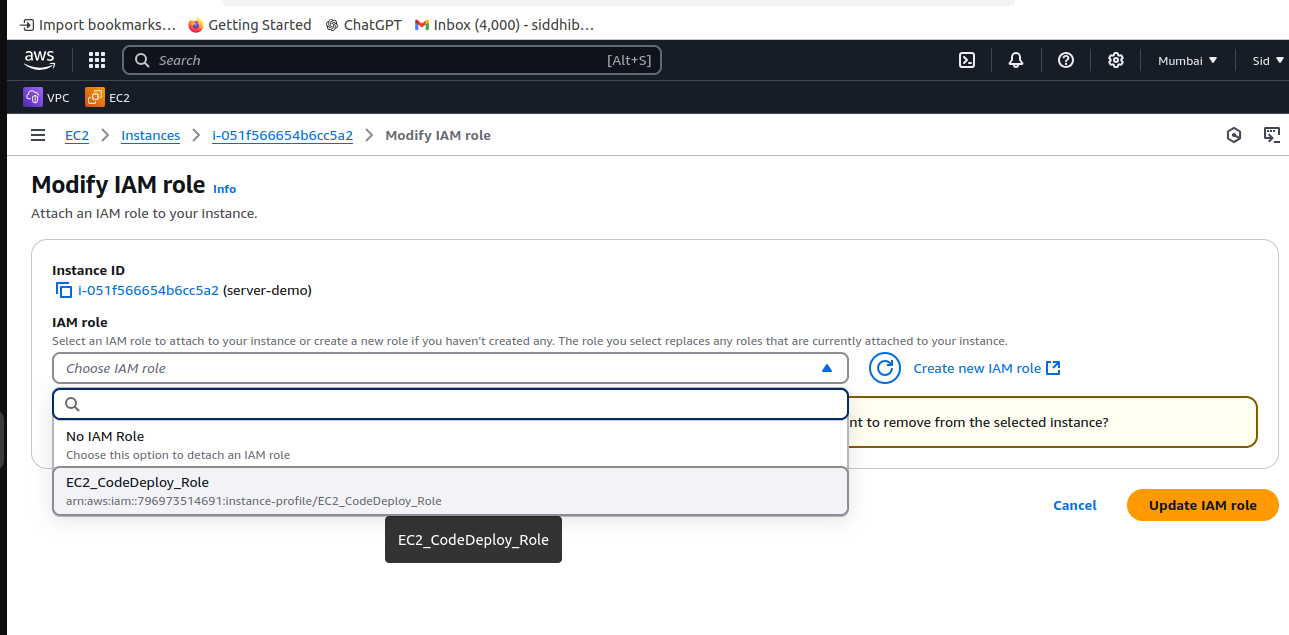






- Assign role to instance



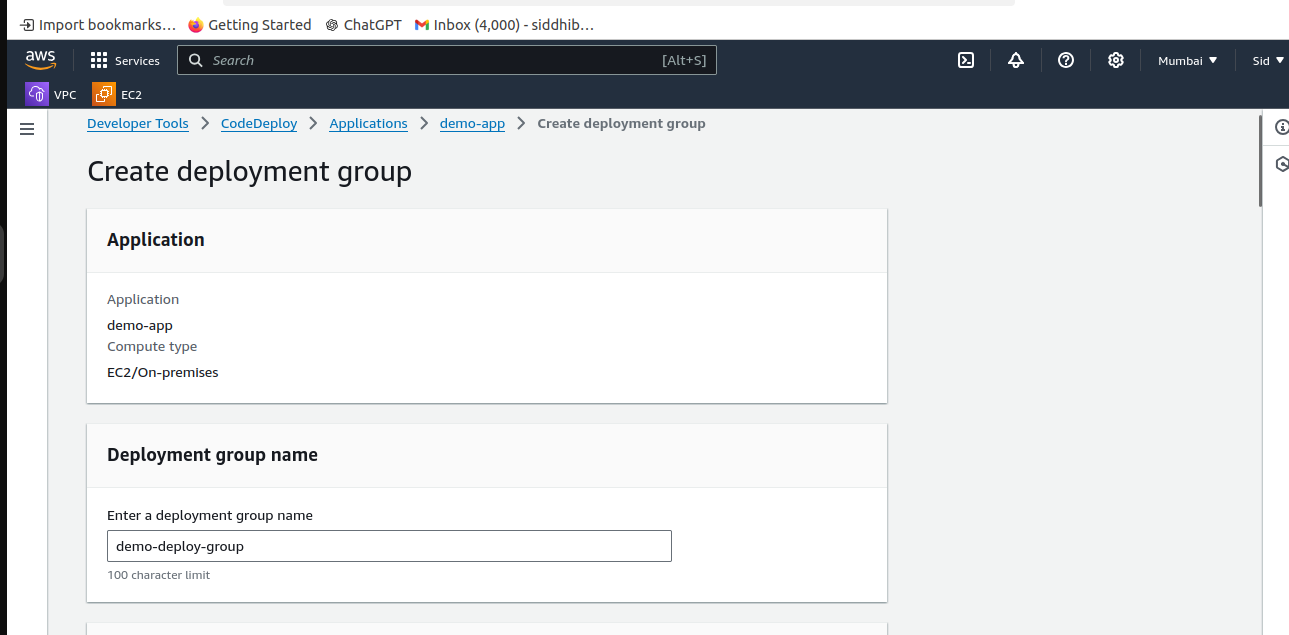


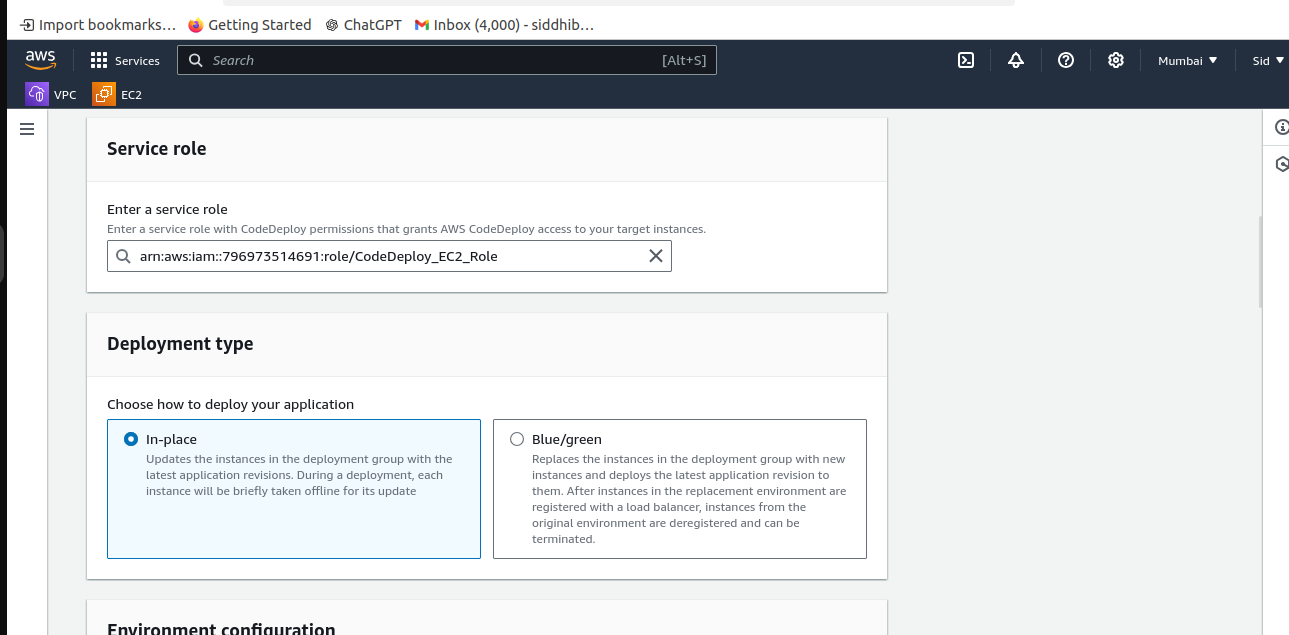
1. **Configure CodeDeploy Deployment Group**

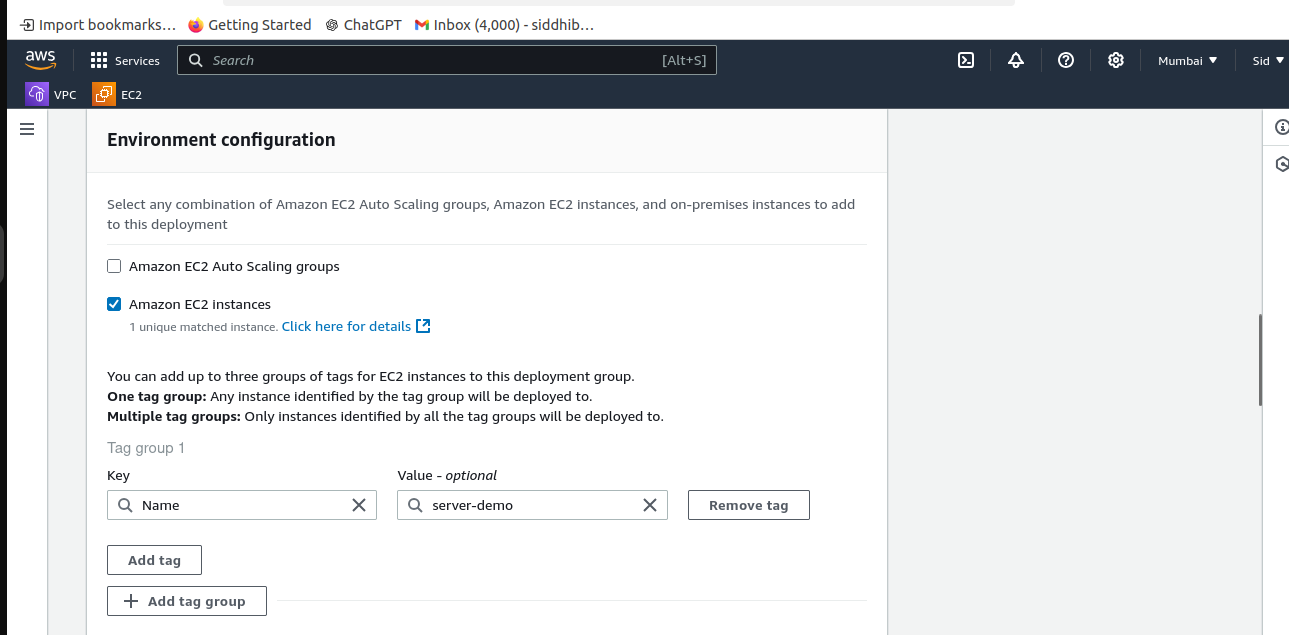
**What is Deployment Group?** A deployment group is a set of EC2 instances where CodeDeploy deploys the application.

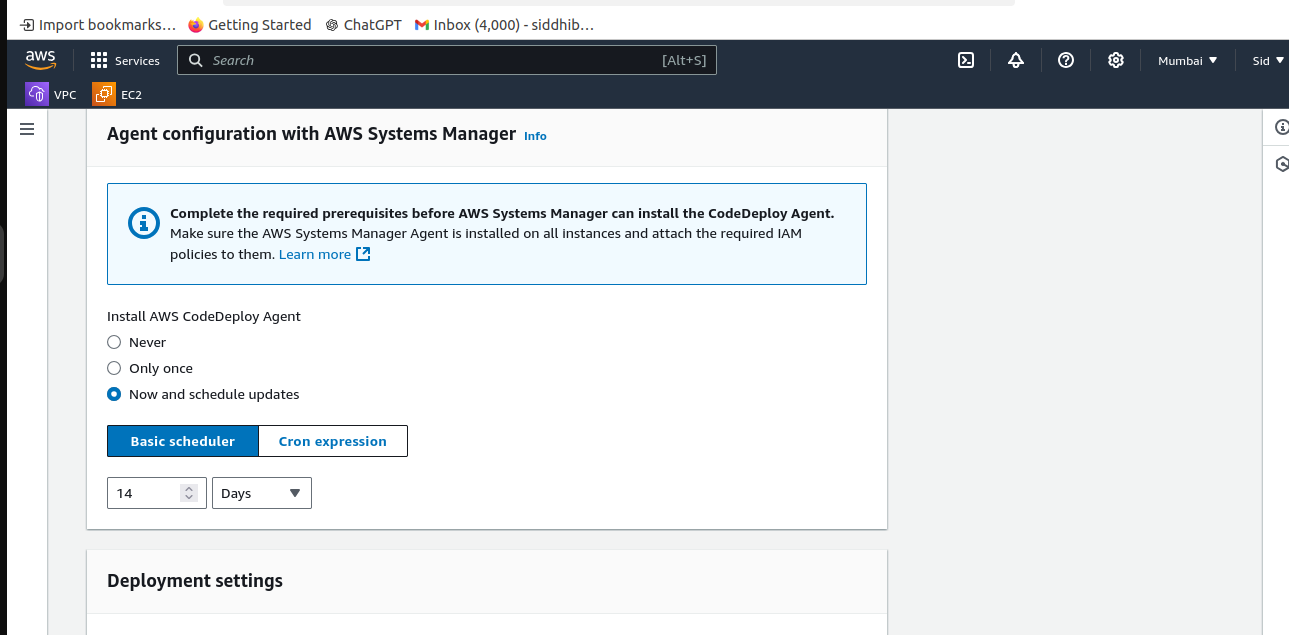
· **How to do it:**

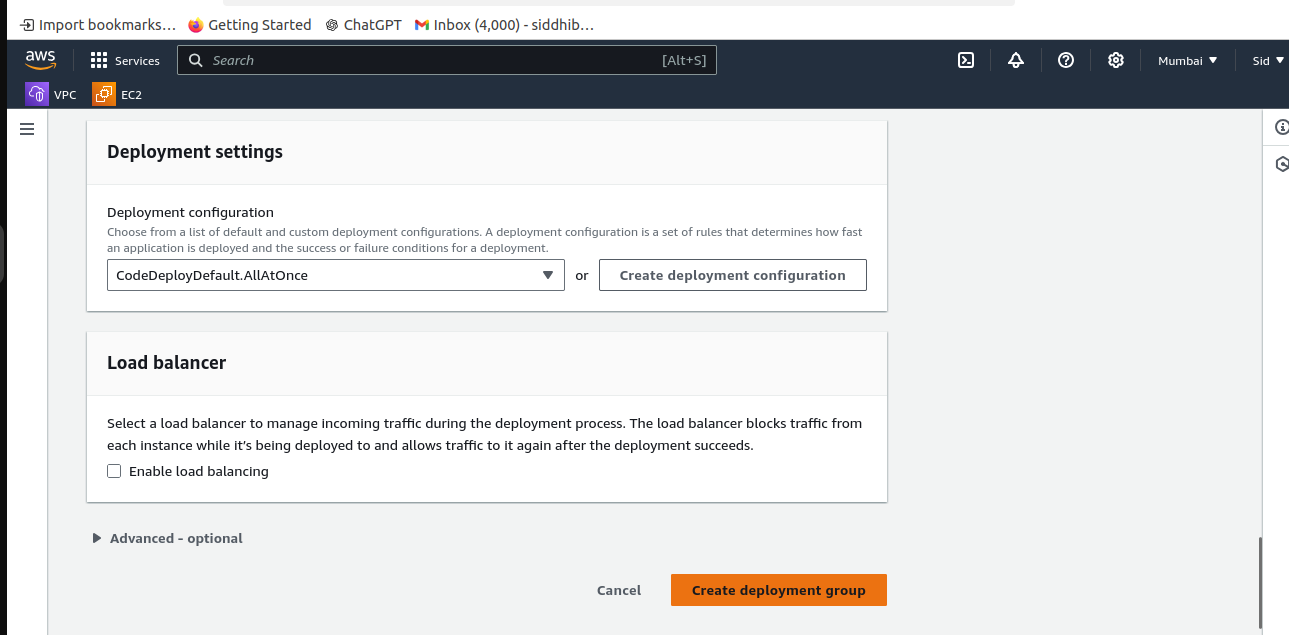
1. Define the deployment group in CodeDeploy (e.g., demo-deploy-group).
2. Choose **In-Place** deployment type, which updates the existing EC2 instance.









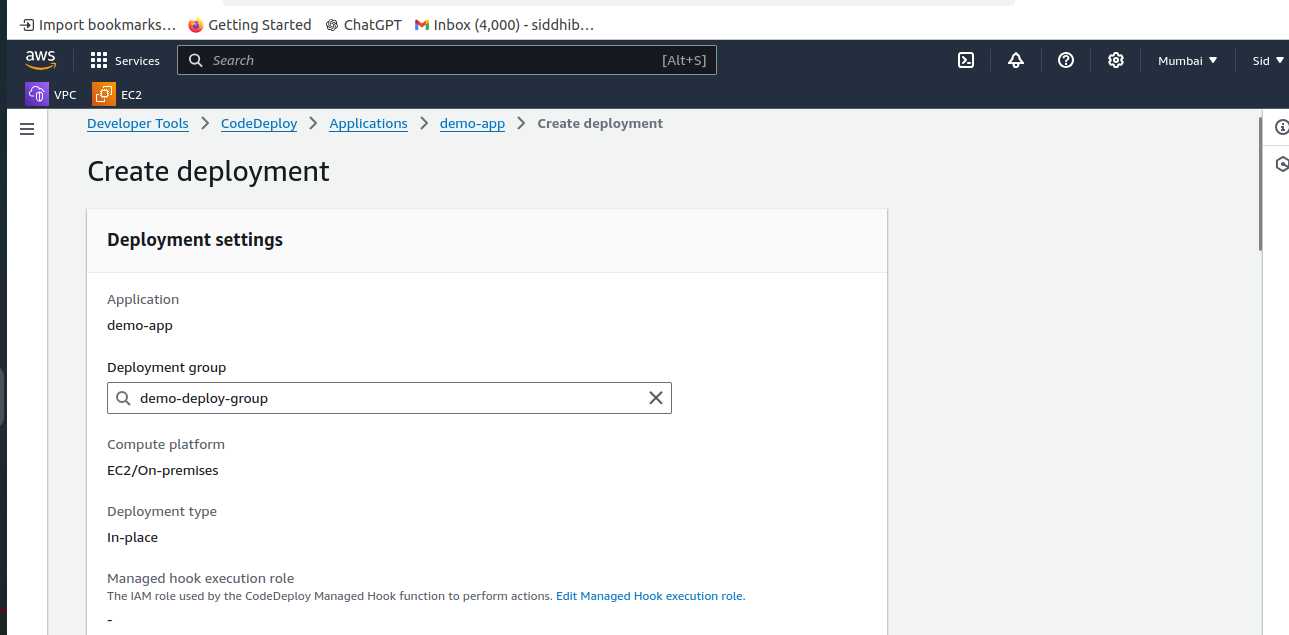


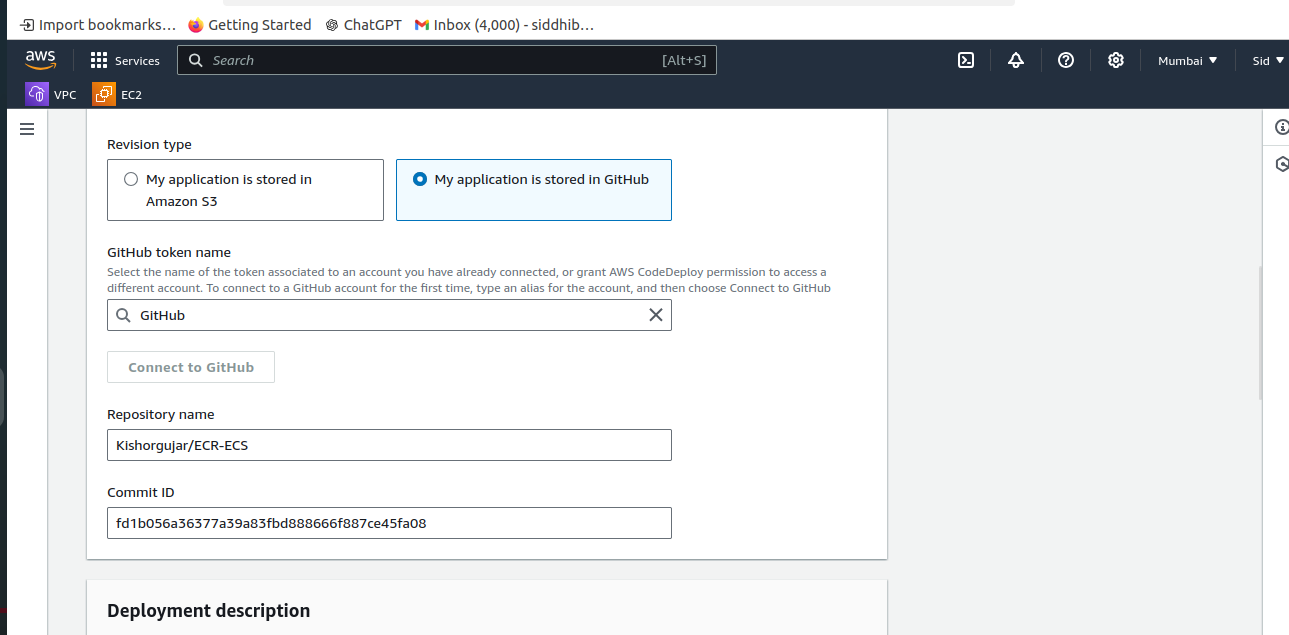
**8] Create a Deployment Using CodeDeploy**

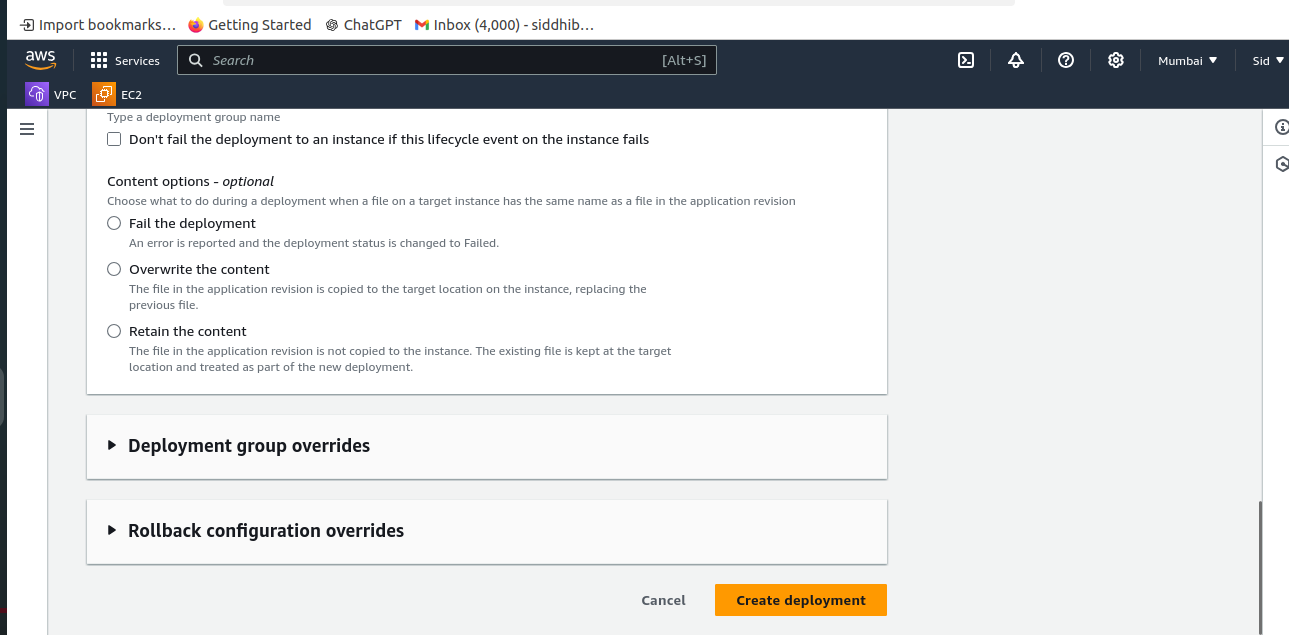
**What is a Deployment?** A deployment is the process of transferring the built Docker image to the EC2 instance.

· **How to do it:**

1. Create a deployment in CodeDeploy, choosing the GitHub repository as the source.
2. Select the deployment group demo-deploy-group.



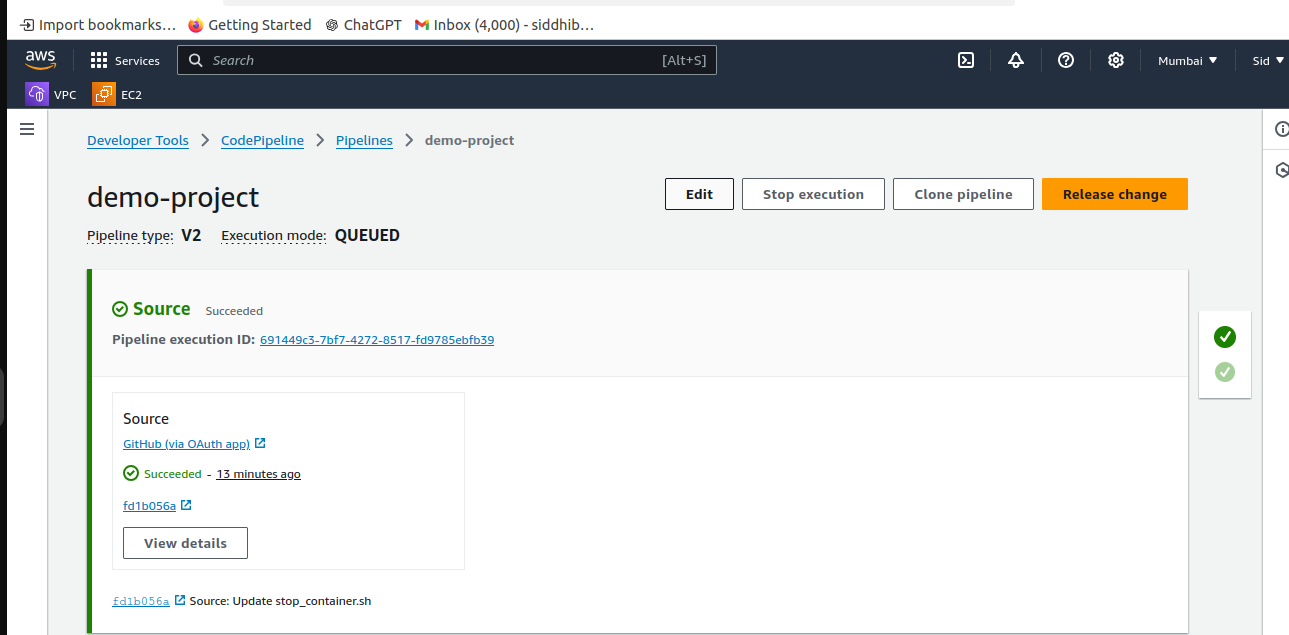


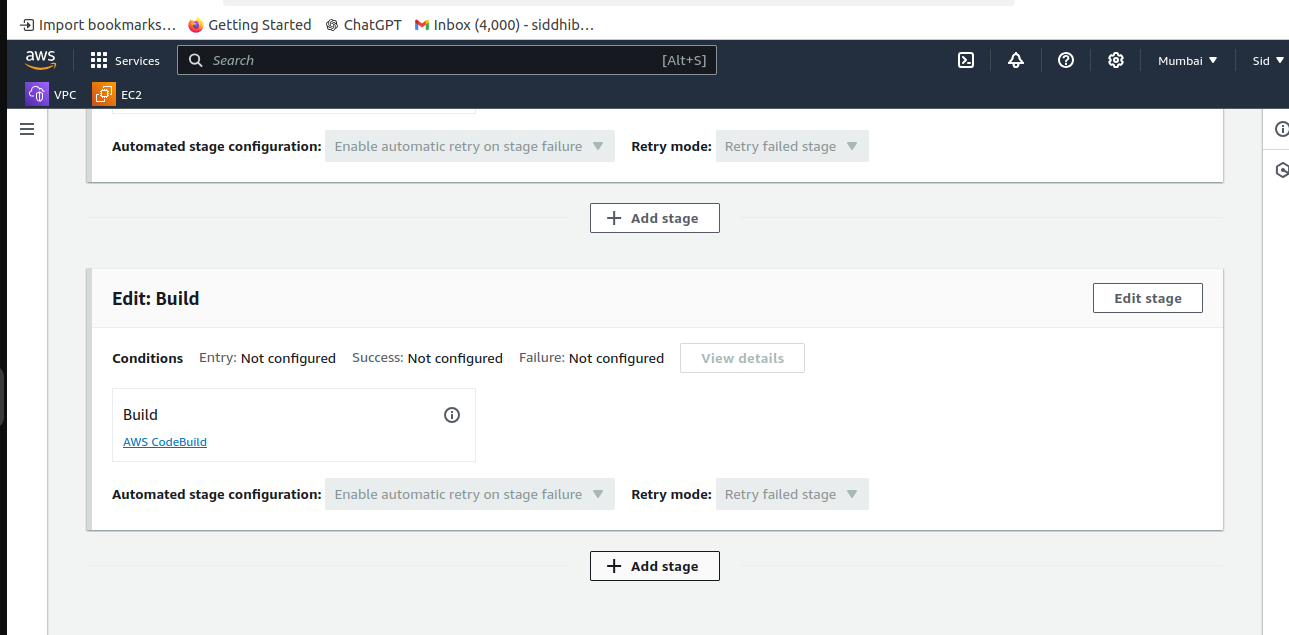


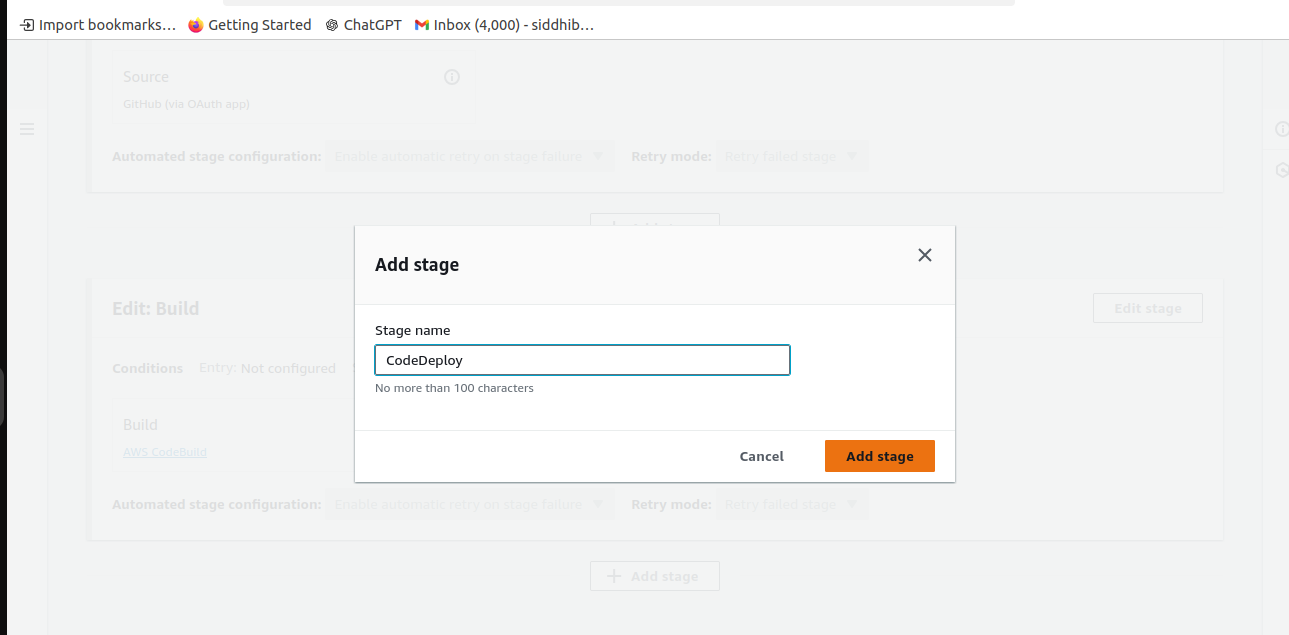
**9] Add CodeDeploy Stage in CodePipeline**

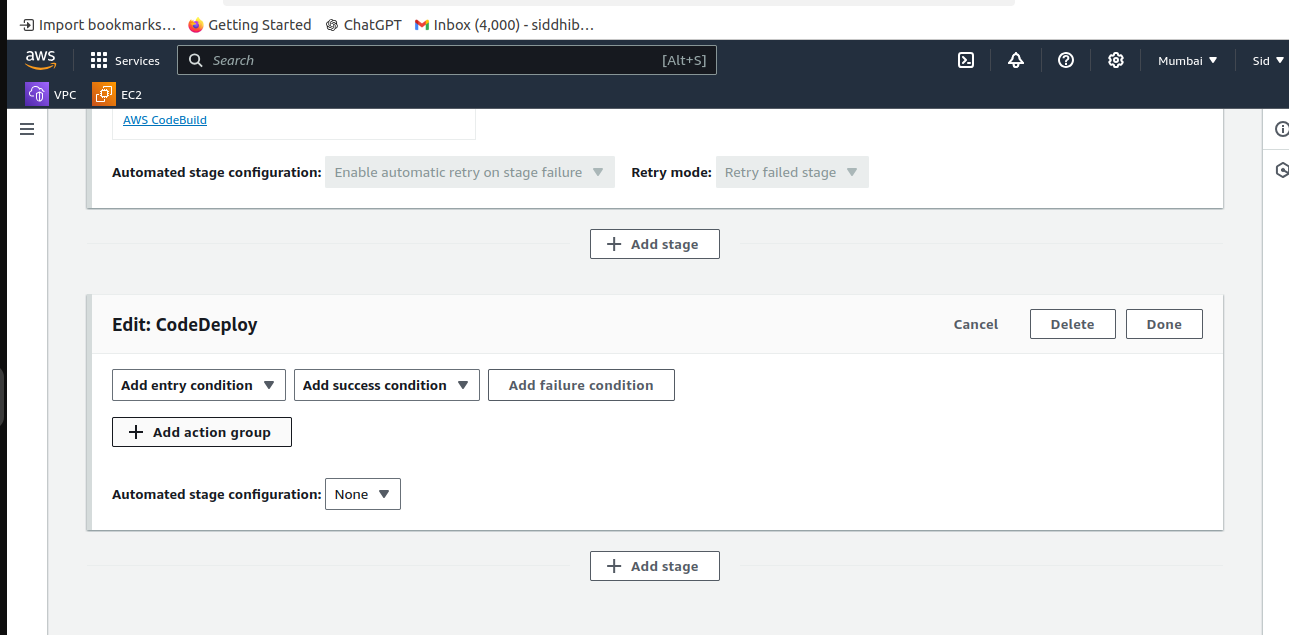
**How to do it:**

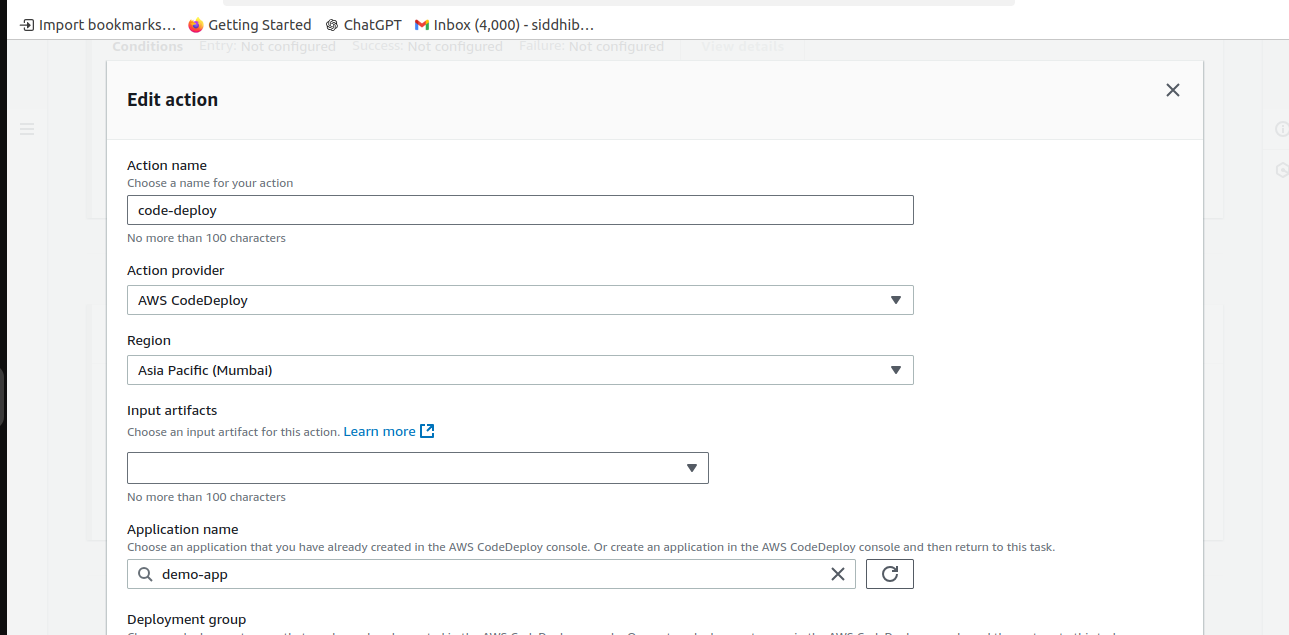
1. In **AWS CodePipeline**, add a new stage for **CodeDeploy** after the build step.
2. Provide details like the application name and deployment group to the pipeline.

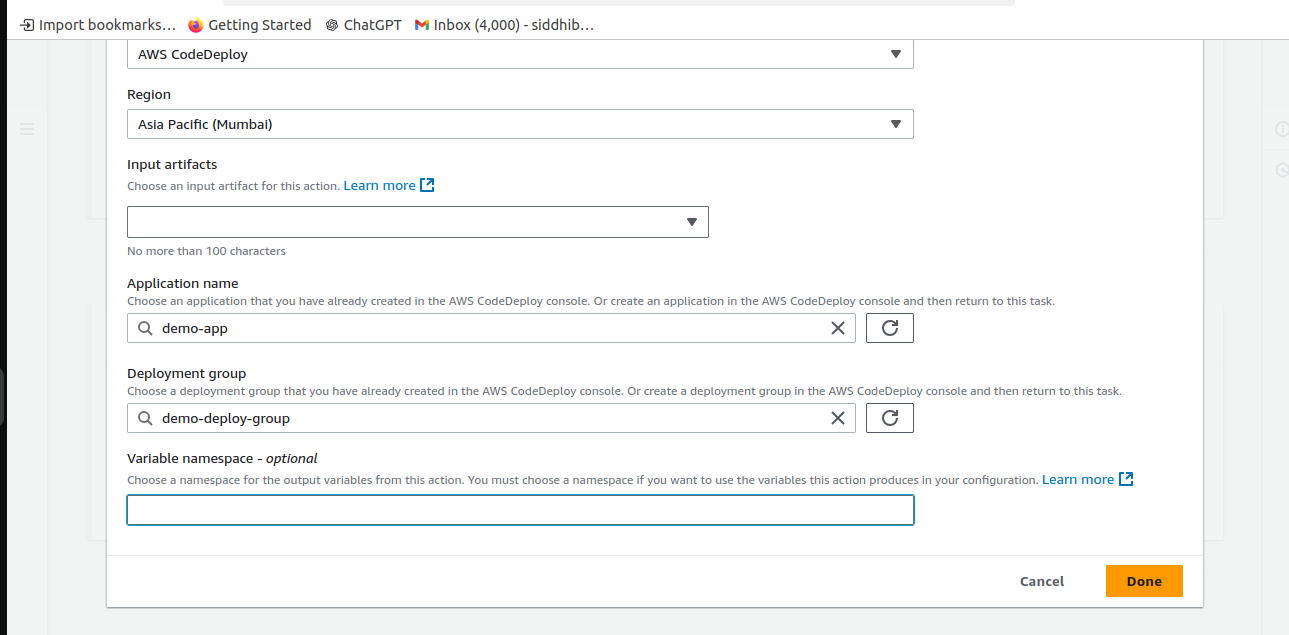








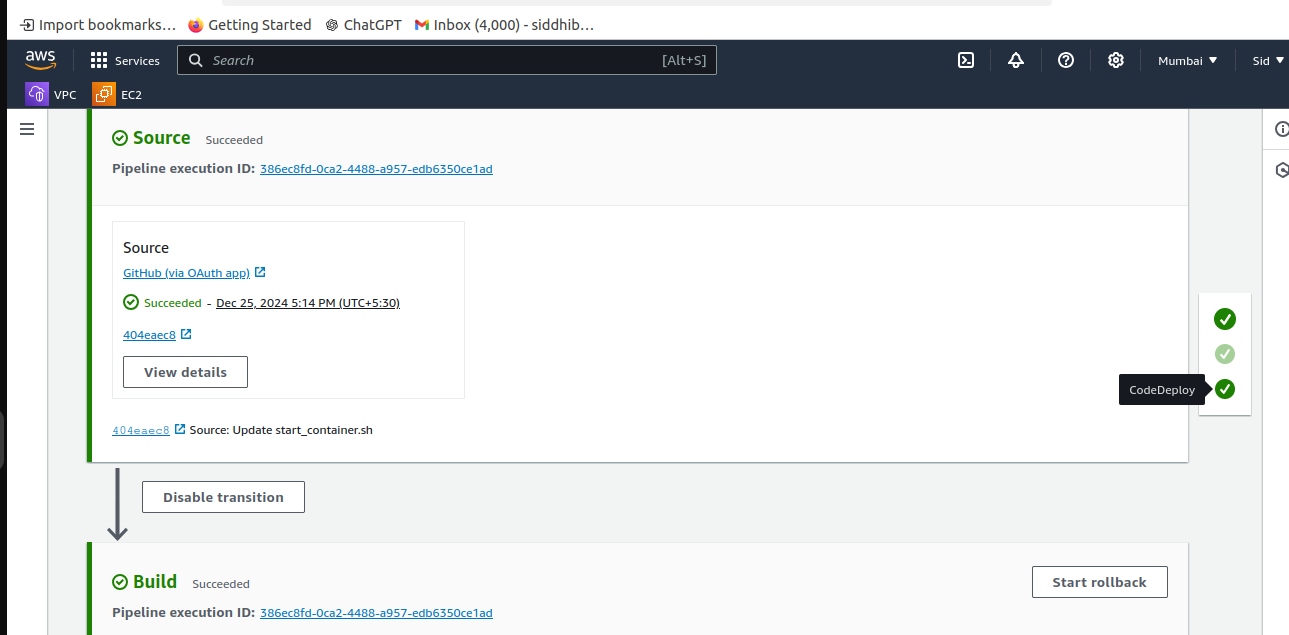




**10] Test the Pipeline**

**How to do it:**

1. Make changes to your GitHub repository to trigger the pipeline.
2. Check the CodePipeline to ensure the code is built and deployed automatically without any manual intervention.



### ****4. Conclusion****

By following the steps outlined in this guide, you have successfully created a CI/CD pipeline using AWS tools. This pipeline automates the process of building, testing, and deploying a Python application, allowing you to quickly release new updates in a controlled and automated manner. AWS services like CodeBuild, CodeDeploy, and CodePipeline work together seamlessly to ensure fast and reliable software delivery.

This approach significantly reduces manual tasks and helps in delivering production-ready applications quickly and with fewer errors.