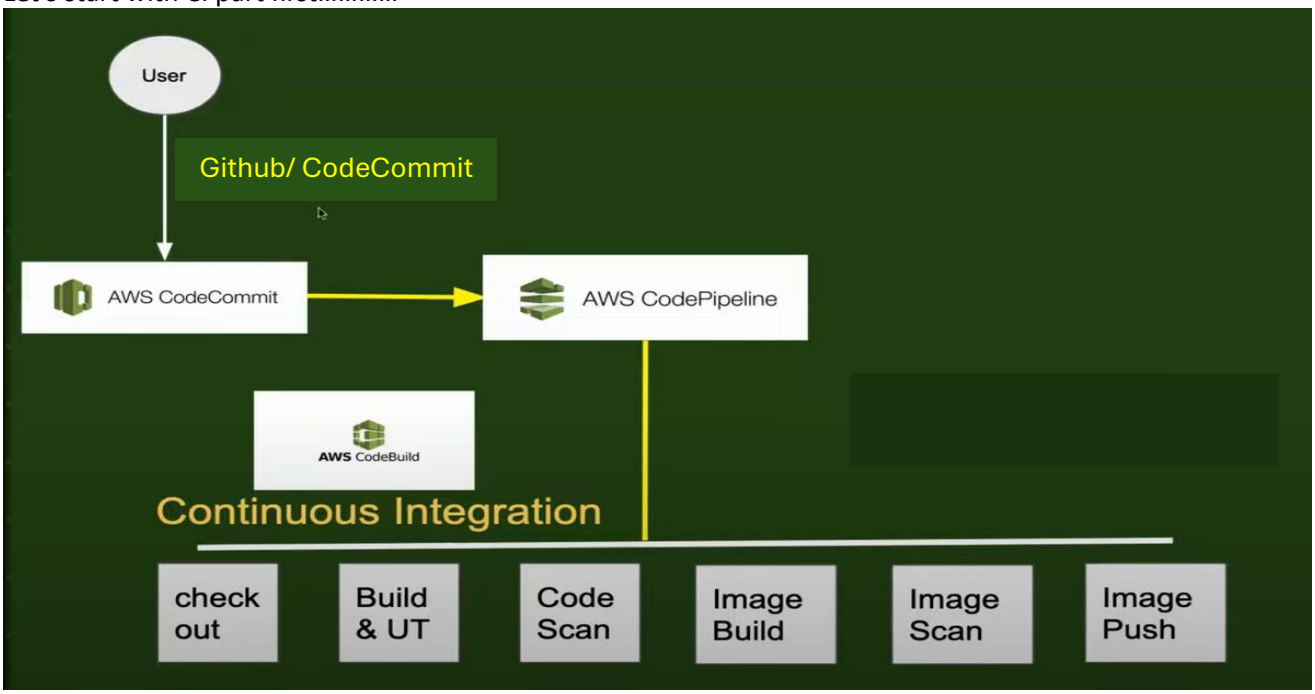
End-to-End CI/CD Implementation for Python Flask Application on AWS

Let's start with CI part first……

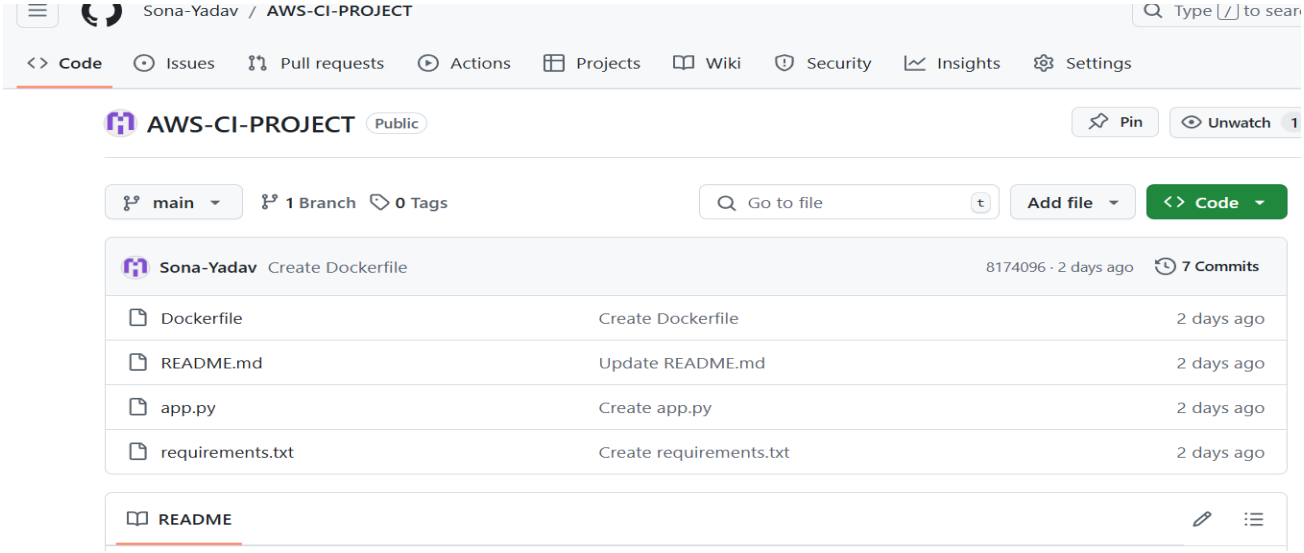


CI Diagram

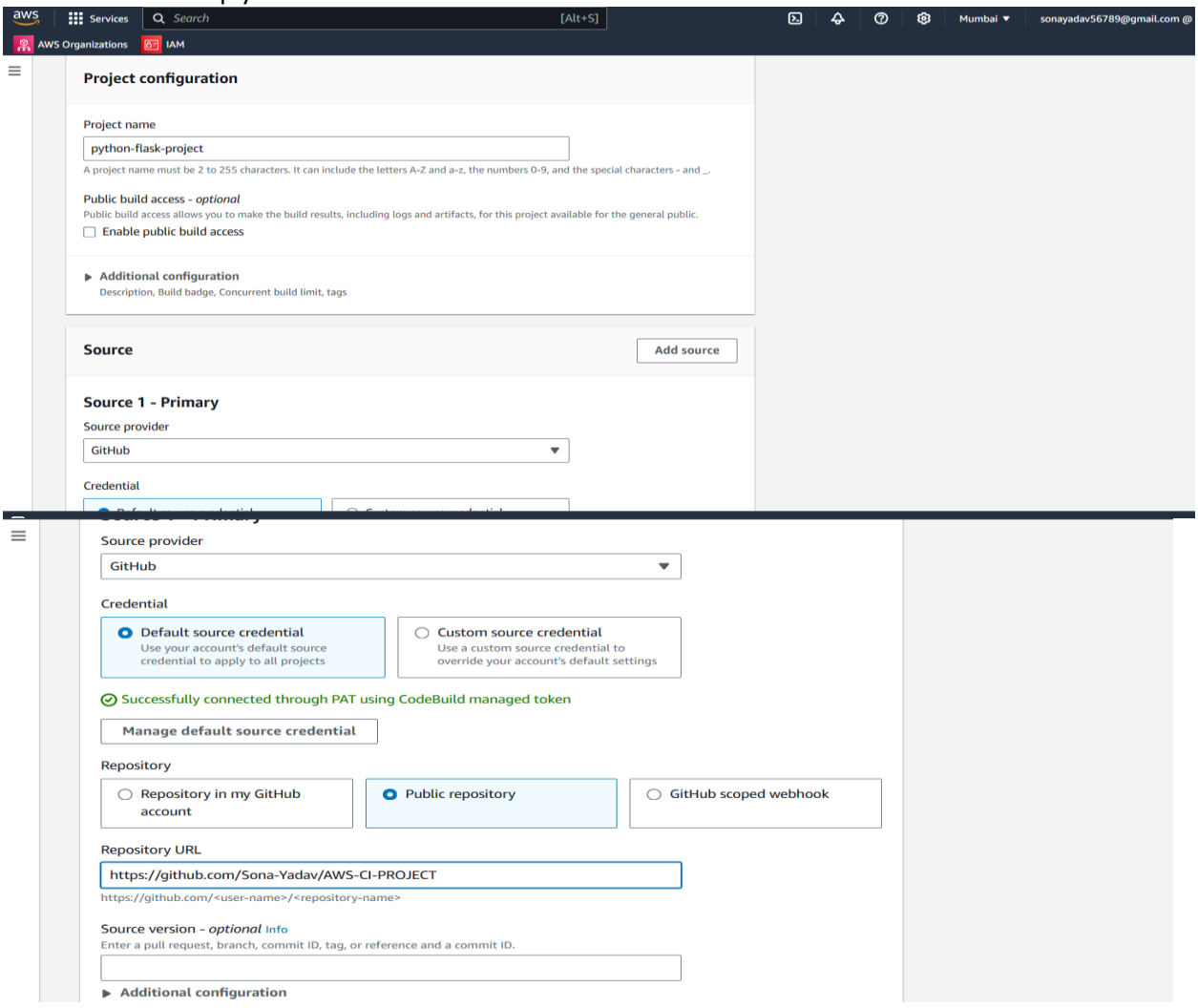
1. Use some files present in our GitHub repo. like app.py, appspec.yml and Dockerfilé etc.
   https://github.com/Sona-Yadav/AWS-CICD-PROJECT

2. Create a dockerfile:

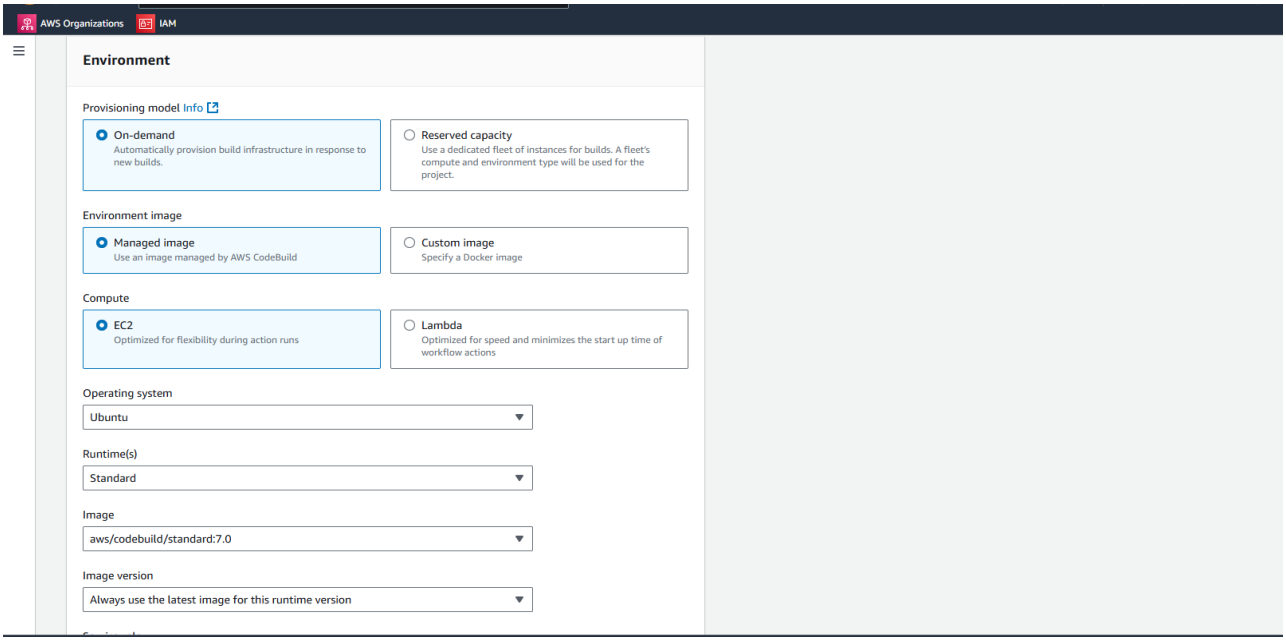
```
# Base image
FROM python:3.8

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file
COPY requirements.txt .

# Install the project dependencies
RUN pip install -r requirements.txt

# Copy the application code into the container
COPY . .

# Expose the port the Flask application will be listening on
EXPOSE 5000

# Set environment variables, if necessary
# ENV MY_ENV_VAR=value

# Run the Flask application
CMD ["python", "app.py"]
```



3. (i) Go to AWS CodeBuild and click on "Create Project."
(ii) Set the project name as python-flask-project.
(iii) For now, do not modify the setting for restricting the number of concurrent builds. If multiple developers push commits to a GitHub or AWS CodeCommit repository, and 100 developers trigger code changes simultaneously, you can choose whether to run all 100 pipelines at once or limit the number of concurrent builds. We will leave this setting as-is for now.
(iv) Set the source provider to GitHub.
(v) For the environment, when using Jenkins, you would need to create worker nodes and install an operating system on top of those nodes. However, with CodeBuild, AWS provides managed images, so you don't need to handle that setup yourself.

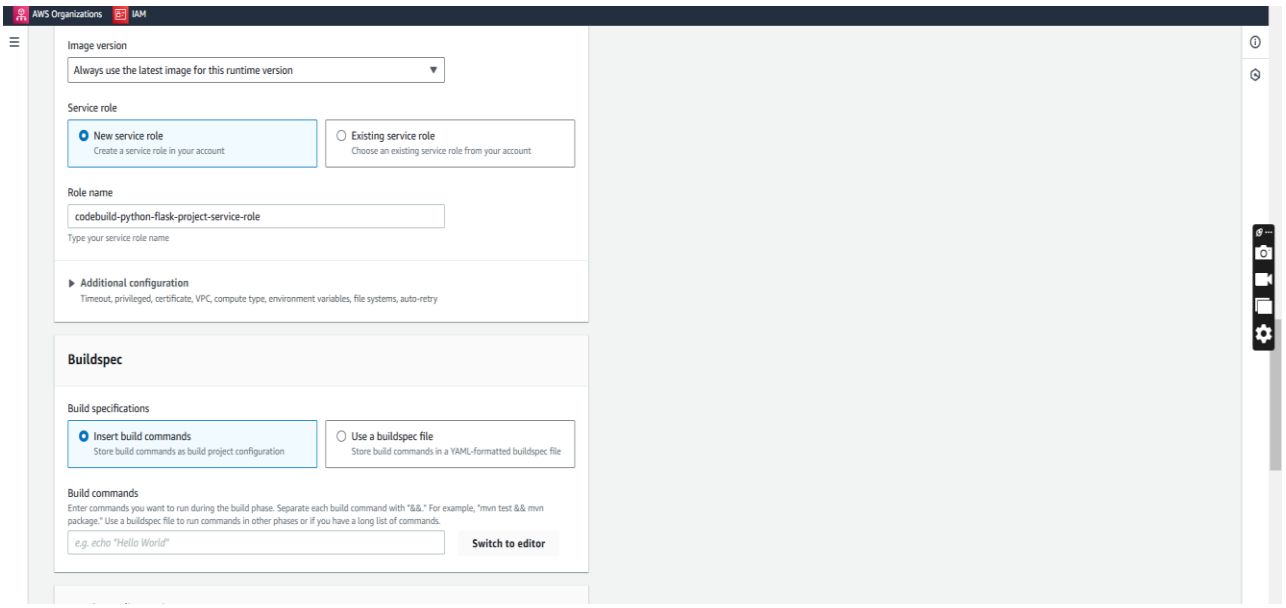


(vi) Env.



(vii) Created a new service role.

(viii) The **buildspec** file in AWS CodeBuild is similar to a Jenkinsfile in Jenkins. You can either use a buildspec file from your GitHub repository or create one manually by using the Insert build commands option provided in the CodeBuild interface.

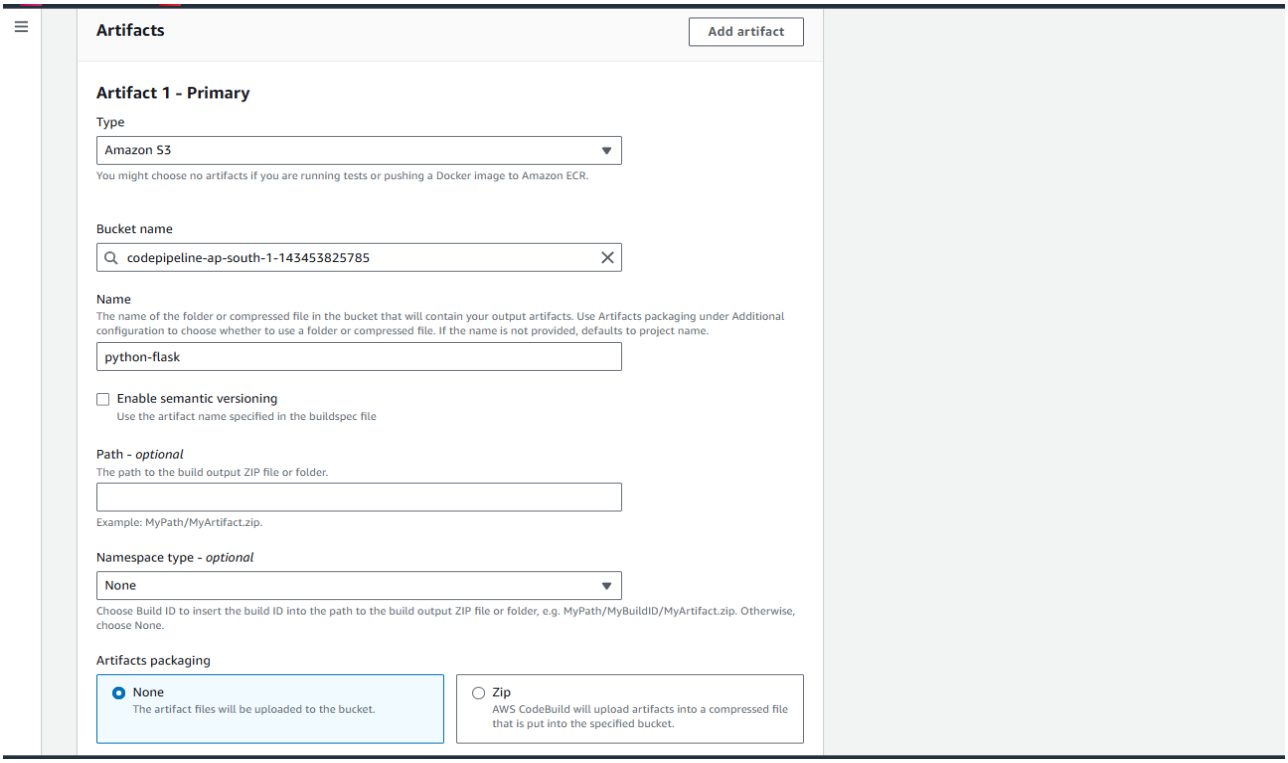


- Click on switch to editor (keep the image tag in Systems Manager to store credentials).
- Parameterstore use to provide the sensitive info like credentials in a secret location.
- CI code:

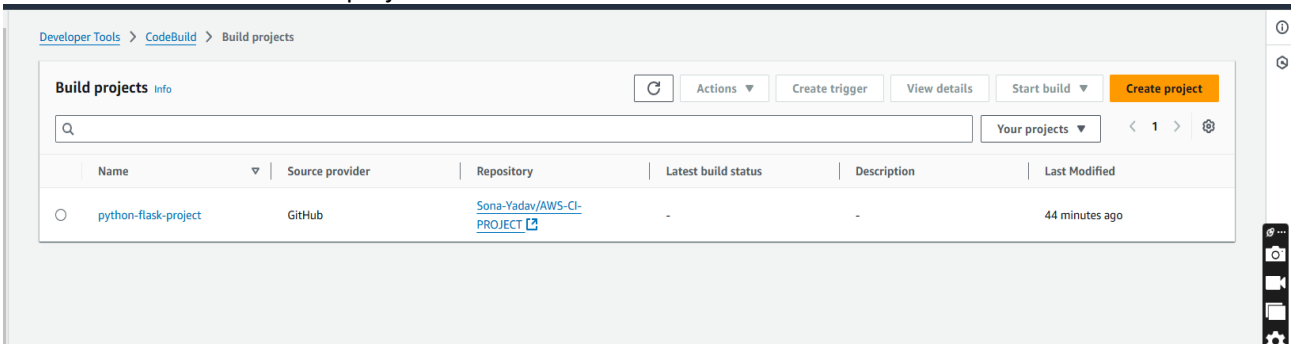
```
version: 0.2
phases:
  install:
    runtime-versions:
      python: 3.11

  pre_build:
    commands:
      - pip install -r requirements.txt

  build:
    commands:
      - echo "Building the Docker image"
      - docker build -t "<>"
      - docker push <>
```

```
post_build:
  commands:
    - echo "Build is successful"
```

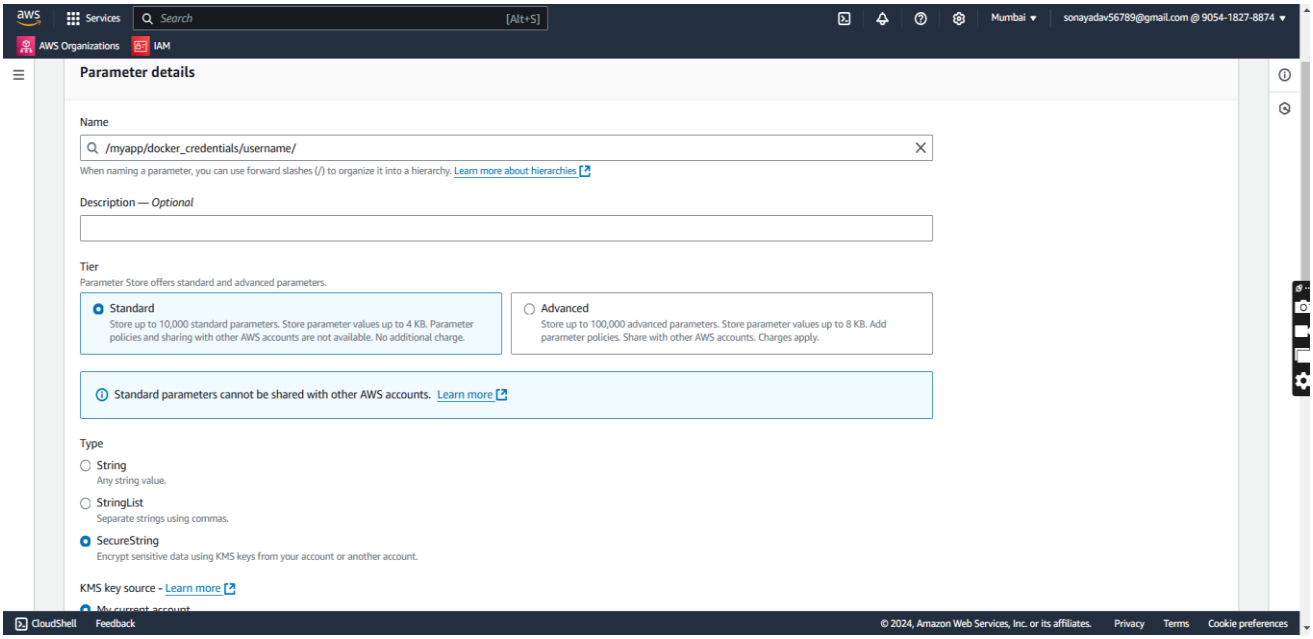
➔ Add S3 to the artifacts.



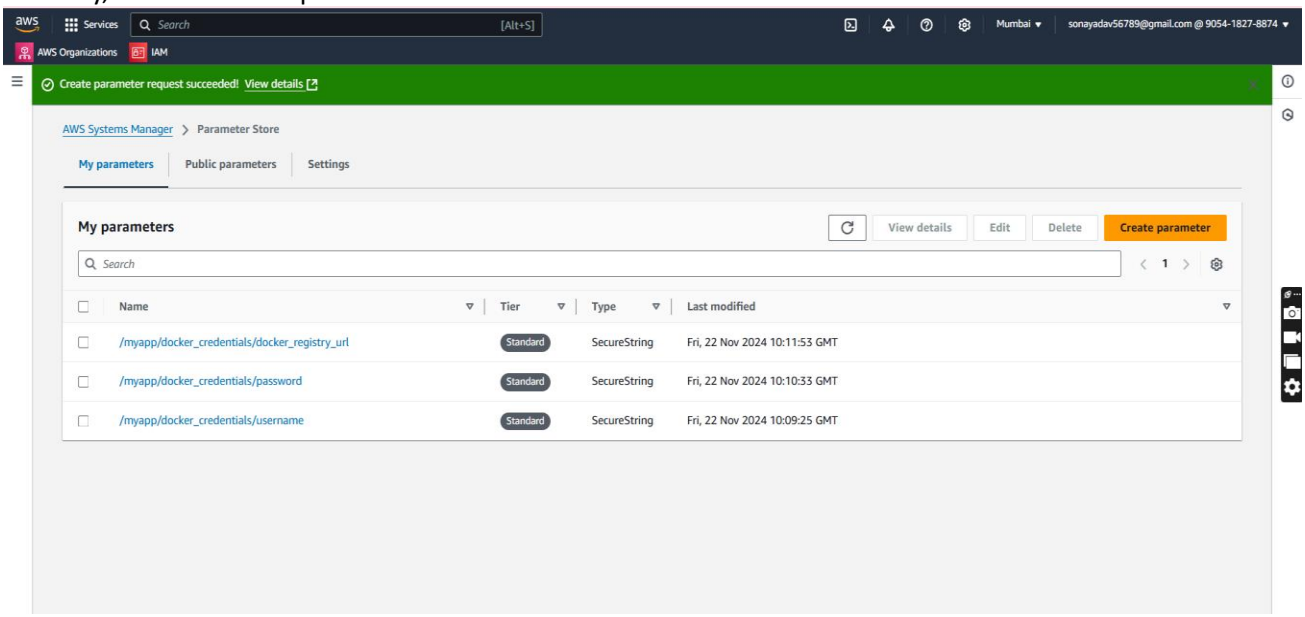
- Click on create build project.



4. Store the sensitive information:
Search for system manager → Find the option called parameter store → click on create parameter → value = docker username → click on create parameter.



Similarly, create another parameter store

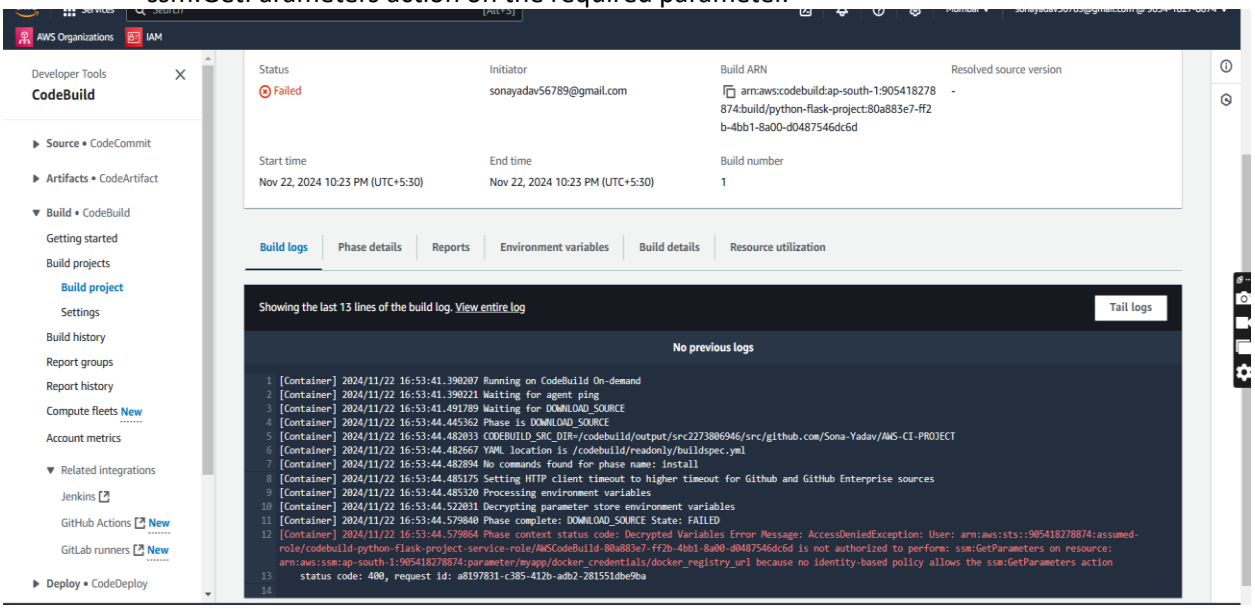


5. Search for CodeBuild:
Go to the edit build details/project details → edit buildspec file → update it

```
version: 0.2
env:
  parameter-store:
    DOCKER_REGISTRY_USERNAME: /myapp/docker_credentials/username
    DOCKER_REGISTRY_PASSWORD: /myapp/docker_credentials/password
    DOCKER_REGISTRY_URL: /myapp/docker_credentials/docker_registry_url

phases:
  install:
    runtime-versions:
      python: 3.11
    commands:
      - echo "Installing dependencies..."

  pre_build:
    commands:
      - echo "Installing Python dependencies..."
```

```
      - pip install -r requirements.txt
      - echo "Logging in to Docker registry..."
      - docker login -u "$DOCKER_REGISTRY_USERNAME" -p "$DOCKER_REGISTRY_PASSWORD"
"$DOCKER_REGISTRY_URL"

  build:
    commands:
      - echo "Building the Docker image..."
      - docker build -t "$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/python-flask-project:latest"
.

  post_build:
    commands:
      - echo "Pushing the Docker image to the registry..."
      - docker push "$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/python-flask-project:latest"
      - echo "Build and push completed successfully."

artifacts:
  files:
    - "**/*"        # Upload all files in the project directory
    - "!**/.git/**"   # Exclude Git metadata
```

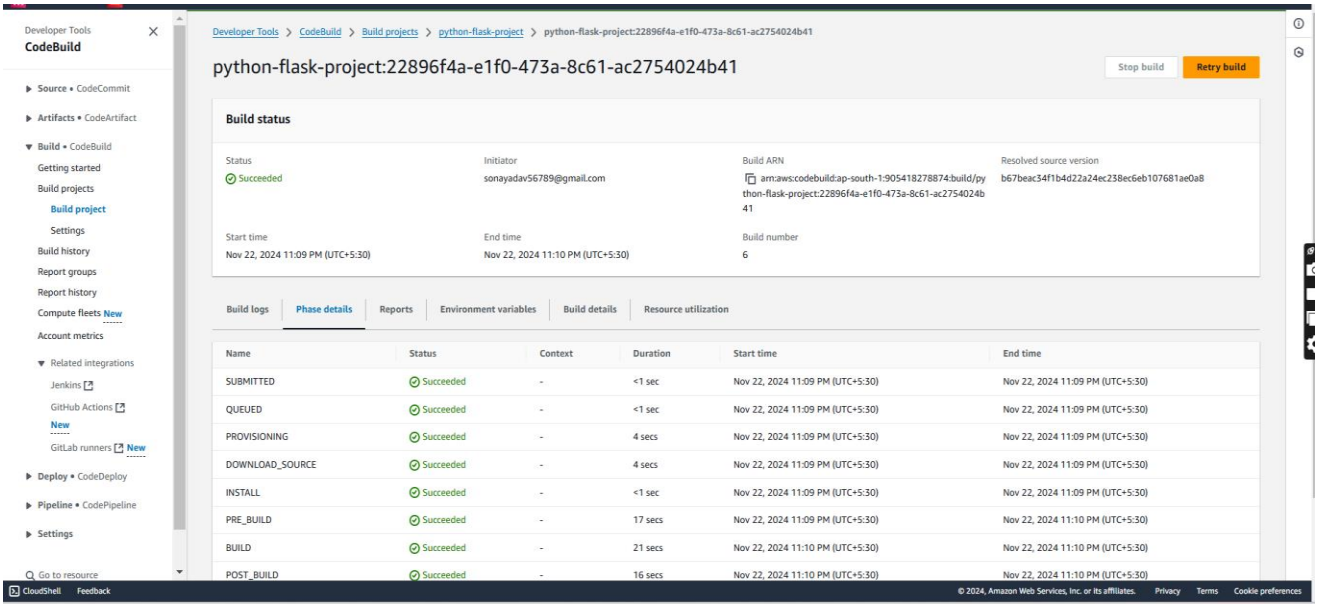
6. Click on start build.
   Errors:
   (i) This occurs becoz codebuild doesn't have access to Systems Manager. The CodeBuild project needs access to the docker_registry_url parameter stored in the SSM Parameter Store to proceed with the build process. However, the IAM role assigned to CodeBuild (codebuild-python-flask-project-service-role) does not have a policy granting it permission to perform the ssm:GetParameters action on the required parameter.

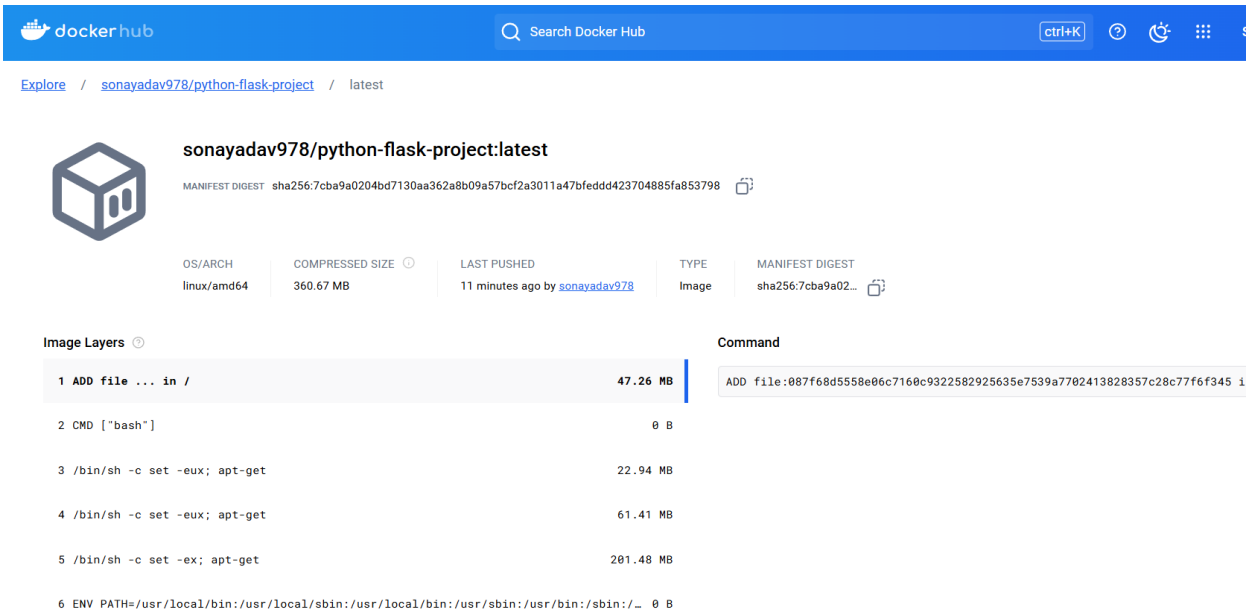


Solution: Go to IAM → Roles (check the roles what was we created) → Add permission → Attach Policy → search for SSM → Grant SSM full access → save.

   (ii) Attach S3 full permission access to this service role.

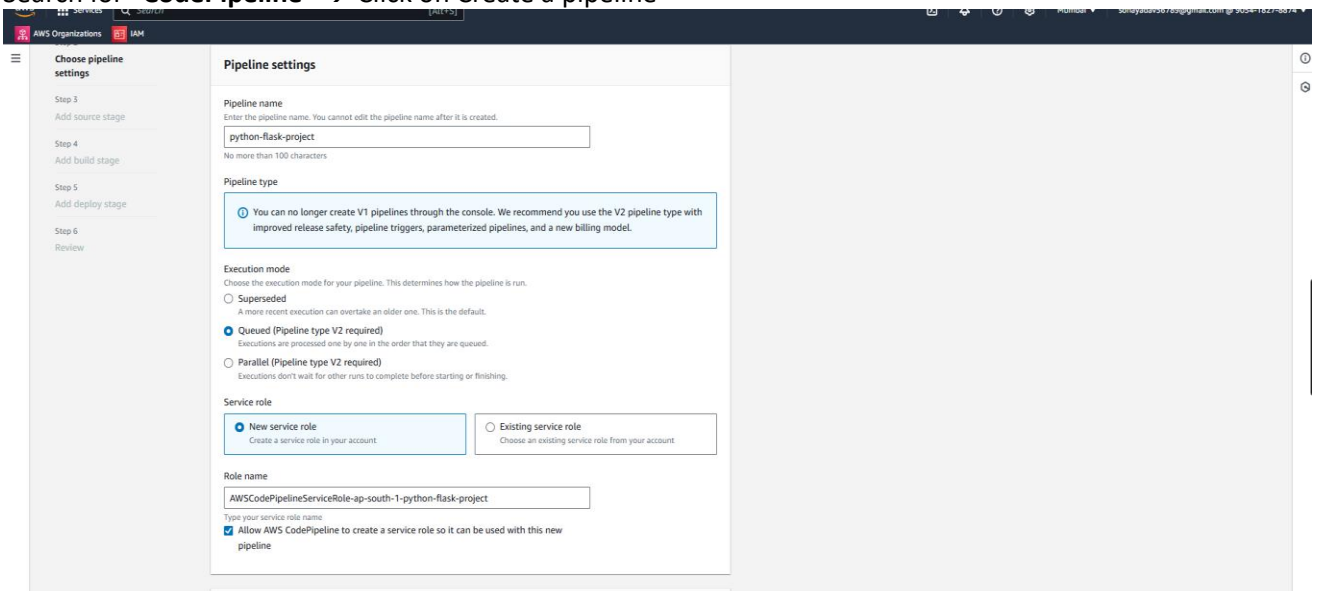
7. Great!!! Build is successful.





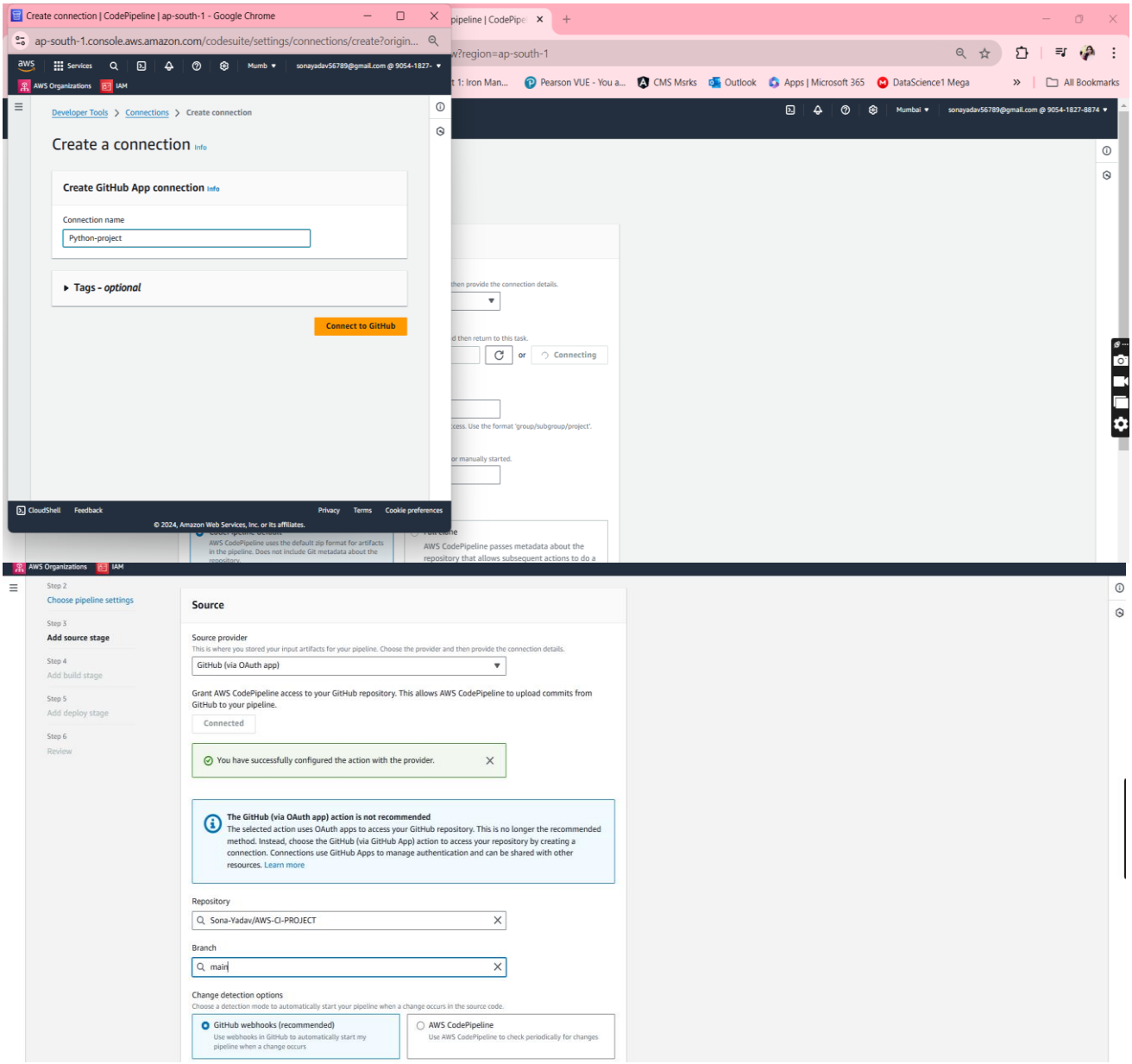
8. Now, let's integrate it with pipeline to invoke automatically when a code change is made to Github repo.

Search for "**CodePipeline**" → Click on Create a pipeline

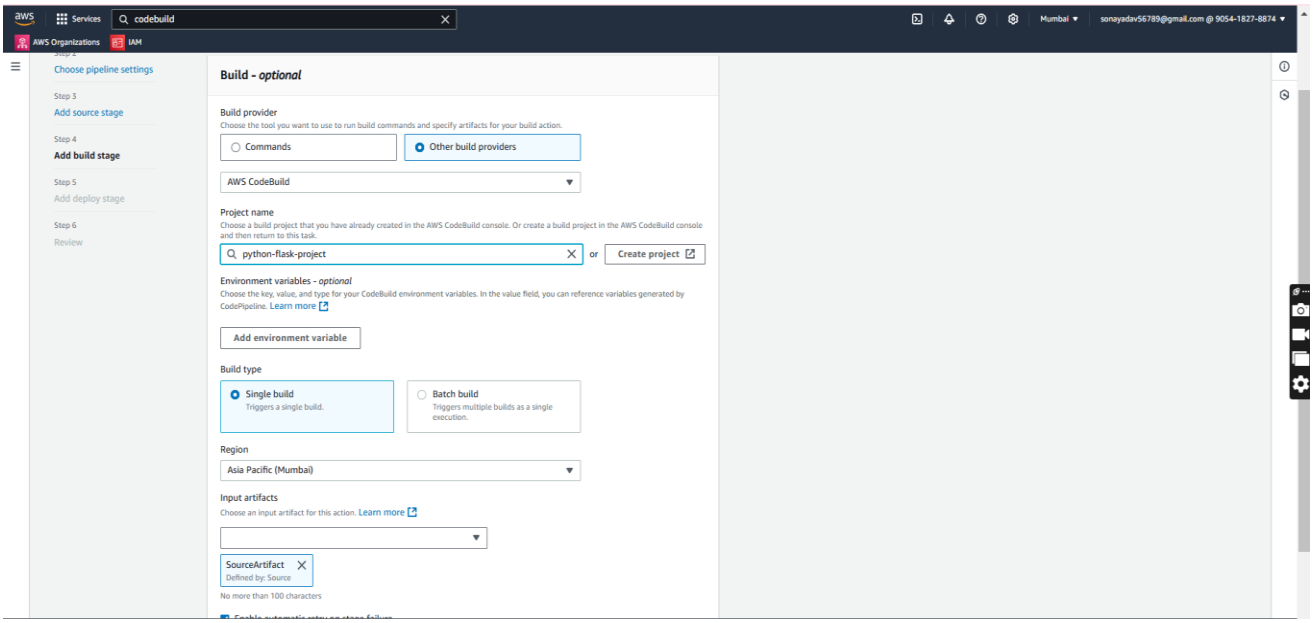


➔ Click on next.

➔ Choose sources as github → click on connect to github



➔ Click on Next.



➔ Click on Next.

➔ For deployment provider, this comes in the CD part so skip this stage.

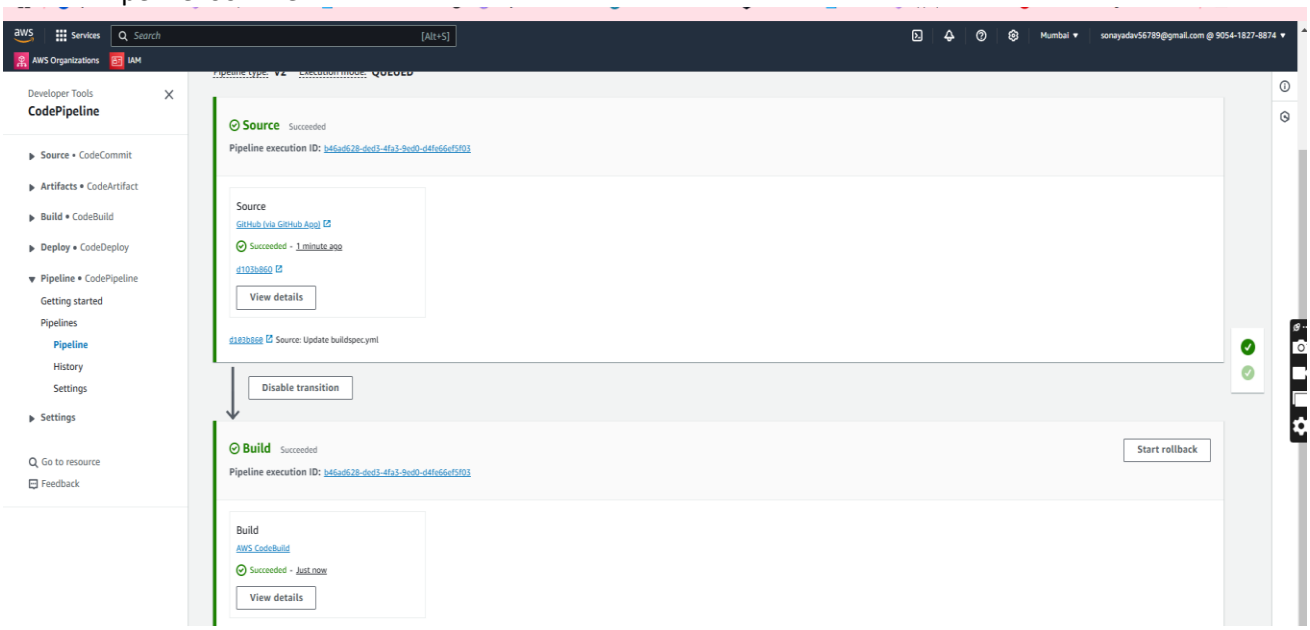
➔ Review and click on create pipeline.

Encounter issue: Action execution failed Error message Error calling startBuild: Invalid characters in project name. Only alphanumeric characters, dash, and underscore are supported (Service: AWSCodeBuild; Status Code: 400; Error Code: InvalidInputException; Request ID: 9dd43b06-b469-4ba9-853b-cce1eb491f4d; Proxy: null)

Solution: Again, create a new CodeBuild and CodePipeline with different project name instead of using hyphen, I used underscore. Now it is working.

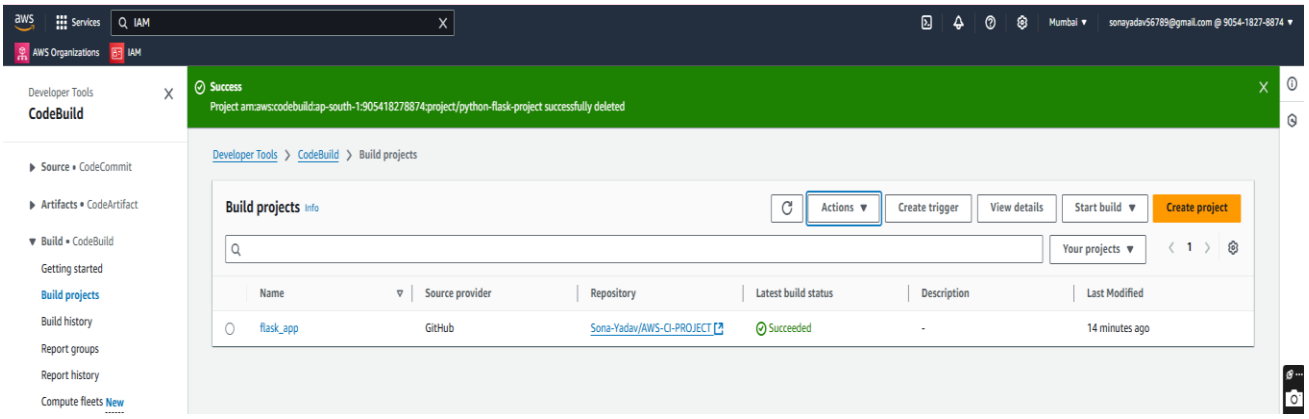
➔ Make any change in your code. It will trigger your pipeline automatically.

➔ Pipeline look like:

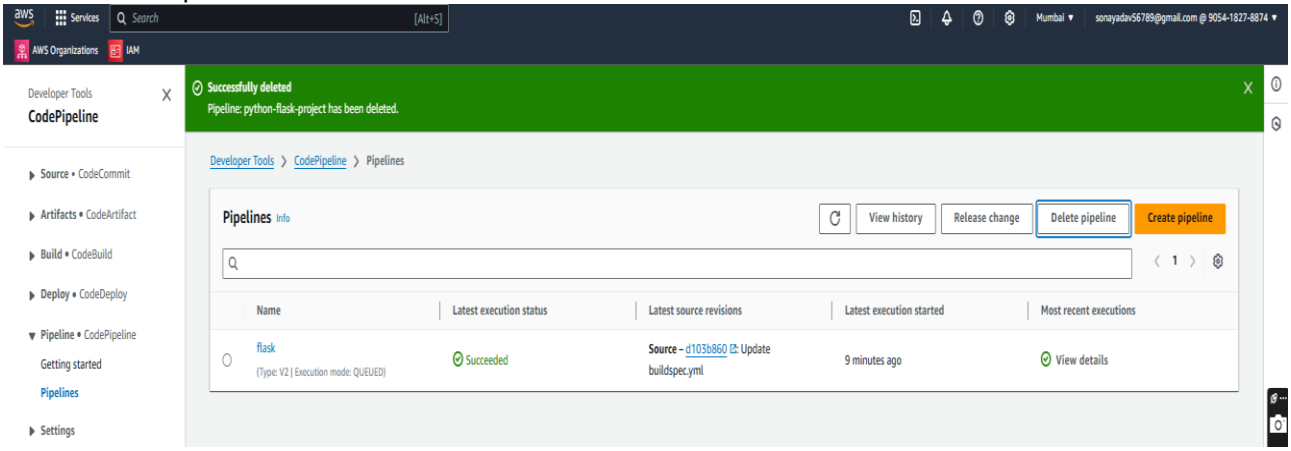


9 **Result:**

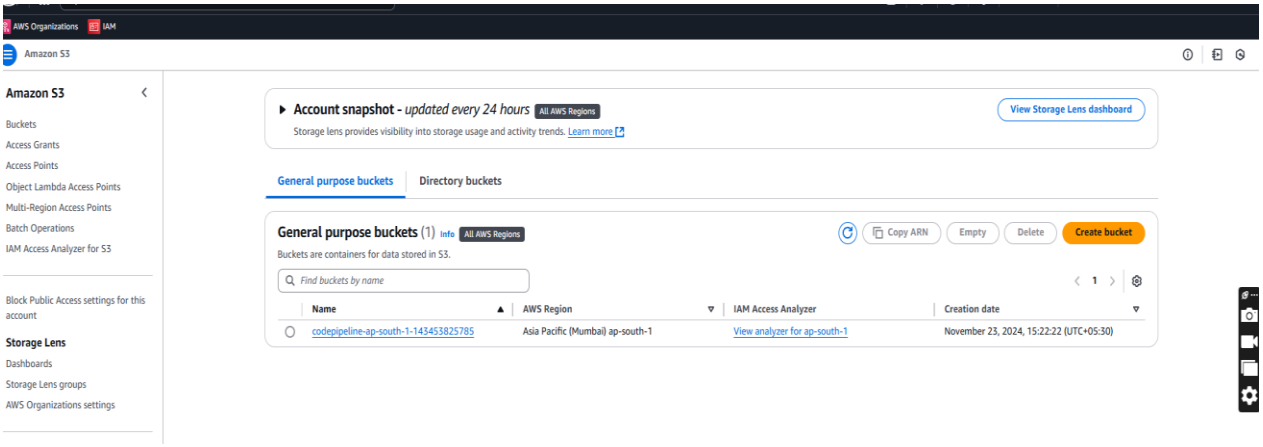
a. CodeBuild



b. CodePipeline



c. S3 Bucket



CD

10

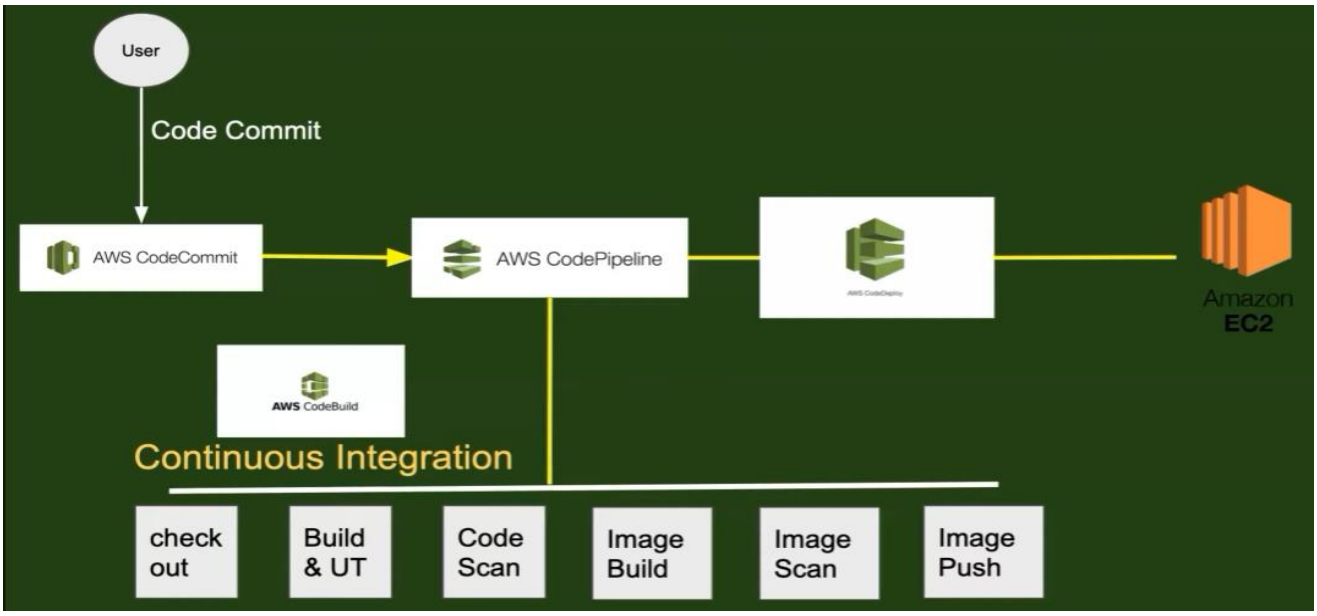
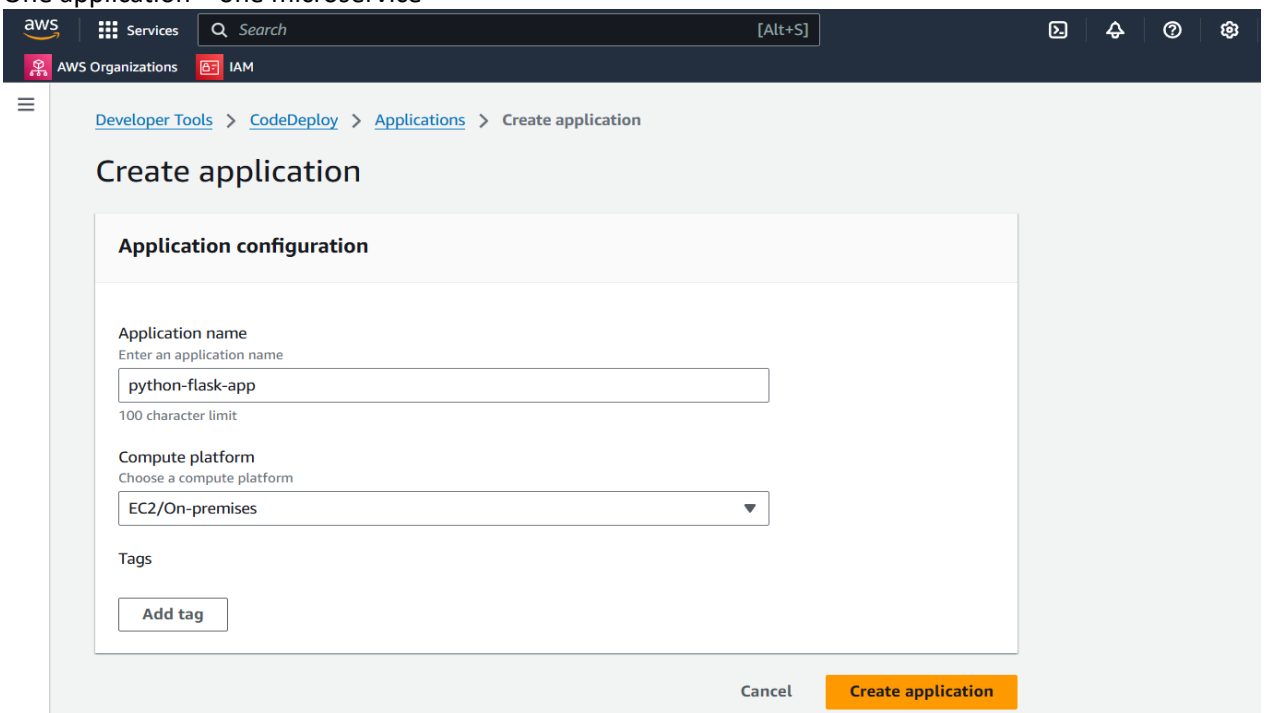


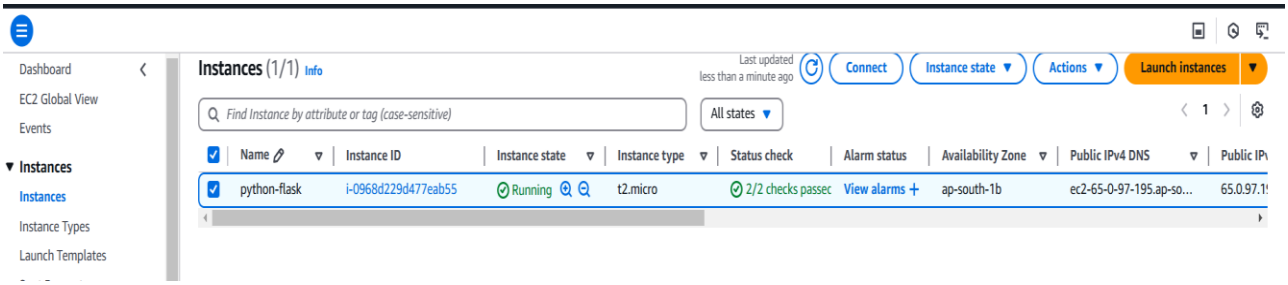
Diagram of CICD

11. Search for CodeDeploy→ click on getting started and create an application → Choose a compute platform as EC2 → click on create application.

One application = one microservice

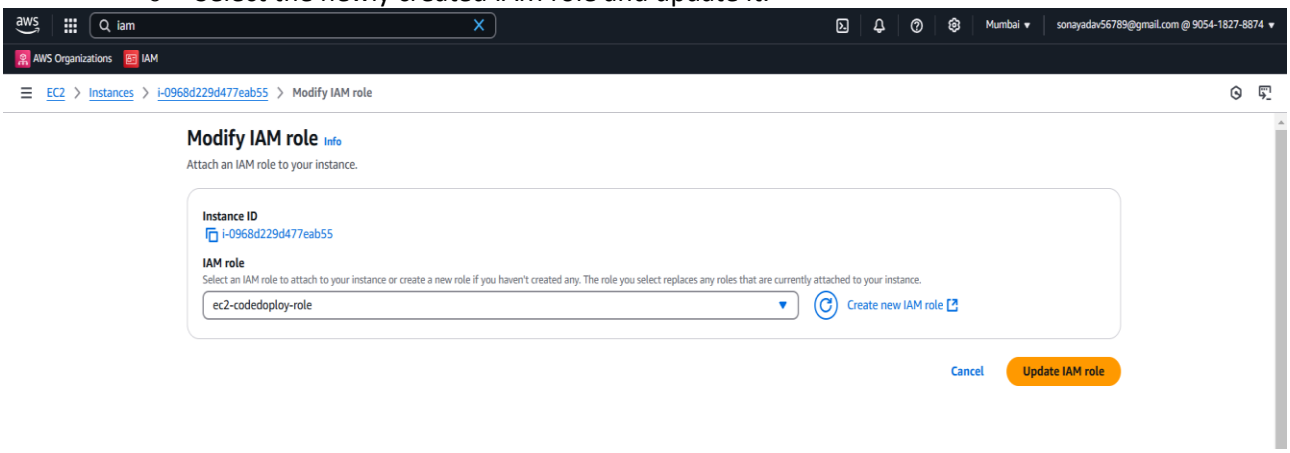


11 Go to the EC2 instance→ Launch an instance



12

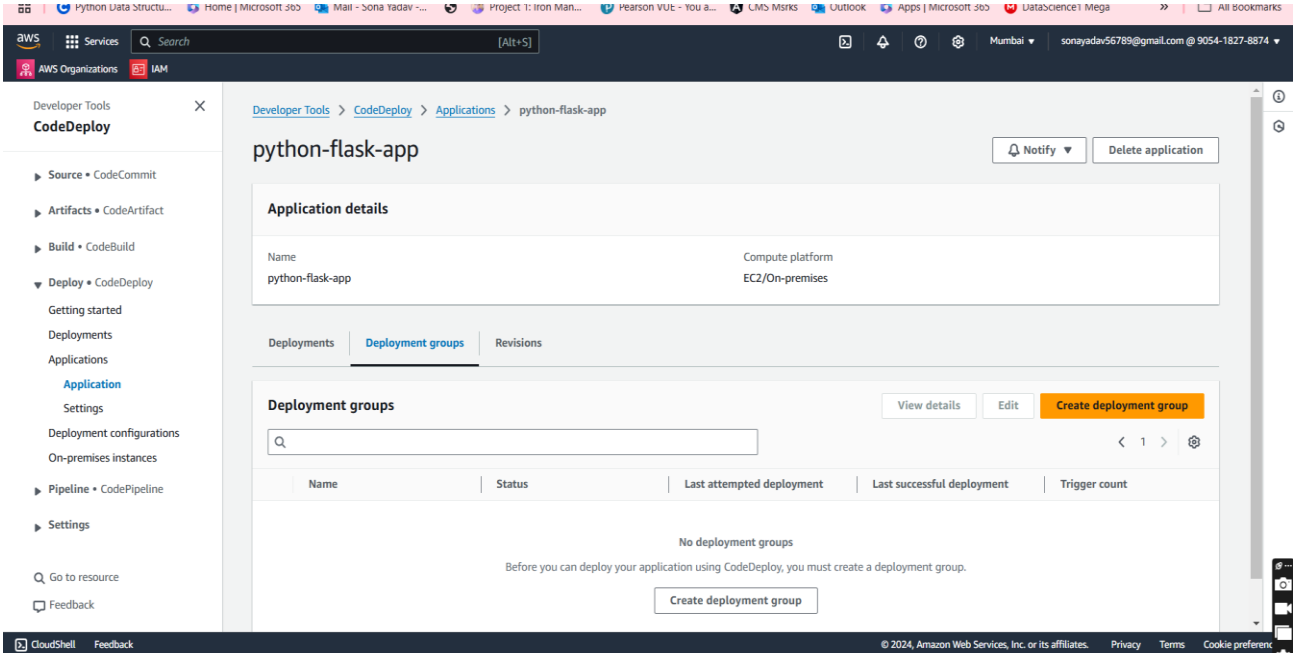
1. **CodeDeploy Deployment**:
   - CodeDeploy can be used to deploy applications on a single EC2 instance or multiple EC2 instances.
   - Tags in EC2 instances play a critical role in identifying and managing the deployment targets.
2. **Installing the CodeDeploy Agent**:
   - Log in to the EC2 instance.
   - Run the script available in the GitHub repository to install the CodeDeploy agent: Agent Installation Script.
   - Additionally, install Docker using the following command:
   - sudo apt install docker.io -y
3. **Assigning IAM Role to EC2**:
   - Search for IAM in the AWS Management Console.
   - Go to Roles and create a new role:
     - Trusted Entity Type: AWS Service
     - Use Case: EC2
     - Click Next.
     - Attach the AWSCodeDeployRole policy along with full EC2 and CodeDeploy access.
     - Click Next and then Create Role.
   - The role now has the necessary permissions for EC2 and CodeDeploy.
4. **Assign Role to the EC2 Instance**:
   - Go to EC2 in the AWS Management Console.
   - Select the instance you want to assign the role to.
   - Click Actions → Security → Modify IAM Role.
   - Select the newly created IAM role and update it.

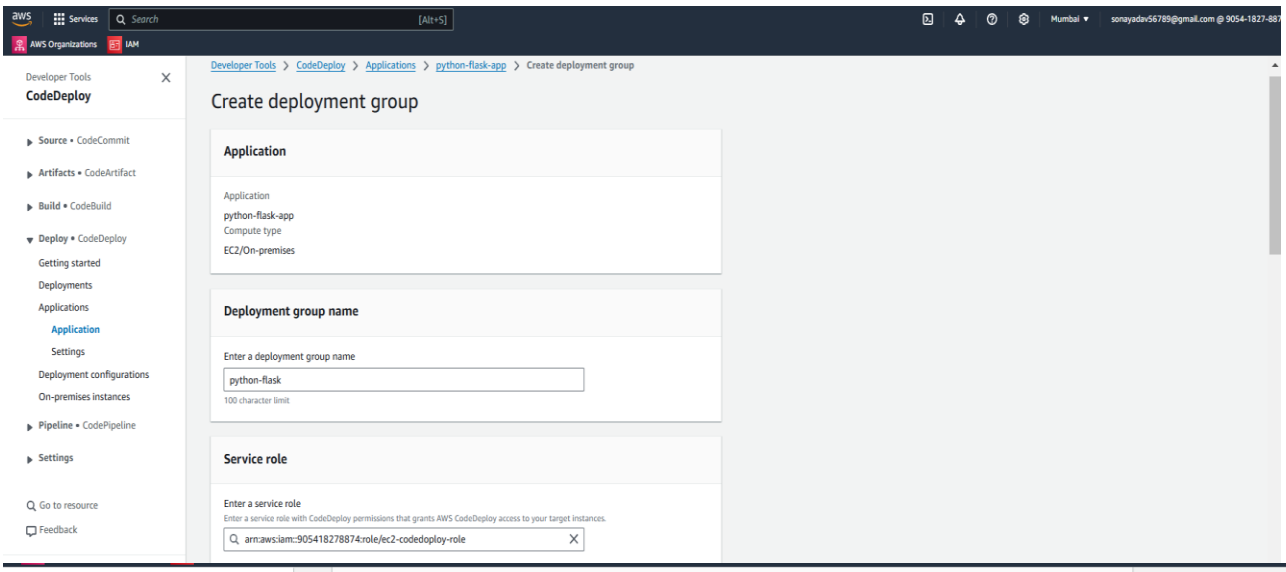


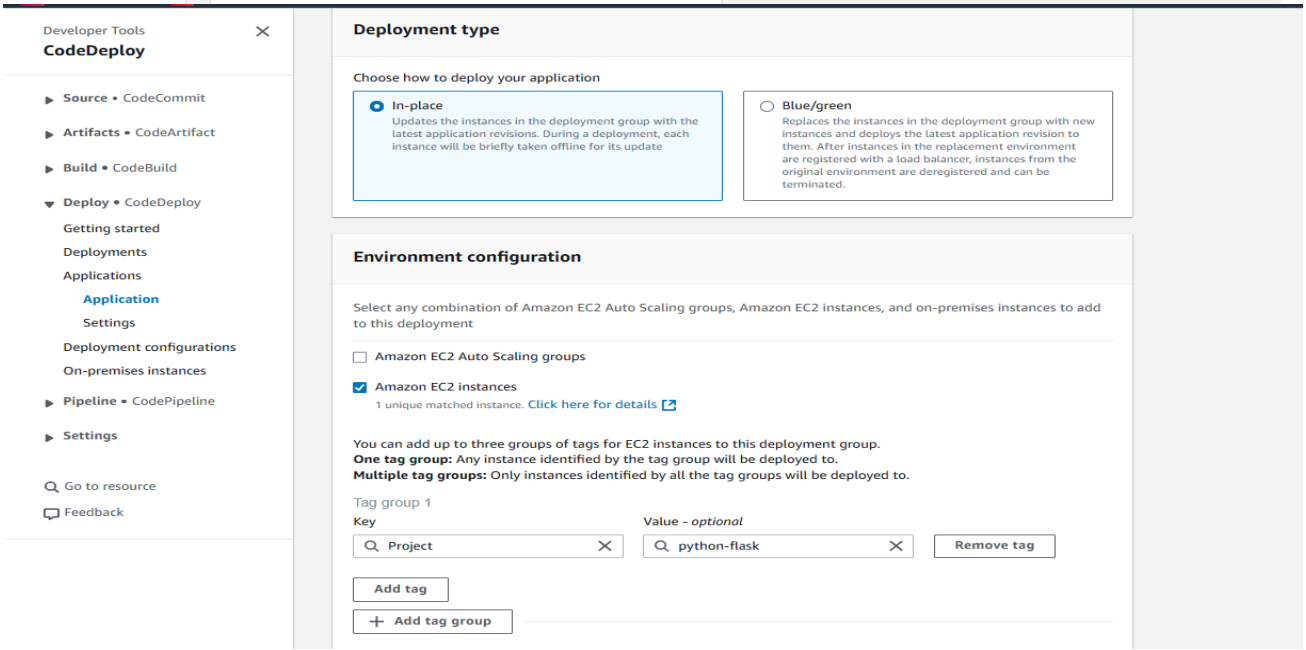
5. **Restart the CodeDeploy Agent**:
   - sudo service codedeploy-agent restart

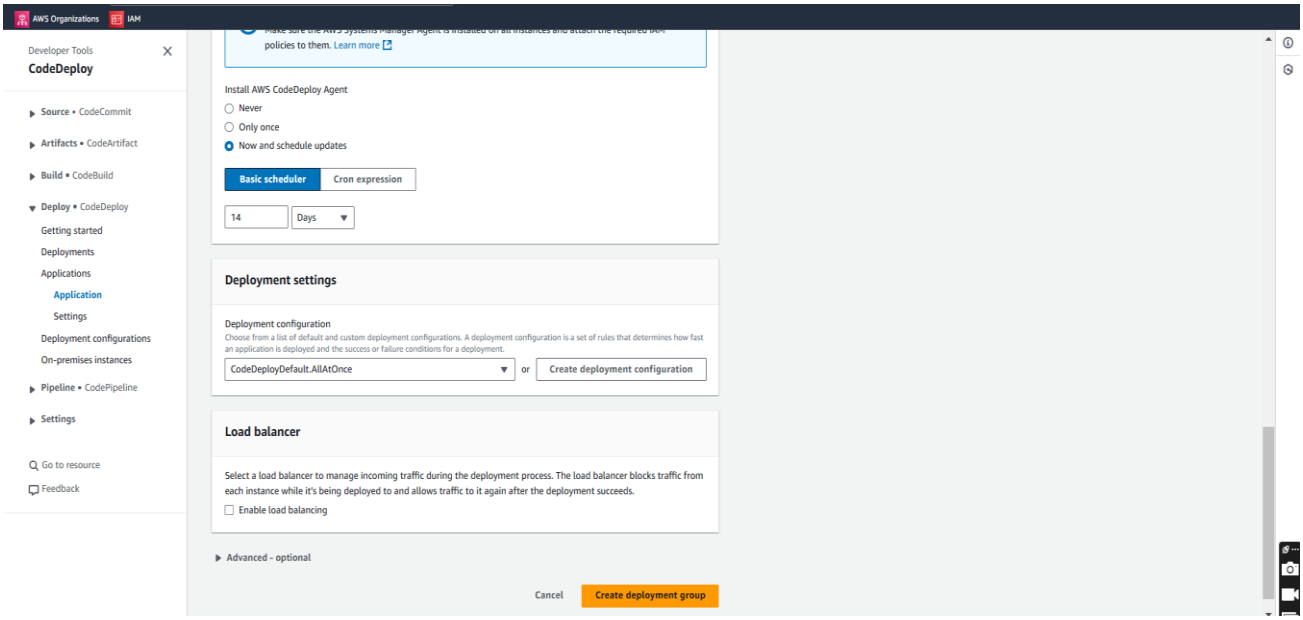
13 Configure the codedeploy:

Go to CodeDeploy application section → click on create deployment group and provide the ec2 instance.

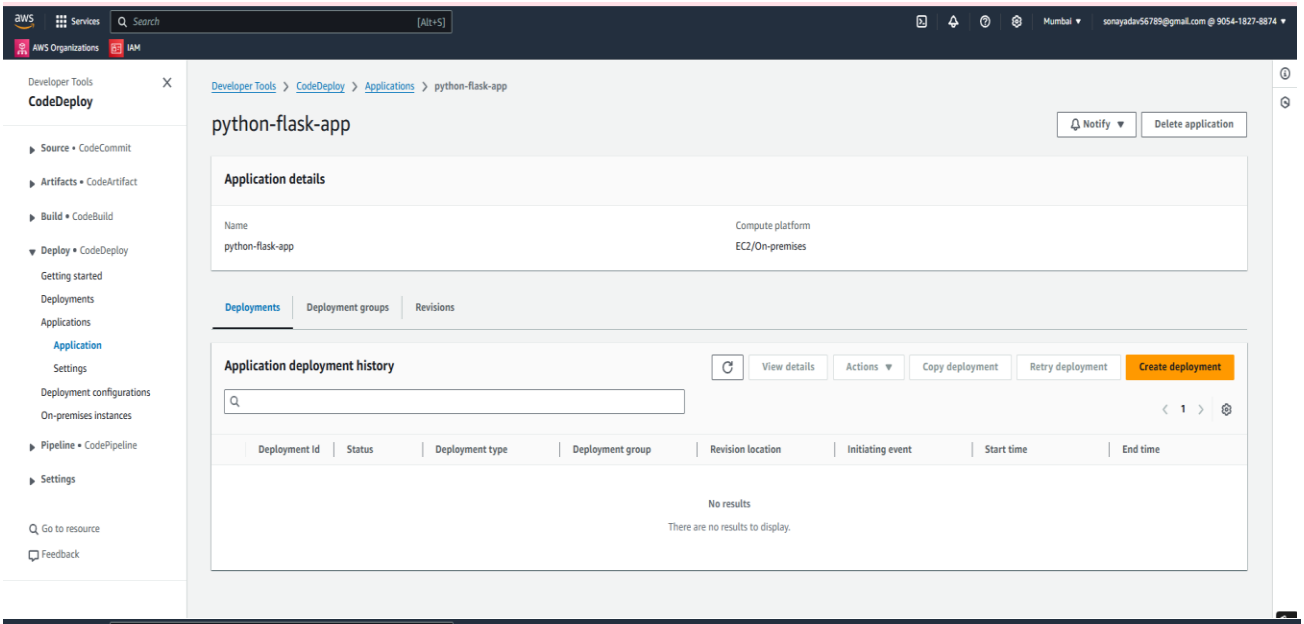

Developer Tools
CodeDeploy
Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Getting started
Deployments
Applications
Application
Settings
Deployment configurations
On-premises instances
Pipeline • CodePipeline
Settings
Go to resource
Feedback
Developer Tools > CodeDeploy > Applications > python-flask-app
python-flask-app
Notify
Delete application
Application details
Name
python-flask-app
Compute platform
EC2/On-premises
Deployments
Deployment groups
Revisions
Deployment groups
View details
Edit
Create deployment group
Name
Status
Last attempted deployment
Last successful deployment
Trigger count
No deployment groups
Before you can deploy your application using CodeDeploy, you must create a deployment group.
Create deployment group
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms

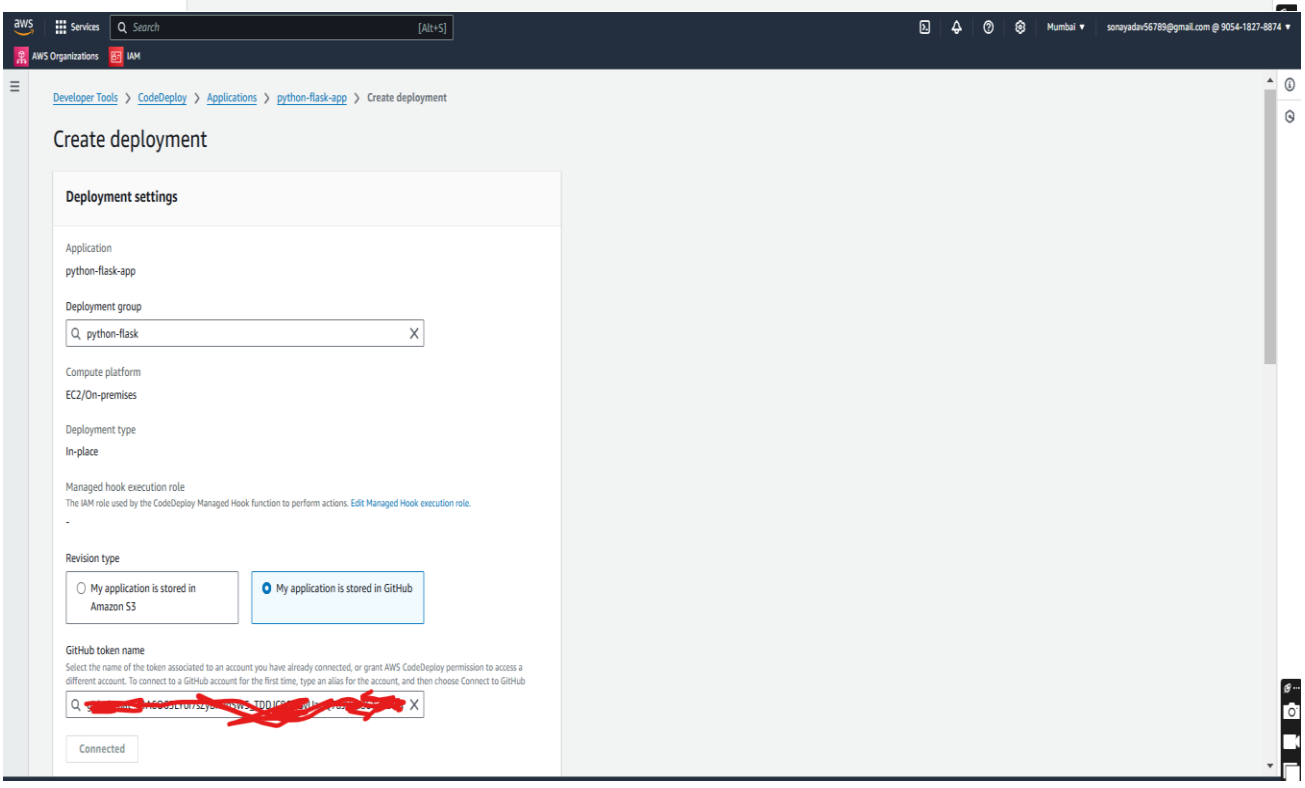

Developer Tools > CodeDeploy > Applications > python-flask-app > Create deployment group
Create deployment group
Application
Application
python-flask-app
Compute type
EC2/On-premises
Deployment group name
Enter a deployment group name
python-flask
100 character limit
Service role
Enter a service role
arn:aws:iam::905418278874:role/ec2-codedoploy-role

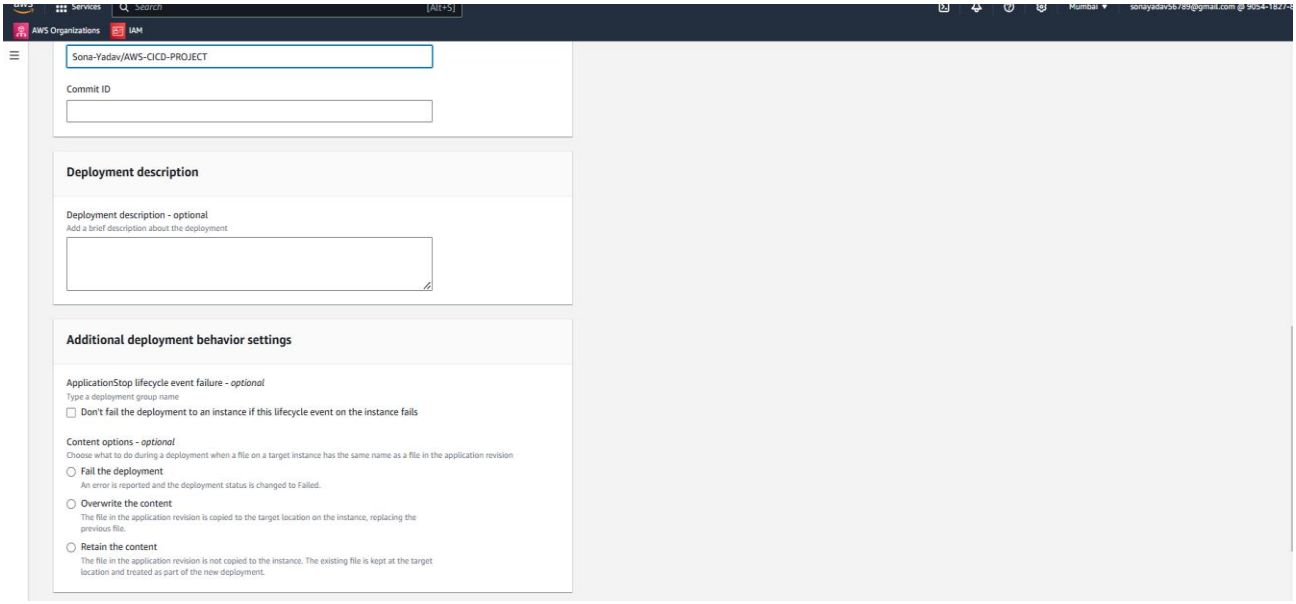

Developer Tools
CodeDeploy
Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Getting started
Deployments
Applications
Application
Settings
Deployment configurations
On-premises instances
Pipeline • CodePipeline
Settings
Go to resource
Feedback
Deployment type
Choose how to deploy your application
In-place
Updates the instances in the deployment group with the latest application revisions. During a deployment, each instance will be briefly taken offline for its update
Blue/green
Replaces the instances in the deployment group with new instances and deploys the latest application revision to them. After instances in the replacement environment are registered with a load balancer, instances from the original environment are deregistered and can be terminated.
Environment configuration
Select any combination of Amazon EC2 Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add to this deployment
Amazon EC2 Auto Scaling groups
Amazon EC2 instances
1 unique matched instance. Click here for details
You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.
Tag group 1
Key
Project
Value - optional
python-flask
Remove tag
Add tag
Add tag group



14 Deploy our application

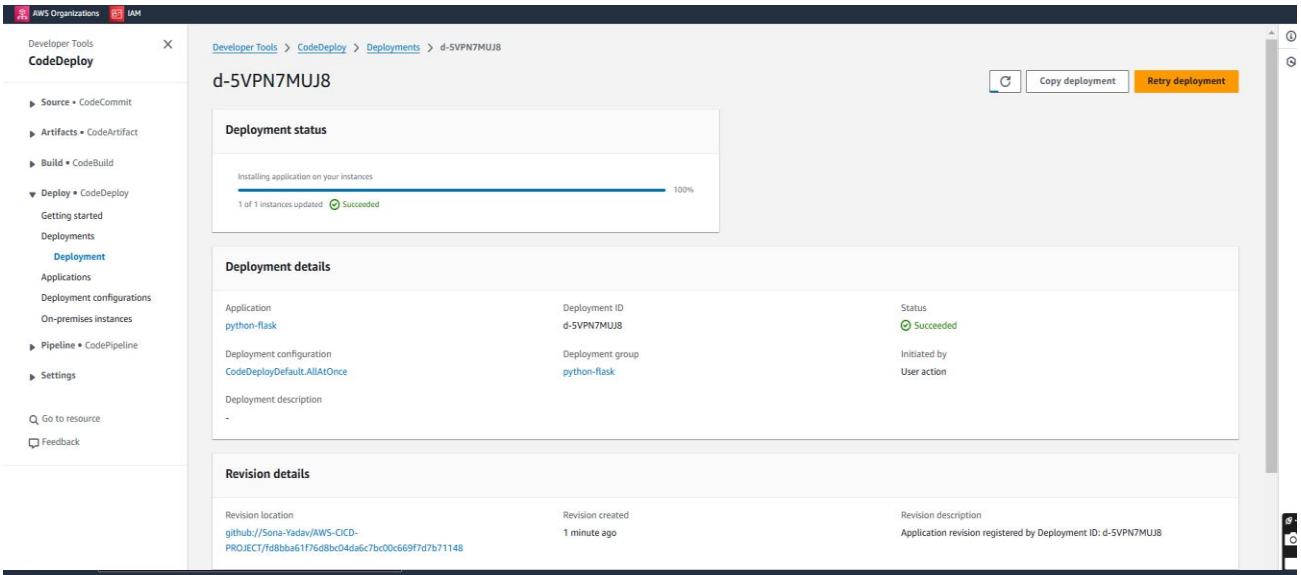


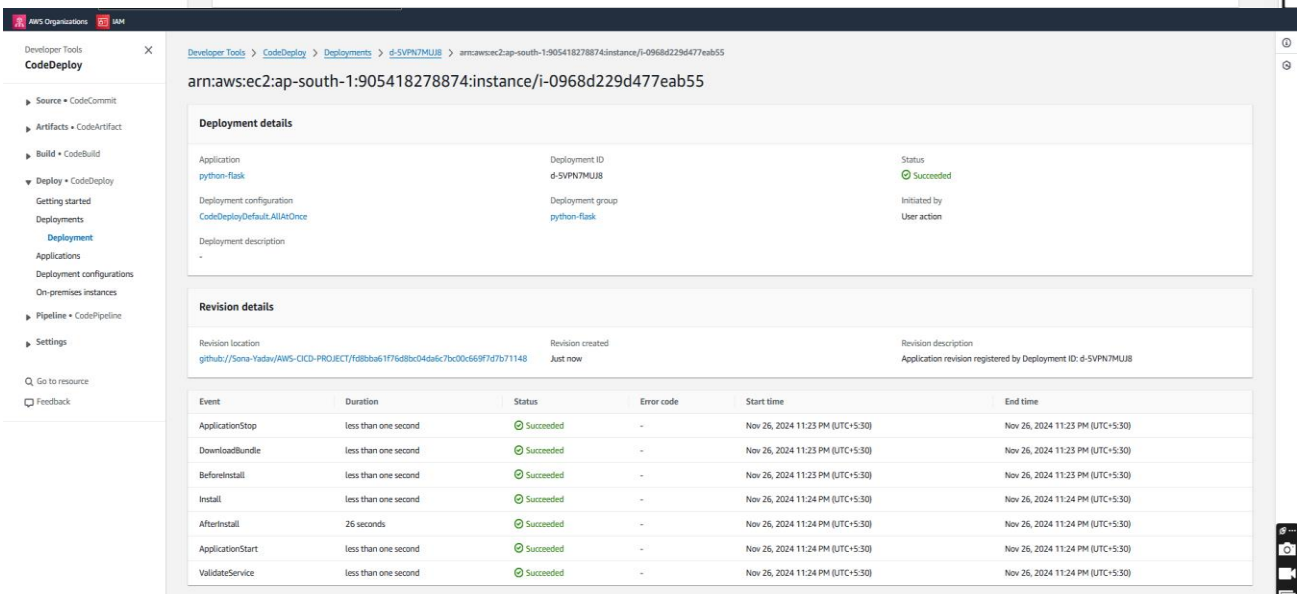




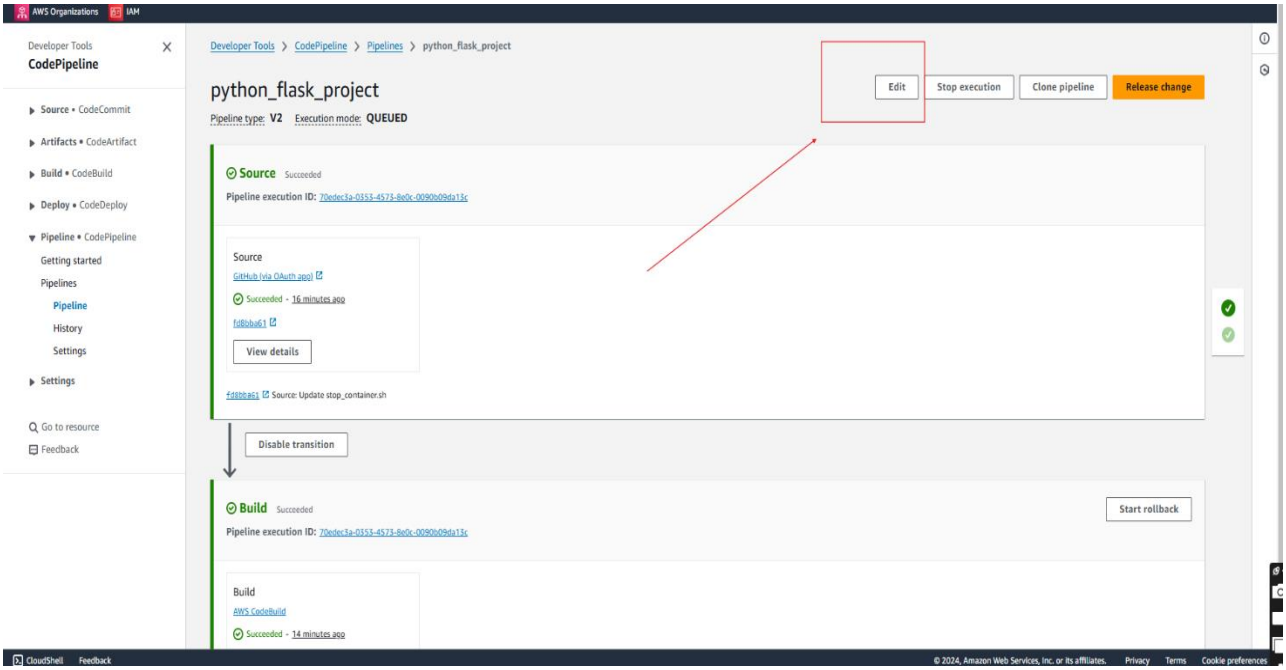
- Connect to Github
- Provide the last commit id.
- Click on create deployment.

15 Deployment is succeeded.

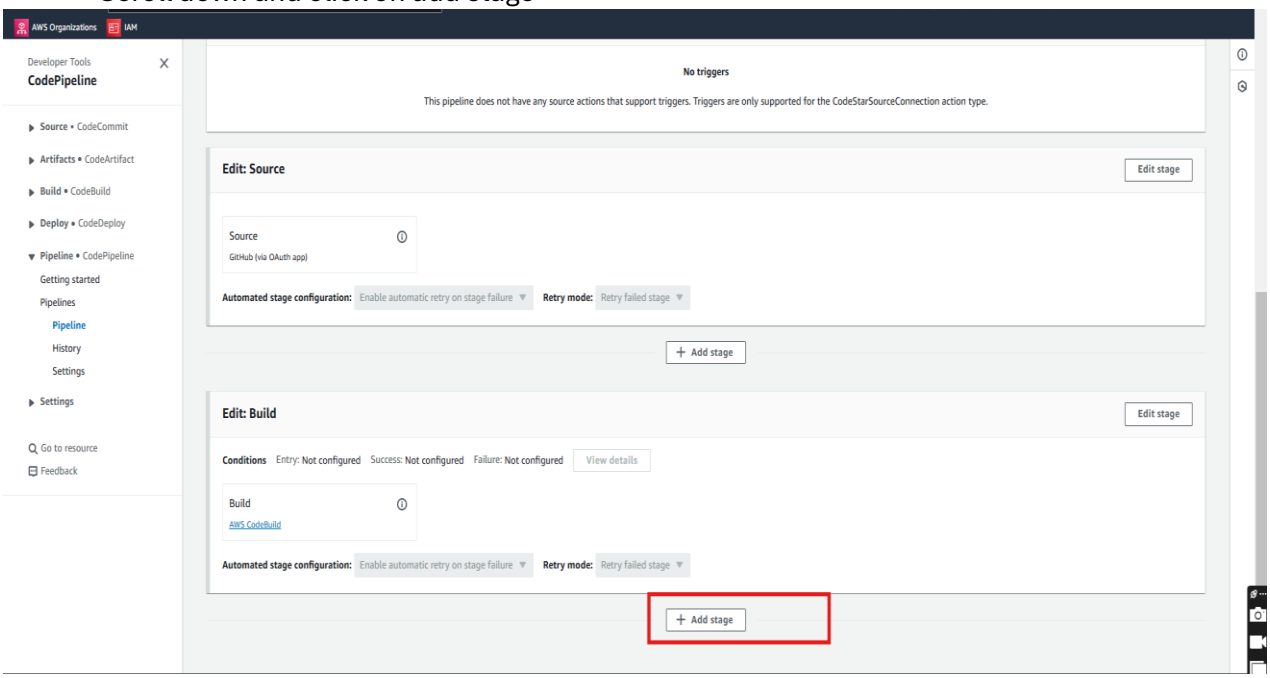




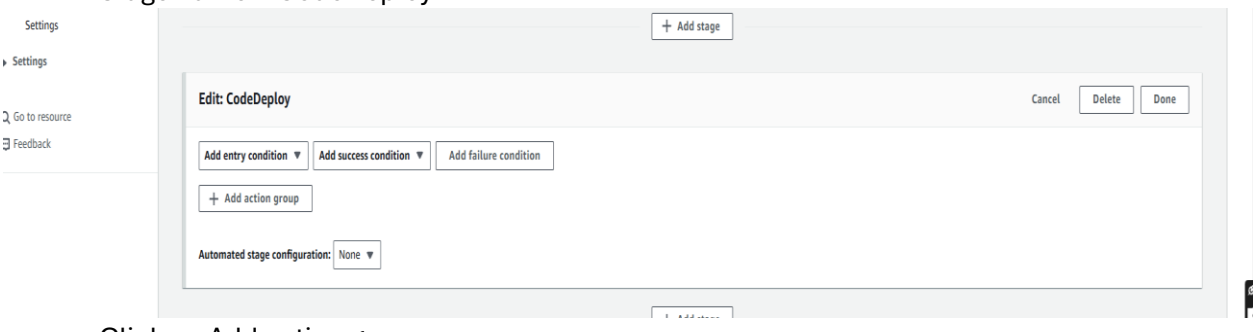
16 Go to the CodePipeline → add a new stage → Click on edit



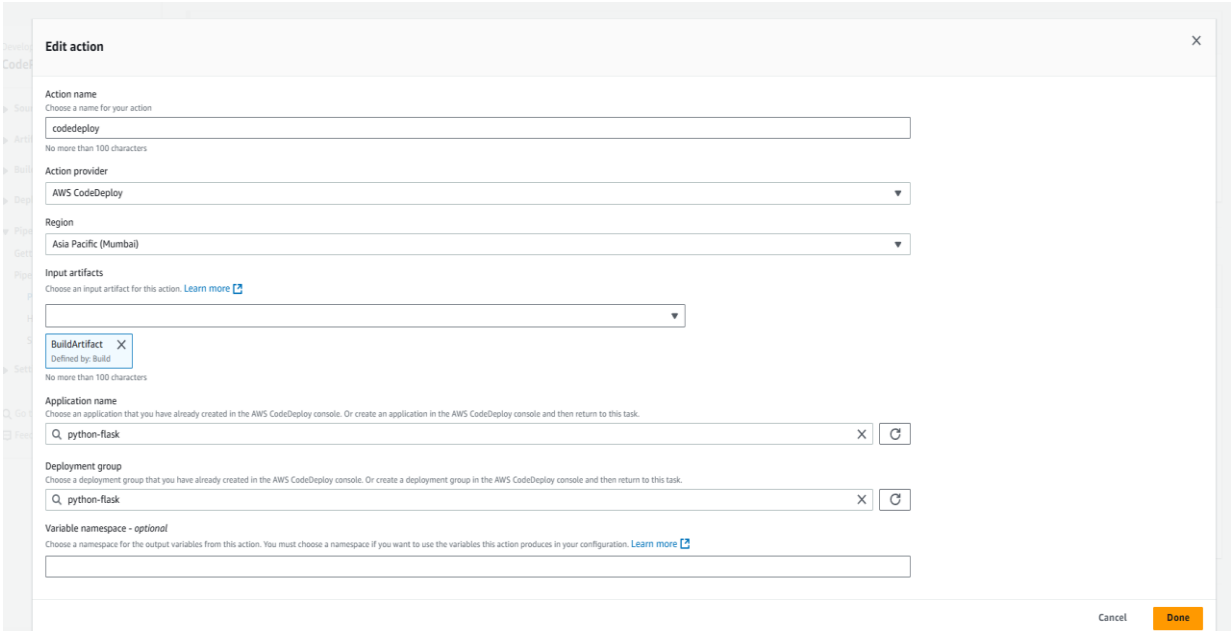
- Scroll down and click on add stage



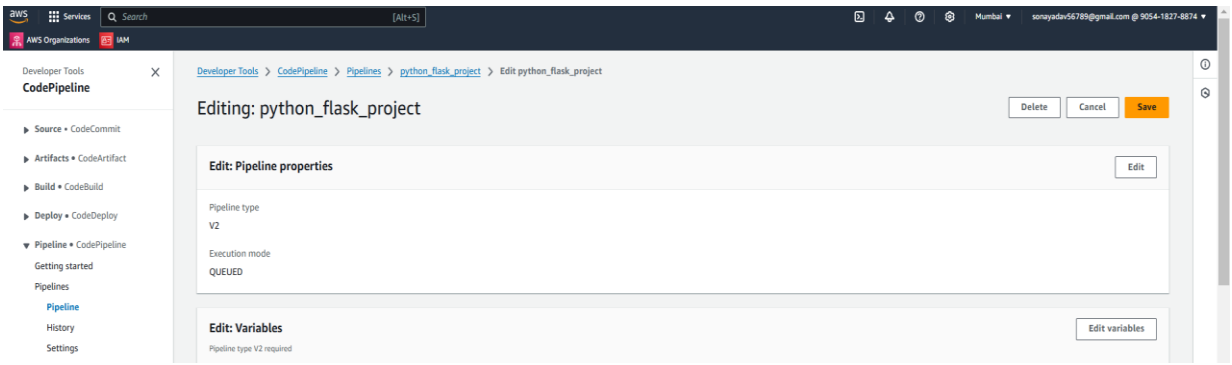
- Stage name = CodeDeploy



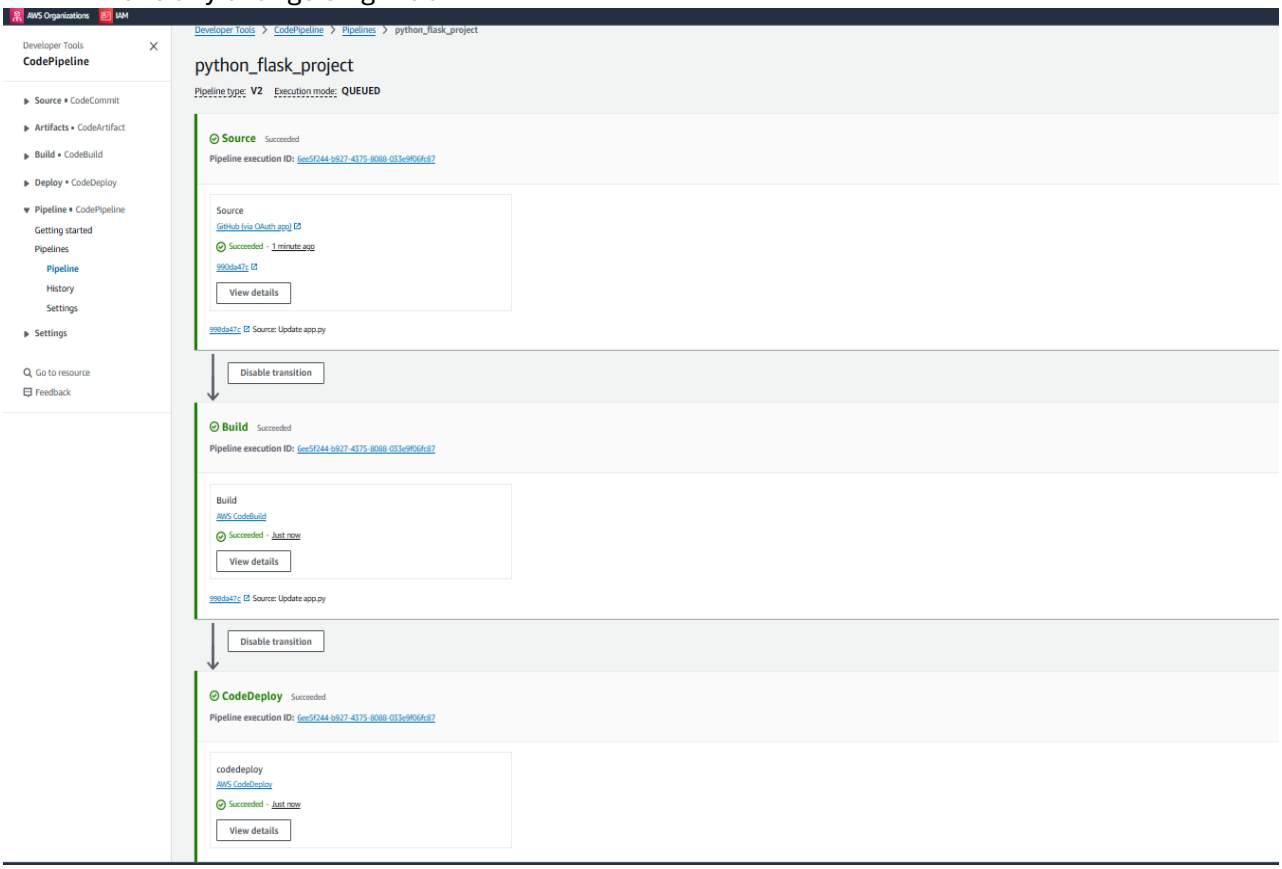
- Click on Add action group



- Click on done



- Click on the save button
- Make any change on github.



Hurray!!!!!! Completed.....