

# Project 1

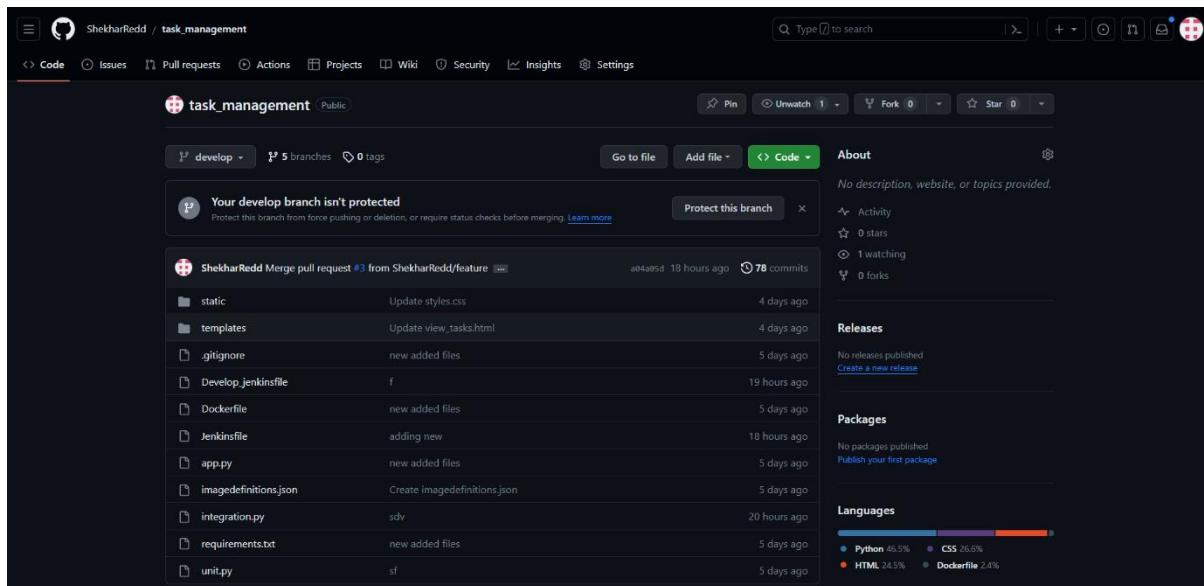
## Continuous Integration and Continuous Deployment (CI/CD) Pipeline

### Sub Task 1:

#### Set up a version control system using Git.

Create a New Repository on GitHub:

- Go to GitHub.
- Log in or create a new account.
- Click the "New" in the top left corner to create a New repository.



Pushing code to a Git repository involves a few steps using command line.

- **Navigate to Your Project Directory:**  
cd /path/to/your/project
- **Initialize a Git Repository:**  
git init
- **Add Your Files:**  
git add .
- **Commit Your Changes:**  
git commit -m "Initial commit"
- **Link Your Local Repository to GitHub:**  
git remote add origin your\_repository\_url.git
- **Push Your Changes to GitHub:**  
git push -u origin master

## Establish branching strategies (e.g., feature branches, release branches).

- Branching strategies in Git refer to the ways in which developers organize and manage branches in a Git repository.

### Feature branches:

- A feature branch is a copy of the main codebase where an individual or team of software developers can work on a new feature until it is complete.
- Create a new feature branch from the main branch
- `git checkout -b feature`

Now, you're on the new feature branch. Start making changes to your code, add new files, or implement the feature.

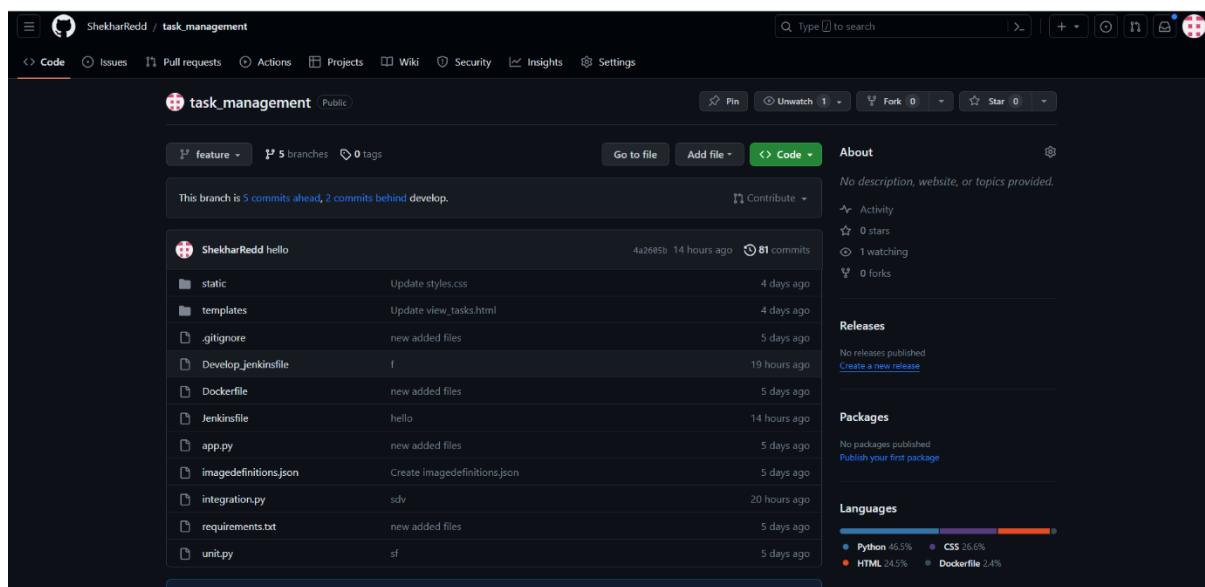
- **Once you've made changes, stage and commit them to the feature branch.**

```
git add .
```

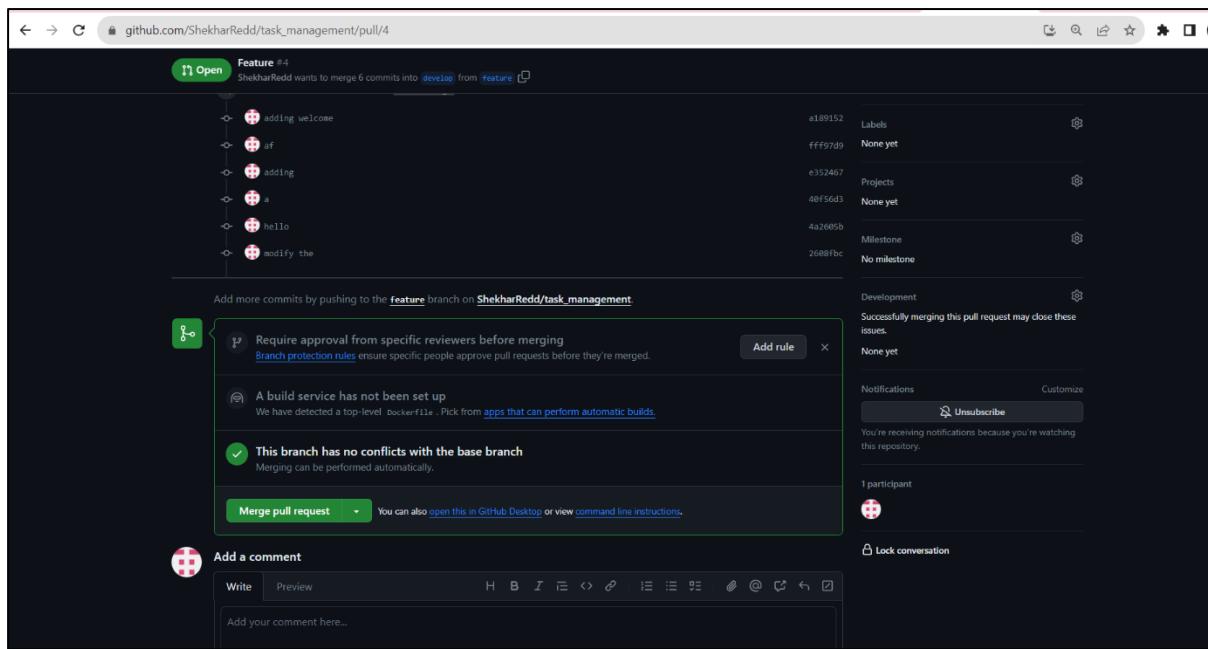
```
git commit -m "Implement new feature"
```

- **If you want to push the feature branch to the repository use the command.**

```
git push origin feature
```



- After pushing to the feature branch in your git repo it shows a banner at the top “feature” has recent pushes less than a minute ago”.
- Click on Compare & Pull request and merge the pull request. Then your feature branch is successfully merged to the main branch.



### Release branch:

A release branch in Git is a branch specifically created to prepare and stabilize a software release. This branch is used to isolate the work related to finalizing and testing the code for a new version before it is deployed to production.

- **Create a Release Branch:**

```
git checkout main
```

```
git pull origin main
```

```
git checkout -b release/1.0.0
```

- **Make Changes and Bug Fixes:**

```
git add .
```

```
git commit -m "Fix issues and prepare for release"
```

- **Merge Release Branch into master:**

```
git checkout master
```

```
git merge release/1.0.0
```

- **Tag the Release:**

```
git tag -a v1.0.0 -m "Release version 1.0.0"
```

- **Push Changes and Tags:**

```
git push origin main
```

```
git push --tags
```

## **Git commit message conventions:**

Git Conventional Commit messages follow a specific format and convention to standardize the way developers write commit messages. This convention is particularly popular in projects that use semantic versioning and automated release processes.

A Conventional Commit message follows a structured format:

<type>(<scope>): <description>

[optional body]

[optional footer]

- **<type>**: Describes the purpose of the commit (e.g., feat, fix, chore).
- **<scope>**: Specifies the scope of the commit (optional but recommended).
- **<description>**: Briefly describes the changes introduced by the commit.
- **[optional body]**: Provides additional details about the changes (optional).
- **[optional footer]**: Includes any additional information or references (optional).

## **Example Conventional Commit:**

**git commit -m "feat(user-auth): implement two-factor authentication"**

- Added support for two-factor authentication in the user authentication module.
- Implemented time-based one-time passwords (TOTP) for enhanced security.
- Updated user interface to prompt users for additional verification during login.
- Unit and integration tests have been added to ensure the security and functionality of two-factor authentication.
- Resolves #123
  - **Type (feat)**: Indicates that it's a new feature.
  - **Scope (user-auth)**: Specifies the scope of the changes, in this case, the user authentication module.
  - **Description (implement two-factor authentication)**: Briefly describes the purpose of the commit.
  - **Details (- Added support...)**: Provides additional details about what the commit does. Bullet points are used for clarity.
  - **Reference (Resolves #123)**: Includes a reference to an associated issue or task in your issue tracking system

## Sub Task 2 :

### Configuring Jenkins on AWS EC2 – LINUX:

#### Launching an Amazon EC2 instance:

- Sign in to the AWS Management Console.
- Open the Amazon EC2 console by selecting EC2 under Compute.
- From the Amazon EC2 dashboard, select **Launch Instance**.

The screenshot shows the AWS EC2 Dashboard in the US East (Ohio) Region. The left sidebar includes links for EC2 Dashboard, Events, Tags, Limits, Instances (with sub-links for Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs), and Elastic Block Store (Volumes, Snapshots). The main panel displays 'Resources' with counts for Instances (running), Elastic IPs, Key pairs, Placement groups, and Snapshots. A central box says 'Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. Learn more'. Below it is the 'Launch instance' section, which contains a note: 'To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.' A large orange 'Launch Instance' button is highlighted with a red box. To the right, the 'Service health' status shows 'Region: US East (Ohio)' and 'Status: This service is operating normally'. The top right corner shows account attributes like 'Supported platforms: VPC' and 'Default VPC: vpc-6129910a'.

- Choose an Amazon Machine Image (AMI). Select Amazon Linux AMI

The screenshot shows the AWS Marketplace search results for 'Application and OS Images (Amazon Machine Image)'. A search bar at the top has the placeholder 'Search our full catalog including 1000s of application and OS images'. Below it, there are tabs for 'Recents' and 'Quick Start'. Under 'Quick Start', several AMI icons are shown: macOS (Mac), Ubuntu (Ubuntu), Windows (Microsoft), Red Hat (Red Hat), and Amazon Linux (AWS logo). The 'Amazon Linux' icon is highlighted with a red box. To the right, a 'Browse more AMIs' section with a magnifying glass icon and the text 'Including AMIs from AWS, Marketplace and the Community' is visible. A detailed view of the 'Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type' is shown, with its AMI ID 'ami-09d3b3274b6c5d4aa' and other details like 'Virtualization: hvm', 'ENA enabled: true', and 'Root device type: ebs'. This view is also highlighted with a red box. At the bottom, there's a 'Description' section for 'Amazon Linux 2 Kernel 5.10 AMI 2.0.20221004.0 x86\_64 HVM gp2', an 'Architecture' dropdown set to '64-bit (x86)', an 'AMI ID' field containing 'ami-09d3b3274b6c5d4aa', and a 'Verified provider' badge.

- Scroll down and select the key pair you have already created or create a new key pair.
- Select an existing security group or create a new security group
- Select the security group.
- Select **Launch Instance**.

**Key pair (login) Info**  
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - **required**

**Network settings Info**  
  
Network Info  
vpc-5d7a3227  
Subnet Info  
No preference (Default subnet in any availability zone)  
Auto-assign public IP Info  
Enable  
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.  
 Create security group  Select existing security group   
Common security groups Info

**Summary**  
Number of instances Info  
1  
Software Image (AMI)  
Amazon Linux 2 Kernel 5.10 AMI...[read more](#)  
ami-09d3b3274b6c5d4aa  
Virtual server type (instance type)  
t2.micro  
Firewall (security group)  
-  
Storage (volumes)  
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

- In the left-hand navigation bar, choose **Instances** to view the status of your instance. Initially, the status of your instance is pending. After the status changes to running, your instance is ready for use.

**Instances (1/1) Info**

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input checked="" type="checkbox"/>	-	i-05bb8d08747b18bad	<span>Running</span>	t2.micro	<span>2/2 checks ...</span>	No alarms	us-east-2a	ec2-3-137-1

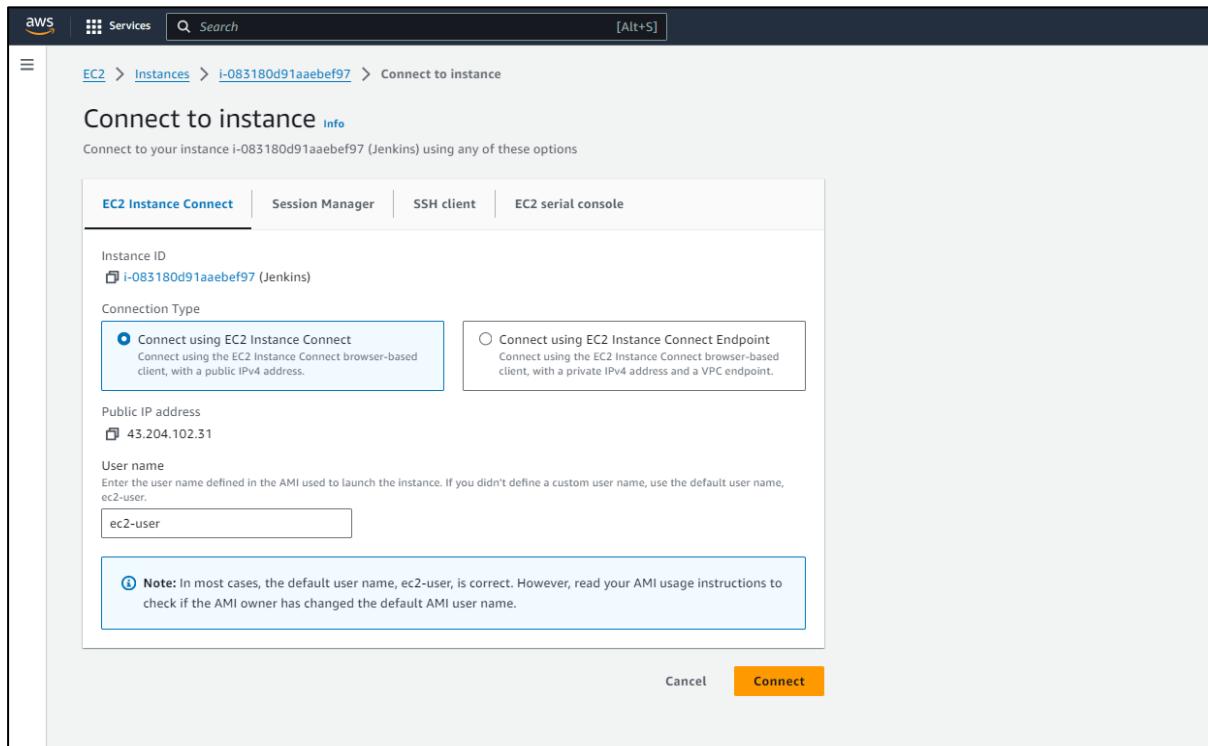
- Now that the Amazon EC2 instance has been launched, Jenkins can be installed properly.

In this step you will deploy Jenkins on your EC2 instance by completing the following tasks:

1. Connecting to your Linux instance
2. Downloading and installing Jenkins using docker
3. Configuring Jenkins

**To connect to your instance using the browser-based client from the Amazon EC2 console:**

- In the navigation pane, choose **Instances**.
- Select the instance and choose **Connect**.
- Choose **EC2 Instance Connect**.



- Verify the User name and choose **Connect** to open a terminal window.

```
A newer release of "Amazon Linux" is available.
Version 2023.2.20231113:
Run "/usr/bin/dnf check-release-update" for full release and version update info
.
.
.
Last login: Thu Nov 16 04:57:10 2023 from 13.233.177.4
[ec2-user@ip-172-31-12-85 ~]$
```

## Downloading and installing Jenkins:

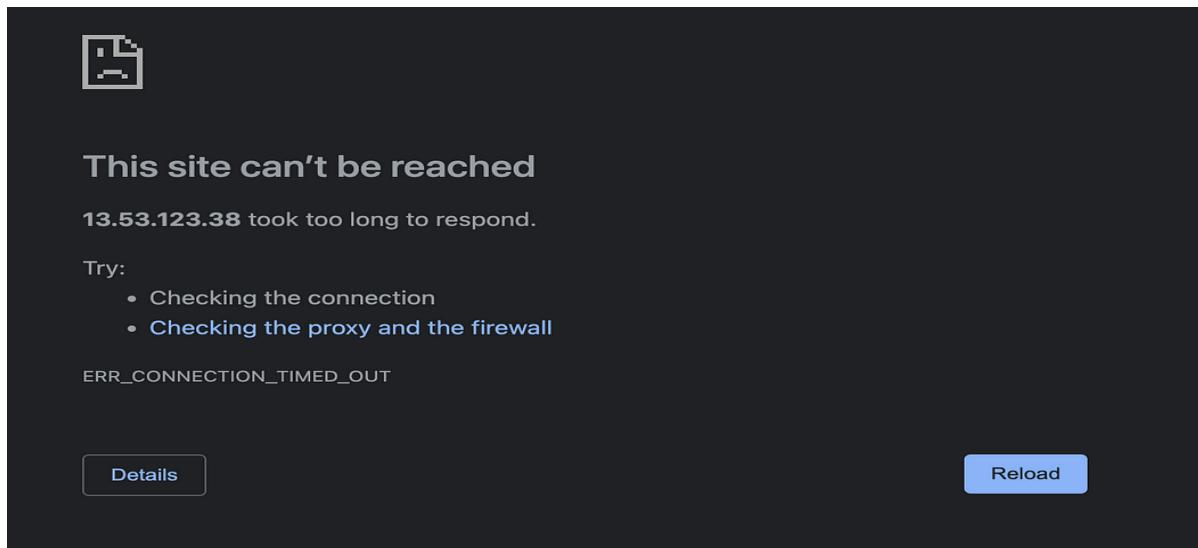
- Ensure that your software packages are up to date on your instance  
**sudo yum update**
- First, we need to complete the Docker installation.  
**sudo yum install docker**
- To run Docker commands without using sudo each time, we should execute the commands below.  
**sudo groupadd docker**  
**sudo usermod -aG docker \${USER}**
- To check Docker, let's run the command below.  
**docker info**

```
[ec2-user@ip-172-31-21-151 ~]$ docker info
Client:
Version: 24.0.5
Context: default
Debug Mode: false
Plugins:
buildx: Docker Buildx (Docker Inc.)
  Version: v0.0.0+unknown
  Path: /usr/libexec/docker/cli-plugins/docker-buildx

Server:
ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
errors pretty printing info
[ec2-user@ip-172-31-21-151 ~]$
```

- To start docker  
**sudo service docker start**  
**sudo systemctl enable docker.service**
- Next, we can start the Jenkins container.
- Before starting the container we need to create the volume to attach the volume to the Jenkins container.
- Jenkins volumes provides a way to persist and manage data generated by docker containers.
- To create a volume  
**docker volume create jenkins\_home**
- Run the container  
**docker run -u root -d -p 80:8080 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/Jenkins**
- The purpose of mounting the /var/run/docker.sock file into a Jenkins container is to allow the Jenkins instance to interact with the Docker daemon on the host machine.
- The -v /var/run/docker.sock:/var/run/docker.sock option maps the Docker daemon's socket on the host (/var/run/docker.sock) to the same path inside the container.

- When we try to access Jenkins, as seen in the image, the page is unreachable.



- For this, we need to add inbound rules to the Security Group.

Inbound rules <small>Info</small>						
Security group rule ID	Type <small>Info</small>	Protocol <small>Info</small>	Port range <small>Info</small>	Source <small>Info</small>	Description - optional <small>Info</small>	
sgr-04b795001383ee019	HTTP	TCP	80	Cus... ▾	<input type="text"/> <input type="button" value="Delete"/>	<input type="button" value="0.0.0/0 X"/>
sgr-0ed22528c9bca4326	SSH	TCP	22	Cus... ▾	<input type="text"/> <input type="button" value="Delete"/>	<input type="button" value="0.0.0/0 X"/>
<input type="button" value="Add rule"/>						

- Now we can access Jenkins.

Getting Started

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

**Administrator password**

- To obtain the password, we first need to connect to the Jenkins container and print the "initialAdminPassword" to the console.

```
docker exec -it jenkins bash
```

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

Now Copy the text and paste in the Administrator password

- After successful authentication we can access the Jenkins page:

The screenshot shows the Jenkins dashboard with the following details:

- Sidebar:** New Item, People, Build History, Manage Jenkins, My Views, Build Queue (No builds in the queue), Build Executor Status (1 Idle, 2 Idle).
- Central Area:** A weather report showing aggregated status of recent builds for three projects:
 

S	W	Name	Last Success	Last Failure	Last Duration
Green checkmark	Sun icon	JenkinsBuild	1 day 14 hr #1	N/A	0.69 sec
Green checkmark	Sun icon	Project1	4 days 17 hr #2	N/A	0.3 sec
Green checkmark	Cloud icon	SamplecodePipeline	3 days 20 hr #3	3 days 20 hr #2	0.15 sec
- Bottom:** Icon legend (S, M, L), Atom feed links for all, failures, and latest builds.

- Execute this command in the Jenkins container to install the docker inside the Jenkins container.

```
curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall
```

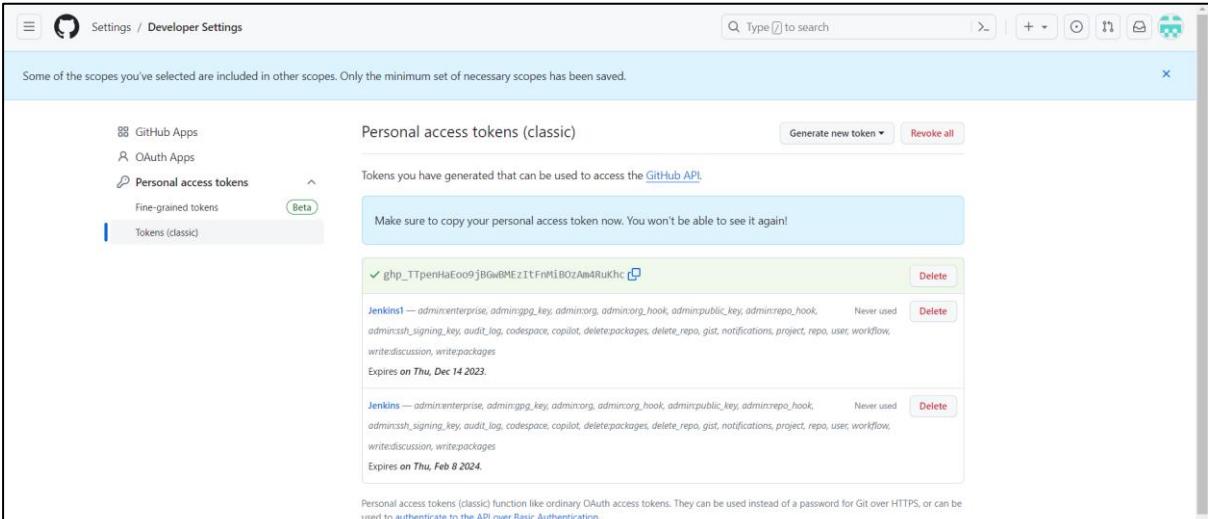
- chmod 666 /var/run/docker.sock to give the permissions to the user.

## Sub Task 2:

### Configuring Jenkins to monitor the Git repository:

#### Creating Personal access token:

- Go to [GitHub](#) and log in to your account.
- Click on your profile picture in the top right corner and select "Settings."
- In the left sidebar, click on "Developer settings."
- Click on "Personal access tokens."

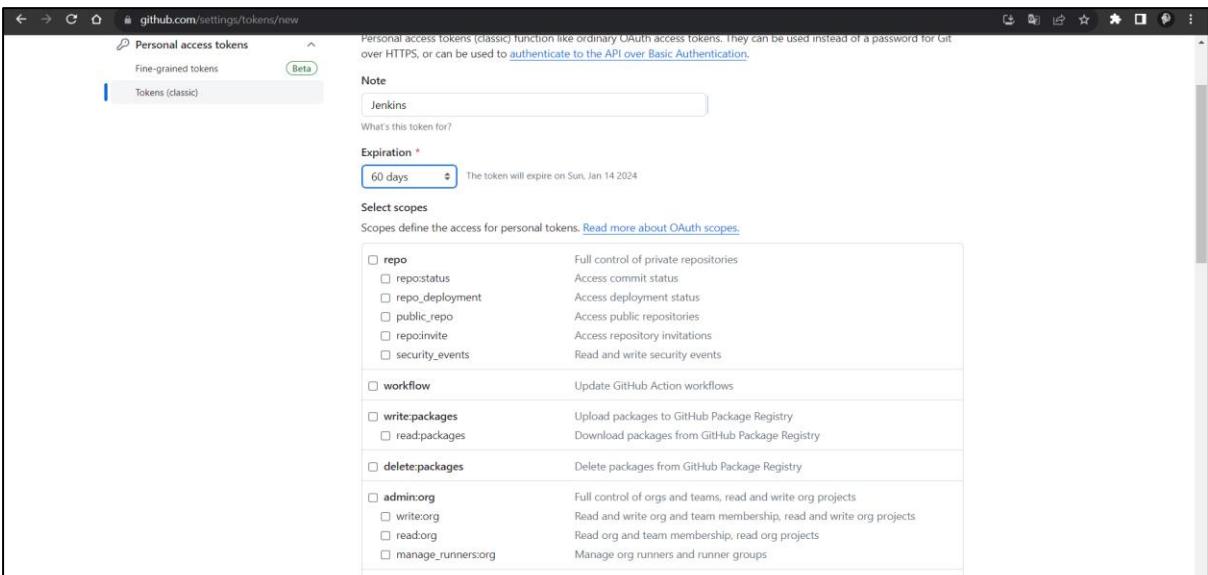


The screenshot shows the GitHub Developer Settings page under Personal access tokens (classic). It displays two tokens:

- Jenkins1** — Never used  
Scopes: admin:enterprise, admin:pgp\_key, admin:org, admin:org\_hook, admin:public\_key, admin:repo\_hook, admin:ssh\_signing\_key, audit\_log, codespace, capitol, delete:packages, delete\_repo, gist, notifications, project, repo, user, workflow.  
Expires on Thu, Dec 14 2023.
- Jenkins** — Never used  
Scopes: admin:enterprise, admin:pgp\_key, admin:org, admin:org\_hook, admin:public\_key, admin:repo\_hook, admin:ssh\_signing\_key, audit\_log, codespace, capitol, delete:packages, delete\_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages  
Expires on Thu, Feb 8 2024.

A note at the bottom states: "Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#)".

- Click the "Generate new token" button.
- Enter a name for your token.



The screenshot shows the GitHub "Personal access tokens" creation form. The "Note" field contains "Jenkins". The "Expiration" dropdown is set to "60 days". The "Select scopes" section lists various GitHub permissions with their descriptions:

Scope	Description
<input type="checkbox"/> <b>repo</b>	Full control of private repositories
<input type="checkbox"/> <b>repo_status</b>	Access commit status
<input type="checkbox"/> <b>repo_deployment</b>	Access deployment status
<input type="checkbox"/> <b>public_repo</b>	Access public repositories
<input type="checkbox"/> <b>repo_invite</b>	Access repository invitations
<input type="checkbox"/> <b>security_events</b>	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write_packages</b>	Upload packages to GitHub Package Registry
<input type="checkbox"/> <b>read_packages</b>	Download packages from GitHub Package Registry
<input type="checkbox"/> <b>delete_packages</b>	Delete packages from GitHub Package Registry
<input type="checkbox"/> <b>admin:org</b>	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> <b>write:org</b>	Read and write org and team membership, read and write org projects
<input type="checkbox"/> <b>read:org</b>	Read org and team membership, read org projects
<input type="checkbox"/> <b>manage_runners:org</b>	Manage org runners and runner groups
<input type="checkbox"/> <b>admin:public_key</b>	Full control of user public keys

- Choose the desired scopes or permissions for your token based on your requirements.

The screenshot shows the GitHub settings page for generating a new token. It lists several permission scopes with their descriptions:

- manage\_runners:enterprise**: Manage enterprise runners and runner groups
- manage\_billing:enterprise**: Read and write enterprise billing data
- readenterprise**: Read enterprise profile data
- audit\_log**: Full control of audit log
- codespace**: Full control of codespaces
- copilot**: Full control of GitHub Copilot settings and seat assignments
- project**: Full control of projects
- admin:gpg\_key**: Full control of public user GPG keys
- admin:ssh\_signing\_key**: Full control of public user SSH signing keys

At the bottom, there are "Generate token" and "Cancel" buttons.

- Scroll down and click the "Generate token" button.
- Copy the generated token and store it in a secure place. Once you navigate away, you won't be able to see it again.

### Webhook:

A webhook is a mechanism to automatically trigger the build of a Jenkins project upon a commit pushed in a Git repository.

In order for builds to be triggered automatically by PUSH and PULL REQUEST events, a Jenkins Web Hook needs to be added to each GitHub repository. You need admin permissions on that repository.

- “Github” Plugin must be installed on Jenkins.
- Create a new Jenkins job or edit an existing one with “pipeline”.

The screenshot shows the Jenkins dashboard with a modal window for creating a new item. The modal has a text input field labeled "Enter an item name" containing "Narasimha". Below the input field are three project types with their descriptions:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

- Run a sample code from pipeline script. This is the sample code you can get.

The screenshot shows the Jenkins Pipeline configuration page. On the left, there's a sidebar with 'Configure' and three tabs: 'General', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is selected. The main area contains a code editor with Groovy pipeline syntax:

```

1+ pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello World'
8       }
9     }
10    }
11  }
12

```

Below the code editor is a checkbox labeled 'Use Groovy Sandbox'. At the bottom are 'Save' and 'Apply' buttons. The status bar at the bottom right shows 'REST API' and 'Jenkins 2.432'.

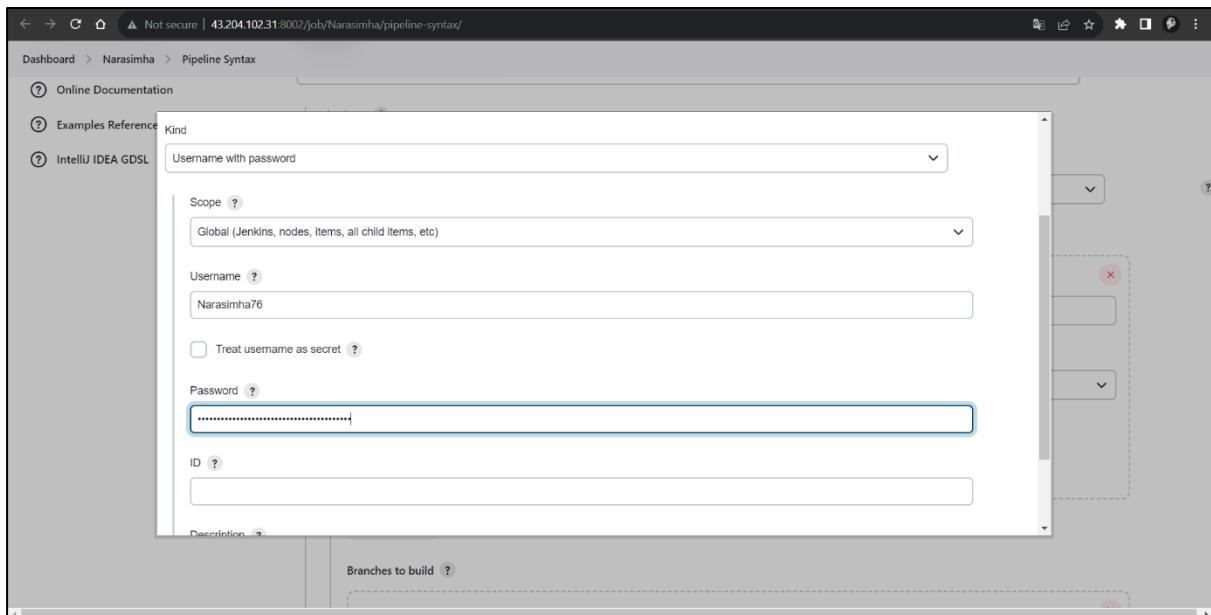
- Click on Pipeline Syntax.
- In the Pipeline syntax , Sample step select “checkout: Check out from version control”.

The screenshot shows the Jenkins Pipeline Syntax dropdown menu. The 'checkout' step is highlighted in blue. Other options visible include archiveArtifacts, bat, build, catchError, cleanWs, configFileProvider, deleteDir, dir, echo, emailext, emalexrecipients, error, fileExists, findBuildScans, fingerprint, git, input, isUnix, junit, and archiveArtifacts.

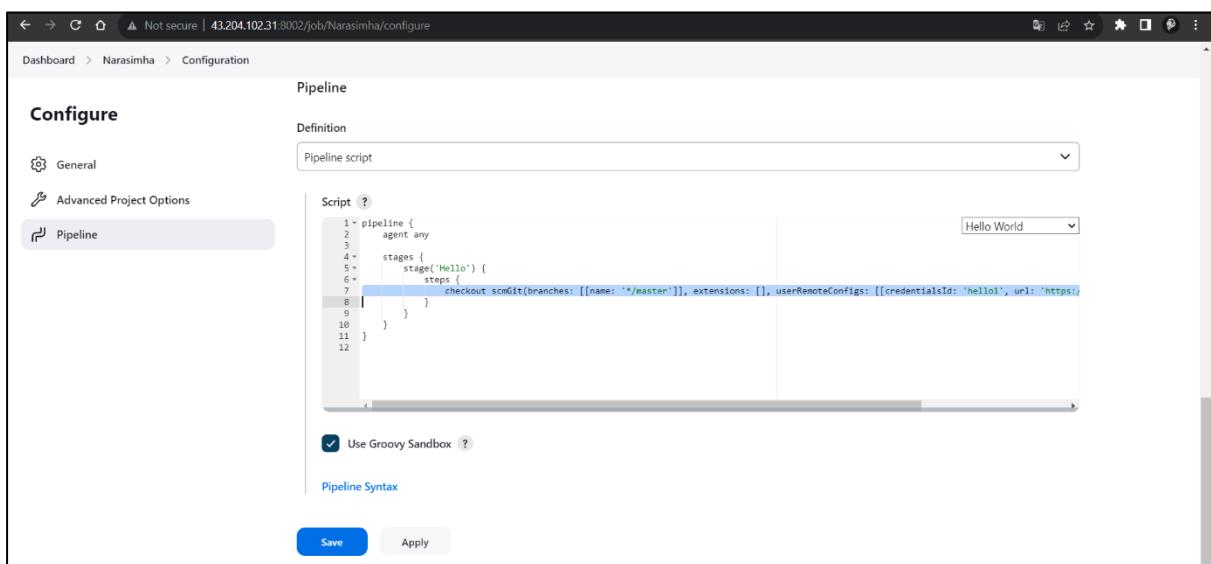
- Add GitHub repository URL here.

The screenshot shows the IntelliJ IDEA GDSL SCM configuration dialog. Under the 'Repositories' section, there is a 'Repository URL' field containing 'https://github.com/Narasimha76/task.git'. Below it, the 'Credentials' dropdown is set to 'none'. A tooltip for 'Jenkins Credentials Provider' is visible.

- In the credentials input add GitHub Username and add the Personal access token that we have created in the previous step.



- Click on Generate Pipeline script. You will get a Script copy it.



- Paste it in the pipeline script definition and click on save.

## Creating a webhook:

- Go to the GitHub repository where you want to set up the webhook.
- Click on the "Settings" tab of your repository.
- In the left sidebar, click on "Webhooks & Services."
- Click the "Add webhook" button.

The screenshot shows the GitHub settings interface for a repository named 'Narasimha76 / task'. On the left, the 'Webhooks' section is selected in the sidebar. The main panel is titled 'Webhooks / Add webhook'. It contains fields for 'Payload URL' (set to 'http://43.204.102.31:8002/github-webhook/'), 'Content type' (set to 'application/x-www-form-urlencoded'), and a 'Secret' field which is empty. Below these, there's a section for selecting events: 'Just the push event' (selected), 'Send me everything', and 'Let me select individual events'. At the bottom, the 'Active' checkbox is checked. A note at the bottom right says 'We will deliver event details when this hook is triggered.'

- In the "Payload URL" field, enter the Jenkins webhook URL. This should be the URL where Jenkins can receive GitHub push events.

**Format:** <http://jenkins-server/github-webhook/>

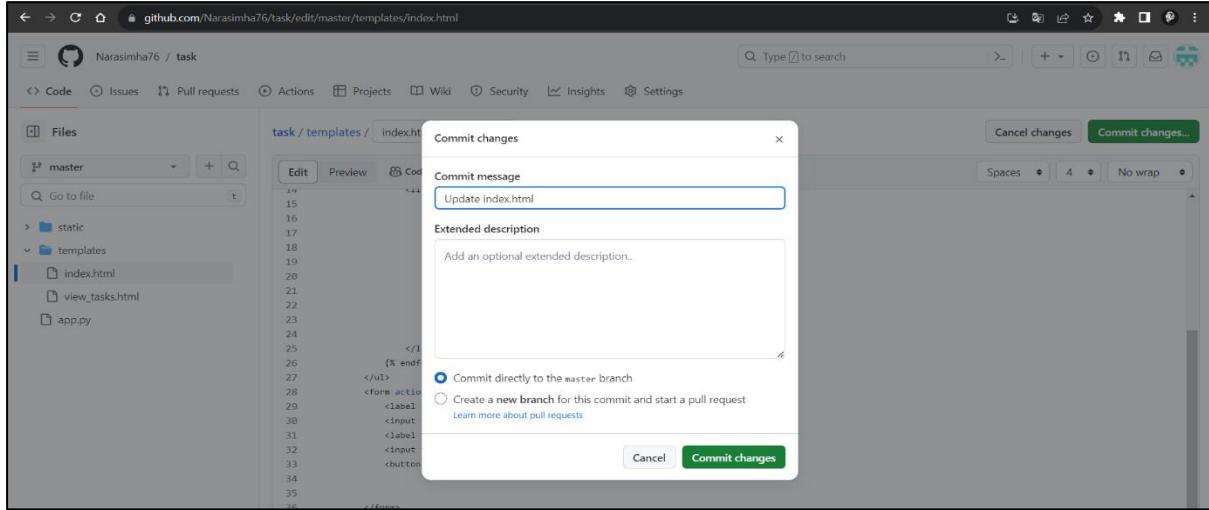
- Replace **jenkins-server** with the actual address of your Jenkins server.
- Choose the events that should trigger the webhook. For CI/CD, you typically want to select at least "Just the push event."
- Click the "Add webhook" button to save the webhook.
- Ensure the webhook is set to be active

The screenshot shows the GitHub settings interface for the same repository. The 'Webhooks' section is now populated with a single entry: 'http://43.204.102.31:8002/github-w... (push)'. To the right of this entry are 'Edit' and 'Delete' buttons. A message at the top of the panel says 'Okay, that hook was successfully created. We sent a ping payload to test it out! Read more about it at https://docs.github.com/webhooks/#ping-event.' The 'Webhooks' section in the sidebar is also highlighted.

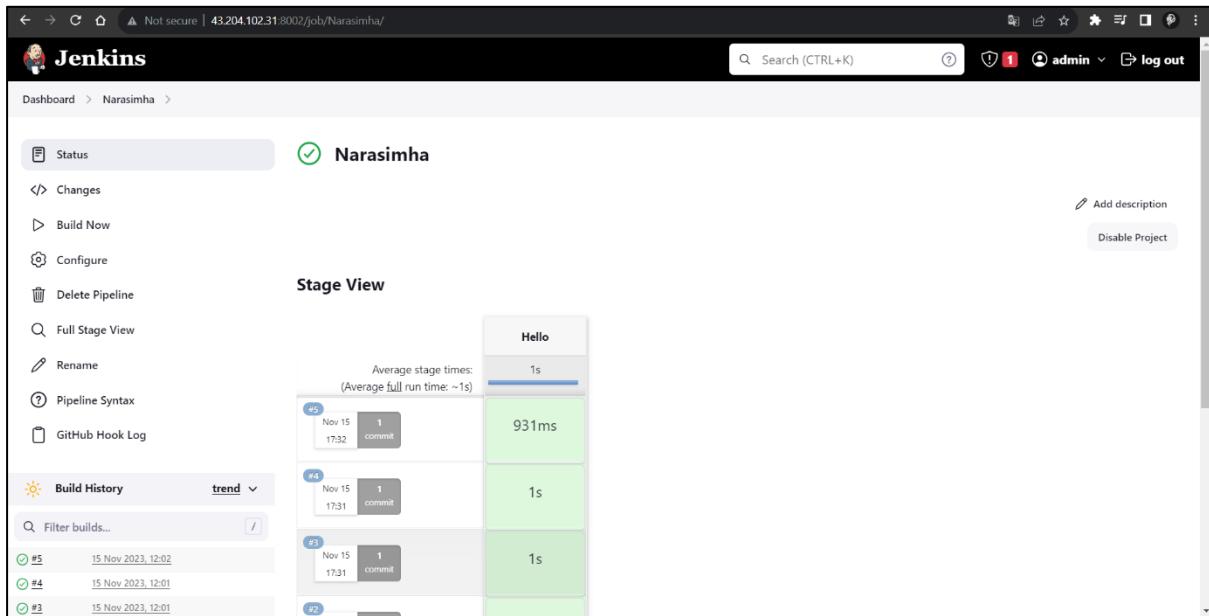
- Ensure the webhook is set to be active.

## Test the setup:

- Make a small change (e.g., push a new commit) to your GitHub repository.



- Go to your Jenkins dashboard.
- You should see your job triggered by the GitHub webhook.



## Set up build jobs for different branches.

### Setting Up Jenkins Build Jobs for Feature Branches

- Prerequisites

Jenkins installed and configured.

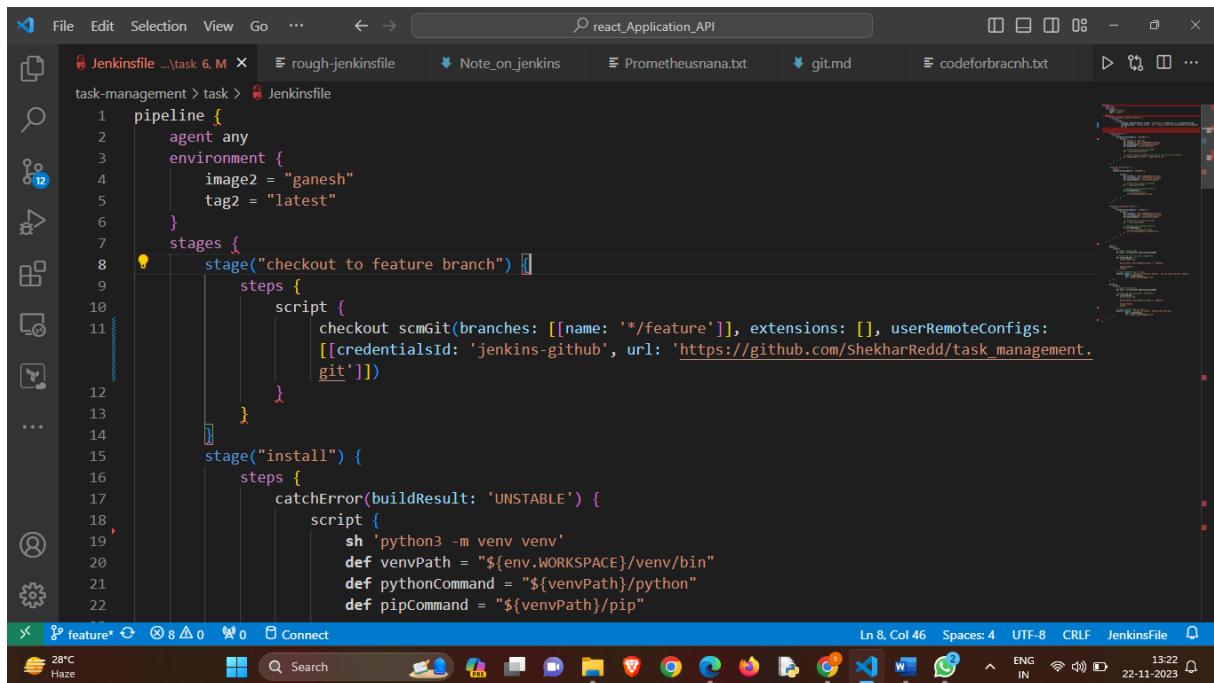
Jenkins plugins: GitHub, Email Extension

Python and required dependencies installed on the Jenkins agent.

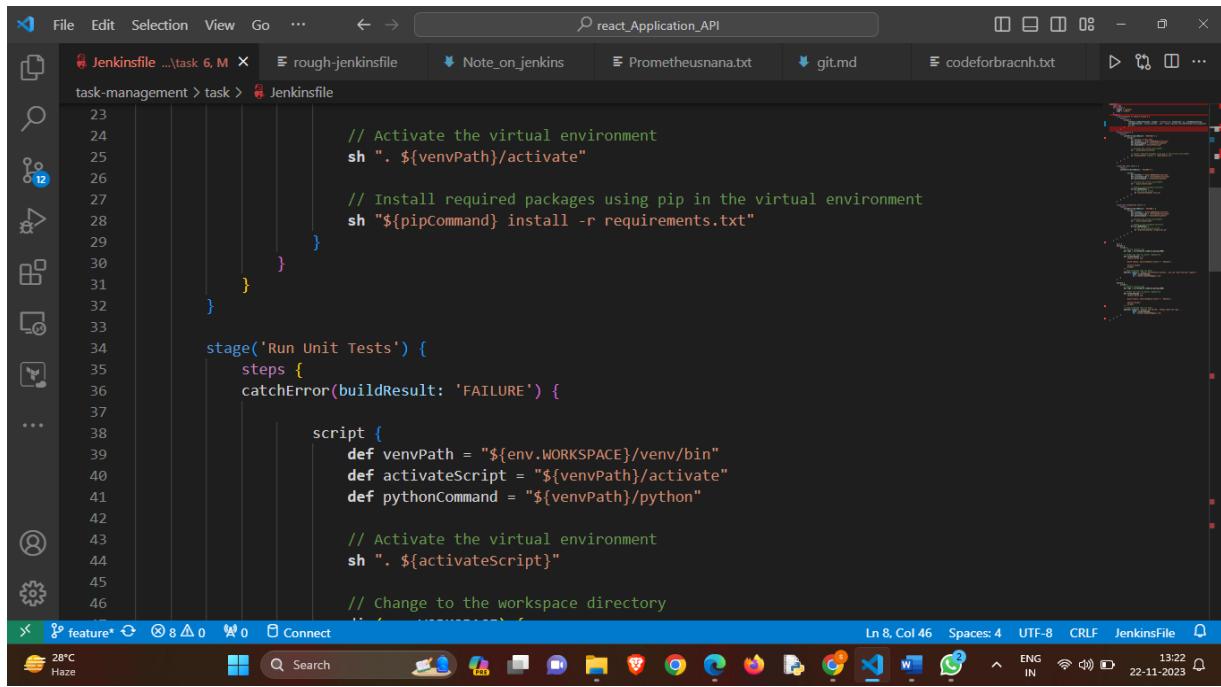
- **Configure Feature Branch Job**

1. Open Jenkins and navigate to the Jenkins dashboard.
2. Click on "New Item" to create a new Jenkins job.
3. Enter a name for the job (e.g., " feature\_branch").
4. Select the "Pipeline" project type.
5. In Build Trigger section, select the "GitHub hook trigger for GITScm polling" option.
6. In the Pipeline section, choose "Pipeline script from SCM" as the Definition.
7. Choose "Git" as the SCM.
8. Set the Repository URL to your GitHub repository (e.g., 'https://github.com/ShekharRedd/task\_management.git').
9. Under "Branches to build," add the branch specifier '\*/feature' to focus on feature branches.
10. In the Script Path section, Select the Jenkinsfile which is in the git repo.
11. Save the job configuration.

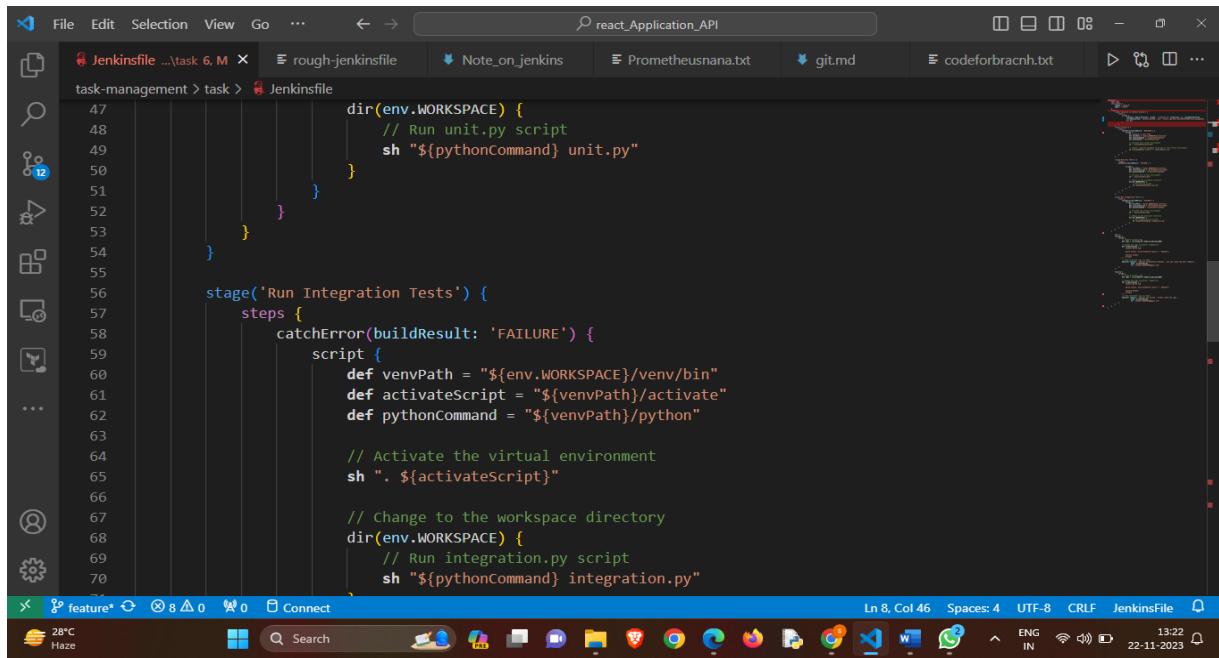
- **Jenkins Pipeline Script**



```
task-management > task > Jenkinsfile
1 pipeline {
2     agent any
3     environment {
4         image2 = "ganesh"
5         tag2 = "latest"
6     }
7     stages {
8         stage("checkout to feature branch") {
9             steps {
10                script {
11                    checkout scmGit(branches: [[name: '*/feature']], extensions: [], userRemoteConfigs: [[credentialsId: 'jenkins-github', url: 'https://github.com/ShekharRedd/task_management.git']])
12                }
13            }
14        }
15        stage("install") {
16            steps {
17                catchError(buildResult: 'UNSTABLE') {
18                    script {
19                        sh 'python3 -m venv venv'
20                        def venvPath = "${env.WORKSPACE}/venv/bin"
21                        def pythonCommand = "${venvPath}/python"
22                        def pipCommand = "${venvPath}/pip"
23                    }
24                }
25            }
26        }
27    }
28 }
```



```
23 // Activate the virtual environment
24 sh ". ${venvPath}/activate"
25
26 // Install required packages using pip in the virtual environment
27 sh "${pipCommand} install -r requirements.txt"
28
29 }
30 }
31 }
32 }
33
34 stage('Run Unit Tests') {
35     steps {
36         catchError(buildResult: 'FAILURE') {
37             script {
38                 def venvPath = "${env.WORKSPACE}/venv/bin"
39                 def activateScript = "${venvPath}/activate"
40                 def pythonCommand = "${venvPath}/python"
41
42                 // Activate the virtual environment
43                 sh "${activateScript}"
44
45                 // Change to the workspace directory
46             }
47         }
48         dir(env.WORKSPACE) {
49             // Run unit.py script
50             sh "${pythonCommand} unit.py"
51         }
52     }
53 }
54
55 stage('Run Integration Tests') {
56     steps {
57         catchError(buildResult: 'FAILURE') {
58             script {
59                 def venvPath = "${env.WORKSPACE}/venv/bin"
60                 def activateScript = "${venvPath}/activate"
61                 def pythonCommand = "${venvPath}/python"
62
63                 // Activate the virtual environment
64                 sh ". ${activateScript}"
65
66                 // Change to the workspace directory
67                 dir(env.WORKSPACE) {
68                     // Run integration.py script
69                     sh "${pythonCommand} integration.py"
70                 }
71             }
72         }
73     }
74 }
```



```
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
```

The screenshot shows a Jenkinsfile script in a code editor. The script defines a post-build step that captures logs, formats them, and sends them via email. The Jenkinsfile is part of a task named 'task 6' in a project named 'task-management'. The code includes Jenkins-specific annotations like `@Task` and `@Post`. The right side of the screen displays a detailed preview of the build log output.

```
task-management > task > Jenkinsfile
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94

  post {
    success {
      script {
        // Capture console logs
        def logs = currentBuild.rawBuild.getLog(1000)

        // Format the logs for better readability
        def formattedLogs = """
          Jenkins Build Log

          Build Status: ${currentBuild.result ?: 'Unknown'}

          Console Output:
          ${logs}
        """

        // Send formatted logs via email
        emailext subject: 'Jenkins Successfully executed, you can raise the pull request'.
      }
    }
  }
```

The screenshot shows a code editor with a Jenkinsfile open. The file contains Groovy script for Jenkins automation. It includes logic to capture build logs, format them, and send them via email if the build fails. The code uses Jenkins' built-in variables like \${currentBuild.result} and \${logs}.

```
task-management > task > Jenkinsfile
93     |         emailext subject: 'Jenkins successfully execute , you can raise the pull request',
94     |         |         body: formattedLogs,
95     |         |         to: 'shekharreddy1010@gmail.com'
96     }
97 }
98 failure {
99     script {
100        // Capture console logs
101        def logs = currentBuild.rawBuild.getLog(1000)
102        // Format the logs for better readability
103        def formattedLogs = """
104            Jenkins Build Log
105            Build Status: ${currentBuild.result ?: 'Unknown'}
106            Console Output:
107            ${logs}
108        """
109        // Send formatted logs via email
110        emailext subject: 'Jenkins job failed , Please check the logs ' ,
111        |         body: formattedLogs,
112        |         to: 'shekharreddy1010@gmail.com'
113    }
114 }
115 }
116 }
```

## EMAIL Configuration

- Install E-mail Notification and Extended E-mail Notification plugins
  - E-mail Notification: This plugin Mailer, allows users to send basic emails.
  - Extended E-mail Notification: This plugin Email Extension Plugin, allows users to customize when emails are sent, who receives them, and the content of the emails.
  - Goto Dashboard >> Manage Jenkins >> System

- Configure the settings as shown in below pictures.

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix

Advanced

Use SMTP Authentication

User Name  
shekharreddy0702@gmail.com

Password  
 Concealed [Change Password](#)

Save [Apply](#)

Use SSL

Use TLS

SMTP Port

465

Reply-To Address

Charset

UTF-8

Test configuration by sending test e-mail

Save [Apply](#)

REST API Jenkins 2.432

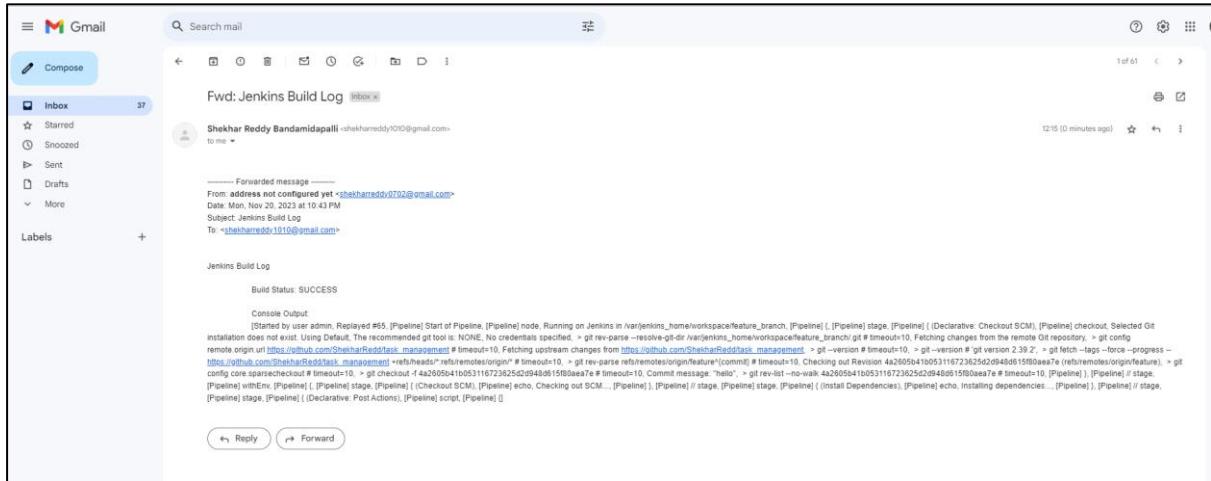
The screenshot shows the 'Manage Jenkins > System > Extended E-mail Notification' configuration page. It includes fields for 'SMTP server' (smtp.gmail.com) and 'SMTP Port' (465). An 'Advanced' dropdown is open, showing a 'Credentials' section with a dropdown menu containing 'shekharreddy0702@gmail.com/\*\*\*\*\*'. There is also a checked 'Use SSL' option. At the bottom are 'Save' and 'Apply' buttons.

## Run the Feature Branch Job

- Trigger a build manually or configure webhooks for automatic triggering on code changes.
- Monitor the build progress on the Jenkins dashboard.
- Review the console output and logs for each stage.
- Receive email notifications on build success or failure.

The screenshot shows the Jenkins 'feature\_branch' pipeline stage view. The left sidebar includes options like 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', 'Pipeline Syntax', 'GitHub Hook Log', and 'Build History' (with a 'trend' dropdown). The main area displays a 'Stage View' table with six columns: 'Declarative: Checkout SCM', 'checkout to feature branch', 'install', 'Run Unit Tests', 'Run Integration Tests', and 'Declarative: Post Actions'. The first two columns show average stage times of 1s and 234ms respectively. The remaining four columns show times of 5s, 1s, 1s, and 4s. Below the table, two build history entries are shown: #71 (Nov 21 21:10, 1 commit) and #70 (Nov 21 13:07, 1 commit). The bottom navigation bar includes links for 'Dashboard', 'feature\_branch', and 'Logout'.

- Jenkins sends an email notification when a build completes, whether it's a successful build, unstable build, or a failed build.



## UNIT TEST :

- Unit testing in Python is a crucial step in making sure each part of your code works correctly. It involves checking the smallest units, or components, of your program to ensure they function as expected.
- Python's built-in "unittest" module helps you create and run these tests. This process is essential for building reliable and error-free software

## Use case:

- Unit testing ensures that each isolated part (unit) of a software application functions correctly in isolation, catching bugs early in the development process.

```

44     #     unittest.main()
45
46     from datetime import datetime
47     import unittest
48     from app import app, tasks
49
50     class TestApp(unittest.TestCase):
51
52         def setUp(self):
53             self.app = app.test_client()
54             self.app.testing = True
55             tasks.clear() # Clear tasks before each test
56
57         def test_index(self):
58             response = self.app.get('/')
59             self.assertEqual(response.status_code, 200)
60
61         def test_add(self):
62             task_data = {'task': 'Test Task', 'due_date': '2023-11-30'}
63             response = self.app.post('/add', data=task_data, follow_redirects=True)
64
65             self.assertEqual(response.status_code, 200)
66             self.assertIn({'task': 'Test Task', 'due_date': datetime(2023, 11, 30)}, tasks)
67
68         def test_delete_valid_id(self):
69             tasks.append({'task': 'Test Task', 'due_date': datetime.now()})
70             response = self.app.get('/delete/0')
71             self.assertEqual(response.status_code, 302)
72             self.assertEqual(len(tasks), 0)
73
74         def test_delete_invalid_id(self):
75             response = self.app.get('/delete/0')
76             self.assertEqual(response.status_code, 302)
77             self.assertEqual(len(tasks), 0)
78
79         def test_view_tasks(self):
80             response = self.app.get('/view_tasks')
81             self.assertEqual(response.status_code, 200)
82
83     if __name__ == '__main__':
84         unittest.main()

```

- If all assertions within a test case are true, the test case is considered to have passed.
- The testing framework will typically output a message indicating that the test has passed, and the overall result of the test run is considered successful.



The screenshot shows a Jenkins job named 'feature\_branch' with build number 23. The console output window displays a series of command-line commands being executed. These commands include activating a virtual environment, setting environment variables like VIRTUAL\_ENV and PATH, and changing the prompt to show the venv context. The output concludes with a summary of test results: "Ran 5 tests in 0.026s".

```

+ . /var/jenkins_home/workspace/feature_branch/venv/bin/activate
+ deactivate nondestructive
+ [ -n ]
+ [ -n ]
+ [ -n -o -n ]
+ [ -n ]
+ unset VIRTUAL_ENV
+ unset VIRTUAL_ENV_PROMPT
+ [ ! nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/jenkins_home/workspace/feature_branch/venv
+ export VIRTUAL_ENV
+ _OLD_VIRTUAL_PATH=/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
+ PATH=/var/jenkins_home/workspace/feature_branch/venv/bin:/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
+ export PATH
+ [ -n ]
+ [ -z ]
+ _OLD_VIRTUAL_PSL=$
+ PS1=(venv) $
+ export PS1
+ VIRTUAL_ENV_PROMPT=(venv)
+ export VIRTUAL_ENV_PROMPT
+ [ -n -o -n ]
[Pipeline] dir
Running in /var/jenkins_home/workspace/feature_branch
[Pipeline] {
[Pipeline] sh
+ /var/jenkins_home/workspace/feature_branch/venv/bin/python unit.py
.....
-----
Ran 5 tests in 0.026s

```

- If any assertion within a test case is false, the test case is considered to have failed.
- The testing framework will output information about which specific assertion failed, along with the expected and actual values.



The screenshot shows a Jenkins console log for a job named 'feature\_branch' with build number #27. The log output is as follows:

```

Not secure | 43.205.118.190:8080/job/feature_branch/27/console

Dashboard > feature_branch > #27

+ _OLD_VIRTUAL_PS1=$
+ PS1=(venv) $
+ export PS1
+ VIRTUAL_ENV_PROMPT=(venv)
+ export VIRTUAL_ENV_PROMPT
+ [ -n -o -n ]
[Pipeline] dir
Running in /var/jenkins_home/workspace/feature_branch
[Pipeline] {
[Pipeline] sh
+ /var/jenkins_home/workspace/feature_branch/venv/bin/python unit.py
F.....
=====
FAIL: test_add (_main__TestApp.test_add)

-----
Traceback (most recent call last):
  File "/var/jenkins_home/workspace/feature_branch/unit.py", line 65, in test_add
    self.assertEqual(response.status_code, 200)
AssertionError: 404 != 200

-----
Ran 5 tests in 0.019s

FAILED (failures=1)
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] }
ERROR: script returned exit code 1

```

## INTEGRATION TEST :

- Integration testing is a level of software testing where individual units or components of a system are combined and tested as a group to ensure that they work together seamlessly.

## UseCase:

- Integration testing helps catch issues related to the combination of components early in the development process, reducing the risk of problems in the production environment.

The screenshot shows a GitHub Copilot interface for a file named `integration.py`. The left sidebar lists project files like `develop`, `static`, `templates`, `qignore`, `Dockerfile`, `Jenkinsfile`, `app.py`, `imagedefinitions.json`, `integration.py`, `requirements.txt`, and `unit.py`. The main area displays Python code for a test class `TestIntegration` using the `unittest` framework. The code includes methods for adding and viewing tasks, and deleting and viewing tasks. A `Symbols` panel on the right shows definitions for `TestIntegration`, `setUp`, `test_add_and_view_tasks`, and `test_delete_and_view_tasks`.

```

import unittest
from datetime import datetime
from app import app, tasks

class TestIntegration(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        tasks.clear() # Clear tasks before each test

    def test_add_and_view_tasks(self):
        # Simulate adding a task
        task_data = {"task": "Test Task", "due_date": '2023-11-30'}
        self.app.post('/add', data=task_data, follow_redirects=True)

        # Simulate viewing tasks
        response = self.app.get('/view_tasks')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Test Task', response.data)
        self.assertIn(b'2023-11-30', response.data)

    def test_delete_and_view_tasks(self):
        # Add a task for testing
        tasks.append({"task": "Test Task", "due_date": datetime.now()})
        # Simulate deleting the task
        response = self.app.get('/delete/0', follow_redirects=True)

        self.assertEqual(response.status_code, 200)
        self.assertEqual(len(tasks), 0)
        # Simulate viewing tasks after deletion
        response = self.app.get('/view_tasks')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Test Task', response.data)

if __name__ == '__main__':
    unittest.main()

```

- If all the test cases in your integration test suite pass, it indicates that the integrated components are working together as expected.
- The testing framework or test runner will typically output a message indicating that the integration tests have passed.

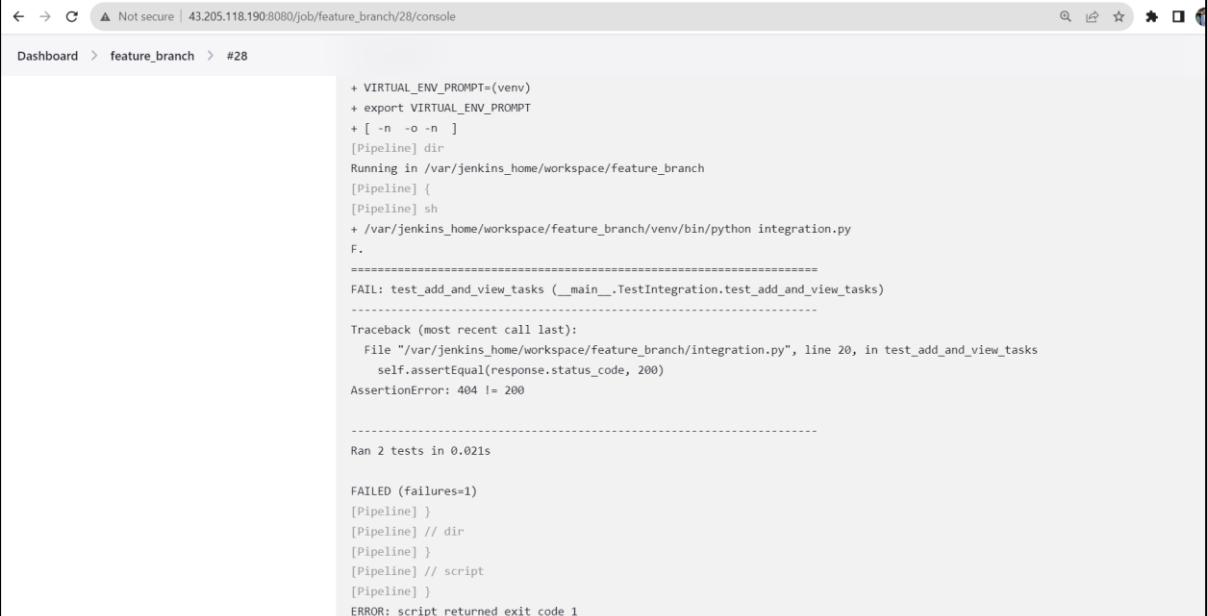
The screenshot shows a Jenkins console log for a job named `feature_branch`. The log output shows the execution of a script that sets up a virtual environment, runs integration tests, and concludes with an `OK` status message.

```

+ unset VIRTUAL_ENV
+ unset VIRTUAL_ENV_PROMPT
+ [ ! nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/jenkins_home/workspace/feature_branch/venv
+ export VIRTUAL_ENV
+ _OLD_VIRTUAL_PATH=/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
+
PATH=/var/jenkins_home/workspace/feature_branch/venv/bin:/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/sbin:/usr/bin:/sbin:/bin
+ export PATH
+ [ -n ]
+ [ -z ]
+ _OLD_VIRTUAL_PSL=$
+ PSL=(venv)
+ export PSL
+ VIRTUAL_ENV_PROMPT=(venv)
+ export VIRTUAL_ENV_PROMPT
+ [ -n -o -n ]
[Pipeline] dir
Running in /var/jenkins_home/workspace/feature_branch
[Pipeline] {
[Pipeline] sh
+ /var/jenkins_home/workspace/feature_branch/venv/bin/python integration.py
+ +
-----
Ran 2 tests in 0.020s
OK

```

- If any test case fails during integration testing, it indicates that there is an issue with the interaction between components.
- The testing framework will output information about which specific test case failed, along with the expected and actual results.



The screenshot shows a Jenkins console log for a build named 'feature\_branch' (Build #28). The log output is as follows:

```
+ VIRTUAL_ENV_PROMPT=(venv)
+ export VIRTUAL_ENV_PROMPT
+ [ -n -o -n ]
[Pipeline] dir
Running in /var/jenkins_home/workspace/feature_branch
[Pipeline] {
[Pipeline] sh
+ /var/jenkins_home/workspace/feature_branch/venv/bin/python integration.py
F.
=====
FAIL: test_add_and_view_tasks (__main__.TestIntegration.test_add_and_view_tasks)
-----
Traceback (most recent call last):
  File "/var/jenkins_home/workspace/feature_branch/integration.py", line 20, in test_add_and_view_tasks
    self.assertEqual(response.status_code, 200)
AssertionError: 404 != 200
-----
Ran 2 tests in 0.021s

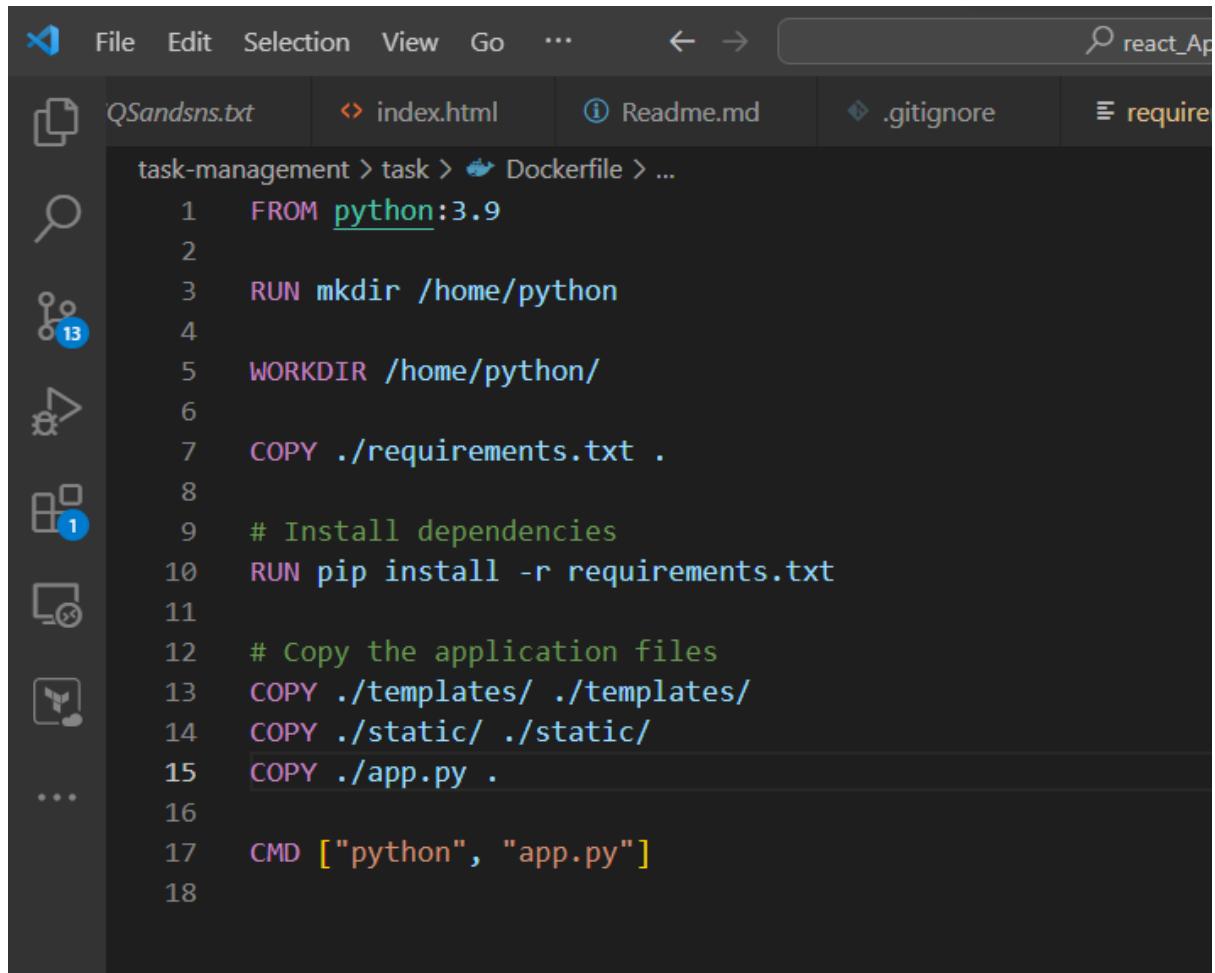
FAILED (failures=1)
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] }
ERROR: script returned exit code 1
```

- The build status will be marked as "Success" or "Stable." This indicates that all the defined stages, including unit and integration tests, have completed successfully.

### **Sub Task 3:**

#### **Dockerfile:**

- Writing a Dockerfile for a web application involves specifying the necessary steps to create a Docker image that can run your web application. Here's a basic example of a Dockerfile for a web application using a Python-based web framework (Flask)



The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with icons for file operations like copy, paste, search, and refresh. The main area displays a Dockerfile with the following content:

```
FROM python:3.9
RUN mkdir /home/python
WORKDIR /home/python/
COPY ./requirements.txt .
# Install dependencies
RUN pip install -r requirements.txt
# Copy the application files
COPY ./templates/ ./templates/
COPY ./static/ ./static/
COPY ./app.py .
CMD ["python", "app.py"]
```

The Dockerfile starts with `FROM python:3.9`, sets the working directory to `/home/python`, copies the `requirements.txt` file, installs dependencies using `pip`, copies the application files (templates and static), and finally runs the application with `CMD ["python", "app.py"]`.

#### **Build docker image and store in docker hub :**

- Jenkinsfile that we can use to build Docker images for different components of a web application (frontend and backend) and push those images to Docker Hub.

```

1  pipeline{
2      agent any
3      environment{
4          image2="ganesh"
5          tag2="latest"
6      }
7      stages{
8          stage("hello")
9          {
10              steps
11              {
12                  echo "checkout to develop branch"
13                  echo "Successfully pulled the changes from the feature branch by reviewing the code through pull request"
14                  echo "Building the docker image with the changes"
15              }
16          }
17          stage("Build the images"){
18              steps{
19                  script{
20                      dir('/var/jenkins_home/workspace/feature_branch')
21

```

```

23          sh "git status"
24          sh "git branch"
25          echo "=====Building docker image ====="
26          echo "adding echo command"
27          withCredentials([usernamePassword(credentialsId: 'dockerhub', passwordVariable: 'PASS',
28          sh "docker build -t ${image2}:${tag2} ."
29          sh '$USER'
30          sh "echo $PASS | docker login -u $USER --password-stdin"
31          sh "docker tag ${image2}:${tag2} $USER/${image2}:${tag2}"
32          sh "docker push $USER/${image2}:${tag2}"
33      }
34      }
35      }
36      }
37      }
38  }
39
40

```

- If all stages in the Jenkinsfile are executed successfully, and there are no errors, the pipeline will be considered a success.

Dashboard > develop > #21

```

15a1784546d1: Layer already exists
5f70bf18a086: Layer already exists
2d788bc47240: Layer already exists
70866f64c03e: Layer already exists
fa83a371445d: Layer already exists
d7b1a4619b1: Layer already exists
86e50e0709ee: Layer already exists
12b956927ba2: Layer already exists
266def75d28e: Layer already exists
29e49b59edda: Layer already exists
1777ac7d307b: Layer already exists
latest: digest: sha256:ad0e12a2c20c52622757afbe30a4433b766a6c2d625d249b953c0a6c73e01012 size: 3459
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // dir
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

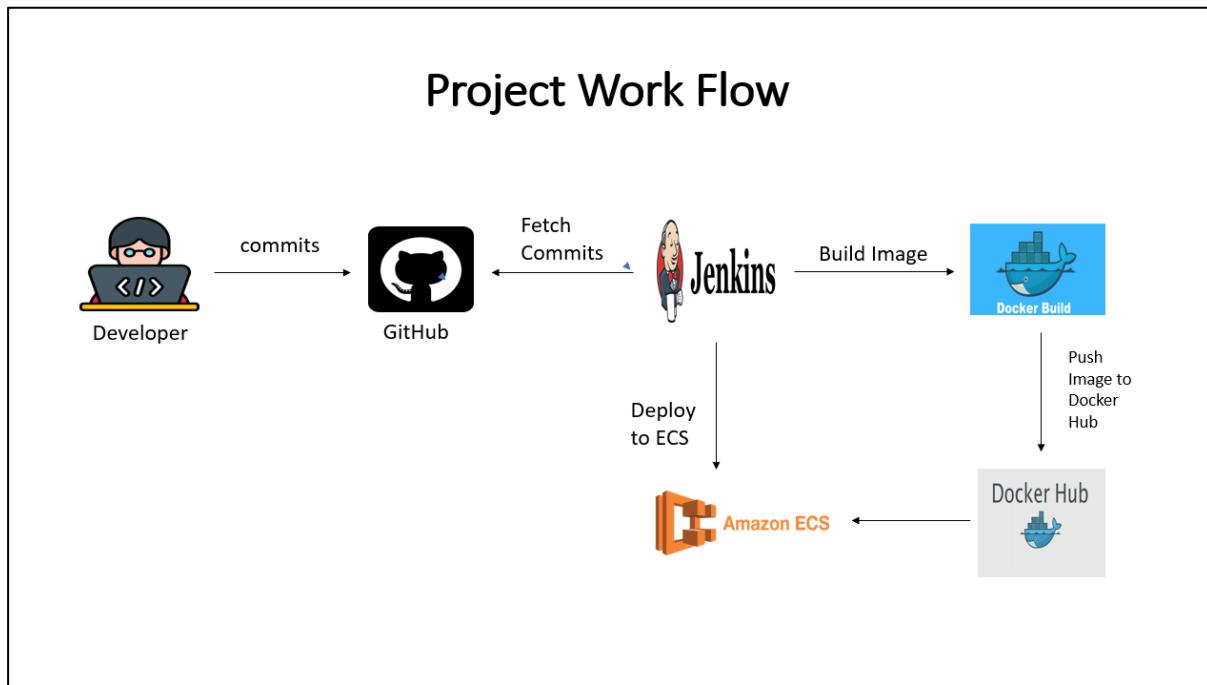
The screenshot shows a web-based CI/CD pipeline interface. At the top, a navigation bar includes links for 'Dashboard', 'develop', 'Status' (which is currently selected), 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', 'Pipeline Syntax', and 'GitHub Hook Log'. The main title is 'develop'. Below the title, there's a 'Stage View' section with three stages: 'Declarative: Checkout SCM' (1s), 'Intro' (79ms), and 'Build the images' (9s). A summary states: 'Average stage times: (Average full run time: ~11s)'. To the left of the stage view is a 'Build History' section showing build #21 from Nov 20 at 15:11, which contained 1 commit. Below the history is a 'Permalinks' section listing recent builds: #21 (Nov 20, 2023, 9:41 AM) and #20.

- If the Docker image is successfully built, the next step in a typical CI/CD pipeline is to push the Docker image to a container registry, such as Docker Hub.

The screenshot shows a Docker Hub repository page for 'shekhar123reddy/ganesh/general'. The top navigation bar includes links for 'Search Docker Hub', 'Explore', 'Repositories', 'Organizations', 'Help', and a user profile for 'shekhar123reddy'. The repository path is 'shekhar123reddy / Repositories / ganesh / General'. A message indicates 'Using 0 of 1 private repositories. [Get more](#)'. The 'General' tab is selected, showing options for 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings'. A note says 'Add a short description for this repository' and 'The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.' An 'Update' button is present. The main content area shows the repository name 'shekhar123reddy / ganesh'. Under 'Description', it says 'This repository does not have a description' with an edit icon. Under 'Docker commands', it says 'To push a new tag to this repository:' followed by a code snippet: 'docker push shekhar123reddy/ganesh:tagname'. There are sections for 'Tags' (showing 1 tag) and 'Automated Builds' (with a note about manually pushing images to Hub). A 'Public View' button is also visible.

## Sub Task 4:

Setting up a Code Pipeline with source (Git), build (Jenkins), and deploy ECS stages.



### Prerequisites:

#### 1. AWS Account:

- Ensure you have an AWS account with the necessary permissions to create and configure AWS resources.

#### 2. Git Repository:

- Have a Git repository containing your application code.

#### 3. Jenkins Server:

- Set up a Jenkins server that can be accessed by AWS Code Pipeline.

#### 4. Docker and ECS:

- Ensure you have Docker installed for building containers.
- Have an ECS cluster, task definition, and service ready for deploying your application.

### Procedure:

#### Step 1: Create IAM Role for Code Pipeline:

In AWS CodePipeline, an IAM (Identity and Access Management) role is used to define permissions for the pipeline to interact with AWS services and resources. This role specifies what actions the pipeline can perform and what resources it can access during its execution. Here are the general steps to create an IAM role for AWS CodePipeline.

## 1. Open the IAM Console:

- Log in to the AWS Management Console and navigate to the IAM service.

## 2. Create a New Role:

- Click on "Roles" in the left navigation pane.
- Click on the "Create role" button.

## 3. Choose the Service that Will Use the Role:

- In the "Select type of trusted entity" section, choose "AWS service" as the type of trusted entity.
- In the "Choose a use case" section, find and select "CodePipeline."

## 4. Attach Permissions Policies:

- In the "Attach permissions policies" step, you can either choose existing policies or attach policies later. Policies define what actions the role can perform. At a minimum, the role should have permissions to access the artifacts stored in Amazon S3 and perform actions on the CodePipeline itself.
- Common policies to attach include `AWSCodePipelineFullAccess` and `AmazonS3ReadOnlyAccess`.

## 5. Review:

- Provide a meaningful name for the role and optionally add a description.
- Review your choices and click "Create role."

The screenshot shows the AWS IAM Roles page. On the left, the navigation pane includes 'Identity and Access Management (IAM)' and sections like 'Access management' (User groups, Users, Roles), 'Policies', 'Identity providers', and 'Account settings'. The main area displays the 'JenkinsRole' details. The 'Summary' tab shows the role was created on November 11, 2023, at 17:21 UTC+05:30, with its ARN as arn:aws:iam:862547479026:role/JenkinsRole. It has a maximum session duration of 1 hour and last activity 16 minutes ago. Below this are tabs for 'Permissions', 'Trust relationships', 'Tags', 'Access Advisor', and 'Revoke sessions'. Under 'Permissions', it shows 'Permissions policies (6) info' with a table of attached policies. The table includes columns for Policy name, Type, and Attached entities. The policies listed are: AmazonEC2FullAccess, AmazonECS\_FullAccess, AmazonS3FullAccess, AWSCodeDeployFullAccess, AWSCodeDeployRoleForECS, and AWSCodePipeline\_FullAccess. Each policy is marked as AWS managed and has 1 attached entity.

## Step 2: Configure Jenkins for Code Pipeline Integration:

### 1. Install the Jenkins AWS CodePipeline plugin

- Click **Manage Jenkins**.
- Select **Manage Plugins**.
- Choose the **Available** tab, then search for the **AWS CodePipeline** plugin in the **Filter** search box.

The screenshot shows the Jenkins Manage Plugins interface. The 'Available' tab is selected. A search bar at the top right contains the text 'AWS CodePipeline'. Below the tabs, there's a table with columns 'Name' and 'Version'. One row in the table is for the 'AWS CodePipeline Plugin', which is described as 'a continuous delivery service for fast and reliable application updates.' The version listed is 0.27. At the bottom of the table, there are three buttons: 'Install without restart' (highlighted in blue), 'Download now and install after restart', and 'Check now'. A status message 'Update information obtained: 1 hr 17 min ago' is displayed below the table.

- Select the plugin, then click **install without restart**

## 2. Create the “JenkinsBuild” Jenkins project:

- Click **New Item**.
- Enter an Item Name: “**JenkinsBuild**”
- Choose **Freestyle Project**, then click **OK**
- In **Source Code Management** section, select the AWS CodePipeline, then configure the values as shown below:

The screenshot shows the Jenkins Configuration page for the 'JenkinsBuild' project. On the left, there's a sidebar with sections like General, Source Code Management (which is selected and highlighted in grey), Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main area is titled 'Source Code Management'. It has a 'Configure' section with a 'None' radio button and an 'AWS CodePipeline' radio button (which is selected). Under 'AWS CodePipeline', there are fields for 'AWS Config' (selected), 'AWS Region' (set to 'ap-south-1'), 'Proxy Host' (empty), 'Proxy Port' (empty), and 'Credentials'. The 'Credentials' section notes that if keys are left blank, the plugin will attempt to use default provider chain. It shows 'AWS Access Key' (set to 'AKIA4RU6WBHZANTW3G45') and 'AWS Secret Key' (redacted with dots). A checkbox 'Clear workspace before copying' is checked. At the bottom of the configuration page, there's a 'Save' button.

- In the **Build Triggers** section, select **Poll SCM** and configure the schedule to poll the SCM as often as you want, for example every minute.

Dashboard > JenkinsBuild > Configuration

### Configure

- General
- Source Code Management
- Build Triggers**
- Build Environment
- Build Steps
- Post-build Actions

**CodePipeline Action Type** This value must match the Category field that is on the Custom Action in your corresponding Pipeline.

**Category**

**Provider**

**Version**

Git

**Build Triggers**

- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM
- Schedule

- In **Build Environment** section, select the “Use secret text(s) or file(s)”

Dashboard > shekhar\_reddy > Configuration

### Configure

- General
- Source Code Management
- Build Environment**
- Build Steps
- Post-build Actions

Delete workspace before build starts

Use secret text(s) or file(s)

**Bindings**

**Username and password (separated)**

Username Variable

Password Variable

Credentials Specific credentials Parameter expression

Specific credentials  Parameter expression  
shekhar123reddy/\*\*\*\*\*

+ Add

Add timestamps to the Console Output

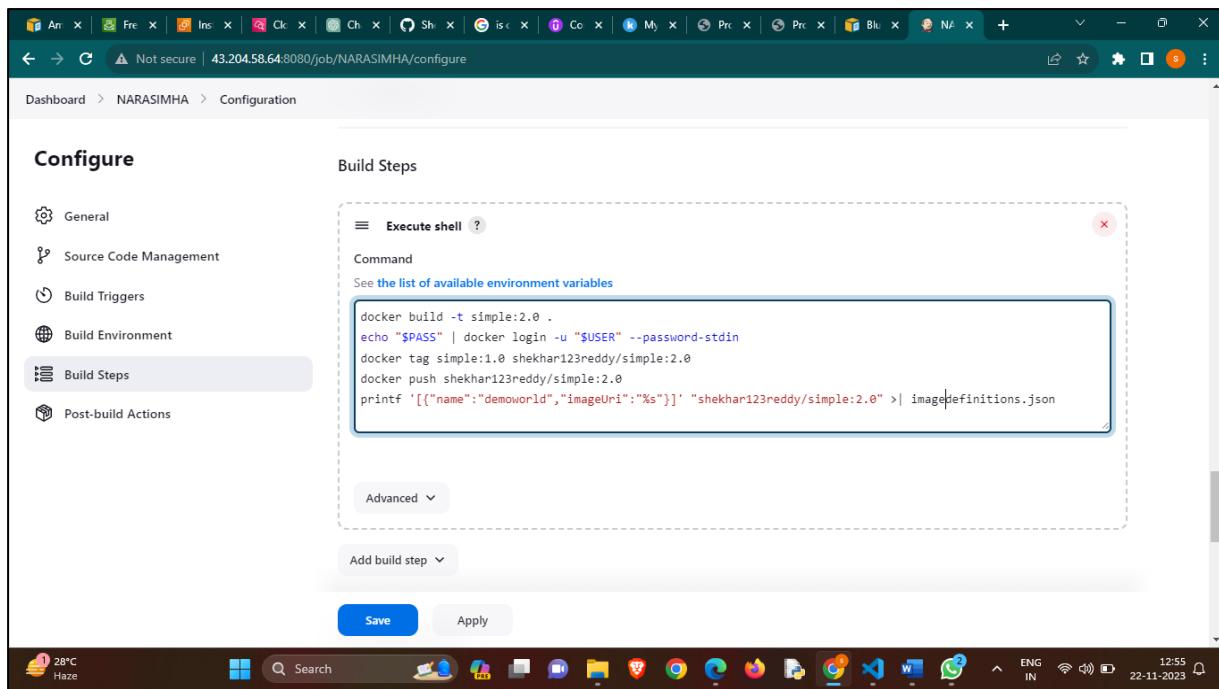
Inspect build log for published build scans

Terminate a build if it's stuck

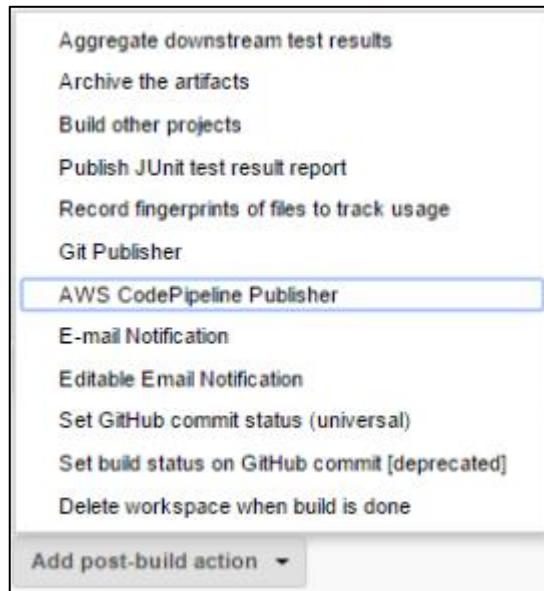
With Ant

**Build Steps**

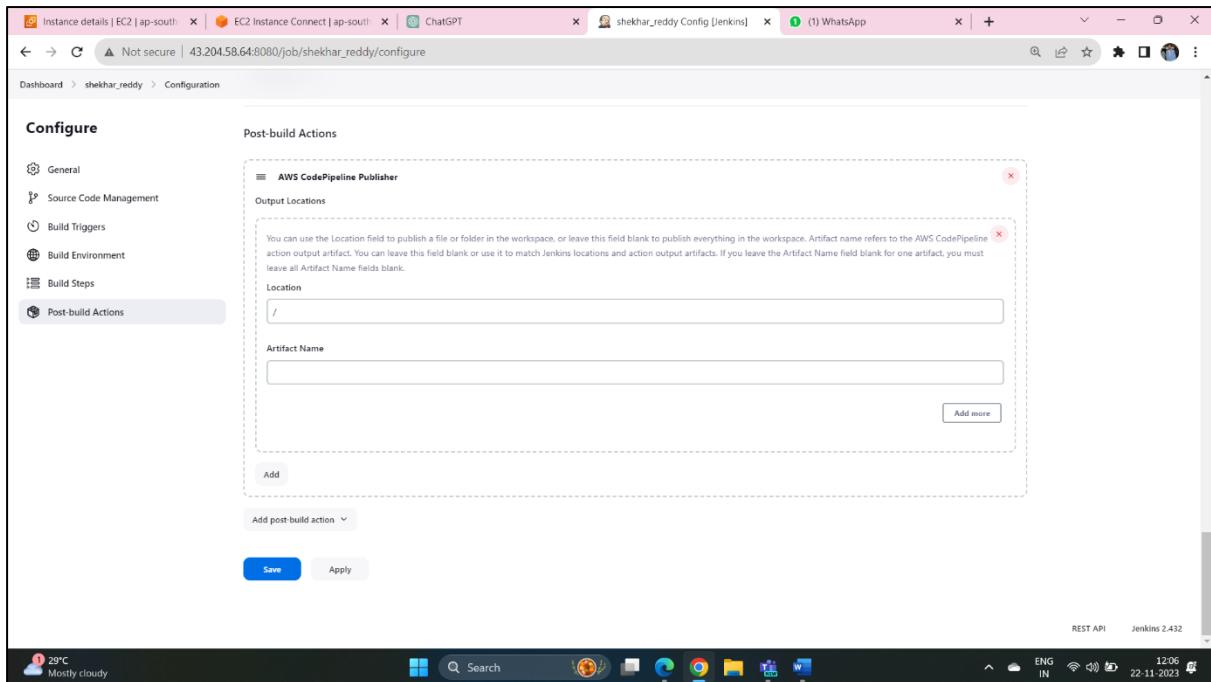
- In **Build steps** section, select add Execute shell commands.



- On the **Post-build Actions** tab, add a post-build action.
- Choose **AWS CodePipeline Publisher**.



- Do not add any output locations.



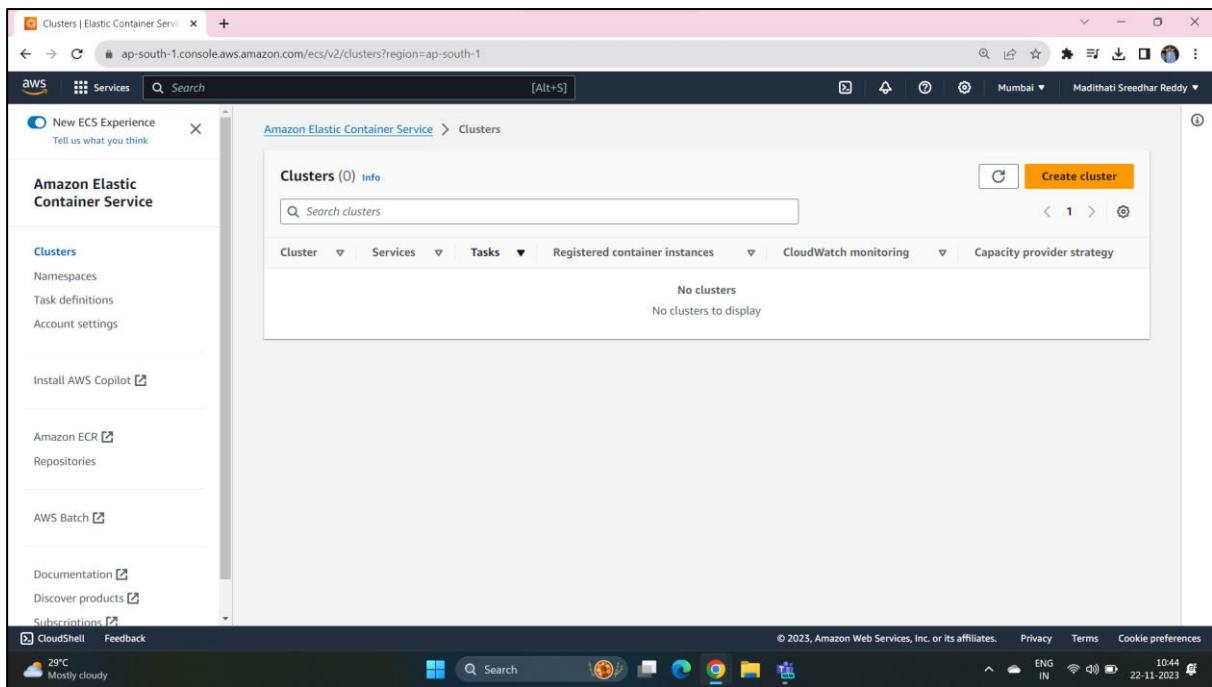
- Click **Save**.

### Step-3: Creating an ECS Cluster, Task definition, Service in AWS:

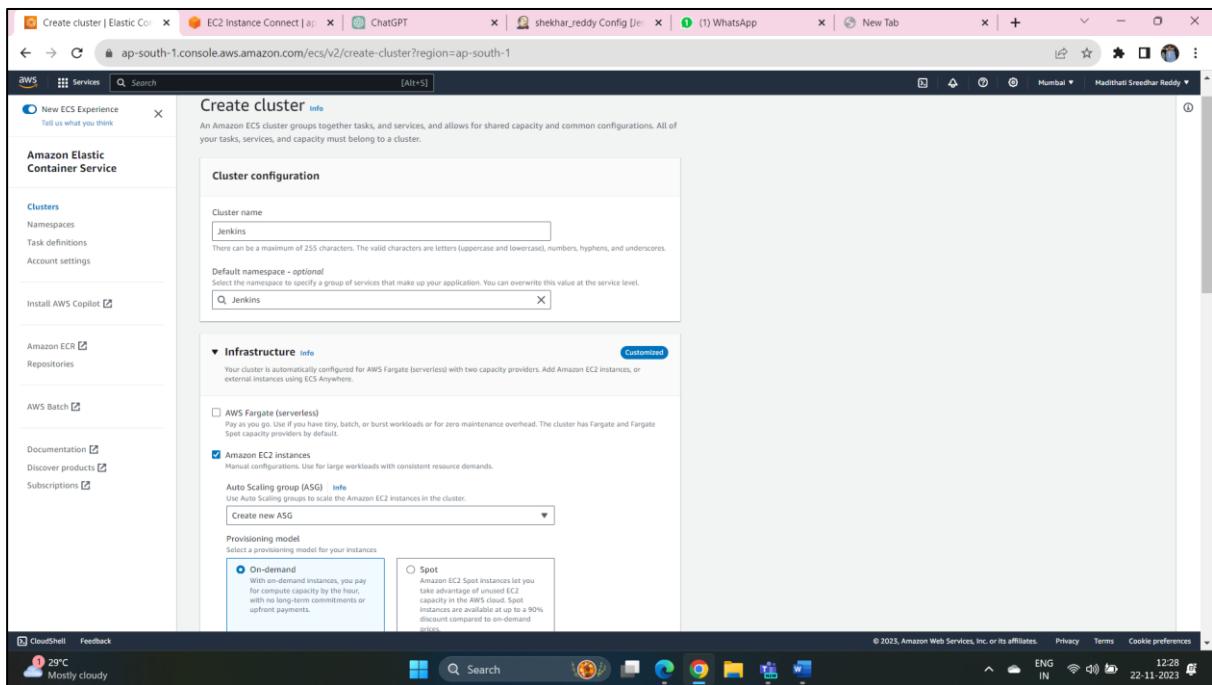
#### 1. Creating ECS Cluster:

Creating an Amazon ECS (Elastic Container Service) cluster involves several steps. ECS is a fully managed container orchestration service that allows you to run, stop, and manage Docker containers on a cluster. Here are the general steps to create an ECS cluster:

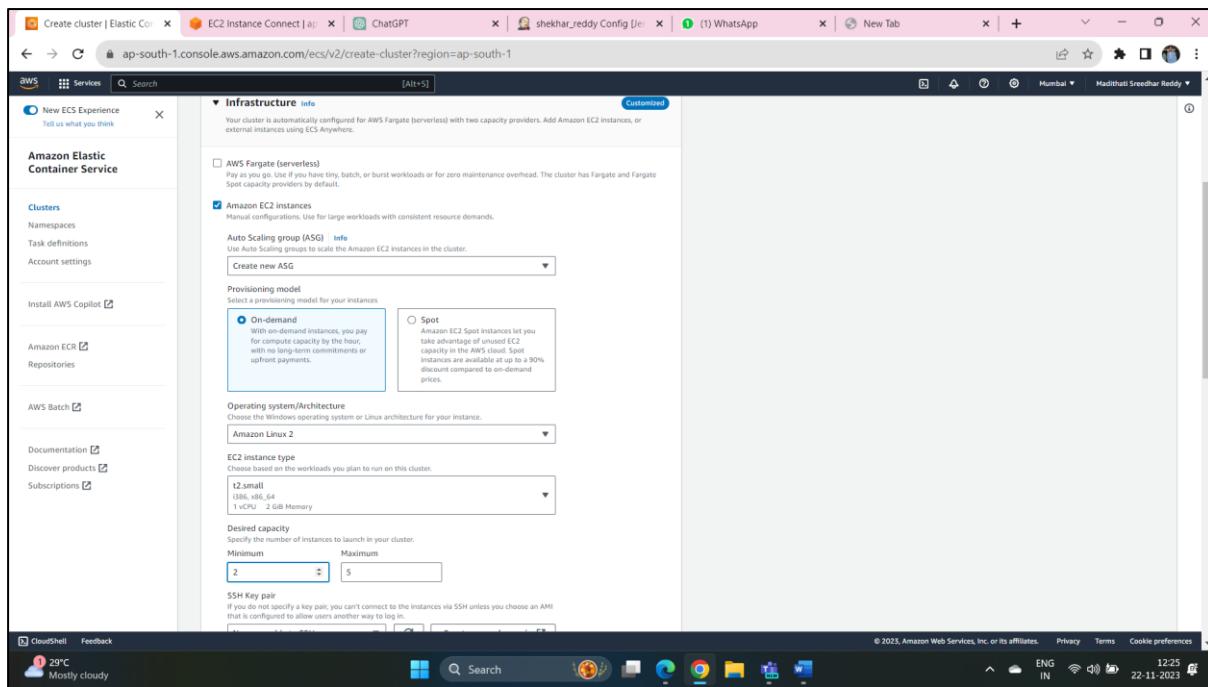
- **Open the AWS Console >>>** Navigate to the Amazon ECS console.
- **Create a Cluster**
- In the ECS console, click on "Clusters" in the left navigation pane.
- Click the "Create Cluster" button.



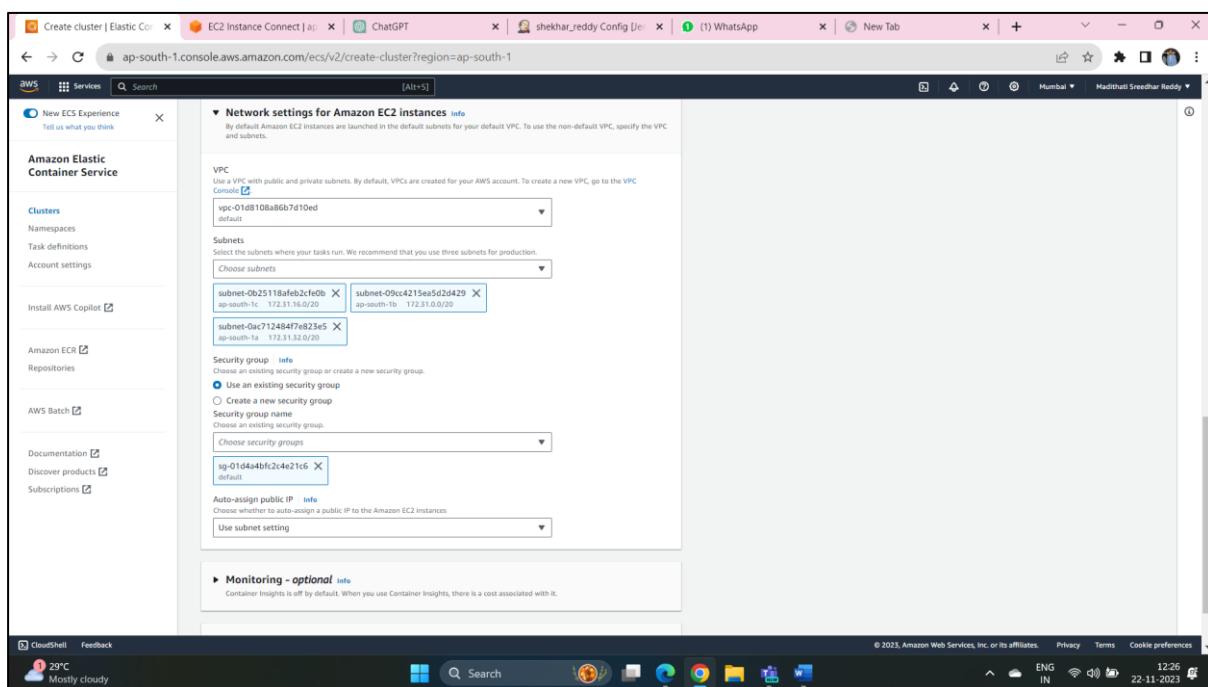
- Configure the cluster settings:
- Cluster Name:** Give your cluster a unique name.



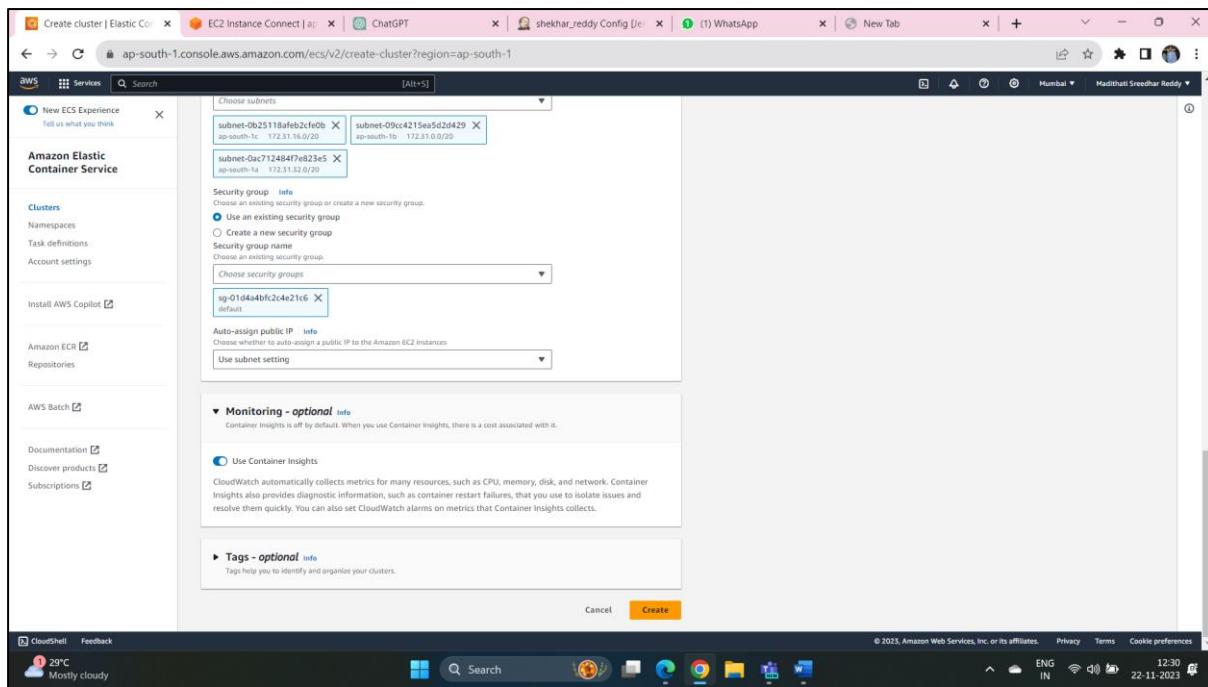
- Choose a cluster Infrastructure based on your requirements:
  - Amazon EC2 Instance:** For EC2 instances using Linux and manual networking.
  - Fargate:** For serverless containers without the need to manage EC2 instances.
- Create an EC2 Instance based on your requirements.



- Number of Instances (if using EC2 instances): Set the initial number of instances in your cluster.
- VPC, Subnets, and other networking settings.



- Click on Monitoring and then turn on it for the CloudWatch Monitoring



- Click "Create" to create the cluster.

### Creating ECS Task definition:

Creating an Amazon ECS (Elastic Container Service) Task Definition is a crucial step before launching containers on ECS. A task definition is a blueprint for your application, specifying various parameters such as the Docker image, CPU and memory requirements, environment variables, networking information, and more. Here how you can create an ECS Task Definition.

- Open the AWS Management Console and navigate to the ECS service.
- In the ECS console, click on "Task Definitions" in the left navigation pane.

The screenshot shows the AWS Elastic Container Service (ECS) Task definitions page. On the left, there's a sidebar with options like Clusters, Namespaces, Task definitions (which is selected), Account settings, and links to ECR, Batch, Documentation, Discover products, and Subscriptions. The main area displays a table of task definitions with columns for Task definition name, Status of last revision, and Action buttons. The task definitions listed are NARASIMHA, TaskApp, ganesh-1, and ramesh-1, all marked as ACTIVE. At the top right, there are buttons for Create new revision, Create new task definition (which is highlighted in orange), and Create new task definition with JSON.

- Click the "Create new Task Definition" button.
- Specify a unique task definition Family name.
- Choose whether your task definition is compatible with EC2 instances, Fargate.

The screenshot shows the 'Create task definition' wizard, step 1: Task definition family. The sidebar is identical to the previous screenshot. The main area has a text input field for 'Task definition family' containing 'Jenkins'. Below it, there's a section titled 'Infrastructure requirements' with a 'Launch type' dropdown set to 'AWS Fargate' (unchecked) and 'Amazon EC2 instances' (checked). Other settings include 'OS, Architecture, Network mode' (Network mode selected), 'Operating system/Architecture' (Linux/X86\_64), 'Network mode' (awsvpc), and 'Task size' (1 vCPU, 1 GB Memory).

- Select Network mode as Default
- Optionally, specify an IAM role for your task execution role.

**Provide details for your container, including:**

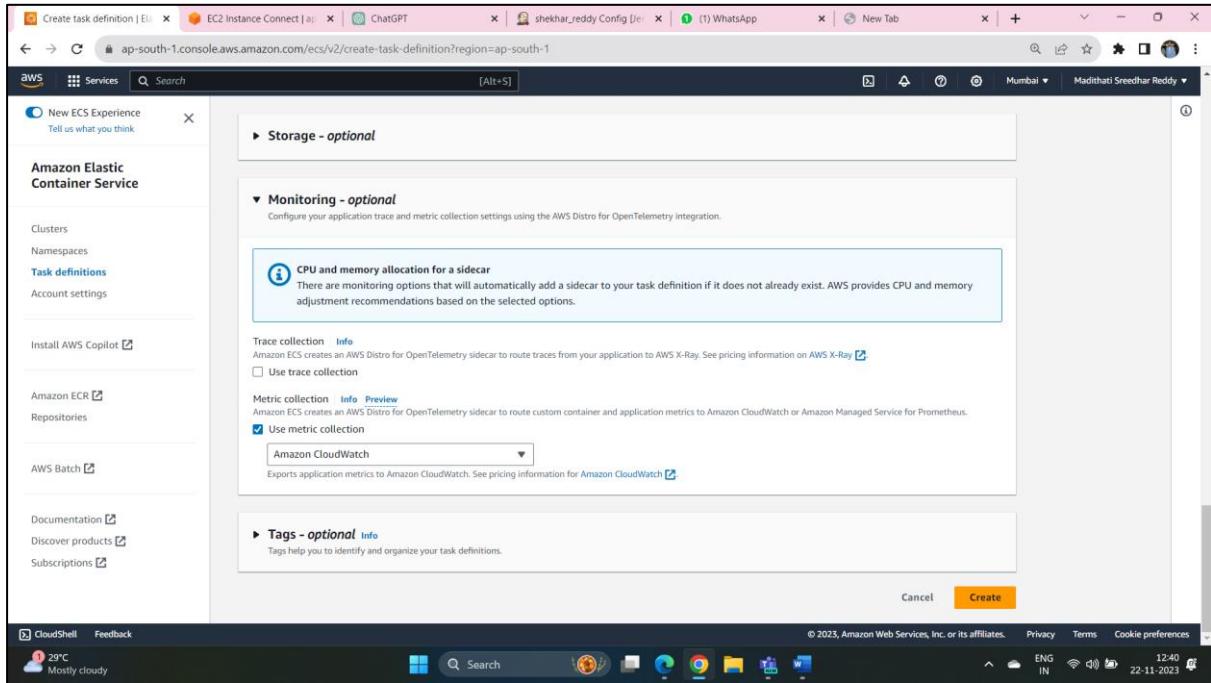
- **Container Name:** A unique name for your container.
- **Image URI:** The Docker image to use.
- **Port Mappings:** Give Host Port and Container Port

The screenshot shows the AWS Cloud Console interface for creating a task definition. The left sidebar lists various services like Clusters, Namespaces, Task definitions, and others. The main area is titled 'Create task definition' and shows the configuration for a task named 'Container - 1'. Key fields include the container name 'ganesh', the image URI 'shekhar123reddy/ganesh:latest', and the setting that it is an 'Essential container'. Under 'Port mappings', a single mapping is defined from host port 5004 to container port 5004, using both TCP and HTTP protocols. The AWS navigation bar at the bottom includes CloudShell, Feedback, and various status icons.

- **Environment:** Set environment variables if needed
- Configure Logging, Health Check, Docker configuration and resource limits depends on your need.

This screenshot continues the task definition creation process. The 'Task definitions' section is now expanded, revealing several optional configuration sections: 'Add environment variable', 'Add environment file', and a list of optional features: Logging, HealthCheck, Container timeouts, Container network settings, Docker configuration, Resource limits (Ulimits), Docker labels, and Storage. The 'Storage' option is currently highlighted. Below these, a '+ Add container' button is visible. The AWS navigation bar at the bottom remains consistent with the previous screenshot.

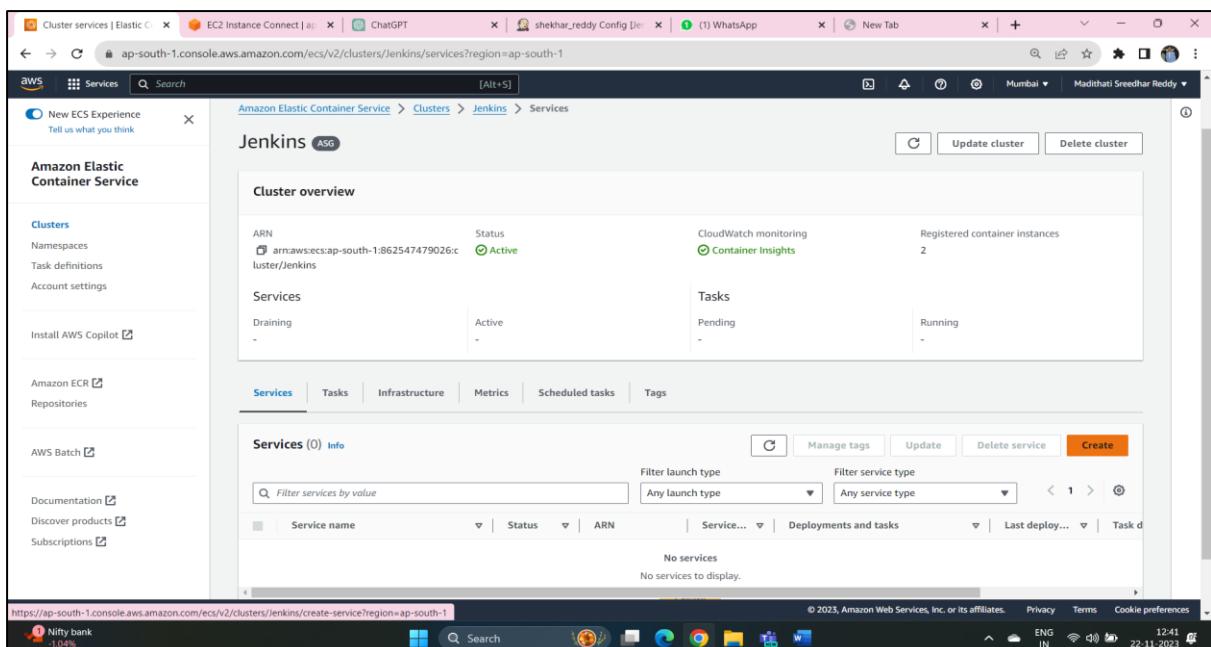
- Set up CloudWatch to monitor key metrics during deployments. For ECS, common metrics include CPU utilization, memory utilization, and the number of running tasks.



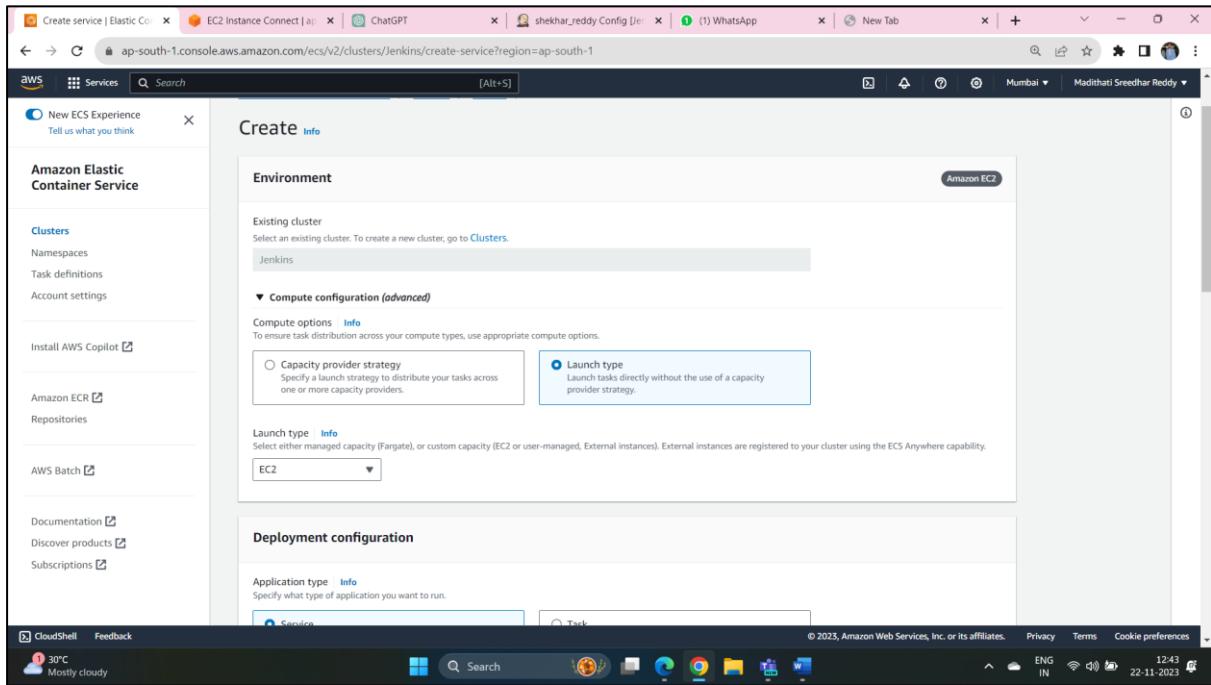
- Review the task definition configuration.
- Click the "Create" button to create the task definition.

### Creating an ECS (Elastic Container Service) service in AWS:

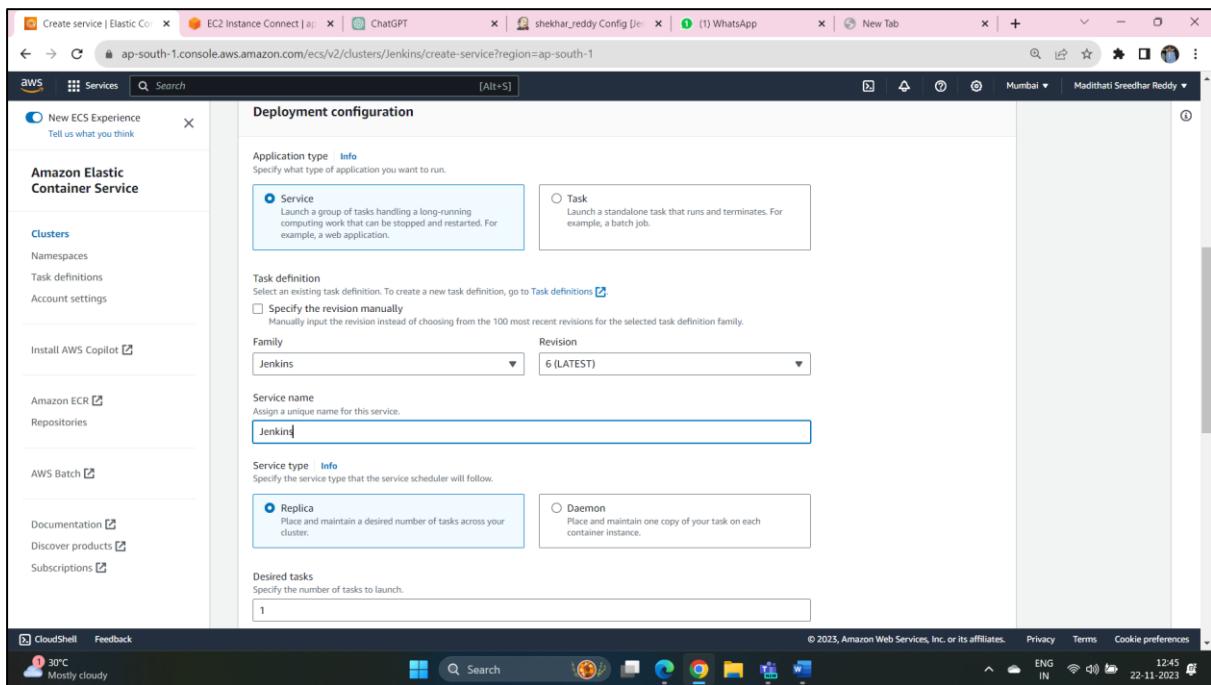
- Open the AWS Management Console and navigate to the ECS service.
- In the ECS console, click on "Clusters" in the left navigation pane.
- Click on the name of the ECS cluster where you want to create the service.
- In the cluster details page, click the "Create" button next to "Services."



- Select the Configuration.



- Configure as below.



In Amazon ECS (Elastic Container Service), there are two primary deployment types:

### Blue/Green Deployment:

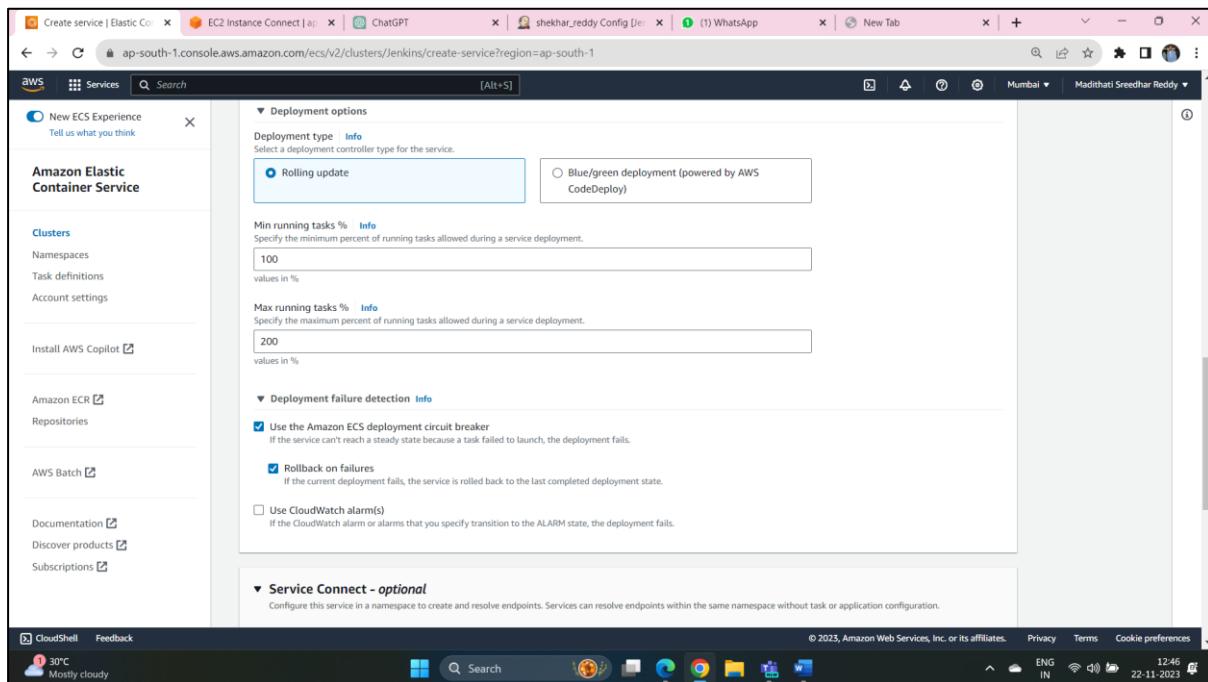
Blue/Green Deployment is Both the old ("blue") and new ("green") versions of the application are deployed in parallel. Provides minimal downtime during the deployment process. Enables thorough testing and validation of the new version before switching traffic. Allows for a quick rollback to the previous version if issues are discovered.

### Rolling Deployment:

Updates are applied incrementally to a subset of tasks while maintaining a specified number of healthy tasks. Provides a continuous deployment process, gradually updating tasks without downtime. Allows for a controlled rollout of the new version across the ECS cluster. Optimizes resource usage by replacing tasks gradually.

Rollback procedures in ECS involve reverting to a previous version of your application or infrastructure in case of issues.

- Here use the Rolling Update Option by configuring as below.



## Create an Application Load Balancer (ALB)

### 1. Navigate to the EC2 Console:

- Go to the EC2 Dashboard.

### 2. Create a Load Balancer:

- Click on "Load Balancers" in the left sidebar.
- Click "Create Load Balancer."
- Choose "Application Load Balancer."

The screenshot shows the AWS EC2 Load Balancers page. On the left, there's a navigation sidebar with various services like Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, Load Balancing, Load Balancers, Target Groups, Auto Scaling, and Auto Scaling Groups. The 'Load Balancers' section is selected. The main area displays a table titled 'Load balancers (1)'. The table has columns for Name, DNS name, State, VPC ID, and Availability Zones. One row is shown for a load balancer named '5004' with the state 'Active', VPC ID 'vpc-01d8108a86b7d1...', and 2 Availability Zones. An 'Actions' dropdown menu is open, showing options: 'Create load balancer' (highlighted in orange), 'Compare load balancer types', 'Create Application Load Balance' (highlighted in blue), 'Create Network Load Balancer', 'Create Gateway Load Balancer', and 'Create Classic Load Balancer'. At the bottom of the main area, a modal window says '0 load balancers selected' and 'Select a load balancer above.' The bottom of the screen shows the Windows taskbar with various pinned icons.

- Give a name for it.

The screenshot shows the 'Create Application Load Balancer' wizard. The first step is 'Basic configuration'. It asks for a 'Load balancer name' which must be unique within the account. A text input field contains '5004'. Below it, a note says 'A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.' Under 'Scheme', the 'Internet-facing' option is selected, with a note: 'An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet.' The 'Internal' option is also available with its own note. Under 'IP address type', 'IPv4' is selected, with a note: 'Recommended for internal load balancers.' The 'Dualstack' option is also listed. The bottom of the screen shows the Windows taskbar.

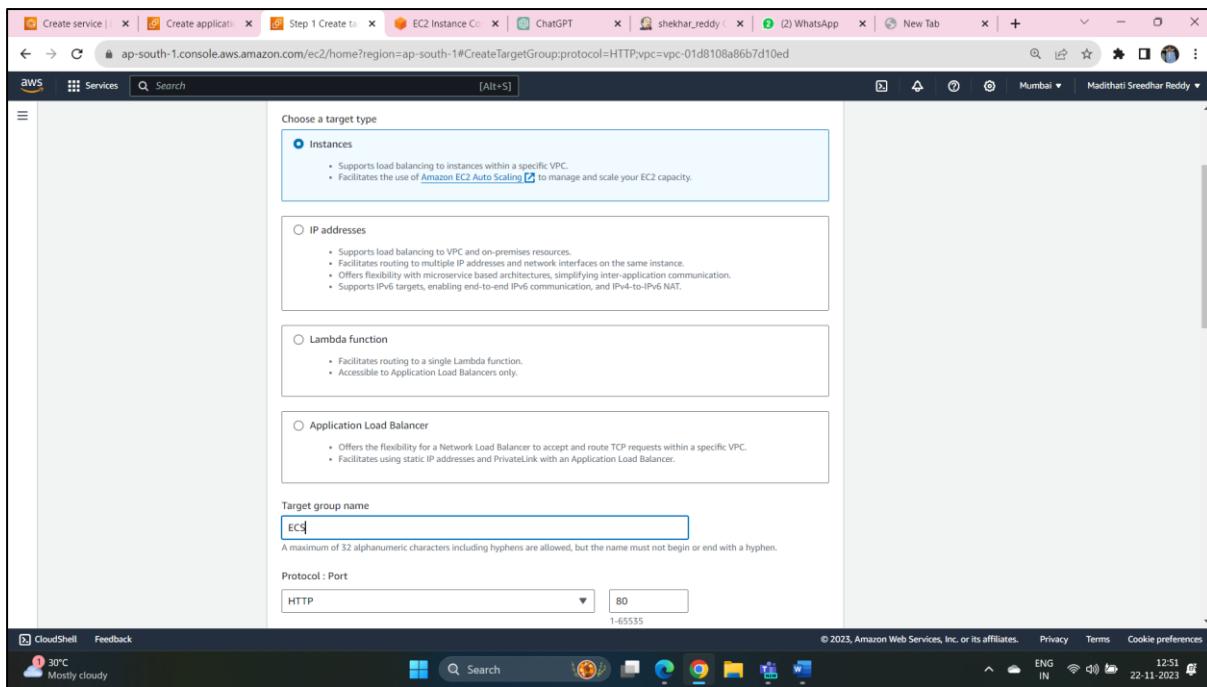
- Configure the load balancer settings (VPC, Subnets, Security Groups).

The screenshot shows the 'Network mapping' step of the ALB Wizard. It displays a VPC selection dropdown containing 'vpc-01d8108a86b7d10ed' with 'IPv4: 172.31.0.0/16'. Below this, three subnets are mapped to Availability Zones: 'ap-south-1a (aps1-az1)', 'ap-south-1b (aps1-az3)', and 'ap-south-1c (aps1-az2)'. Each mapping includes a dropdown for selecting a subnet and an 'IPv4 address' field showing 'Assigned by AWS'. The bottom of the screen shows the AWS navigation bar and a Windows taskbar.

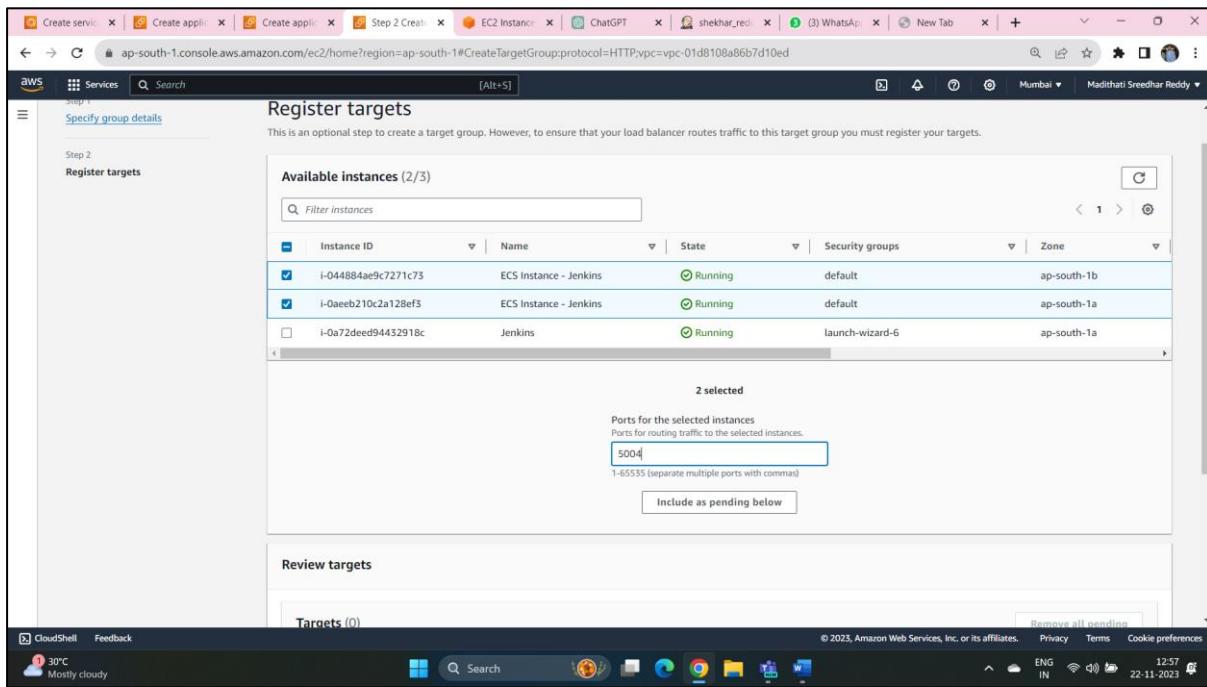
- Configure routing by adding listeners and target groups.

The screenshot shows the 'Security groups' and 'Listeners and routing' steps of the ALB Wizard. In the security group section, a single security group 'default' is selected from a dropdown. In the 'Listeners and routing' section, a new listener for 'HTTP:80' is being configured. The 'Protocol' is set to 'HTTP' and the 'Port' is '80'. The 'Default action' dropdown is set to 'Forward to' with a placeholder 'Select a target group'. A blue link 'Create target group' is visible next to the dropdown. Below the listener configuration, there is a section for 'Listener tags - optional' with a button 'Add listener tag'.

- For this Create a Target Group by clicking on the Blue Option present there



- Next Choose the targets as the registered instances of the created cluster.



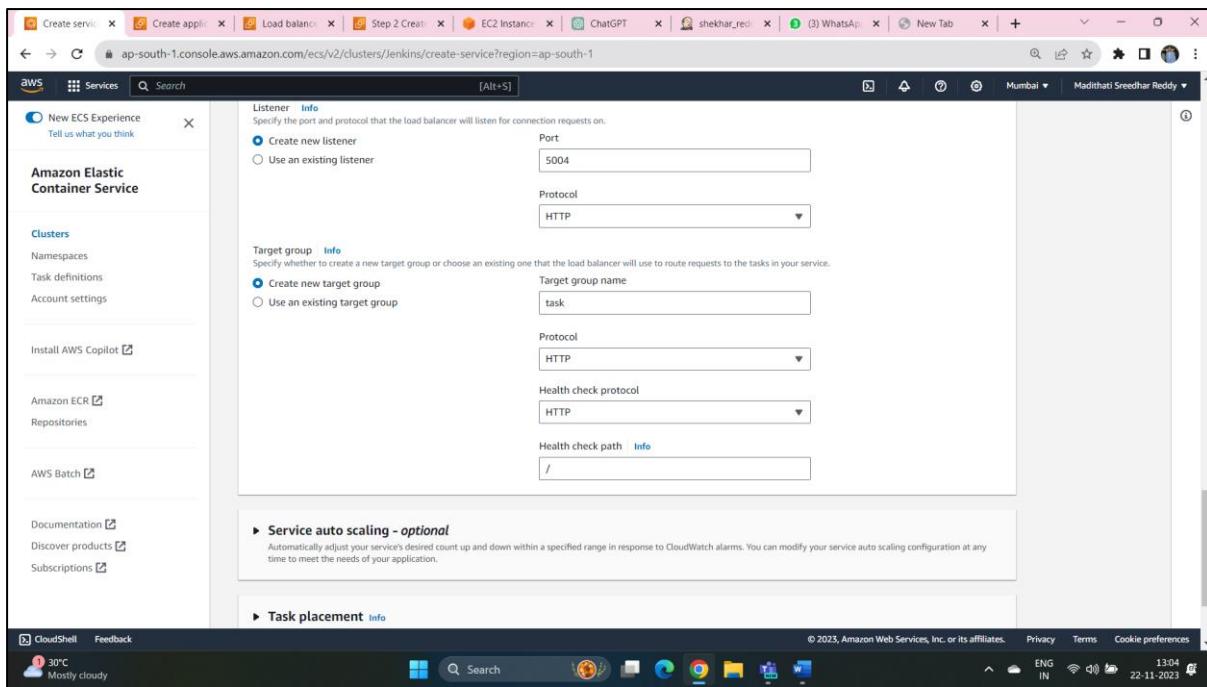
- Then return to the LoadBalancer and assign the created Register target.

The screenshot shows the AWS Lambda Step Functions console. A state machine named 'Create service' is displayed with three states: 'Step 1 Create Service', 'Step 2 Create Load Balancer', and 'Step 3 Create API'. The 'Step 2 Create Load Balancer' state is highlighted. The configuration for this step includes an IAM role 'arn:aws:lambda:ap-south-1:916444444444:role/lambdaBasicExecutionRole' and a CloudWatch Metrics destination 'arn:aws:logs:ap-south-1:916444444444:log-group:/aws/lambda/CreateService'. The 'Step 2 Create Load Balancer' state has a duration of 10 seconds.

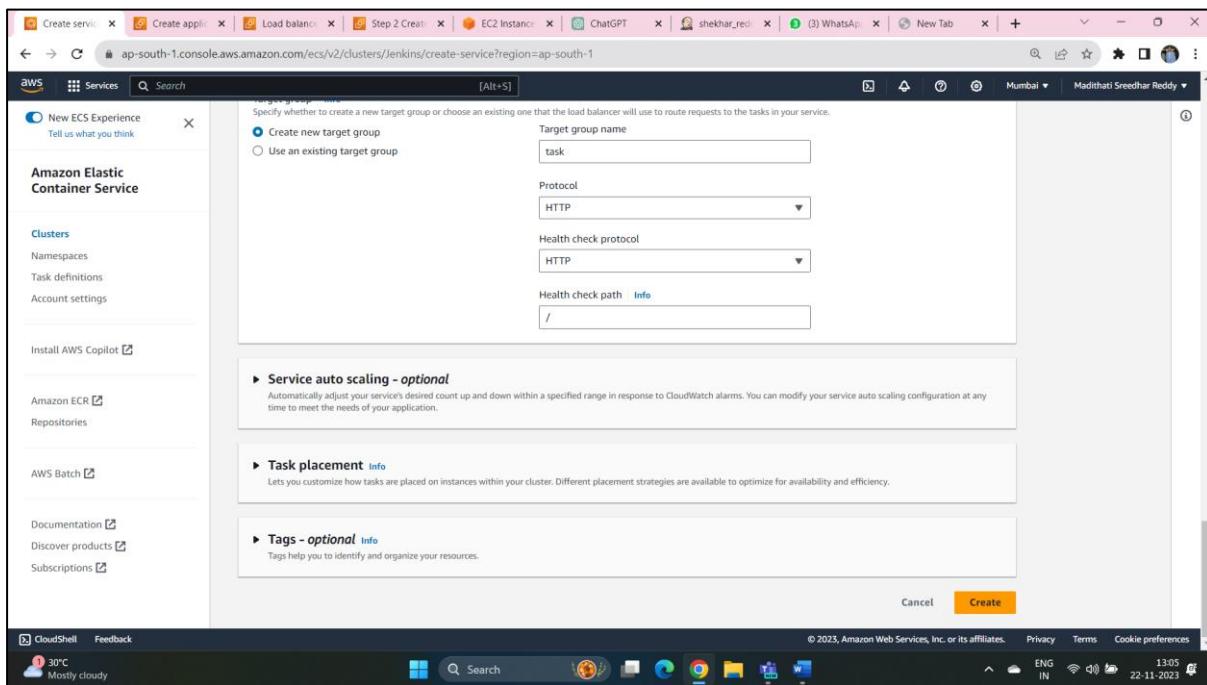
- Give that LoadBalancer under the LoadBalancing Section.

The screenshot shows the AWS ECS Cluster creation wizard. In the 'Load balancing - optional' section, the 'Load balancer type' dropdown is set to 'Application Load Balancer'. Below it, the 'Load balancer' dropdown is set to 'ECS'. The 'Health check grace period' is set to 0 seconds. Under the 'Container' section, the 'Choose container to load balance' dropdown is set to 'ganesh 5004:5004'. In the 'Listener' section, the 'Port' is set to 5004 and the 'Protocol' is set to HTTP. The 'Create new listener' radio button is selected.

- Configure other Options as Below.



- Leave the Other Options as Optional.



- Review all configurations.
- Click the "Create" button to create the ECS service.
- Once created, you can view the details of your service, including task status, events, and more
- If Everything goes right Once you open the cluster, It will Look As below.

**Cluster overview**

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:ap-south-1:862547479026:custer/Jenkins	Active	Container Insights	2

**Services**

Draining	Active	Pending	Running
-	1	-	1

**Services (1) Info**

Service name	Status	ARN	Service type	Deployments and tasks	Last deploy...	Task d...
JenkinsJobs	Active	arn:aws:ec...	REPLICA	1/1 Tasks ru...	In progress	Jenkin...

## Create CodePipeline:

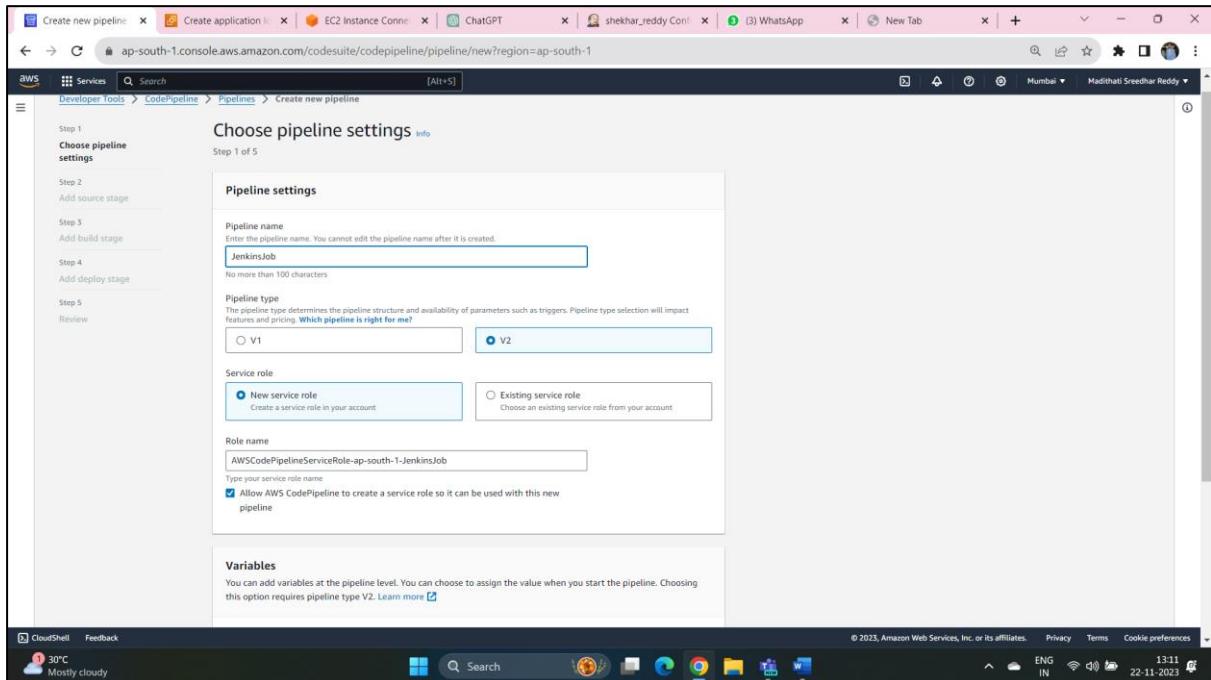
- In the AWS Management Console, navigate to CodePipeline.
- Click "Create pipeline."

**Pipelines**

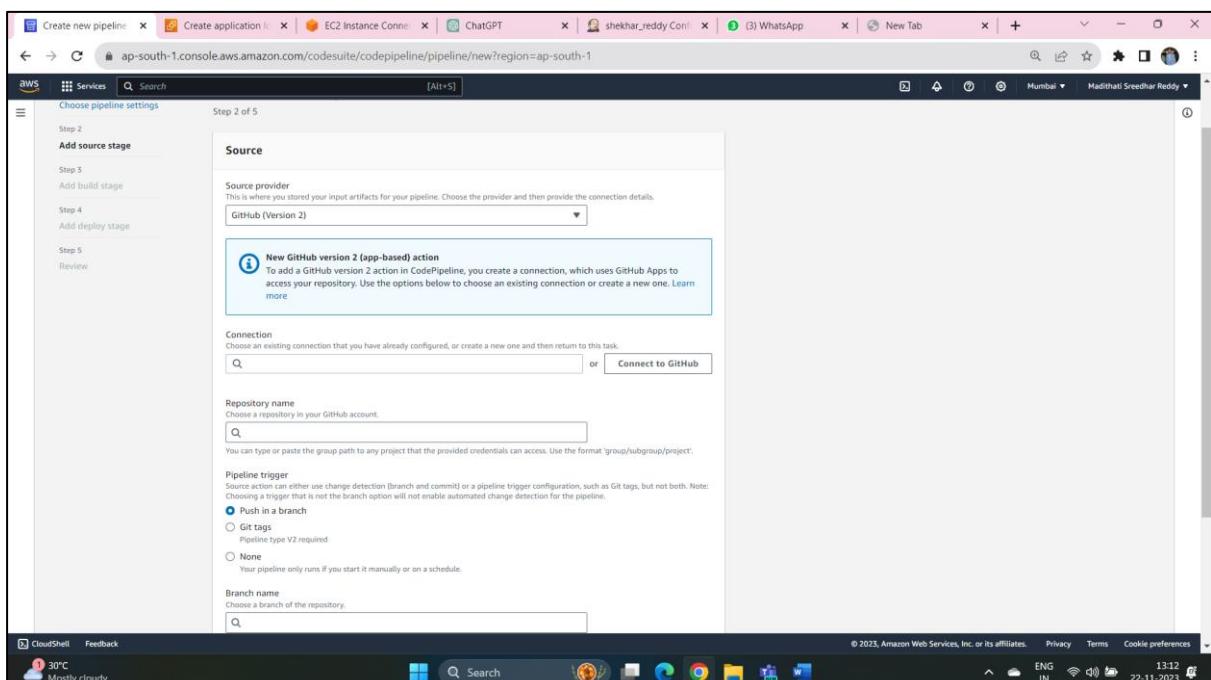
Name	Type	Most recent execution	Latest source revisions	Last executed
NARASIMHA8	V2	Failed	Source - 6d08324c: remove debug	15 hours ago
Jenkins	V2	Failed	Source - a04a05dc: Merge pull request #3 from ShekharRedd/feature	1 hour ago
TaskWebApp	V2	In progress	Source - e173df9c: as	19 hours ago

- Configure your pipeline with the following stages:
  - **Source Stage:** Connect to your “GitHub” repository.
  - **Build Stage:** Use “Jenkins” as the build provider.

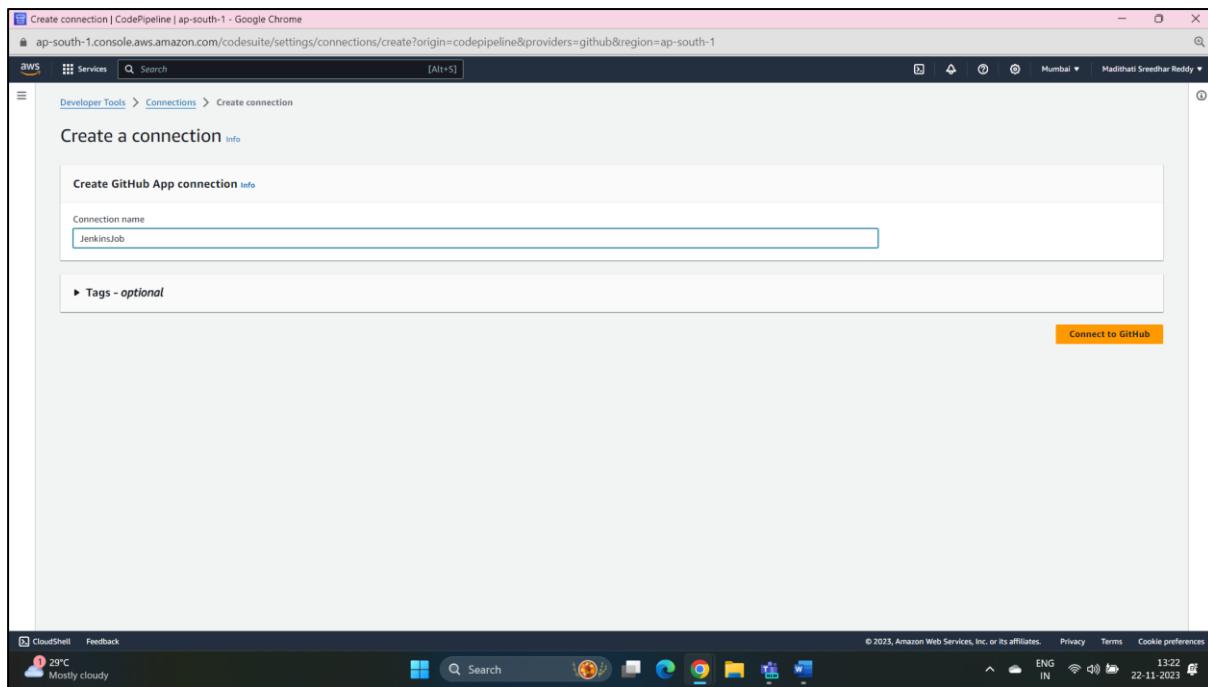
- **Deploy Stage:** Choose “ECS” as the deployment provider.
- Enter the Unique Pipeline name.



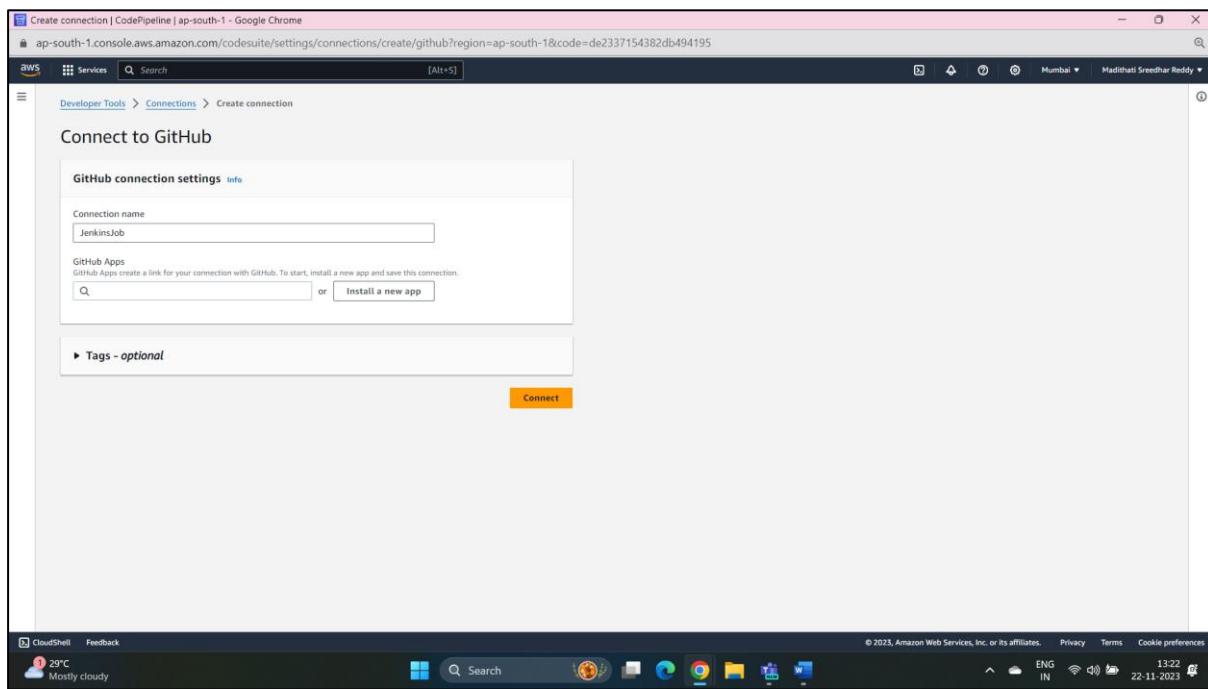
- Click next
- Choose the source provider (e.g., GitHub)



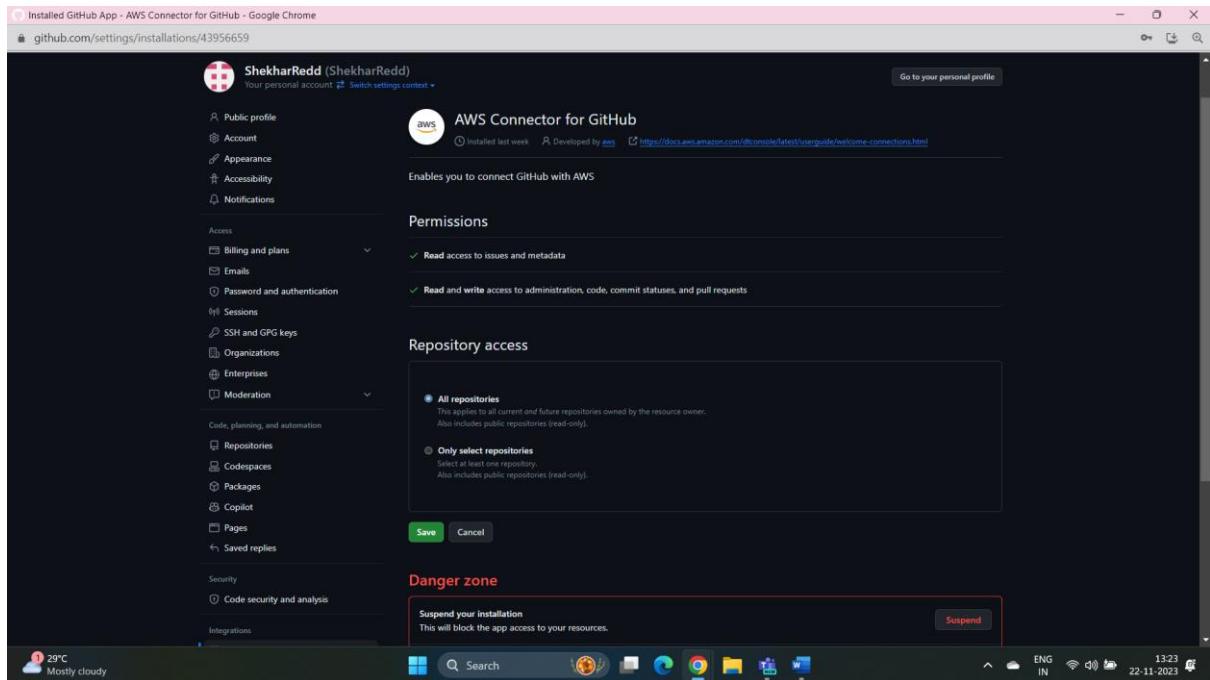
- Click on “Connect to GitHub”



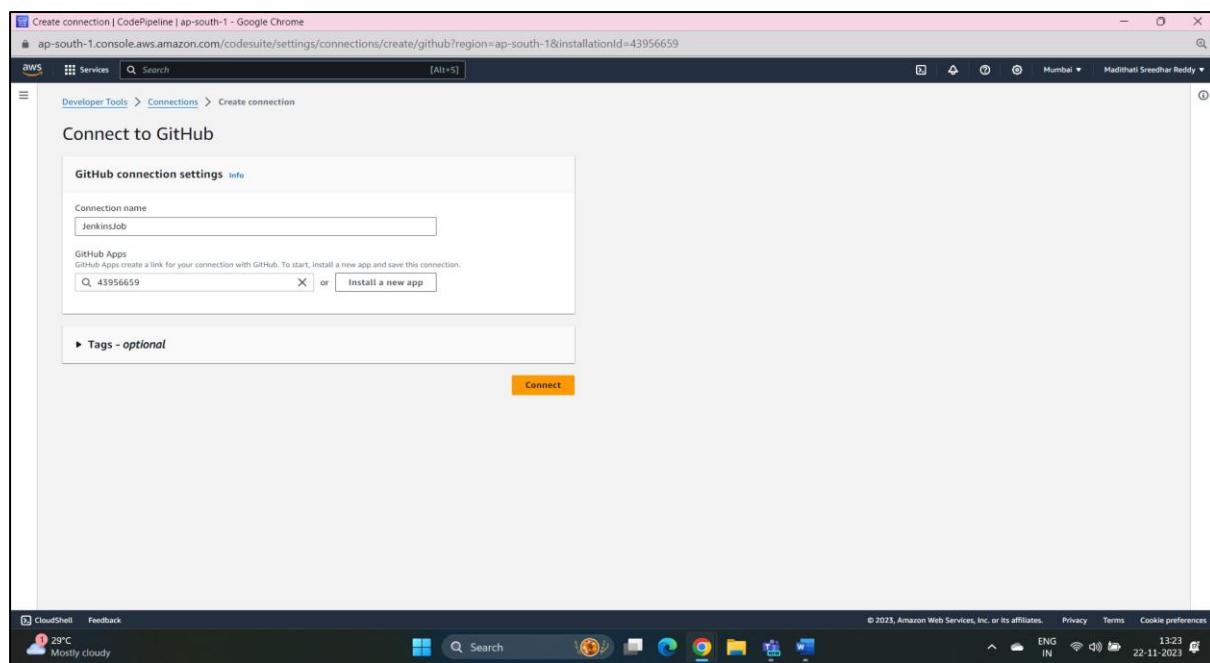
- Click on the tab mentioned there.



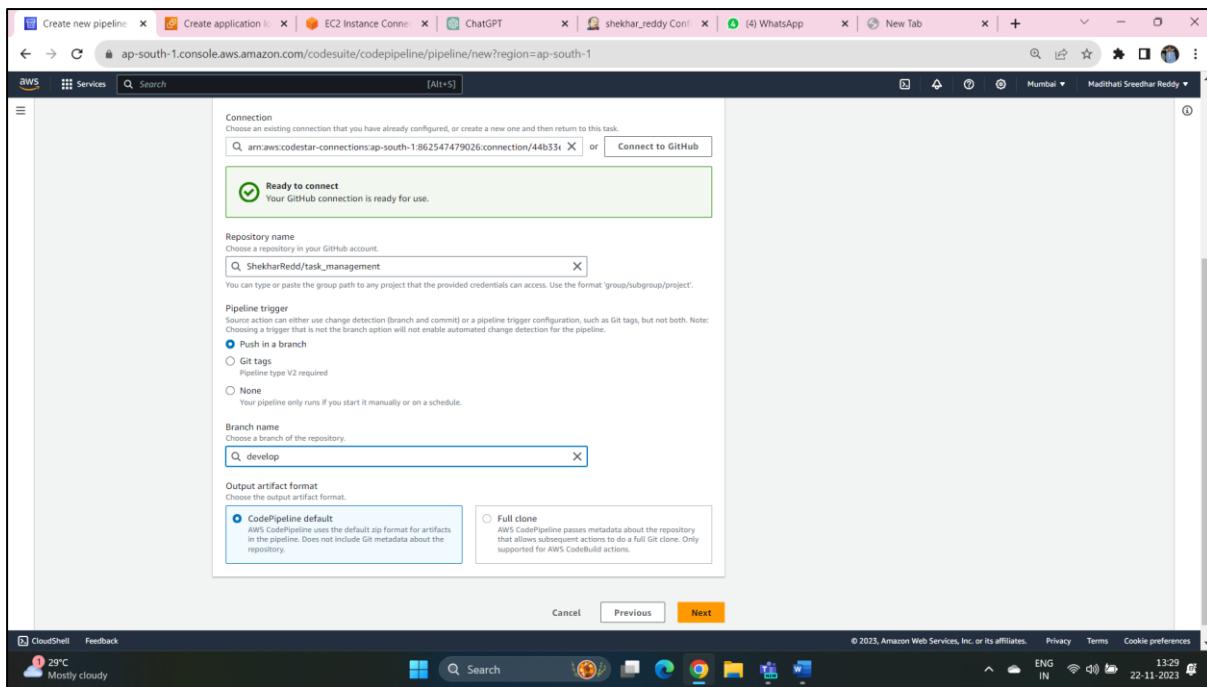
- Click on Install New App
- Next Enter the credentials of your GitHub
- Then Choose the Repo where your SourceCode relies.



- Click on save and then return to the AWS Console.



- Connect to your repository and select the branch to monitor.

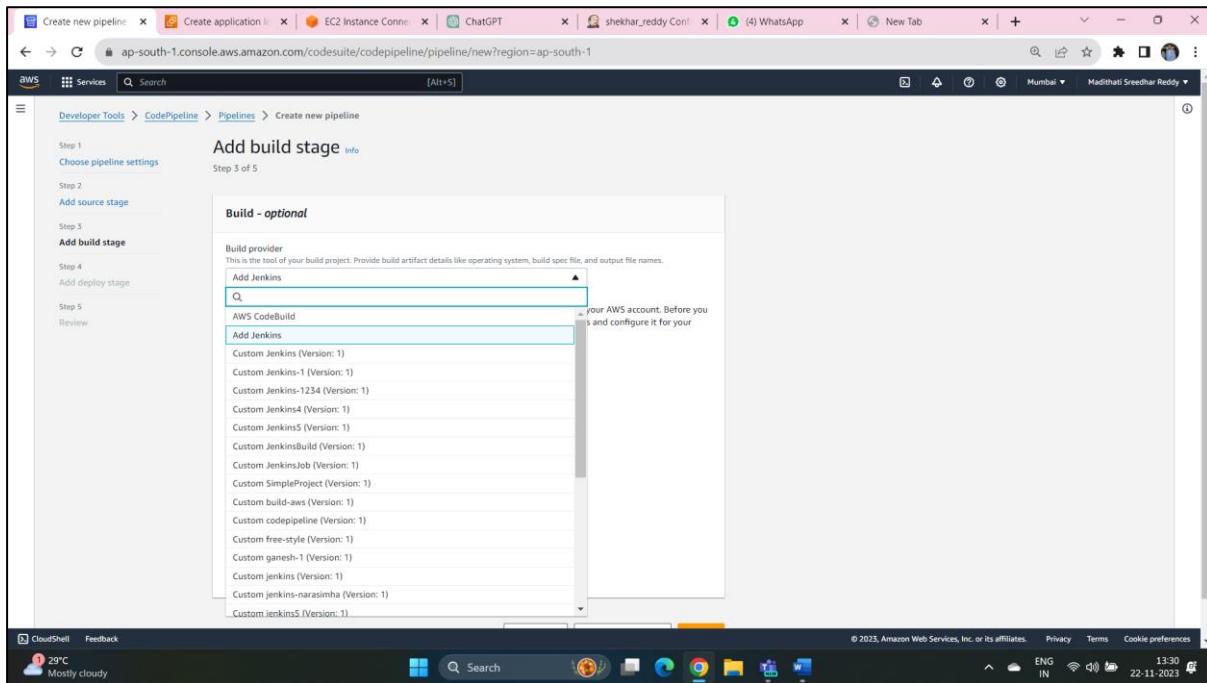


- Click on next

### Configure Build Stage :

Configure the build settings

- Build Provider: Choose Build provider (Jenkins)

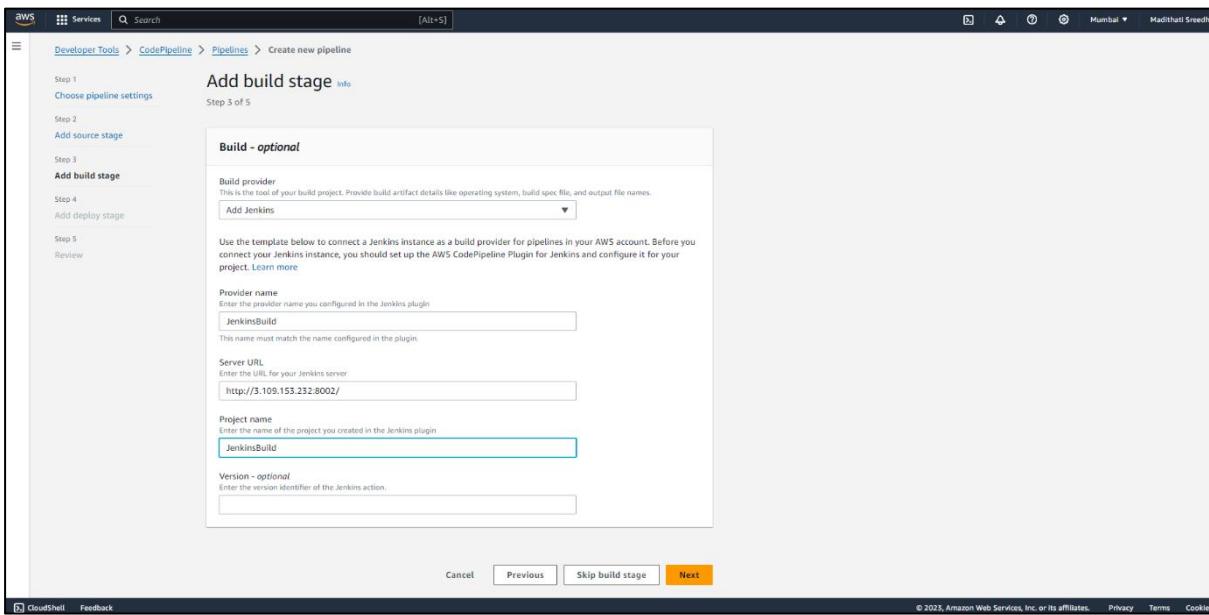


- Provide Name: Enter the provider name you configured in the Jenkins plugin

(Note: This name must match the name configured in the Pipeline)

- Server URL: Enter the URL of your Jenkins server

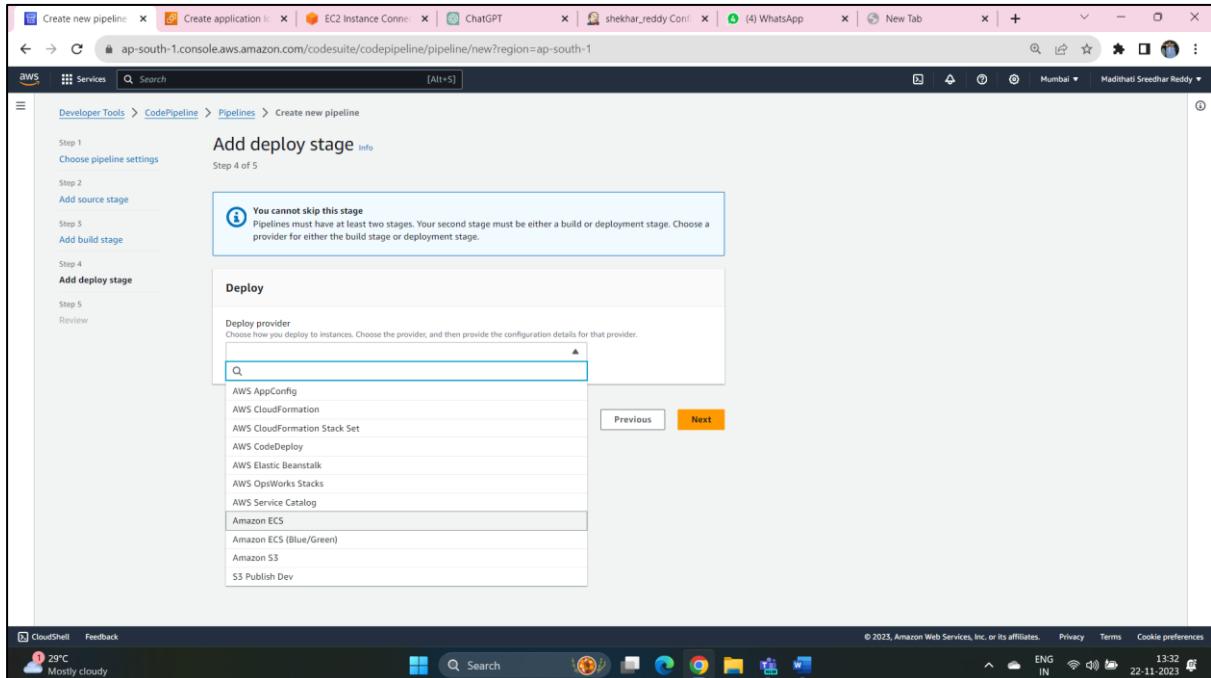
- Project name: Enter the project name you created in the Jenkins plugin



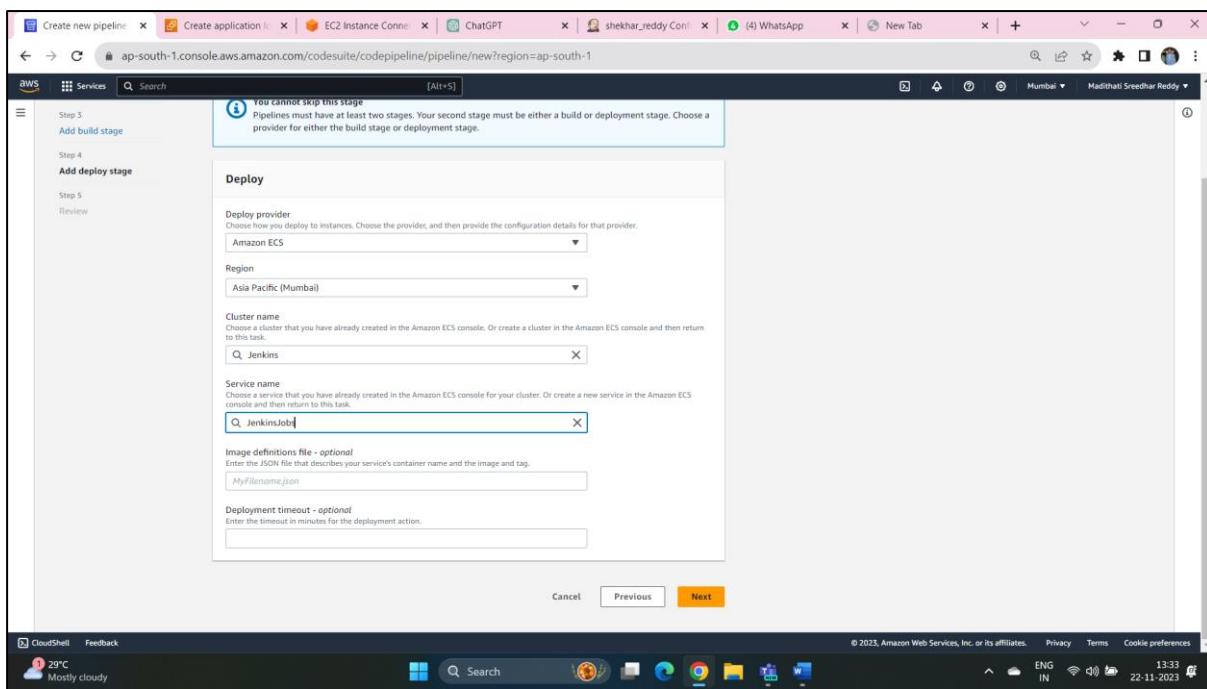
- Click next

### Configure Deploy Stage:

- Add a deployment stage based on your deploy provider (ECS).



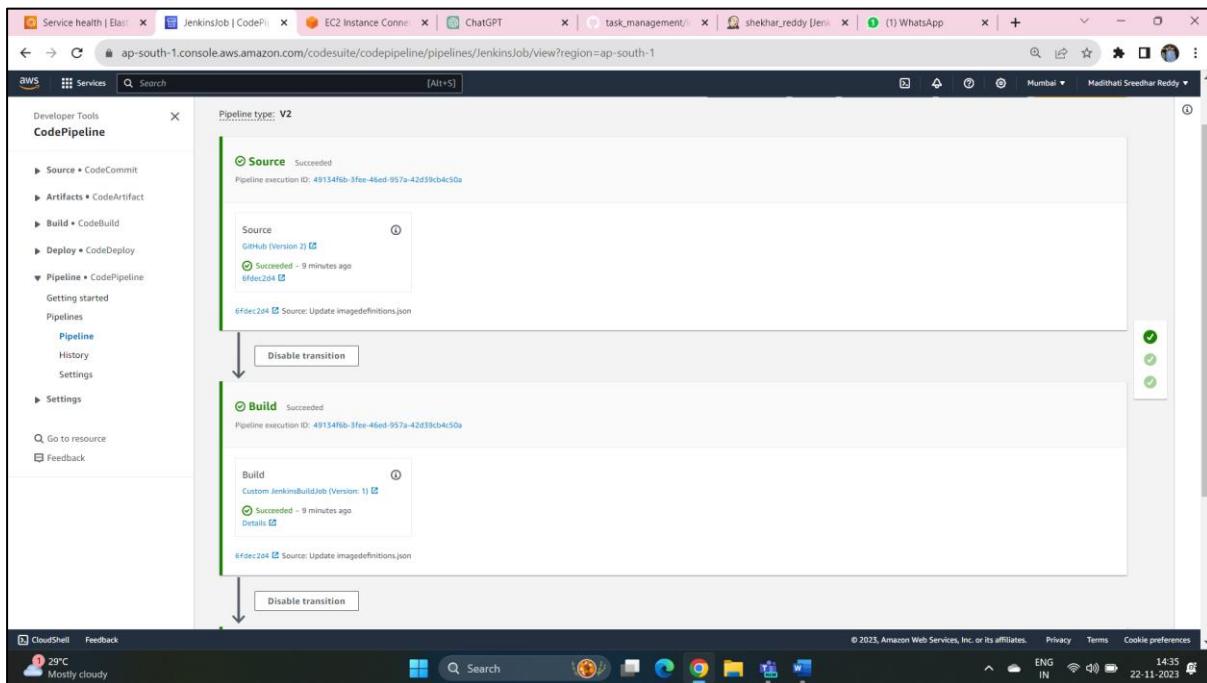
- Configure deployment settings, including the Cluster name, service name and any required parameters  
(Defined as per your ECS cluster).

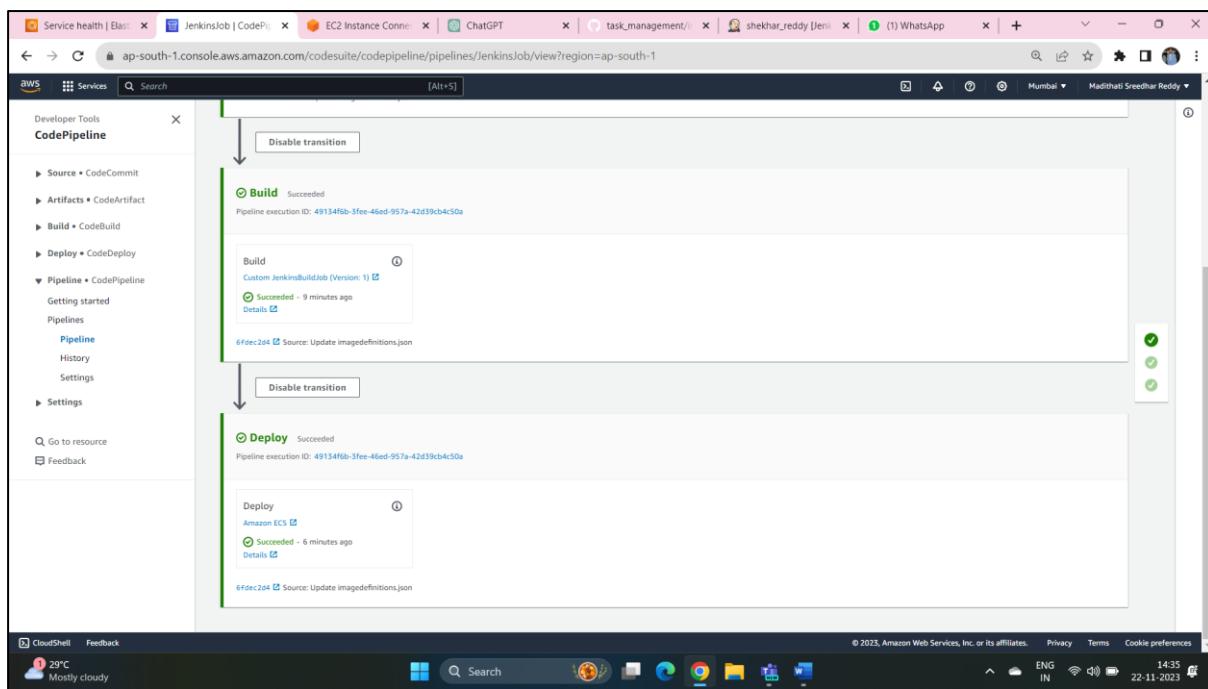


- Click on Next.
- Add additional stages if you have more steps in your pipeline (e.g., testing, approval).
- Review the pipeline configuration.
- Click the "Next" button to create the pipeline.

The completion of a pipeline depends on the successful execution of all stages and actions within the pipeline. If all stages and actions succeed, the pipeline execution is considered complete. If any stage or action fails, the pipeline execution may be marked as failed.

- The successfully executed code pipeline looks like this.





- In the Jenkins Server that is Configured in the AWS CodePipeline see the Console Output
- The main purpose is that This will take the SourceArtifact as Input and run the JenkinsJob that will in return produce the BuildArtifact that is stored in the S3 Bucket

```

Started by an SCM change
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/shekhar_reddy
[AWS CodePipeline Plugin] Job 'd47f31b1-0036-4bff-afbd-5fc8c7e4254' received
[AWS CodePipeline Plugin] Acknowledged job with ID: d47f31b1-0036-4bff-afbd-5fc8c7e4254
[AWS CodePipeline Plugin] Clearing workspace '/var/jenkins_home/workspace/shekhar_reddy' before download
[AWS CodePipeline Plugin] Detected compression type: none
[AWS CodePipeline Plugin] Successfully downloaded artifact from AWS CodePipeline
[AWS CodePipeline Plugin] Extracting '/var/jenkins_home/workspace/shekhar_reddy/WIoA7z?' to '/var/jenkins_home/workspace/shekhar_reddy'
[AWS CodePipeline Plugin] Artifact uncompressed successfully
[shekhar_reddy] $ /bin/sh -xe /tmp/jenkins50092329920073020503.sh
+ docker build -t simple
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring Dockerfile: 370B done
#1 DONE 0.0s

#2 [internal] load .dockerignore
#2 transferring context: 28 done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/python:3.9
#3 DONE 0.0s

#4 [1/8] FROM docker.io/library/python:3.9@sha256:b6cc878074fdcc6aff44867f42bf4cc8d18f4e71ed44027649856355b0e23dbe4
#4 DONE 0.0s

#5 [internal] load build context
#5 transferring context: 7.93kB done
#5 DONE 0.0s

#6 [6/8] COPY ./templates/ ./templates/
#6 CACHED

#7 [2/8] RUN mkdir -p /home/python
#7 CACHED

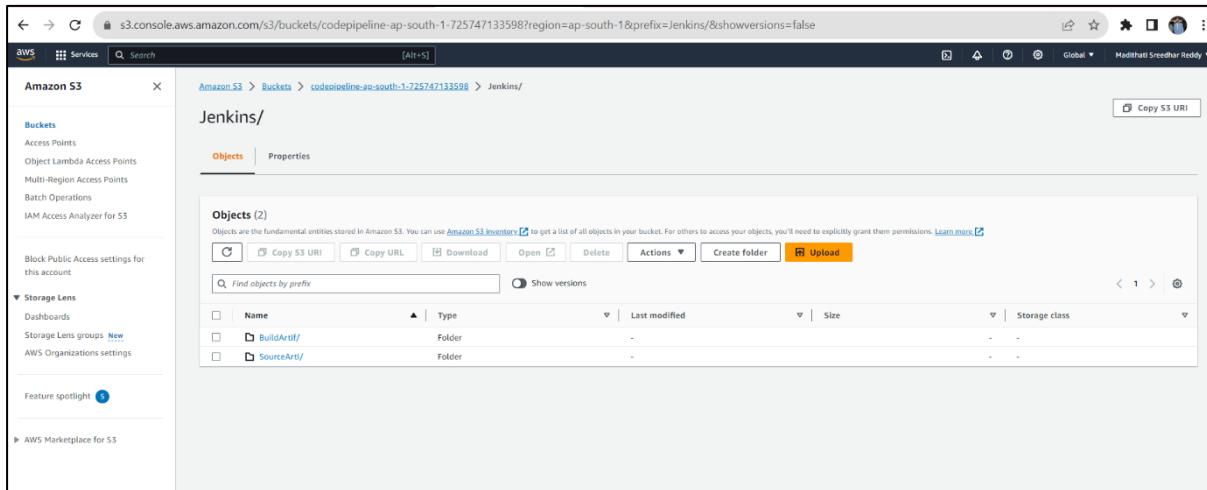
```

```

Dashboard > shekhar_reddy > #8 > Console Output
[...]
260d0f7/d2de: Preparing
29e49b059edda: Preparing
1777ac7d3d0b: Preparing
5f7fbef18a0861: Waiting
0751ba4610b1: Waiting
f8a3a371445d: Waiting
70866f74c09e: Waiting
2d7d8b0c472404: Waiting
86e50e0709ee: Waiting
12b956027b2a1: Waiting
266edf75d0c8: Waiting
29e49b059edda: Waiting
1777ac7d3d0b: Waiting
e27b52b85e99: Layer already exists
8a3c6a610b95c: Layer already exists
448549066ef5: Layer already exists
15a178a454d1: Layer already exists
e7959e1c56b0: Layer already exists
5f7fbef18a0861: Layer already exists
70866f74c09e: Layer already exists
f8a3a371445d: Layer already exists
0751ba4610b1: Layer already exists
2d7d8b0c472404: Waiting
266edf75d0c8: Layer already exists
1777ac7d3d0b: Layer already exists
12b956027b2a1: Layer already exists
86e50e0709ee: Layer already exists
29e49b059edda: Layer already exists
latest: digest: sha256:67b4114fa25588d173f5e42fc17c7ba37d4814f402a127cc8d81dba56c2425 size: 3461
[AWSCodePipelinePlugin] Publishing artifacts
[AWSCodePipelinePlugin] Compressing directory '/var/jenkins_home/workspace/shekhar_reddy' as 'Zip' archive
[AWSCodePipelinePlugin] Uploading artifact: {Name: BuildArtifact, Location: {Type: S3, S3Location: {BucketName: codepipeline-ap-south-1-725747133598, ObjectKey: Jenkins/BuildArtifact/yOkaTKy}}}, file: /tmp/shekhar_reddy-125292794069835516.zip
[AWSCodePipelinePlugin] Upload successful
[AWSCodePipelinePlugin] Build succeeded, calling PutJobSuccessResult
Finished: SUCCESS

```

- If your application produces output files or data within the container, you might want to configure a mechanism to retrieve them. This could involve using shared volumes, S3, or other storage solutions.



- These BuildArtifact is used by the Deploy Stage i.e, ECS for deploying the application
- If all went right the cluster UI will look Like this

The screenshot shows the AWS ECS console with the following details:

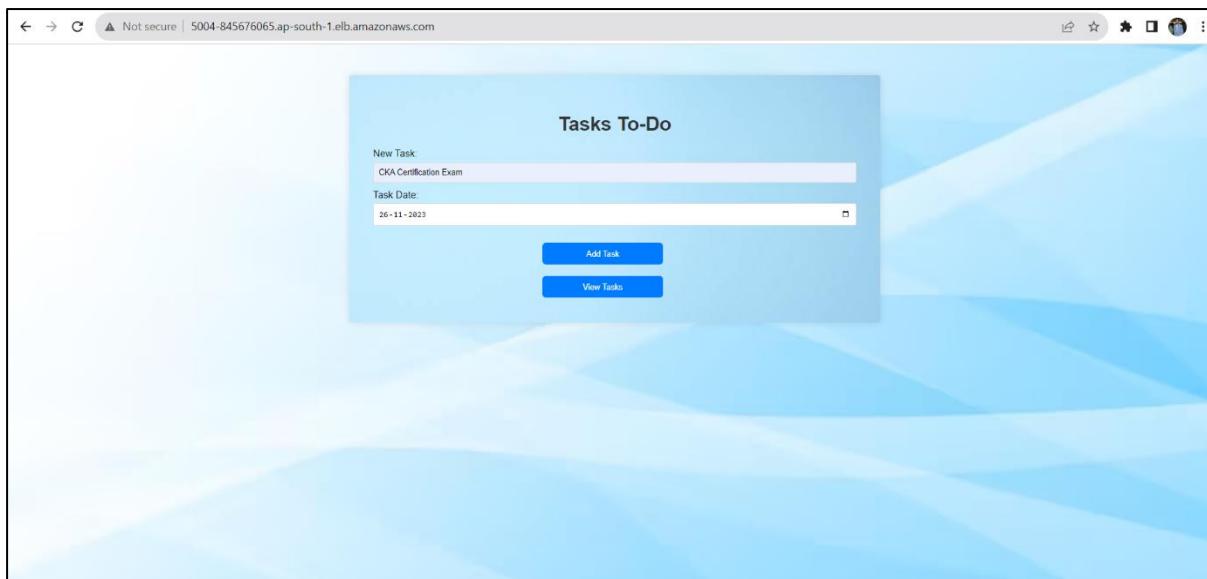
- Clusters:** JenkinsJob
- Status:** ARN: arn:aws:ecs:ap-south-1:862547479026:service/jenkins/JenkinsJobs Status: Active
- Deployments:** 1 Completed
- Health:** Health check grace period: 0 seconds
- Load balancer health:** Application Load Balancer (ALB) named protocol. Listener protocol: HTTP, port: 5004. Target group name: taskHTTP. Health check path: /. Targets (1 total): 1 Healthy, 0 Unhealthy.
- Metrics:** CPU utilization and Memory utilization graphs over the last 3 hours.

- Click on View Load Balancer.
- Then Copy the DNS Name of it and paste it in your browser

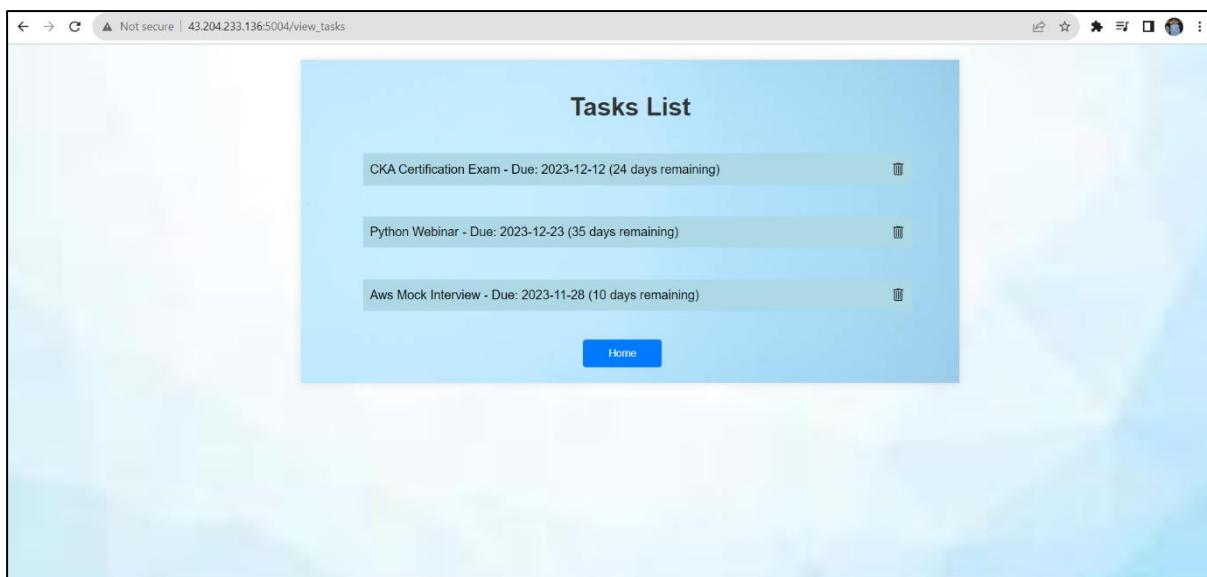
The screenshot shows the AWS Load Balancing console with the following details:

- Details:** Load balancer type: Application. Status: Active. Scheme: Internet-facing. Hosted zone: ZP97RAFLXTNZK.
- Load balancer ARN:** arn:aws:elasticloadbalancing:ap-south-1:862547479026:loadbalancer/app/ECS/e1c533dc25719ef
- DNS name:** ECS-542256558.ap-south-1.elb.amazonaws.com (A Record)
- Listeners and rules:** 2 listeners defined:
  - Protocol:Port:** HTTP:80, **Default action:** Forward to target group (ECS 1 (100%))
  - Protocol:Port:** HTTP:5004, **Default action:** Forward to target group (task 1 (100%))

- We can access our Web application using DNS name of the Load balancer.
- The web UI will Look Like below.
- Now add Some Tasks as per your Requirement.



- Click on View Tasks that will look as below.



#### Define deployment actions, including application parameterization.

In Amazon Elastic Container Service (ECS), deployment actions refer to the steps and processes involved in releasing and updating containerized applications. ECS is a fully managed container orchestration service that simplifies the deployment, management, and scaling of Docker containers on Amazon EC2 instances or AWS Fargate.

Here are the key components and concepts related to deployment actions and application parameterization in ECS:

## **1. Task Definition:**

- A task definition is a blueprint for your application. It defines various parameters, such as which Docker image to use, how much CPU and memory to allocate, networking information, and more.
- When updating an application, you can revise the task definition to include new container images or modify other settings.

## **2. Service:**

- A service in ECS allows you to run and maintain a specified number of instances of a task definition simultaneously.
- During deployment, ECS manages the desired count of tasks, ensuring that the specified number of tasks are running.

## **3. Deployment Types:**

- ECS supports two deployment types: rolling updates and blue/green deployments.
- **Rolling Updates:** ECS gradually replaces instances of the previous version of your task with the new version, minimizing downtime.
- **Blue/Green Deployments:** This involves running two separate environments (blue and green) and switching traffic from one to the other when deploying updates.

## **4. Application Load Balancer (ALB) or Network Load Balancer (NLB):**

- Load balancers help distribute incoming traffic across multiple tasks in your ECS service. This is crucial for achieving high availability and seamless updates.
- ALBs and NLBs can be used to route traffic to different versions of your application during a blue/green deployment.

## **5. Application Parameterization:**

- Parameterization involves configuring your application with external parameters, allowing you to customize its behavior without modifying the application code.
- In ECS, you can use environment variables and task definition parameters for parameterization.
- For example, you might use environment variables to specify configuration settings or connection strings dynamically.

## **6. ECS CLI and AWS Management Console:**

- The ECS CLI and AWS Management Console provide tools for managing ECS tasks, services, and deployments.
- You can use these interfaces to update task definitions, change desired task counts, and monitor the status of deployments.

## **7. Continuous Integration/Continuous Deployment (CI/CD):**

- Many organizations integrate ECS deployments into their CI/CD pipelines. Tools like AWS CodePipeline and AWS CodeBuild can be used to automate the building and deployment of containerized applications on ECS.

Application parameterization in AWS ECS refers to the practice of configuring and managing application-specific parameters or settings dynamically, allowing for flexibility and customization without modifying the application code. This can include environment variables, configuration files, or other mechanisms to adjust the behaviour of the application.

- Define environment variables in your ECS task definition.
- Store Sensitive Data:

For sensitive information like API keys or database passwords, consider using AWS Secrets Manager.

Store secrets in Secrets Manager and reference them in your ECS task definition.

- Store Configurations in an S3 Bucket:

Store configuration files in an S3 bucket, and download them at runtime.

- Service Discovery:

Use service discovery for dynamic port mapping.

Automatically register services with a service discovery namespace.

## Sub Task 5:

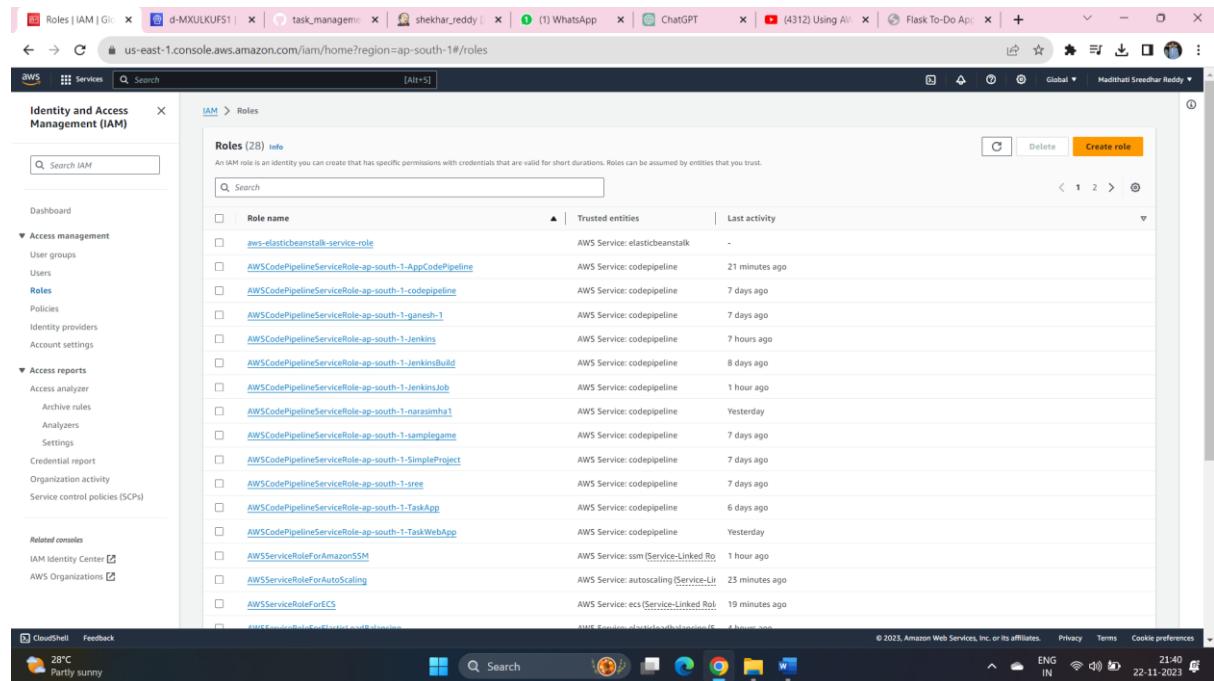
### Deploy the application to AWS ECS or EKS using AWS CodeDeploy.

AWS CodeDeploy is a fully managed deployment service that automates software deployments to a variety of compute services, including Amazon EC2 instances, AWS Lambda functions, and instances running on-premises. It allows you to consistently deploy your applications across your development, test, and production environments.

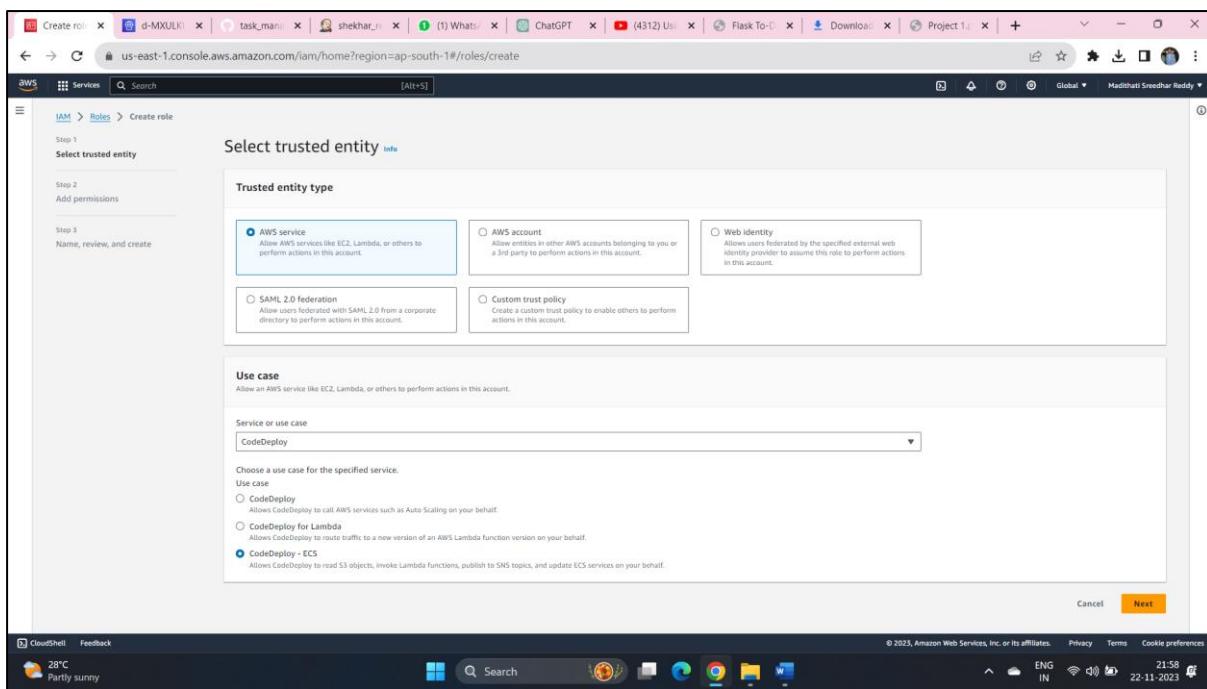
#### Creation of the IAM Role

Creating a role in AWS involves using AWS Identity and Access Management (IAM), which allows you to manage access to AWS services and resources securely. Here are the general steps to create a role in AWS:

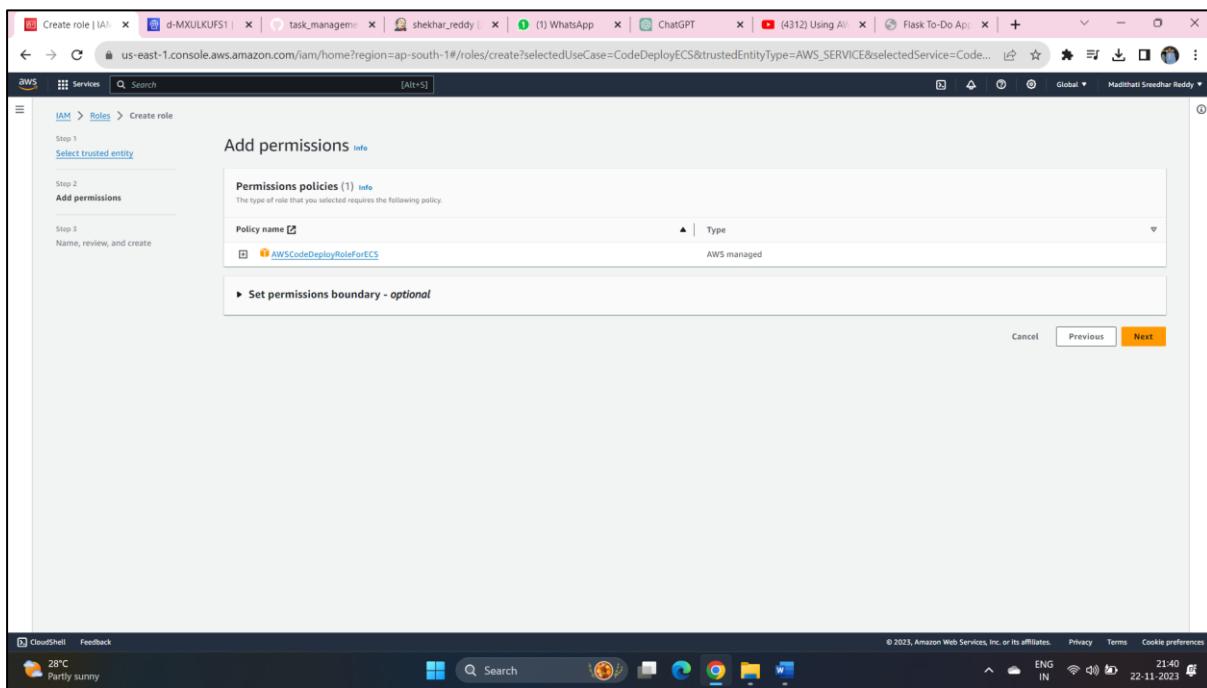
- Navigate to IAM:  
In the AWS Management Console, go to the IAM service. You can find it under the "Services" dropdown or by searching for "IAM."
- Select "Roles" in the IAM Dashboard:  
In the IAM dashboard, select "Roles" from the left navigation pane.



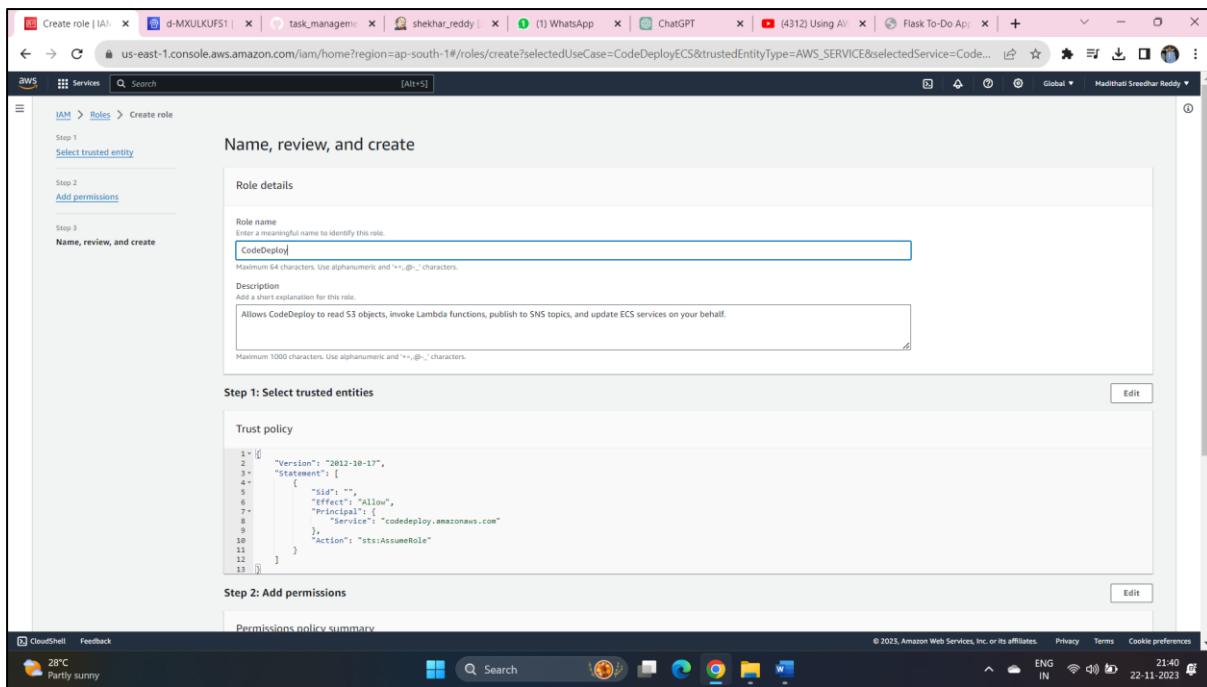
- Click "Create role":
- Click the "Create role" button to start the role creation process.



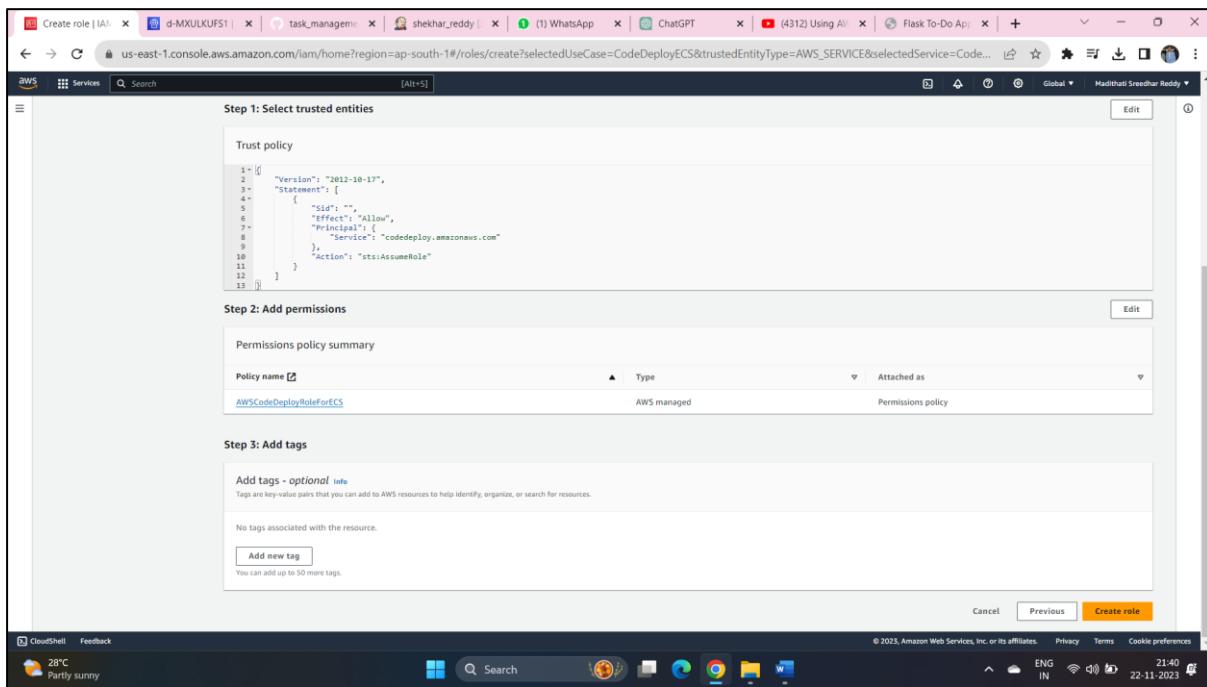
- Choose the use case as CodeDeploy
- It will reveal 3 options on which you need to select the CodeDeploy-ECS
- Click on Next



- Add Permissions as above and click on Next



- Give the Name for the role .



- Click on Create Role

### **Creation of Cluster, Task Definition file:**

- Follow the Same procedure for creating the Cluster and the Task Definition File mentioned in the Sub Task -4

### **Creation of the ECS Service:**

- **Navigate to ECS:**

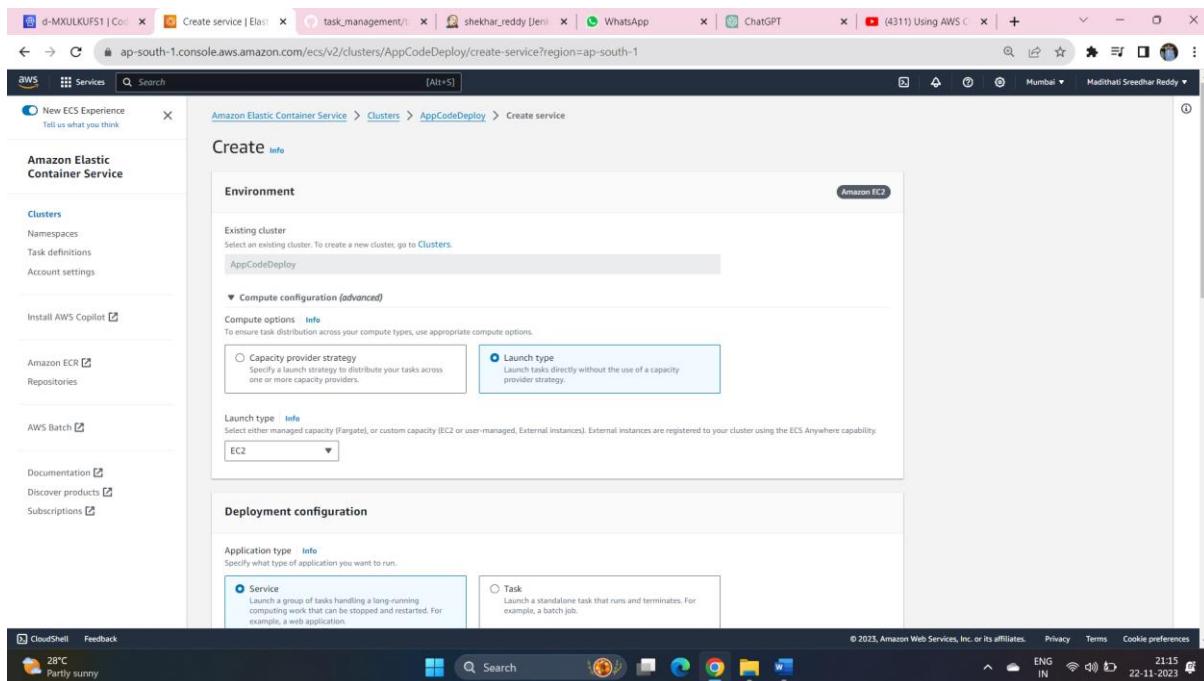
In the AWS Management Console, go to the ECS service. You can find it under the "Services" dropdown or by searching for "ECS."

- **Select your Cluster:**

In the ECS dashboard, select the ECS cluster where you want to create the service.

- **Click "Create" in the "Services" tab:**

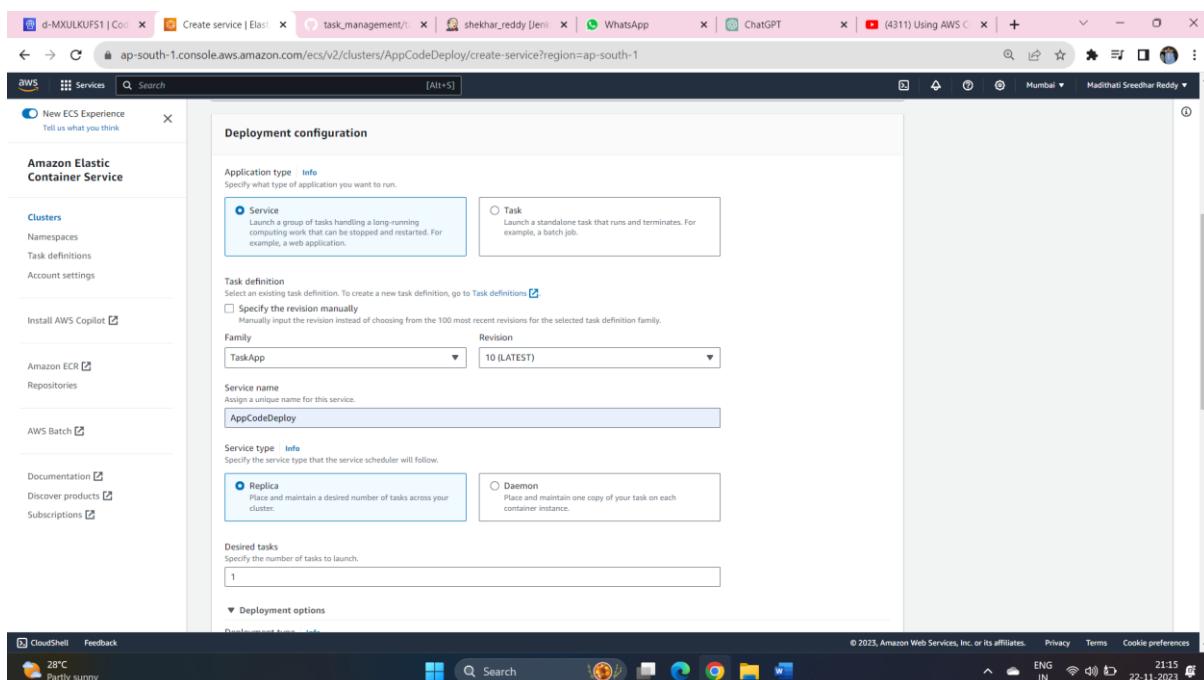
In the "Services" tab of your cluster, click the "Create" button.



- **Configure Service:**

**Launch type:** Choose whether your tasks will run on EC2 instances or as AWS Fargate tasks.

**Task Definition:** Select the task definition that you want to run as part of the service.



- **Service name:** Provide a unique name for your service.
- **Number of tasks:** Specify the number of tasks (containers) to run in the service.
- **Cluster VPC:** Choose the VPC in which to launch your tasks.
- **Subnets:** Select the subnets in which to place your tasks.
- **Security Groups:** Choose security groups for your tasks.

The screenshot shows the 'Create service' step in the AWS ECS wizard. Under 'Deployment options', 'Blue/green deployment (powered by AWS CodeDeploy)' is selected. The 'Deployment configuration' dropdown is set to 'CodeDeployDefault.ECSAlAtOnce'. A service role for CodeDeploy is specified as 'arn:aws:iam:862547479026:role/CodeDeploy'. Other optional sections like Service Connect, Service discovery, Load balancing, and Service auto scaling are shown but not filled.

- **Set Deployment Configuration:**

Choose the deployment configuration that determines how your service tasks are deployed.  
Select the option as blue/green deployments.

- **Select the Service role created earlier**

This screenshot is identical to the one above, showing the 'Create service' step in the AWS ECS wizard. It displays the same deployment configuration settings ('Blue/green deployment (powered by AWS CodeDeploy)', 'CodeDeployDefault.ECSAlAtOnce' for deployment config, and the specified service role). The optional sections remain unconfigured.

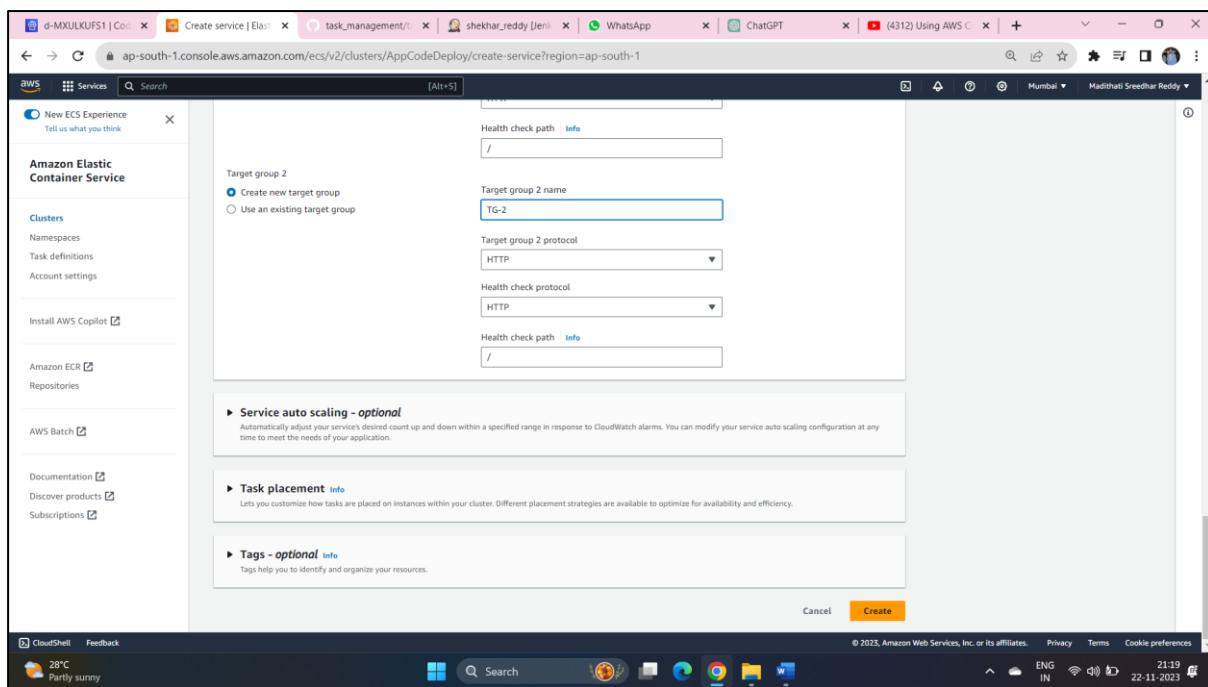
- **Load Balancer:** If you want to distribute traffic to your tasks, configure the load balancer settings.
- Give the configuration as below.

The screenshot shows the AWS Cloud Console interface for creating a new service. On the left, the navigation pane is visible with options like 'Clusters', 'Namespaces', 'Task definitions', and 'Account settings'. The main content area is titled 'Service Connect - optional' and 'Service discovery - optional'. Below these, under 'Load balancing', the 'Load balancer type' is set to 'Application Load Balancer'. The 'CodeDeploy' option is selected for the load balancer. The 'Health check grace period' is set to 0 seconds. Under 'Container', 'task 5004:5004' is chosen. In the 'Listeners' section, 'Production listener' is configured with port 5004 and protocol HTTP. The status bar at the bottom indicates it's 21:21 on 22-11-2023.

- Create the Listeners as below

The screenshot shows the 'Listeners' configuration page. It includes fields for 'Production listener port' (5004) and 'Production listener protocol' (HTTP). Under 'Test listener', there's an option to 'Add a test listener'. The 'Target groups' section shows two target groups: 'Target group 1' (name TG-1, protocol HTTP, health check path '/') and 'Target group 2' (name TG-2, protocol HTTP, health check path '/'). The status bar at the bottom indicates it's 21:19 on 22-11-2023.

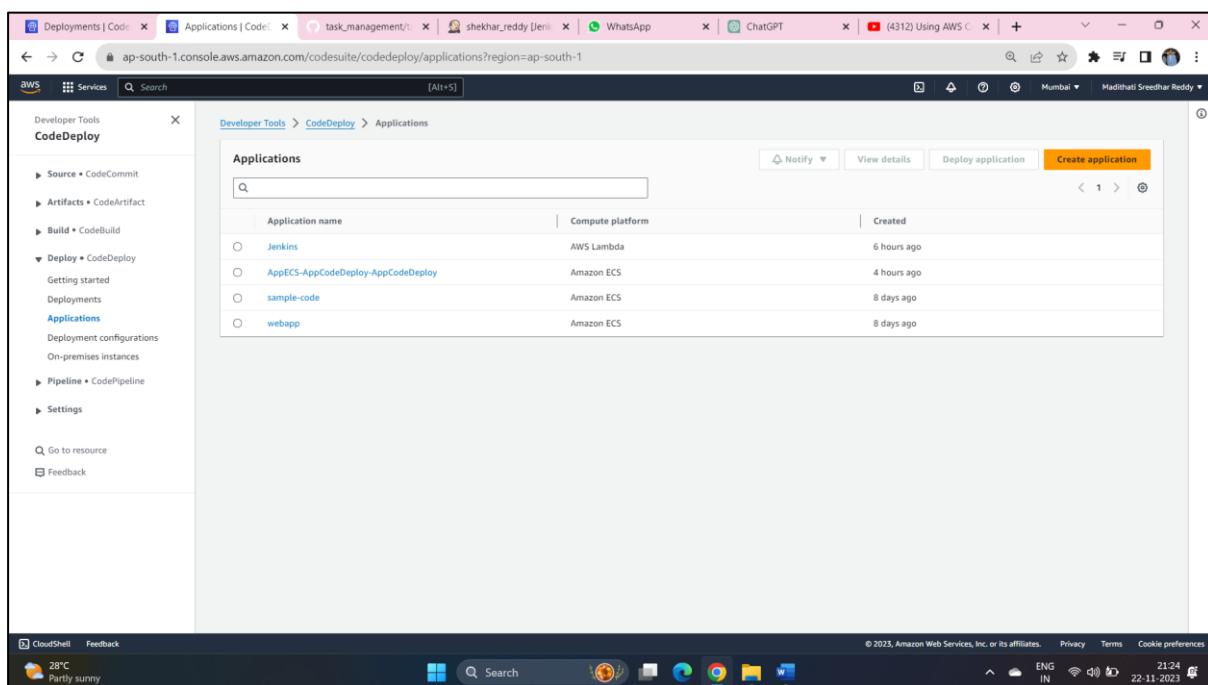
- Create new target groups or Choose the Target Groups from the existing ones configured



- Then Review the configurations to ensure they are correct.
- Click the "Create Service" button to create the ECS service

## Configuring the Code Deploy:

- Navigate to AWS CodeDeploy:  
In the AWS Management Console, go to the CodeDeploy service. You can find it under the "Services" dropdown or by searching for "CodeDeploy."
- Under Applications You can see the application created with the name of our Cluster Created earlier
- Click on it.



- You can see an Deployment Already Created.
- Click on that.

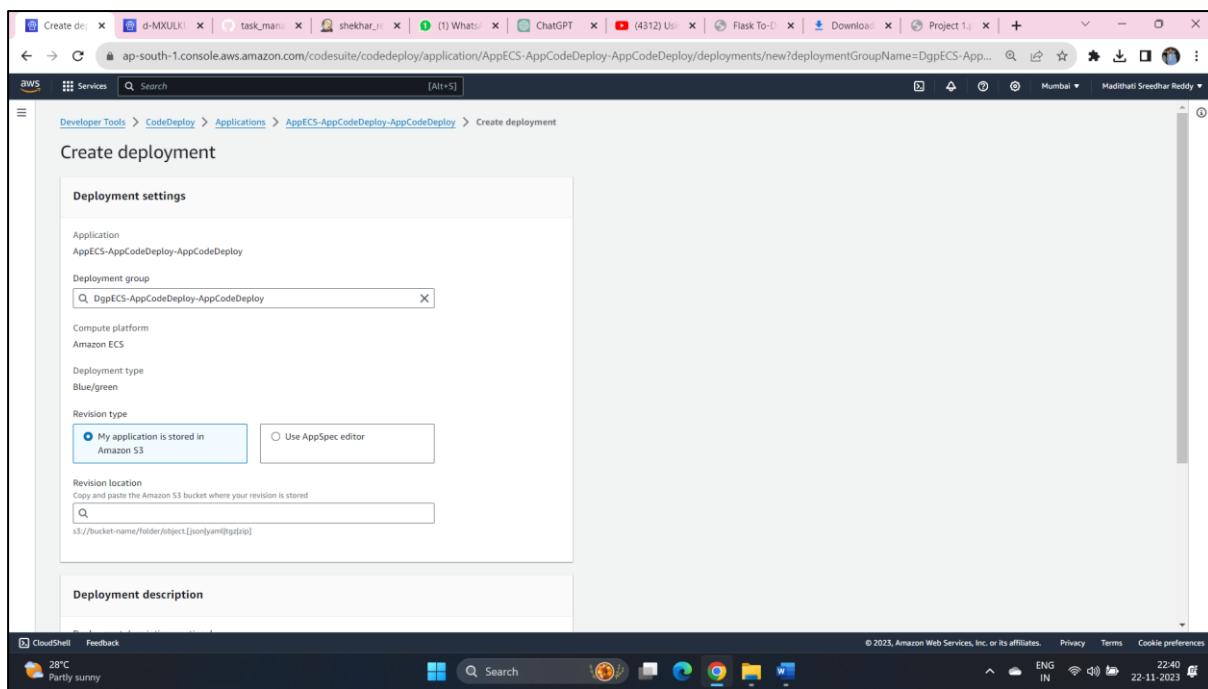
The screenshot shows the AWS CodeDeploy console with the URL [ap-south-1.console.aws.amazon.com/codesuite/codedeploy/applications/AppECS-AppCodeDeploy-AppCodeDeploy?region=ap-south-1](https://ap-south-1.console.aws.amazon.com/codesuite/codedeploy/applications/AppECS-AppCodeDeploy-AppCodeDeploy?region=ap-south-1). The left sidebar is collapsed, and the main content area displays the 'Application details' for the application 'AppECS-AppCodeDeploy-AppCodeDeploy'. The 'Deployment groups' tab is active, showing a table with one row:

Name	Status	Last attempted deployment	Last successful deployment	Trigger count
DgpECS-AppCodeDeploy-AppCode...	In progress	-	Nov 22, 2023 5:25 PM (UTC+5:30)	0

- Then Click on the Create Deployment Option

The screenshot shows the AWS CodeDeploy console with the URL [ap-south-1.console.aws.amazon.com/codesuite/codedeploy/applications/AppECS-AppCodeDeploy-AppCodeDeploy/deployment-groups/DgpECS-AppCodeDeploy-AppCodeDeploy](https://ap-south-1.console.aws.amazon.com/codesuite/codedeploy/applications/AppECS-AppCodeDeploy-AppCodeDeploy/deployment-groups/DgpECS-AppCodeDeploy-AppCodeDeploy). The left sidebar is collapsed, and the main content area displays the 'Deployment group details' for the deployment group 'DgpECS-AppCodeDeploy-AppCodeDeploy'. The 'Deployment group details' section includes fields for deployment group name ('DgpECS-AppCodeDeploy-AppCodeDeploy'), application name ('AppECS-AppCodeDeploy-AppCodeDeploy'), compute platform ('Amazon ECS'), deployment type ('Blue/green'), service role ARN ('arn:aws:iam::862547479026:role/CodeDeploy'), and deployment configuration ('CodeDeployDefault.ECSAllAtOnce'). The 'Environment configuration' section shows the ECS cluster name ('AppCodeDeploy') and ECS service name ('AppCodeDeploy'). The 'Load Balancing' section shows target group 1 name ('AppCodeDeploy'), target group 2 name ('CodeDeploy'), production listener ARN ('arn:aws:elasticloadbalancing:ap-south-1:862547479026:listener/app/CodeDeploy/cd59290e8586a1a4/72f3d5b44052b196'), and test listener ARN ('-').

- Under it Choose the Deployment group



- For the AppSpec file you need to do the Following steps initially
- Create the file with the content as below.

```

version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:ap-south-1:862547479026:task-definition/TaskApp:9"
        LoadBalancerInfo:
          ContainerName: "task"
          ContainerPort: 5004
    
```

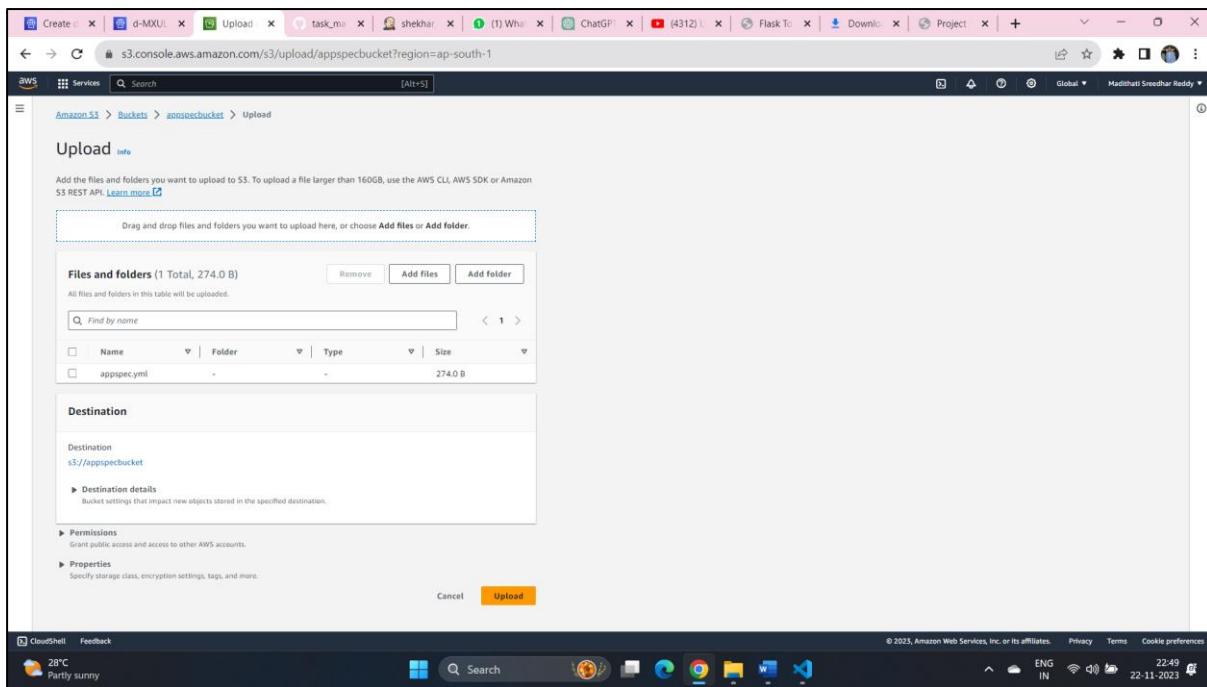
- Create a new Bucket with Unique name in Amazon S3

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with options like Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Storage Lens, Dashboards, and AWS Organizations settings. The main area displays an 'Account snapshot' with a link to 'View Storage Lens dashboard'. Below it is a table titled 'Buckets (4) info' showing four buckets: 'sree-1110', 'sree-1110-staging', 'codepipeline-ap-south-1-25747133598', and 'appspecbucket'. Each row includes columns for Name, AWS Region, Access (all set to Public), and Creation date. A 'Create bucket' button is located at the top right of the table area.

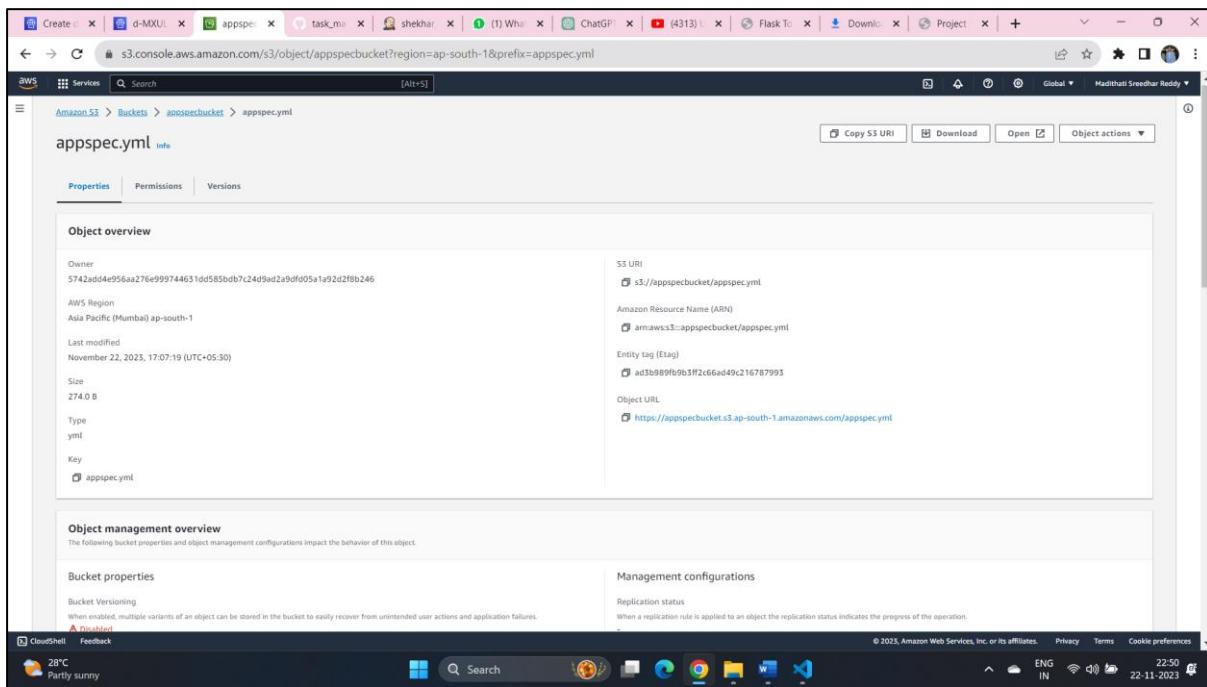
- Click on Create Bucket
- Give an Unique name and click create

The screenshot shows the 'Create bucket' configuration page. At the top, it says 'Create bucket' with a 'Info' link. Below that is a 'General configuration' section with fields for 'Bucket name' (set to 'appspecbucket'), 'AWS Region' (set to 'Asia Pacific (Mumbai) ap-south-1'), and a 'Copy settings from existing bucket - optional' section with a 'Choose bucket' button. Under 'Object Ownership', there are two options: 'ACLs disabled (recommended)' (selected) and 'ACLs enabled'. The 'Block Public Access settings for this bucket' section is at the bottom, with a note about public access being granted through ACLs, bucket policies, or access point policies. The status bar at the bottom shows it's 28°C and partly sunny.

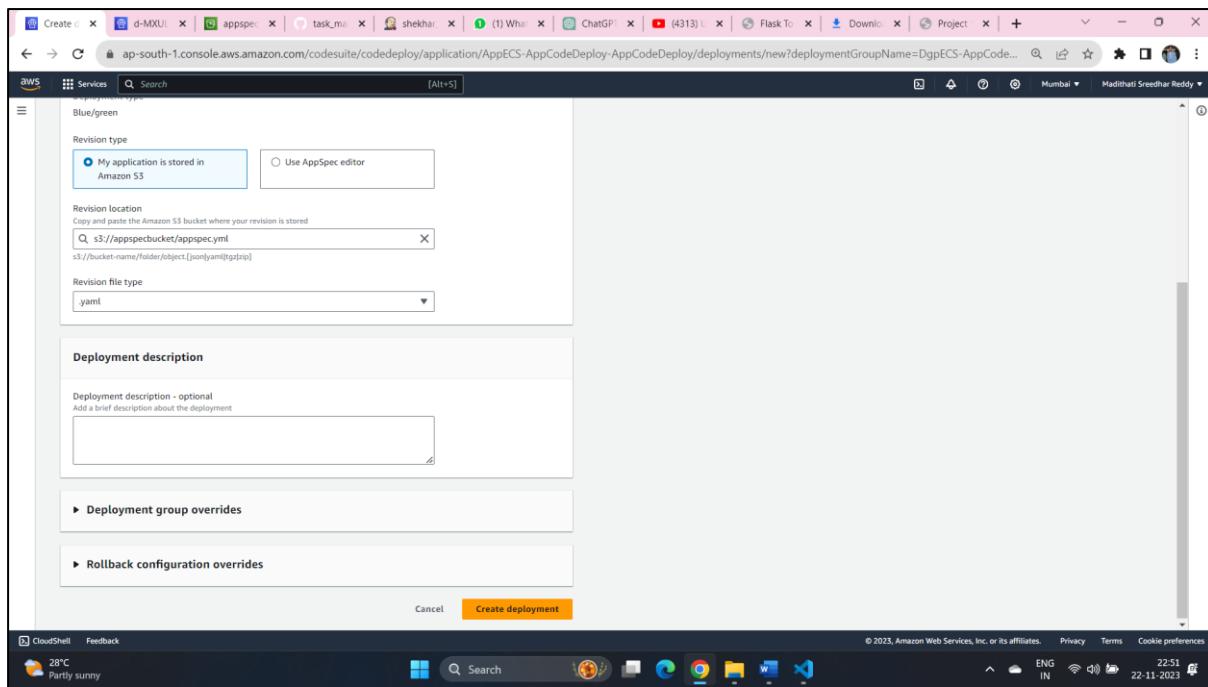
- Upload the file created in the local to it.



- Once Uploaded S3 URI of it



- Paste it under the Revision location
- Choose the Revision File Type as .yaml



- Click on Create Deployment
- Under Deployments Section The one created will be in progress state

Deployment history									
Deployment Id	Status	Deployment ...	Compute pla...	Application	Deployment ...	Revision loca...	Initiating event	Start time	End time
d-MXULKUF51	<span>In progress</span>	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://codepipe...	User action	Nov 22, 2023...	-
d-B1WN4S851	<span>Succeeded</span>	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://appspec...	User action	Nov 22, 2023...	Nov 22, 2023...
d-NEIF2K3GE	<span>Failed</span>	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://appspec...	User action	Nov 22, 2023...	Nov 22, 2023...
d-JOR73ECS1	<span>Failed</span>	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://appspec...	User action	Nov 22, 2023...	Nov 22, 2023...

- Click on our Deployment.

The screenshot shows the AWS CodeDeploy console for a deployment named 'd-MXULKUFS1'. The 'Deployment status' section details four steps: Step 1 (Deploying replacement task set) completed at 100% success; Step 2 (Rerouting production traffic to replacement task set) completed at 100% success; Step 3 (Wait 1 hour 0 minutes) in progress at 18%; and Step 4 (Terminate original task set) not started. The 'Traffic shifting progress' section shows 'Original' at 0% and 'Replacement' at 100%. Deployment details include Application: AppECS-AppCodeDeploy-AppCodeDeploy, Deployment ID: d-MXULKUFS1, Deployment configuration: CodeDeployDefault.ECSAllAtOnce, Deployment group: DgECS-AppCodeDeploy-AppCodeDeploy, and Status: In progress (Initiated by User action).

- Once it is fully Completed and all the above steps are performed correctly It will look as below.

The screenshot shows the AWS CodeDeploy console for the same deployment 'd-MXULKUFS1'. The 'Deployment status' section shows all steps completed at 100% success. The 'Traffic shifting progress' section shows 'Original' at 0% and 'Replacement' at 100%. Deployment details show the final state: Status Succeeded (Completed at 100%).

- And all the deployment events will be marked as Succeeded

The screenshot shows the AWS CodeDeploy console for a deployment named "d-MXULKUFS1".

- Revision details:**
  - Revision location: s3://codipeline-ap-south-1-725747133598/AppCodePipeline/BuildArtifact/OxWbUEk?
  - Revision created: 1 hour ago
  - Revision description: Application revision registered by Deployment ID: d-MXULKUFS1
- Task set activity:**

Task set ID	Environment	Task set status	Traffic	Desired count	Running count	Pending count
ecs-svc://1109913773616991282	Replacement	PRIMARY	100%	1	1	0
ecs-svc://2207025905170465844	Original	ACTIVE	0	1	1	0
- Deployment lifecycle events:**

Event	Duration	Status	Start time	End time
BeforeInstall	less than one second	Succeeded	Nov 22, 2023 8:58 PM (UTC+5:30)	Nov 22, 2023 8:58 PM (UTC+5:30)
Install	2 minutes 22 seconds	Succeeded	Nov 22, 2023 8:58 PM (UTC+5:30)	Nov 22, 2023 9:01 PM (UTC+5:30)
AfterInstall	less than one second	Succeeded	Nov 22, 2023 9:01 PM (UTC+5:30)	Nov 22, 2023 9:01 PM (UTC+5:30)
BeforeAllowTraffic	less than one second	Succeeded	Nov 22, 2023 9:01 PM (UTC+5:30)	Nov 22, 2023 9:01 PM (UTC+5:30)
AllowTraffic	less than one second	Succeeded	Nov 22, 2023 9:01 PM (UTC+5:30)	Nov 22, 2023 9:01 PM (UTC+5:30)
AfterAllowTraffic	less than one second	Succeeded	Nov 22, 2023 9:01 PM (UTC+5:30)	Nov 22, 2023 9:01 PM (UTC+5:30)

- Navigate to the created Cluster it will resemble as below

The screenshot shows the AWS Elastic Container Service (ECS) console for a cluster named "AppCodeDeploy".

- Cluster infrastructure:**
  - ARN: arn:aws:ecs:ap-south-1:862547479026:cluster/A
  - Status: Active
  - CloudWatch monitoring: Container Insights
  - Registered container instances: 2
- Capacity providers:**

Capacity provider	Type	ASG	Managed s...	Managed i...	Current size	Desired size	Min size	Max size
Infra-ECS-Cluster-App...	ASGProvider	Infra-ECS-Clus...	Yes	No	2	2	2	5
- Container instances:**

Container instance	Status	Type	Instance ID	Capacity p...	Availability zo...	Running tasks...	CPU available	Memory available
313530c1e26342d...	Active	EC2	i-0247556bea75...	Infra-ECS-Clu...	ap-south-1b	1	0	954
81e1c7029e46447...	Active	EC2	i-0cbd85df1435...	Infra-ECS-Clu...	ap-south-1a	1	0	954

- For Accessing Our Web Application Navigate to the EC2 Console

The screenshot shows the AWS CloudSearch interface. On the left, there is a sidebar with navigation links for various AWS services like CodeDeploy, Source, Artifacts, Build, Deploy, Pipeline, Settings, and Feedback. The main search bar at the top has the query 'ec2'. Below it, a search result summary says 'Try searching with longer queries for more relevant results'. The 'Services' section lists EC2, EC2 Image Builder, Recycle Bin, and Amazon Inspector. The 'Features' section lists Dashboard, AMIs, and Elastic IPs. To the right, a detailed search results table for EC2 is displayed, showing four deployment entries. The table columns include Deployment ID, Revision location, Initiating event, Start time, and End time. The entries are: DgpECS-App..., s3://codepipe..., User action, Nov 22, 2023..., -; DgpECS-App..., s3://appspec..., User action, Nov 22, 2023..., Nov 22, 2023...; DgpECS-App..., s3://appspec..., User action, Nov 22, 2023..., Nov 22, 2023...; DgpECS-App..., s3://appspec..., User action, Nov 22, 2023..., Nov 22, 2023... . The bottom of the screen shows the Windows taskbar with various pinned icons.

- Choose the Created Load Balancer

The screenshot shows the AWS EC2 Load Balancers page. The sidebar includes links for EC2 Dashboard, EC2 Global View, Events, Instances (with sub-links for Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (with sub-links for AMIs, AMI Catalog), and Network & Security (with sub-links for Security Groups, Volumes, Snapshots, Lifecycle Manager, Placement Groups, Key Pairs, Network Interfaces). The main content area displays a table titled 'Load balancers (1)'. The table has columns for Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created. One row is shown: 'CodeDeploy' (DNS name: CodeDeploy-20207529.ap..., State: Active, VPC ID: vpc-01d8108a86b7d1..., Availability Zones: us-east-1, Type: application, Date created: November 22, 2023, 16:46 (UTC+05:30)). Below the table, a message says '0 load balancers selected' and 'Select a load balancer above.' The bottom of the screen shows the Windows taskbar with various pinned icons.

- Copy the DNS Name of it

The screenshot shows the AWS EC2 Load Balancer Details page. The main section displays the following details:

- Load balancer type:** Application
- Status:** Active
- VPC:** vpc-01d8108a86b7d10ed
- IP address type:** IPv4
- Scheme:** Internet-facing
- Hosted zone:** ZP97RAFLXTNZK
- Availability Zones:** subnet-0b25118afeb2cfe0b, subnet-0cc421sea5d2d429, subnet-0ac712484f7e823e5
- Date created:** November 22, 2023, 16:46 (UTC+05:30)
- Load balancer ARN:** arn:aws:elasticloadbalancing:ap-south-1:862547479026:loadbalancer/app/CodeDeploy/cd59290e8586a1a4
- DNS name:** [Info](#) CodeDeploy-20207529.ap-south-1.elb.amazonaws.com (A Record)

The sidebar on the left includes sections for Instances, Images, Elastic Block Store, and Network & Security.

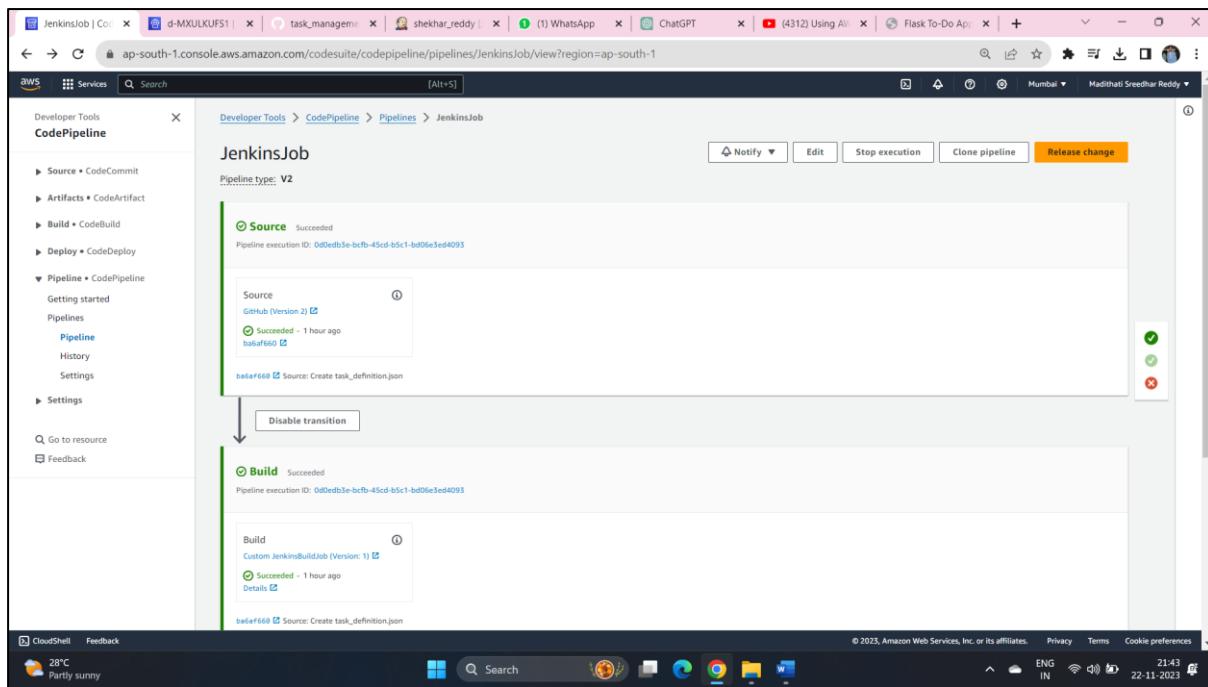
- Paste it on the Web Browser and It will Resembles as below

The screenshot shows a web browser window displaying a simple "Tasks To-Do" application. The interface includes:

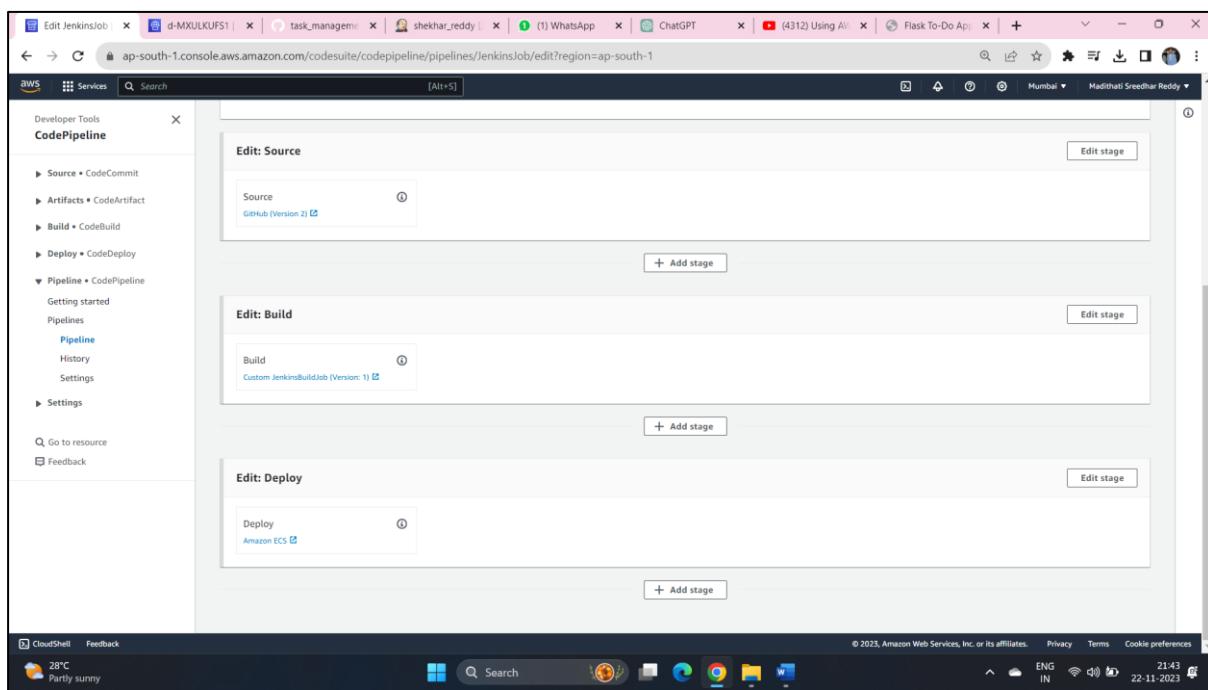
- A header bar with tabs for "Load balancer", "codedeploy-20207529.ap-south-1.elb.amazonaws.com", and "task\_management".
- A main form titled "Tasks To-Do" with fields for "New Task:" and "Task Date:" (with a date input field showing "dd-mm-yyyy").
- Buttons for "Add Task" and "View Tasks".
- A footer bar with icons for search, file, and other browser functions.

## Configuring CodePipeline For Updates:

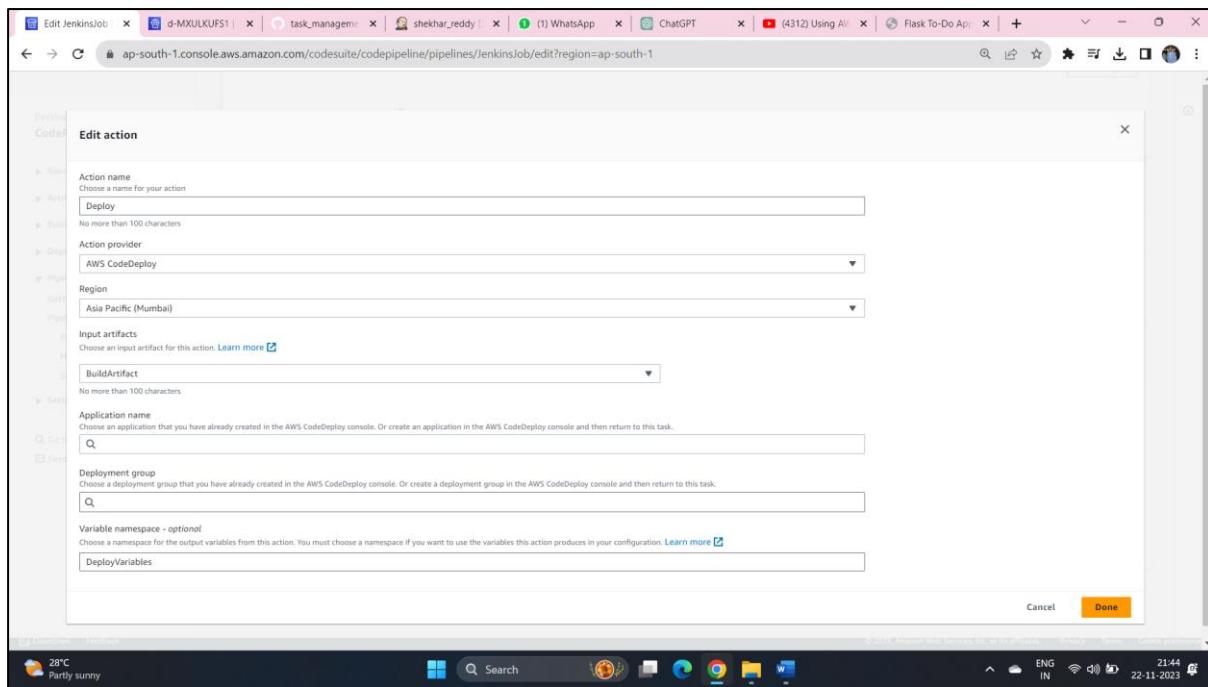
- Navigate to the CodePipeline
- Choose the created pipeline earlier
- Click on Edit



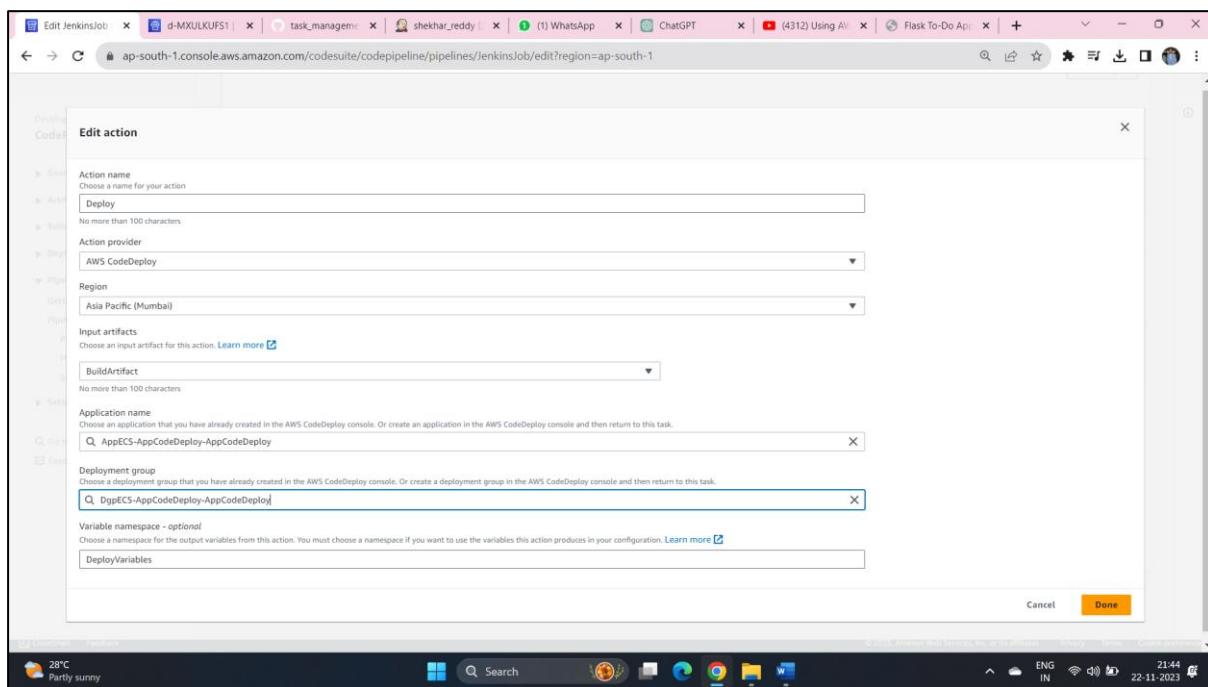
- Navigate to the Deploy Stage
- Choose Edit Stage



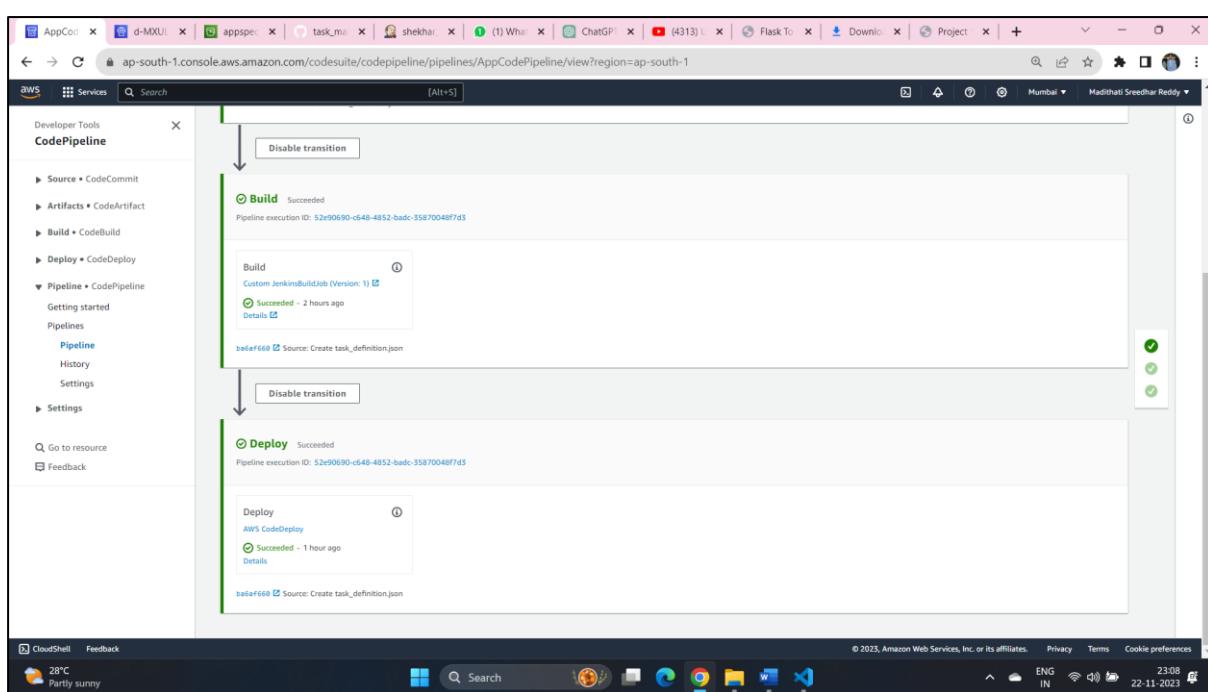
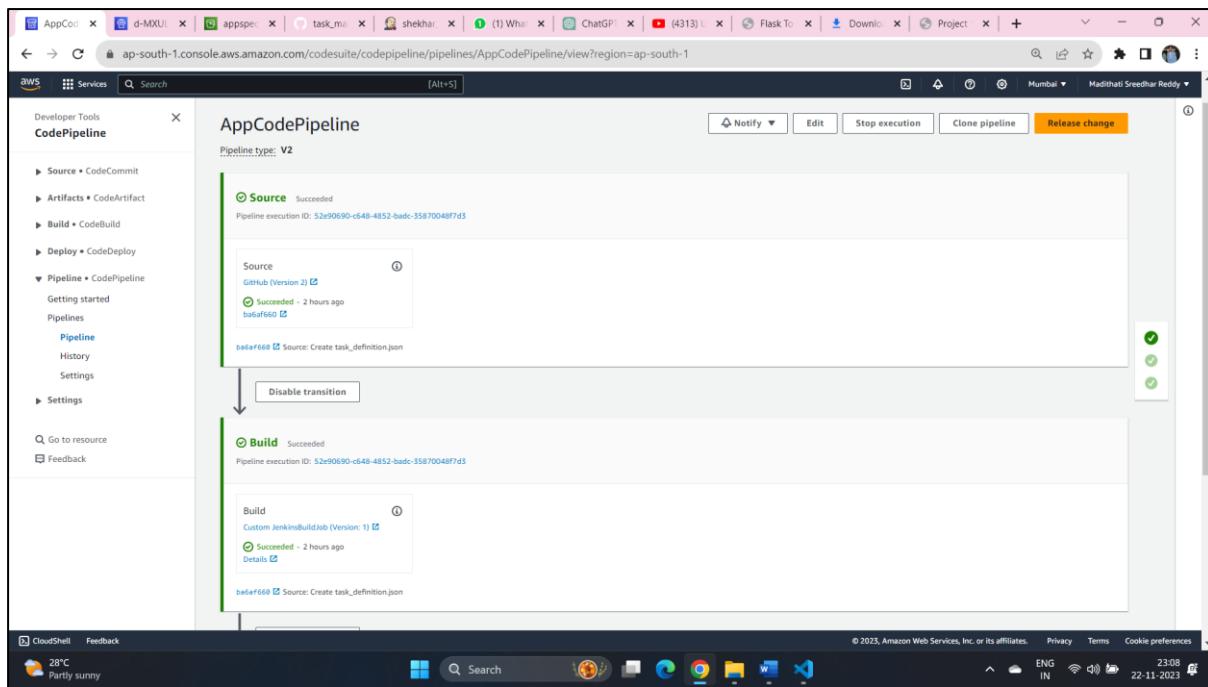
- Choose the Action Provider as AWS CodeDeploy



- Then choose the other options correctly from the existing ones



- Now Rerun the Pipeline Again with the New Image that contain the latest updates
- Once it is Successful it will look as below.

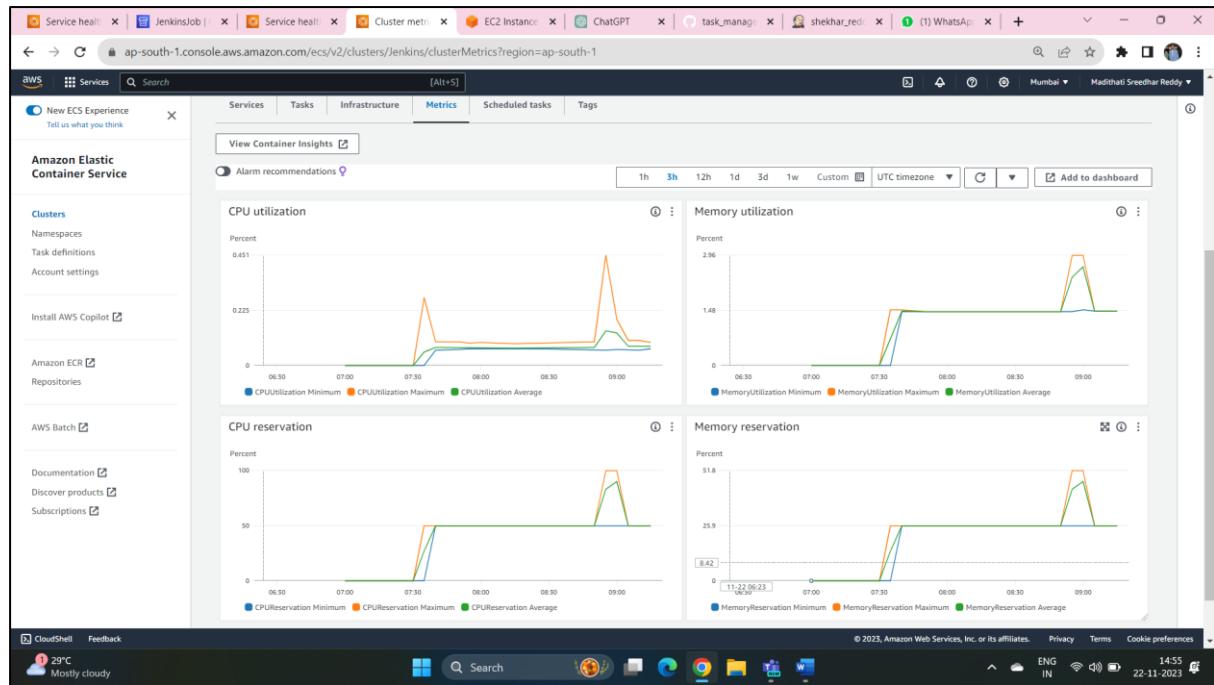


- Everytime we Release the changes It will be running in one instance that is named as Blue with target group TG-1
- Another will be the old Version of our Application running in another instance that is named as Green with Target group TG-2
- If anything goes wrong the loadbalancer will reroute the traffic to the healthy instance.

## Monitoring:

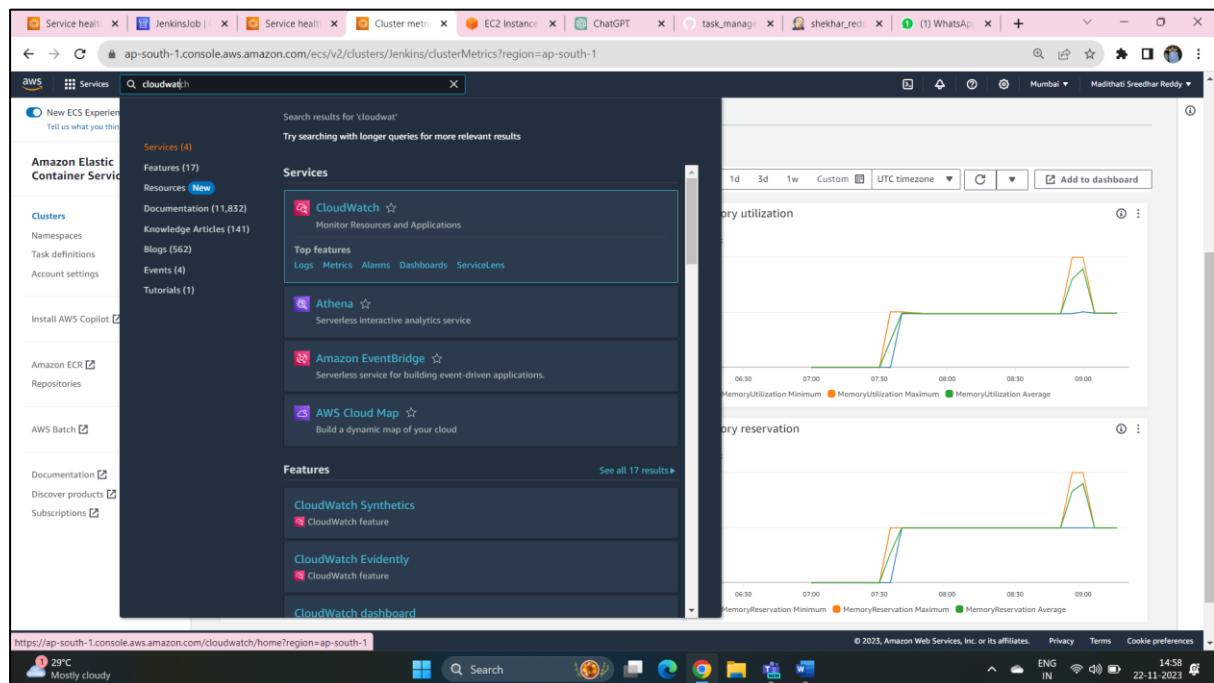
### 1. View the metrics

- Go to the ECS console
- Navigate to our cluster
- Then click on metrics that will show the ECS metrics like CPU Utilization.



### 2. View Logs in CloudWatch Console:

- Search for the CloudWatch Console and Navigate to it.



- Navigate to "Logs" in the left navigation pane.
- In the "Log groups" section, find and select the log group associated with your ECS task.
- View and search logs within the log group. Logs are organized by log streams, with each stream representing an instance of your ECS task.

The screenshot shows the AWS CloudWatch Logs console. On the left, the navigation pane includes sections for CloudWatch, Favorites and recent, Metrics, and Application monitoring. Under Logs, the Log groups section is selected, showing a list of log groups. One group, "arn:aws:logs:ap-south-1:186254747902:log-group:/aws/ecs/containerinsights/Jenkins/performance:", is selected and detailed on the right. This panel shows metrics like stored bytes (1.11 MB), creation time (8 days ago), retention (1 day), and subscription filters (0). Below this, the Log streams section displays a table of 8 log streams, each with a name, last event time, and timestamp. The streams include AgentTelemetry, ClusterTelemetry-Jenkins, ServiceTelemetry-JenkinsJobs, and various Jenkins agent logs.

- Navigate to the path mentioned below
- This helps in knowing the live utilization metrics

This screenshot shows the CloudWatch Logs console with the path "CloudWatch > Log groups > /aws/ecs/containerinsights/Jenkins/performance > AgentTelemetry-66e1817e9a5147d58f7d97172c8aa2ac". The left sidebar shows the same navigation structure as the previous screenshot. The main area displays the "Log events" table. The first event is timestamped "2023-11-22T14:26:09.000+05:30" and contains a JSON object representing a log entry from an ECS container. It includes fields like Version, Type, ContainerName, TaskId, TaskDefinitionFamily, TaskDefinitionRevision, ContainerInstanceArn, EC2InstanceId, ServiceName, Image, ContainerStatus, Timestamp, ContainerHealth, CPUAllocated, CPUReserved, MemoryAllocated, MemoryReserved, StorageAllocated, NetworkRttLatency, NetworkRttDropped, NetworkRttPackets, NetworkTxDrop, NetworkTxDropped, and NetworkTxFrags. Subsequent events show similar log entries for different tasks and instances.

- We can use these logs for troubleshooting issues on any failures.
- We can create Alarm for the events that monitors the cluster and if something goes wrong then It will trigger the tigger the alarm and send an email

- The Steps for doing this is:
  - In the Same Console Navigate to Alarm

The screenshot shows the AWS CloudWatch Alarms page. The left sidebar includes sections for Favorites and recent dashboards, Alarms (with one in alarm), Logs, Metrics (All metrics, Explorer, Streams), X-Ray traces, Events, Rules, Event Buses, and Application monitoring (ServiceLens Map, Resource Health, Internet Monitor). The main content area displays 'Alarms (2)'. A search bar at the top allows filtering by Name, State, Last state update, Conditions, and Actions. Two alarms are listed:

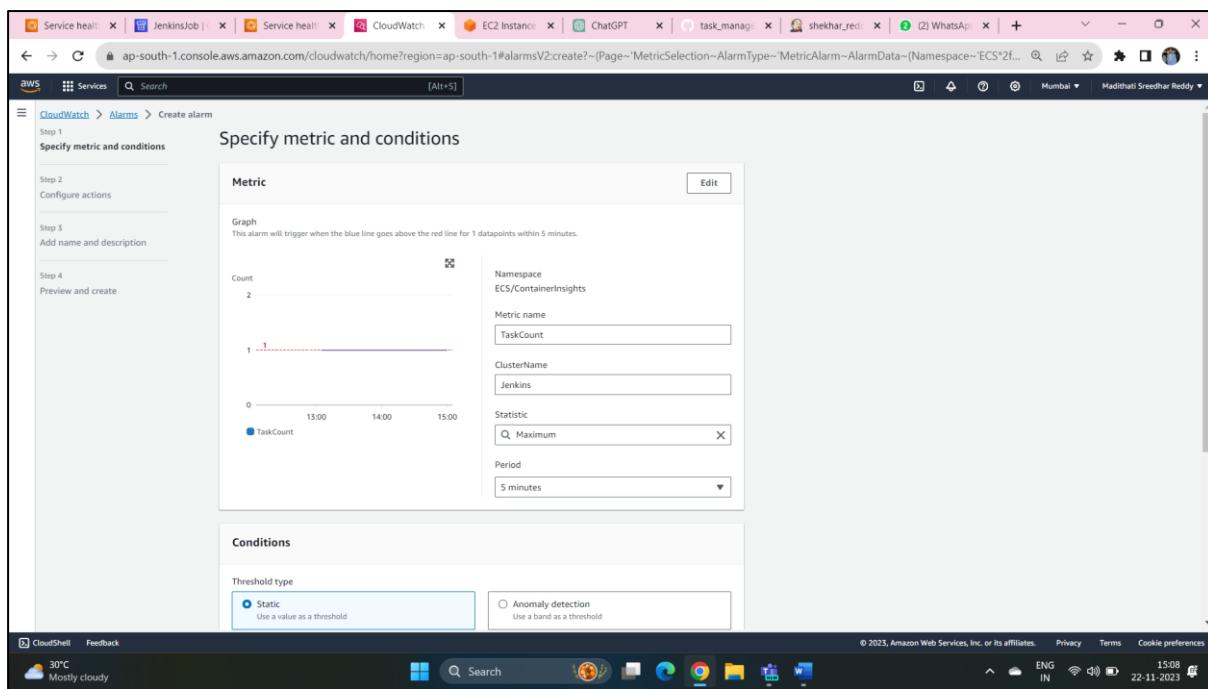
- TargetTracking-Infra-ECS-Cluster-Jenkins-a2feef9:** ECSAutoScalingGroup-y1eHvCScnj-AlarmLow-d078120c-7997-45b6-acf1-417569a6fd67. State: In alarm. Last updated: 2023-11-22 09:19:02. Condition: CapacityProviderReservation < 100 for 15 datapoints within 15 minutes. Actions: Enabled.
- ecs:** ecs. State: In alarm. Last updated: 2023-11-22 07:39:20. Condition: ContainerInstanceCount > 1 for 2 datapoints within 2 minutes. Actions: Enabled.

The bottom status bar shows the URL as <https://ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#alarmsV2:create>, the date as 22-11-2023, and the time as 15:06.

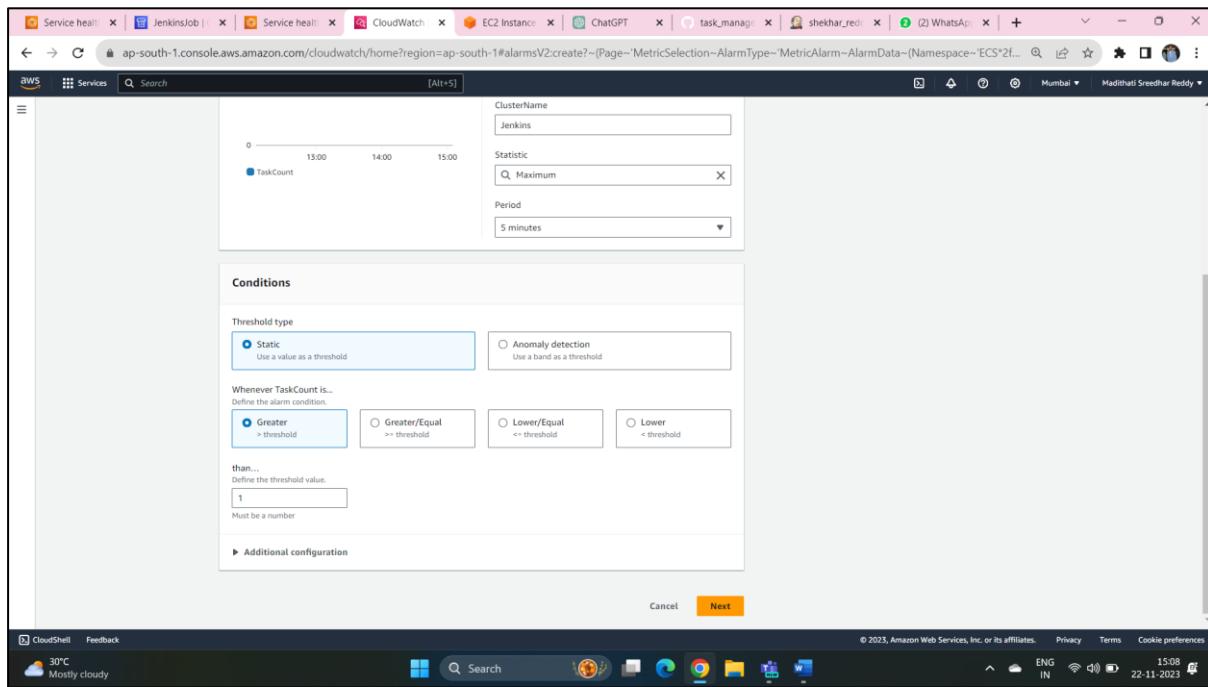
- Click on Create Alarm
- Then select the metric you want to create alarm

The screenshot shows the 'Select metric' dialog box from the AWS CloudWatch Metrics console. The title bar says 'Create alarm' and the URL is [https://ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#alarmsV2:create?~\(Page~MetricSelection~AlarmType~MetricAlarm~AlarmData~\(Metrics~\(~\)~Alarm...](https://ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#alarmsV2:create?~(Page~MetricSelection~AlarmType~MetricAlarm~AlarmData~(Metrics~(~)~Alarm...). The dialog has tabs for 'Browse', 'Query', 'Graphed metrics (1)', 'Options', and 'Source'. The 'Graphed metrics (1)' tab is selected, showing a list of metrics from the Jenkins instance. The 'TaskCount' metric under the Jenkins category is selected and highlighted with a blue border. Other metrics listed include ServiceCount, ContainerInstanceCount, StorageWriteBytes, NetworkTxBytes, CpuUtilized, StorageReadBytes, MemoryReserved, MemoryUtilized, NetworkRxBytes, CpuReserved, and ServiceCount. At the bottom right are 'Cancel' and 'Select metric' buttons.

- Then Configure the metrics and conditions



- Specify the Threshold value for it



- Click on Next

The screenshot shows the AWS CloudWatch Metrics console with the 'Configure actions' step selected. In the 'Notification' section, the 'In alarm' option is chosen. Below it, a new SNS topic is being created with the name 'Default\_CloudWatch\_Alarms\_Topic'. Two email addresses are listed as endpoints: 'mudit@srreddy123@gmail.com' and 'user1@example.com, user2@example.com'. There are buttons for 'Create topic' and 'Add notification'.

- It will send the Subscription mail Accept it

The screenshot shows a Gmail inbox with an email from 'AWS Notifications <no-reply@sns.amazonaws.com>' titled 'AWS Notification - Subscription Confirmation'. The email body contains the following text:  
You have chosen to subscribe to the topic:  
<arn:aws:sns:ap-south-1:862547479026:ecs>  
To confirm this subscription, click or visit the link below (if this was in error no action is necessary):  
[Confirm subscription](#)  
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

- Click on Create topic

The screenshot shows the AWS SNS console with a topic named 'ecs'. The 'Subscriptions' tab is active, displaying a single confirmed subscription. The subscription details are as follows:

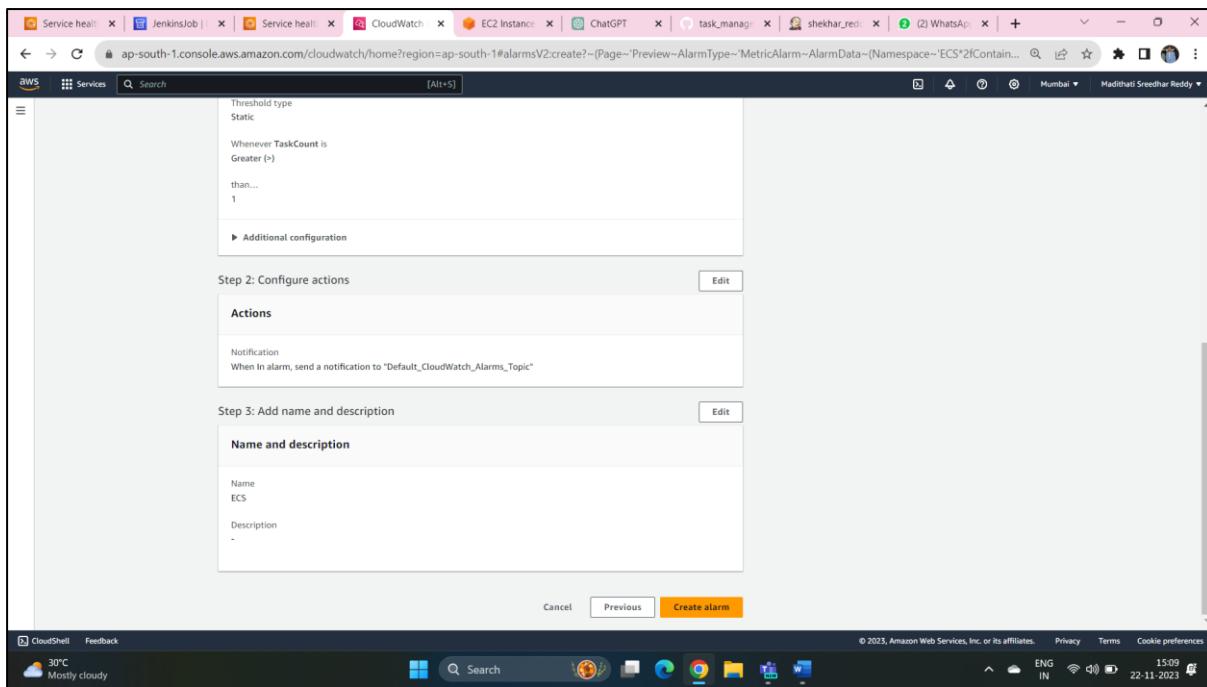
ID	Endpoint	Status	Protocol
pbfd5497-46be-4f8d-aace-2d564a313a0c	madithatreedhar123@gmail.com	Confirmed	EMAIL

The screenshot shows the AWS CloudWatch Metrics console with a new alarm being created. The 'Add name and description' step is active, with the following input fields:

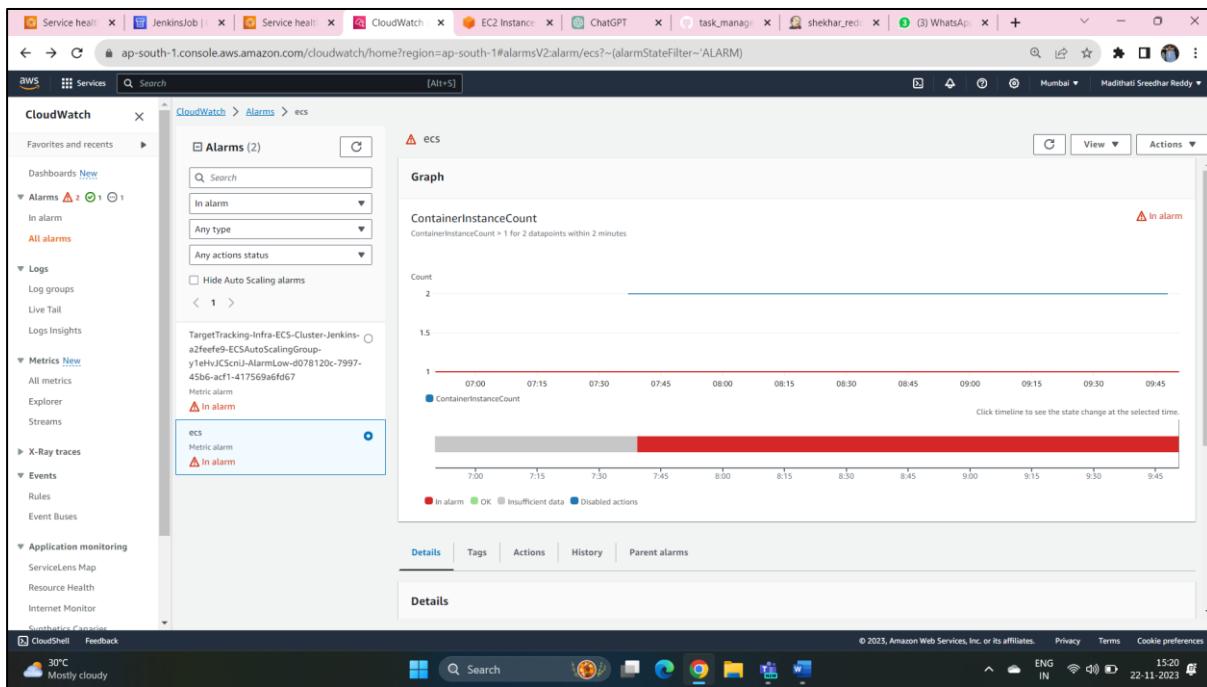
- Alarm name:** EC2
- Alarm description - optional:** # This is an H1  
\*\*double asterisks will produce strong character\*\*  
This is [an example](https://example.com) inline link.

At the bottom, there are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' being highlighted.

- Click on Next for Preview



- Click on Create Alarm
- Initially It would be in Insufficient data
- After the mentioned time is met, it will turn to the Alarm state



- If Anything goes Wrong It will change from Alarm to the OK state
- Then it will send an email defining the breach of the mentioned metric.

