

*****Follow me :- Vinay Pramanik in LinkedIn for more *****

Project 1

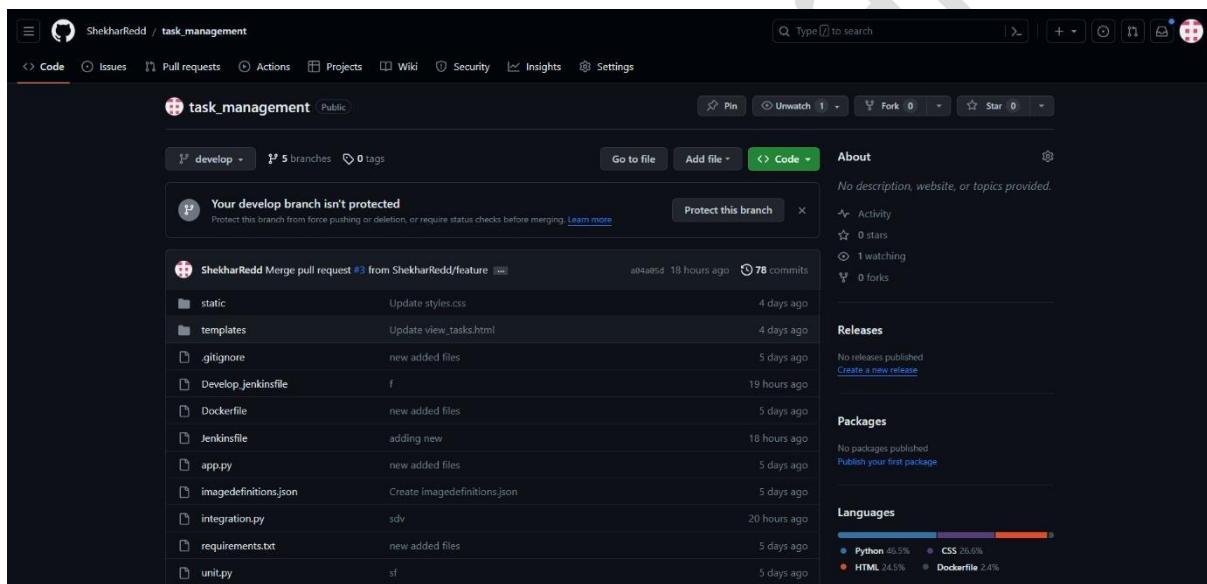
Continuous Integration and Continuous Deployment (CI/CD) Pipeline

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:**

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

```
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.

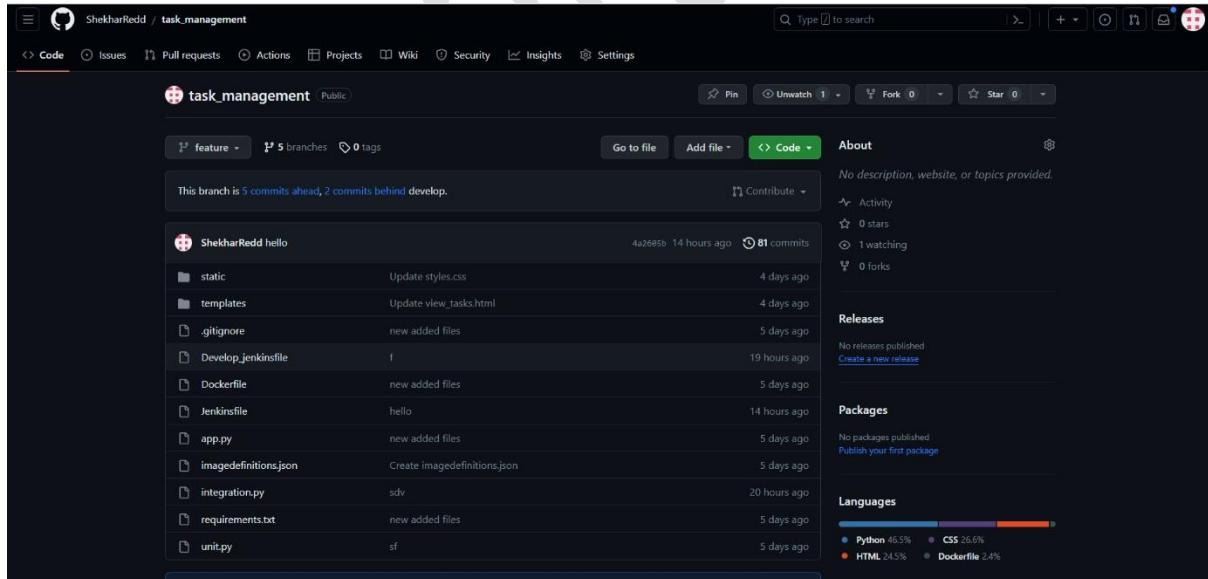
Feature branches:

- 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.**

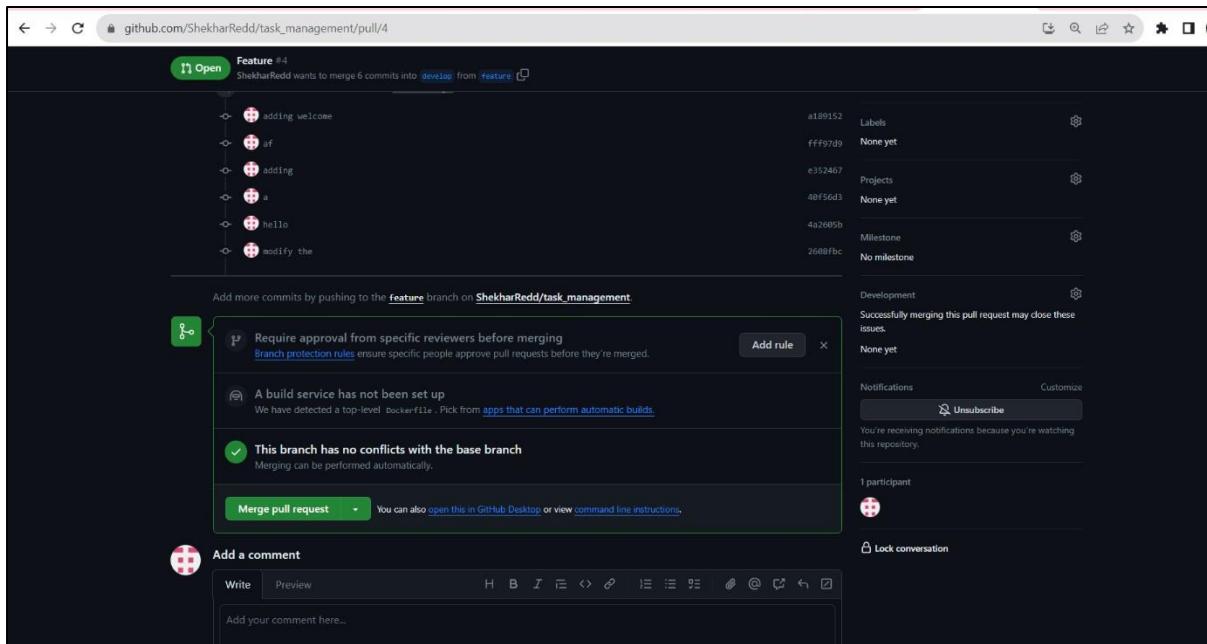
```
git push origin feature
```



- 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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:

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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 :

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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. On the left, there's a sidebar with options like Instances, Images, and Elastic Block Store. The main area has a heading 'Resources' with statistics for Instances (running), Dedicated Hosts, Elastic IPs, Instances (all states), Key pairs, Load balancers, Placement groups, Security groups, Snapshots, and Volumes. Below this is a callout for Microsoft SQL Server Always On availability groups. A large central box contains the 'Launch instance' section, which includes a note about launching a virtual server in the cloud, a 'Launch Instance' button (which is highlighted with a red box), and a note that instances will launch in the US East (Ohio) Region. To the right, there's a 'Service health' section showing the region as US East (Ohio) and status as 'This service is operating normally'. Further right is an 'Account attributes' section with options like VPC, Default VPC, and Settings, and an 'Explore AWS' section with links for custom AMIs and best price-performance.

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

The screenshot shows the AWS Marketplace search results for 'Amazon Machine Image (AMI)'. It features a search bar and tabs for 'Recents' and 'Quick Start'. Several AMI icons are shown: Amazon Linux (selected and highlighted with a red box), macOS, Ubuntu, Windows, Red Hat, and others. A 'Browse more AMIs' link is also present. Below the search bar, a detailed view of the 'Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type' is shown. This view includes the AMI ID (ami-09d3b3274b6c5d4aa), a note that it's 'Free tier eligible', a description (Amazon Linux 2 Kernel 5.10 AMI 2.0.20221004.0 x86_64 HVM gp2), architecture (64-bit (x86)), AMI ID (ami-09d3b3274b6c5d4aa), and a 'Verified provider' badge. The entire 'Amazon Linux 2 AMI (HVM)' section is also highlighted with a red box.

- Scroll down and select the key pair you have already created or create a new key pair.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- Select an existing security group or create a new security group □ Select the security group.
- Select **Launch Instance**.

The screenshot shows the 'Launch Instance' wizard in the AWS Management Console. It's divided into two main sections: 'Key pair (login)' and 'Summary'.

Key pair (login) Info: A dropdown menu for 'Key pair name - required' is highlighted with a red box. Below it, there's a link to 'Create new key pair'.

Network settings Info: This section includes fields for 'Network' (vpc-5d7a3227), 'Subnet' (No preference), 'Auto-assign public IP' (Enable), and 'Firewall (security groups)'. The 'Select existing security group' button is highlighted with a red box. A tooltip for the 'Free tier' is visible on the right.

Summary: Shows 'Number of instances' (1), 'Software Image (AMI)' (Amazon Linux 2 Kernel 5.10 AMI...), 'Virtual server type (instance type)' (t2.micro), 'Firewall (security group)' (None), 'Storage (volumes)' (1 volume(s) - 8 GiB), and a 'Launch Instance' button.

- 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.

The screenshot shows the 'Instances' page in the AWS Management Console. The left sidebar has 'Instances' selected. The main table displays one instance:

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

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

Follow for more: Vinay Pramanik (LinkedIn)

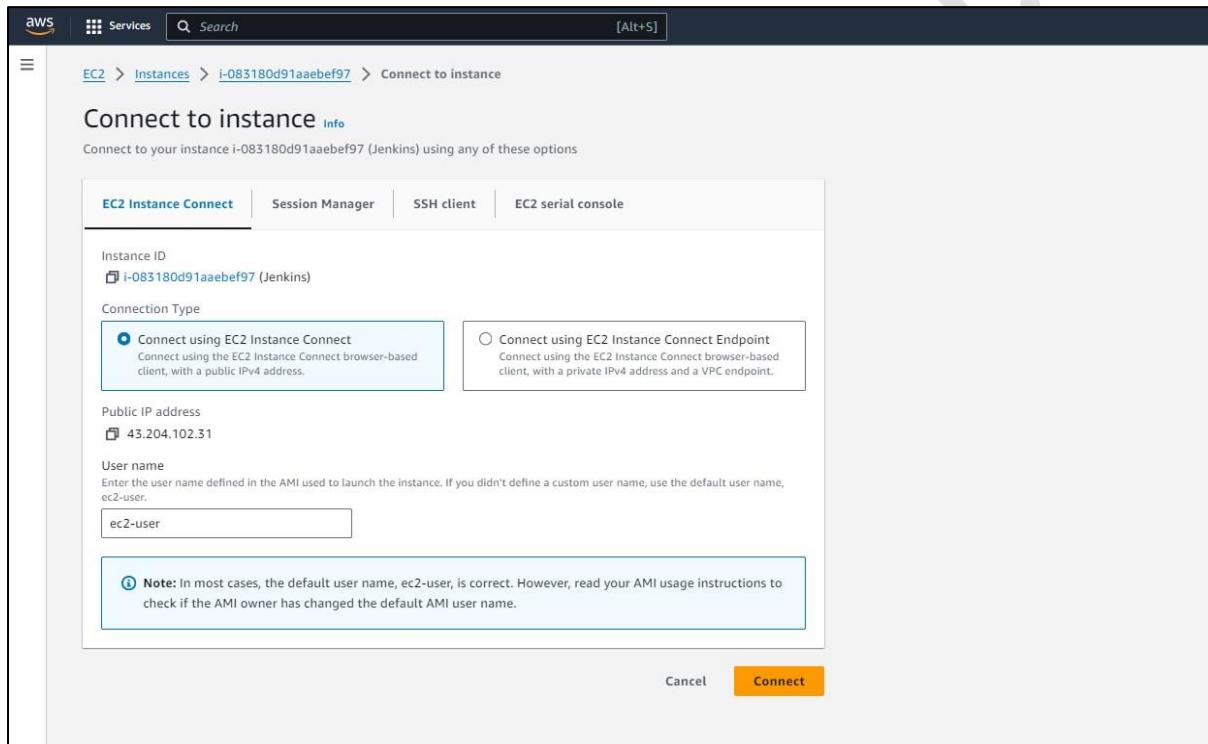
[AWS Cloud Architect]

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
,
~\ #####
~~ \#####
~~ \|#
~~ V-' ->
~~ / \
~~ / \
/m/
Last login: Thu Nov 16 04:57:10 2023 from 13.233.177.4
[ec2-user@ip-172-31-12-85 ~]$
```

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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.

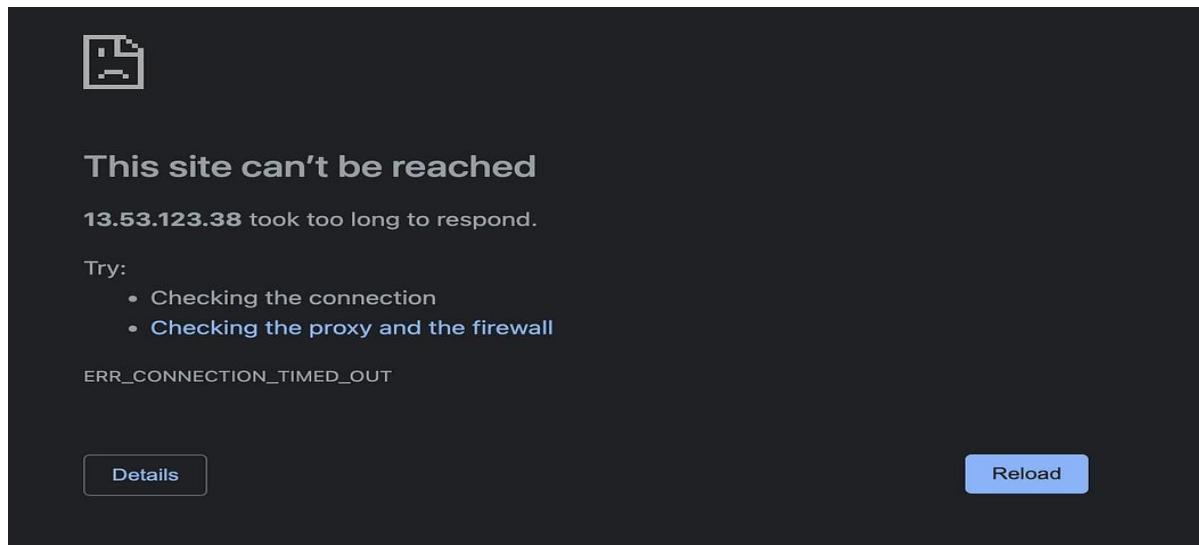
```
sudo groupadd docker sudo
usermod -aG docker ${USER}
```

- 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 Info	Protocol Info	Port range	Source Info	Description - optional Info
sgr-04b795001383ee019	HTTP	TCP	80	Cus... ▾ 0.0.0.0/0 X	<input type="text"/> Delete
sgr-0ed22528c9bca4326	SSH	TCP	22	Cus... ▾ 0.0.0.0/0 X	<input type="text"/> Delete

[Add rule](#)

A screenshot of the AWS CloudFormation console showing the "Inbound rules" section. It displays two existing security group rules. Rule 1 is for HTTP on port 80, allowing traffic from 0.0.0.0/0. Rule 2 is for SSH on port 22, also allowing traffic from 0.0.0.0/0. Both rules have a "Delete" button next to them. At the bottom left is a "Add rule" button.

- 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

A screenshot of the Jenkins "Getting Started" page. It features a large title "Unlock Jenkins" and a descriptive paragraph about finding the initial admin password. It includes the path "/var/jenkins_home/secrets/initialAdminPassword" in red text. Below this, instructions say to copy the password from either the log or this file and paste it into a text input field labeled "Administrator password".

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- 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:

- Left Sidebar:** Includes links for New Item, People, Build History, Manage Jenkins, and My Views.
- Build Queue:** Shows "No builds in the queue."
- Build Executor Status:** Shows 1 Idle and 2 Idle.
- Recent Builds:** A table with columns S (Status), W (Weather), Name, Last Success, Last Failure, and Last Duration. The data is as follows:

S	W	Name	Last Success	Last Failure	Last Duration
Green checkmark	Sunny	JenkinsBuild	1 day 14 hr #1	N/A	0.69 sec
Green checkmark	Sunny	Project1	4 days 17 hr #2	N/A	0.3 sec
Green checkmark	Cloudy	SamplecodePipeline	3 days 20 hr #3	3 days 20 hr #2	0.15 sec

- Bottom Navigation:** Includes links for Icon legend, Atom feed for all, Atom feed for failures, and Atom feed for just 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: one for Jenkins and another for Jenkins. Both tokens have a 'Delete' button next to them. 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 'settings/tokens/new' page for generating a new personal access token. It includes fields for Note (Jenkins), What's this token for?, Expiration (60 days), and Select scopes. The Select scopes section lists various GitHub API permissions with checkboxes. The 'repo' scope is selected, granting full control of private repositories, access to commit status, deployment status, public repositories, repository invitations, and security events.

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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

<input type="checkbox"/> manage_runners:enterprise	Manage enterprise runners and runner groups
<input type="checkbox"/> manage_billing:enterprise	Read and write enterprise billing data
<input type="checkbox"/> read:enterprise	Read enterprise profile data
<input checked="" type="checkbox"/> audit_log	Full control of audit log
<input type="checkbox"/> read:audit_log	Read access of audit log
<input checked="" type="checkbox"/> codespace	Full control of codespaces
<input type="checkbox"/> codespace:secrets	Ability to create, read, update, and delete codespace secrets
<input checked="" type="checkbox"/> copilot	Full control of GitHub Copilot settings and seat assignments
<input type="checkbox"/> manage_billing:copilot	View and edit Copilot for Business seat assignments
<input checked="" type="checkbox"/> project	Full control of projects
<input type="checkbox"/> read:project	Read access of projects
<input checked="" type="checkbox"/> admin:gpg_key	Full control of public user GPG keys
<input type="checkbox"/> write:gpg_key	Write public user GPG keys
<input type="checkbox"/> read:gpg_key	Read public user GPG keys
<input checked="" type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

[Generate token](#) [Cancel](#)

- 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”.

Not secure | 43.204.102.31:8002/view/all/new/job

Jenkins

Dashboard > All >

Enter an item name

Narasimha » Required field

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.

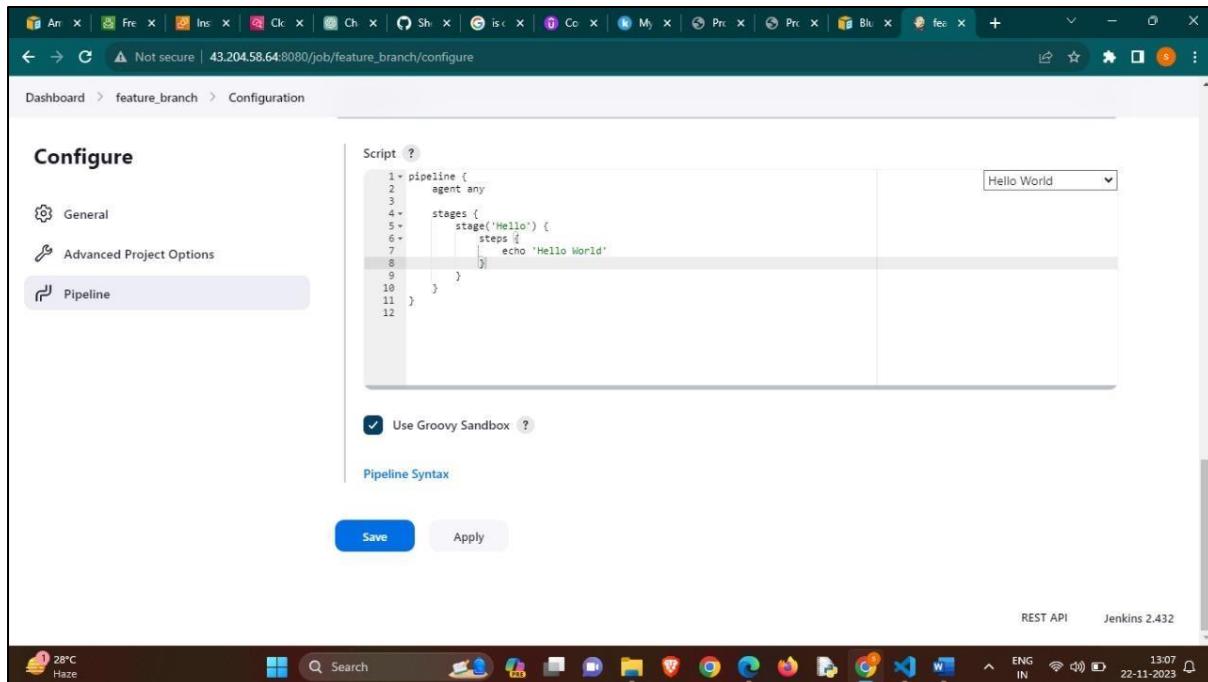
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

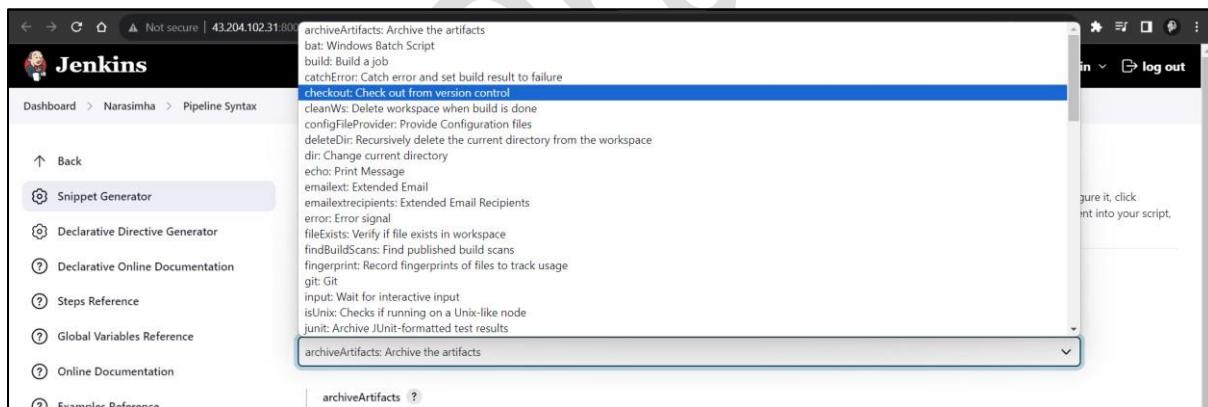
- Run a sample code from pipeline script. This is the sample code you can get.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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



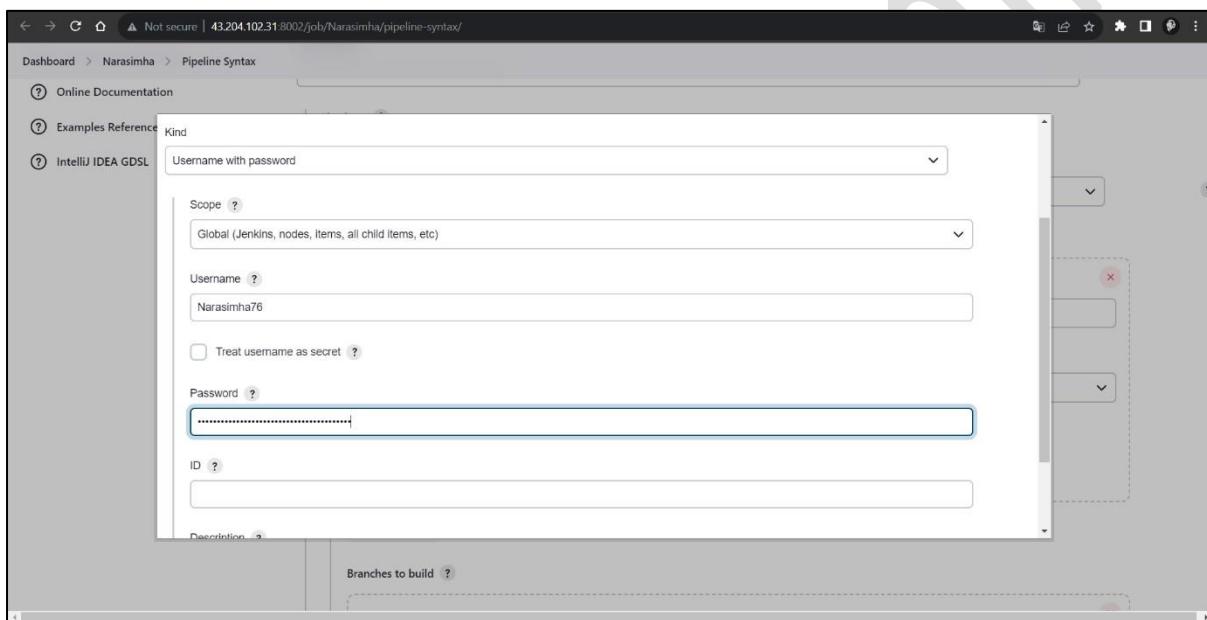
Add GitHub repository URL here.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



- 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

Include in polling? [?](#)

Include in changelog? [?](#)

Generate Pipeline Script

```
checkout scmGit(branches: [[name: '/master']], extensions: [], userRemoteConfigs: [[credentialsId: 'Narasimha9', url: 'https://github.com/Narasimha76/task.git']])
```

Dashboard > Narasimha > Configuration

Pipeline

Configure

General

Advanced Project Options

Pipeline

Definition

Pipeline script

Script ?

```
1+ pipeline {
2+   agent any
3+
4+   stages {
5+     stage('Hello') {
6+       steps {
7+         checkout scmGit(branches: [[name: '/master']], extensions: [], userRemoteConfigs: [[credentialsId: 'hello1', url: 'https://github.com/Narasimha76/task.git']])}
8+       }
9+     }
10+   }
11+ }
```

Use Groovy Sandbox ?

Pipeline Syntax

Save **Apply**

- 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.

Narasimha76 / task

Type to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type

Secret

Which events would you like to trigger this webhook?

Just the push event.

Send me everything

Let me select individual events.

Active

We will deliver event details when this hook is triggered.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- 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 repository settings page for 'Narasimha76 / task'. The 'Webhooks' tab is selected. A new webhook entry is visible, with the URL 'http://43.204.102.31:8002/github-w... (push)' and status 'Active'. There are 'Edit' and 'Delete' buttons next to the entry.

- Ensure the webhook is set to be active.

Test the setup:

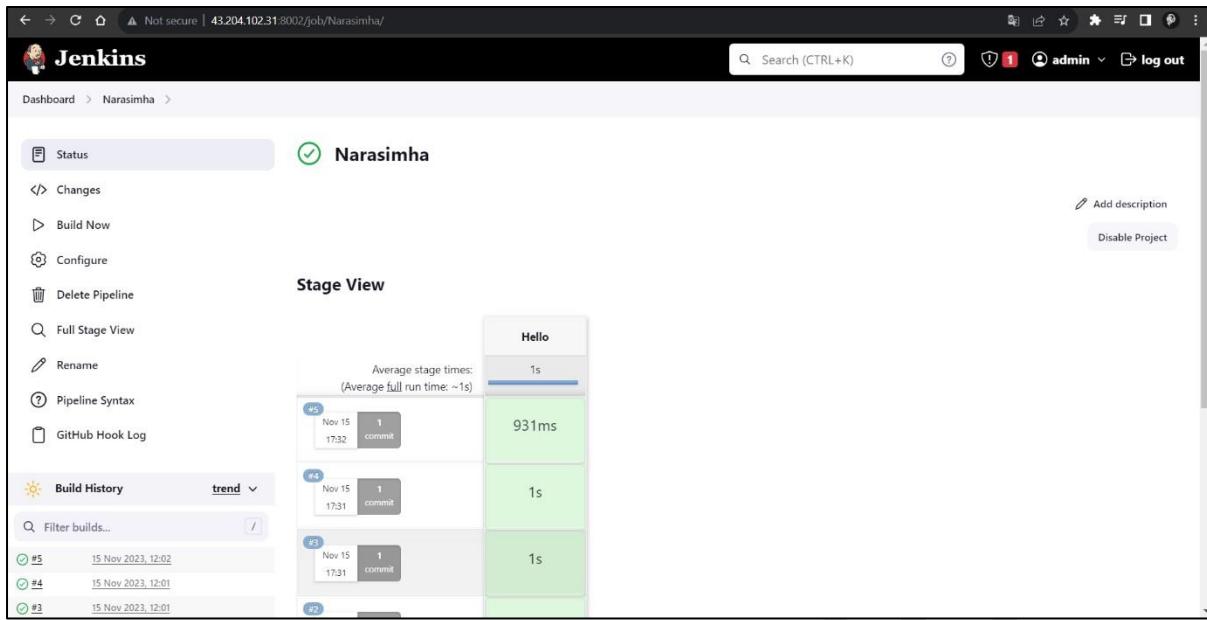
- Make a small change (e.g., push a new commit) to your GitHub repository.

The screenshot shows the GitHub commit changes dialog box. The 'Commit message' field is filled with 'Update index.html'. The 'Extended description' field is empty. Below the fields are two radio button options: 'Commit directly to the master branch' (selected) and 'Create a new branch for this commit and start a pull request'. At the bottom are 'Cancel' and 'Commit changes...' buttons.

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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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.

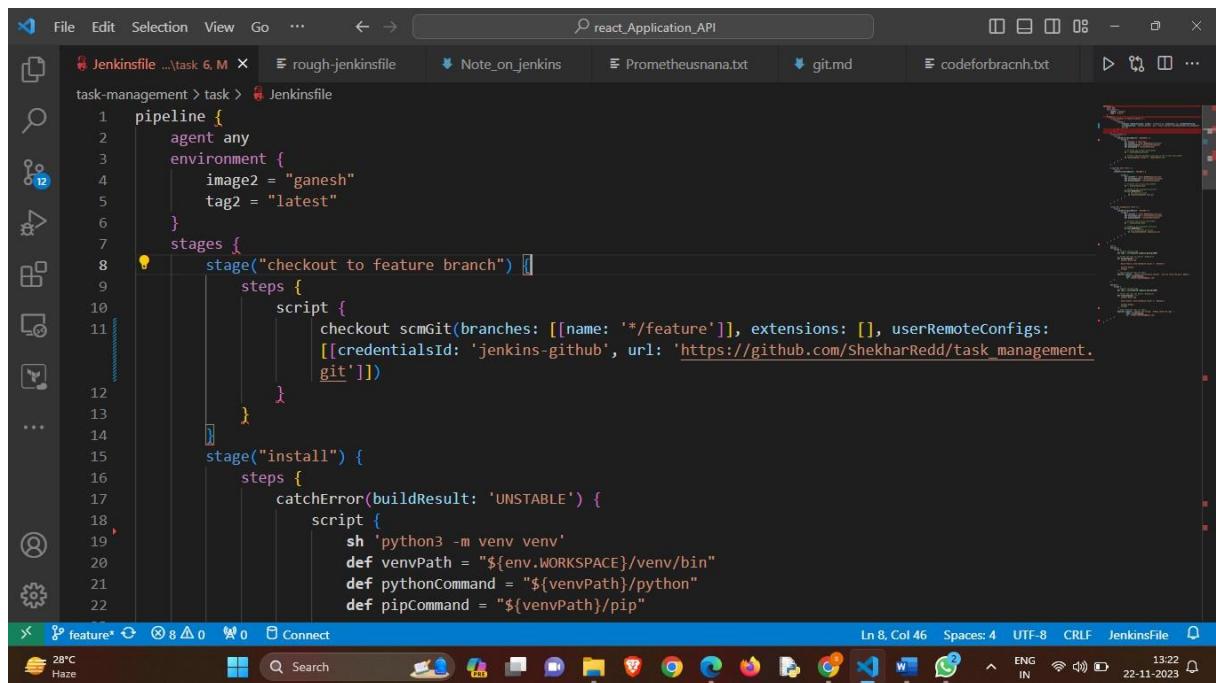
- **Configure Feature Branch Job**

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.

Follow for more: Vinay Pramanik (LinkedIn)

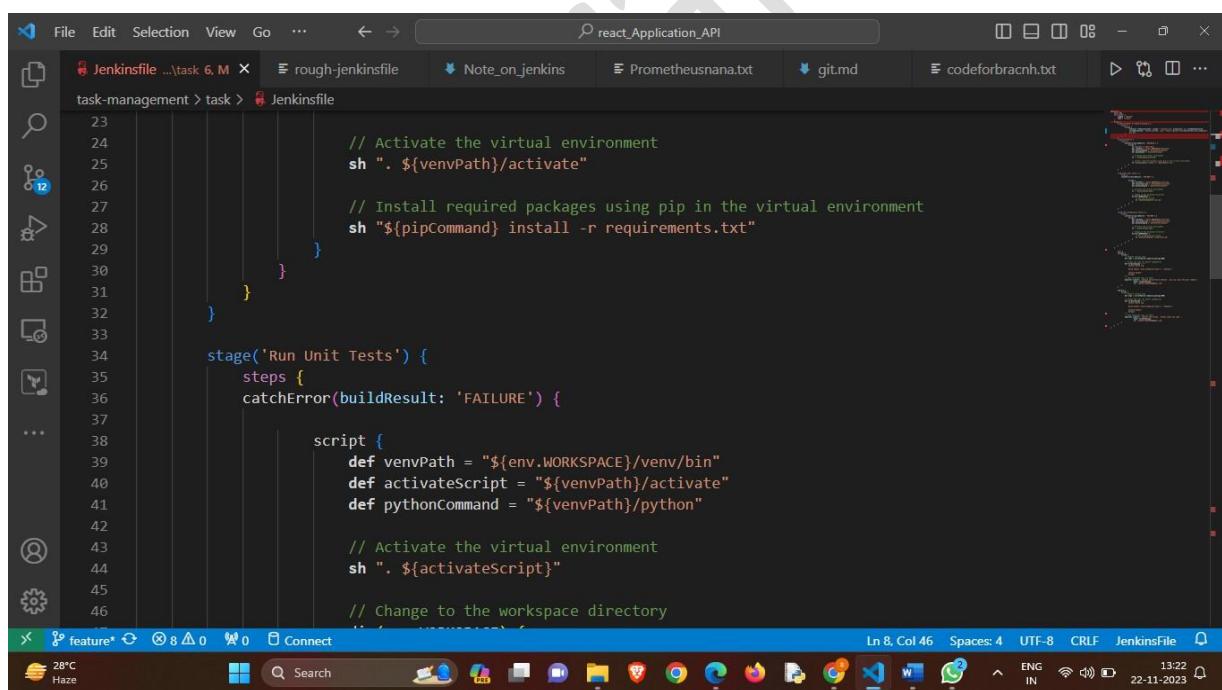
[AWS Cloud Architect]

- Jenkins Pipeline Script



A screenshot of a code editor window titled "react_Application_API". The main pane displays a Jenkins Pipeline script. The script defines a pipeline with an agent "any", environment "image2 = 'ganesh'" and "tag2 = 'latest'", and stages for "checkout to feature branch" and "install". The "install" stage includes catching unstable builds and running pip commands to install packages from requirements.txt.

```
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 = "${venvPath}/pip"
23                     }
24                 }
25             }
26         }
27     }
28 }
29
30 }
31
32 }
33
34 stage('Run Unit Tests') {
35     steps {
36         catchError(buildResult: 'FAILURE') {
37             script {
38                 def venvPath = "${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     }
49 }
```



A screenshot of a code editor window titled "react_Application_API". The main pane displays a Jenkins Pipeline script. The "install" stage now includes activating the virtual environment and installing packages. A new stage "Run Unit Tests" has been added, which activates the virtual environment and changes to the workspace directory.

```
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                         sh ". ${venvPath}/activate"
24                         sh "${pipCommand} install -r requirements.txt"
25                     }
26                 }
27             }
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     }
49 }
```

Follow for more: Vinay Pramanik (LinkedIn)

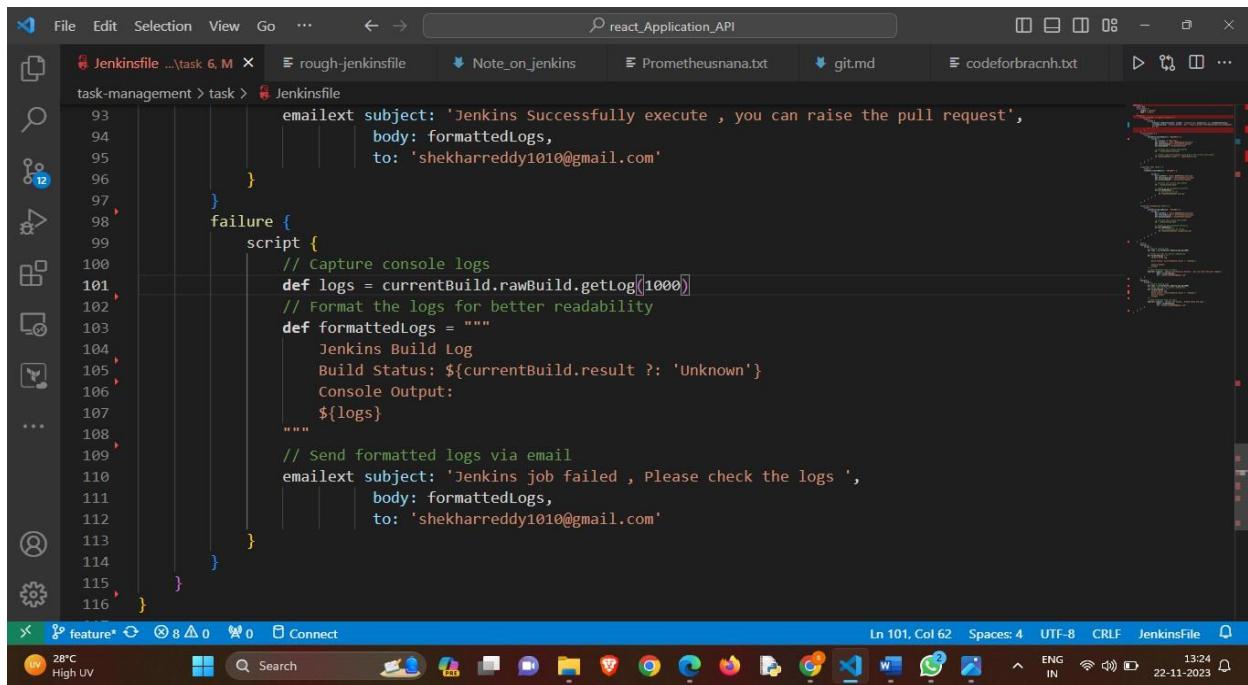
[AWS Cloud Architect]

```
task-management > task > Jenkinsfile
47     dir(env.WORKSPACE) {
48         // Run unit.py script
49         sh "${pythonCommand} unit.py"
50     }
51 }
52 }
53 }
54 }
55
56 stage('Run Integration Tests') {
57     steps {
58         catchError(buildResult: 'FAILURE') {
59             script {
60                 def venvPath = "${env.WORKSPACE}/venv/bin"
61                 def activateScript = "${venvPath}/activate"
62                 def pythonCommand = "${venvPath}/python"
63
64                 // Activate the virtual environment
65                 sh ". ${activateScript}"
66
67                 // Change to the workspace directory
68                 dir(env.WORKSPACE) {
69                     // Run integration.py script
70                     sh "${pythonCommand} integration.py"
71                 }
72             }
73         }
74     }
75 }
76
77
78 post {
79     success {
80         script {
81             // Capture console logs
82             def logs = currentBuild.rawBuild.getLog(1000)
83
84             // Format the logs for better readability
85             def formattedLogs = """
86                 Jenkins Build Log
87
88                 Build Status: ${currentBuild.result ?: 'Unknown'}
89
90                 Console Output:
91                 ${logs}
92
93             """
94
95             // Send formatted logs via email
96             emailext subject: 'Jenkins Successfully executed ., you can raise the pull request'.
97         }
98     }
99 }
```

```
task-management > task > Jenkinsfile
70     }
71 }
72 }
73 }
74 }
75 }
76
77
78 post {
79     success {
80         script {
81             // Capture console logs
82             def logs = currentBuild.rawBuild.getLog(1000)
83
84             // Format the logs for better readability
85             def formattedLogs = """
86                 Jenkins Build Log
87
88                 Build Status: ${currentBuild.result ?: 'Unknown'}
89
90                 Console Output:
91                 ${logs}
92
93             """
94
95             // Send formatted logs via email
96             emailext subject: 'Jenkins Successfully executed ., you can raise the pull request'.
97         }
98     }
99 }
```

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



```
emailext subject: 'Jenkins Successfully execute , you can raise the pull request',  
        body: formattedLogs,  
        to: 'shekharreddy1010@gmail.com'  
  
    }  
failure {  
    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 job failed , Please check the logs ',  
                body: formattedLogs,  
                to: 'shekharreddy1010@gmail.com'  
    }  
}
```

EMAIL Configuration

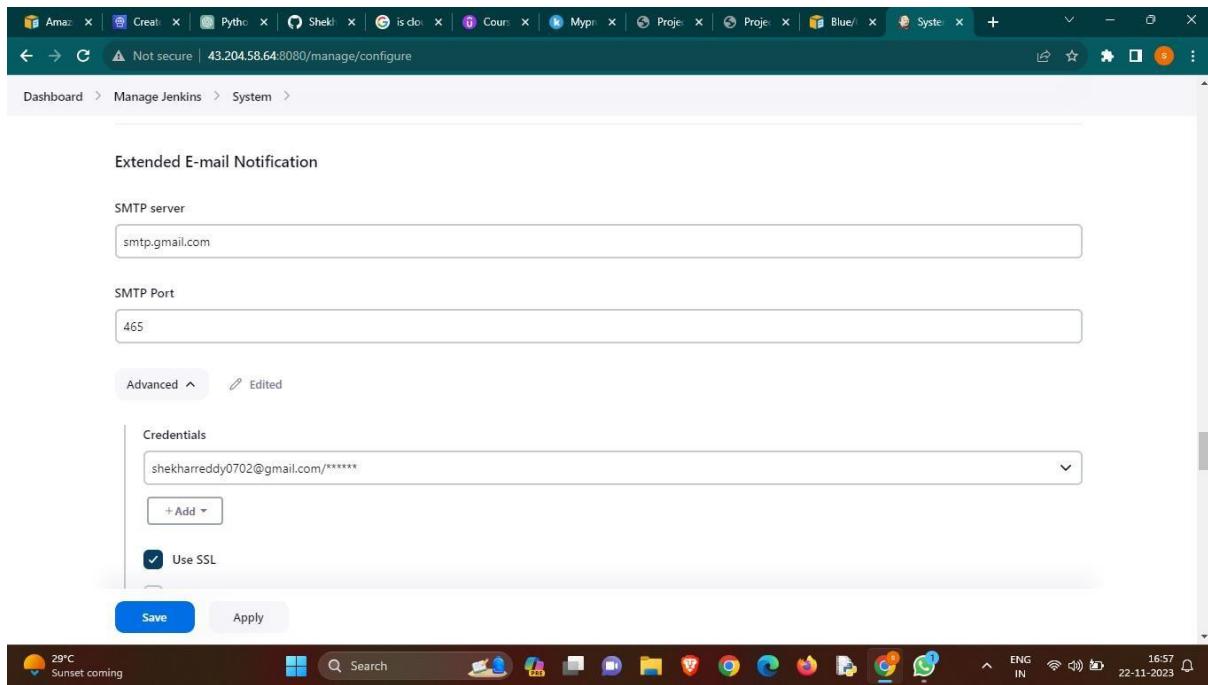
- Install E-mail Notification and Extended E-mail Notification plugins □ E-mail Notification: This plugin Mailer, allows users to send basic emails.
- Extended E-mail Notification: This plugin Email Extension Plugin, allows users to customize when emails are sent, who receives them, and the content of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

The screenshot shows the 'E-mail Notification' configuration page in Jenkins. The 'SMTP server' field contains 'smtp.gmail.com'. The 'Default user e-mail suffix' field is empty. Under the 'Advanced' section, 'Use SMTP Authentication' is checked, and the 'User Name' is 'shekharreddy0702@gmail.com'. The 'Password' field is masked as 'Concealed'. There are 'Save' and 'Apply' buttons at the bottom.

The screenshot shows the 'System' configuration page in Jenkins. Under 'E-mail Configuration', 'Use SSL' is checked, while 'Use TLS' is unchecked. The 'SMTP Port' is set to '465'. The 'Reply-To Address' and 'Charset' fields are empty. A checkbox for 'Test configuration by sending test e-mail' is unchecked. There are 'Save' and 'Apply' buttons at the bottom. The status bar at the bottom right shows 'REST API' and 'Jenkins 2.432'.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



Run the Feature Branch Job

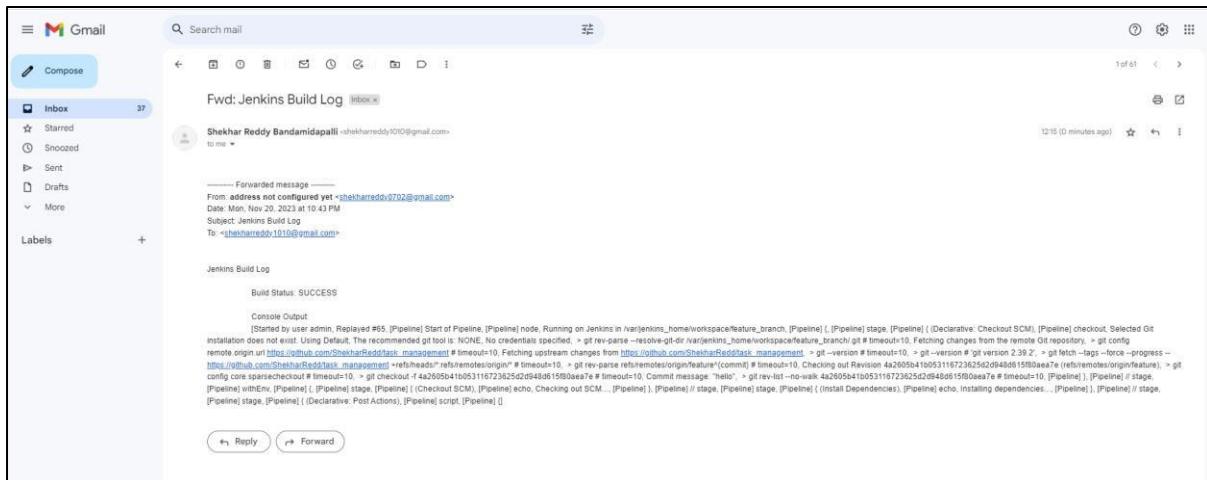
- Trigger a build manually or configure webhooks for automatic triggering on code changes.
- Monitor the build progress on the Jenkins dashboard.
- Review the console output and logs for each stage.
- Receive email notifications on build success or failure.

The screenshot shows the Jenkins pipeline interface for the 'feature_branch' job. On the left, there's a sidebar with options like 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', 'Pipeline Syntax', and 'GitHub Hook Log'. The main area is titled 'Stage View' and displays a grid of stages for two builds: #71 (Nov 21, 21:10) and #70 (Nov 21, 13:07). The stages are: Declarative: Checkout SCM (1s), checkout to feature branch (234ms), install (5s), Run Unit Tests (1s), Run Integration Tests (1s), and Declarative: Post Actions (4s). The grid also shows average stage times: 1s, 137ms, 4s, 1s, 1s, and 3s respectively. A note at the top says 'Average stage times: (Average full run time: ~16s)'.

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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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.

Follow for more: Vinay Pramanik (LinkedIn)

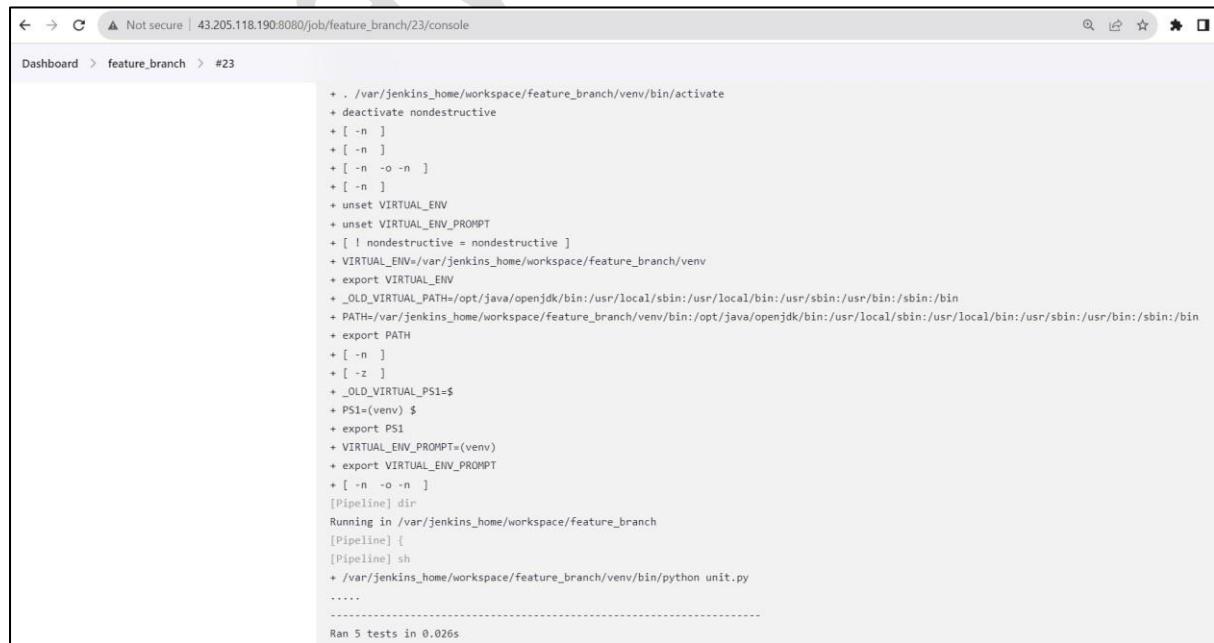
[AWS Cloud Architect]

```

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 console log for a pipeline job named 'feature_branch' with build number #23. The log output is as follows:

```

Dashboard > feature_branch > #23
Not secure | 43.205.118.190:8080/job/feature_branch/23/console

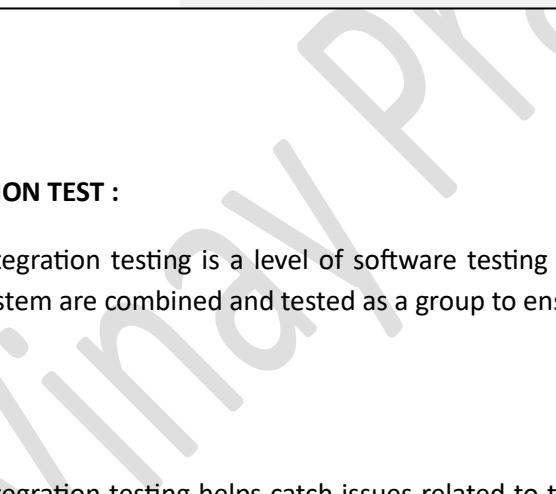
+ . /var/jenkins_home/workspace/feature_branch/venv/bin/activate
+ deactivate nondestructive
+ [ -n ]
+ [ -n ]
+ [ -n -o -n ]
+ [ -n ]
+ unset VIRTUAL_ENV
+ unset VIRTUAL_ENV_PROMPT
+ [ i nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/jenkins_home/workspace/feature_branch/venv
+ export VIRTUAL_ENV
+ _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_PSI=$
+ PSI=(venv) $
+ export PSI
+ 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

```

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- 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.



A screenshot of a Jenkins job console output. The URL is 43.205.118.190:8080/job/feature_branch/27/console. The output shows a terminal session with environment variable setup, directory navigation, and execution of unit.py. It ends with a failure message due to an assertion error, followed by a traceback, and concludes with a failed exit code.

```

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

Dashboard > feature_branch > #27

+ _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
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 the GitHub Code Blame interface for the file `integration.py`. The code is a Python unittest test case for a task management application. It includes methods for adding and viewing tasks, and deleting tasks. The GitHub Copilot logo is visible at the top right.

```

1 import unittest
2 from datetime import datetime
3 from app import app, tasks
4
5 class TestIntegration(unittest.TestCase):
6
7     def setUp(self):
8         self.app = app.test_client()
9         self.app.testing = True
10        tasks.clear() # Clear tasks before each test
11
12    def test_add_and_view_tasks(self):
13        # Simulate adding a task
14        task_data = {'task': 'Test Task', 'due_date': '2023-11-30'}
15        self.app.post('/add', data=task_data, follow_redirects=True)
16
17        # Simulate viewing tasks
18        response = self.app.get('/view_tasks')
19
20        self.assertEqual(response.status_code, 200)
21        self.assertIn('Test Task', response.data)
22        self.assertIn(b'2023-11-30', response.data)
23
24    def test_delete_and_view_tasks(self):
25        # Add a task for testing
26        tasks.append({'task': 'Test Task', 'due_date': datetime.now()})
27
28        # Simulate deleting the task
29        response = self.app.get('/delete/0', follow_redirects=True)
30
31        self.assertEqual(response.status_code, 200)
32        self.assertEqual(len(tasks), 0)
33
34        # Simulate viewing tasks after deletion
35        response = self.app.get('/view_tasks')
36
37        self.assertEqual(response.status_code, 200)
38        self.assertNotIn('Test Task', response.data)
39
40    if __name__ == '__main__':
41        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 the Jenkins console output for a job named `feature_branch`. The log shows the execution of a shell script that sets up a virtual environment and runs an integration test. The test passes, indicated by the message `Ran 2 tests in 0.020s`.

```

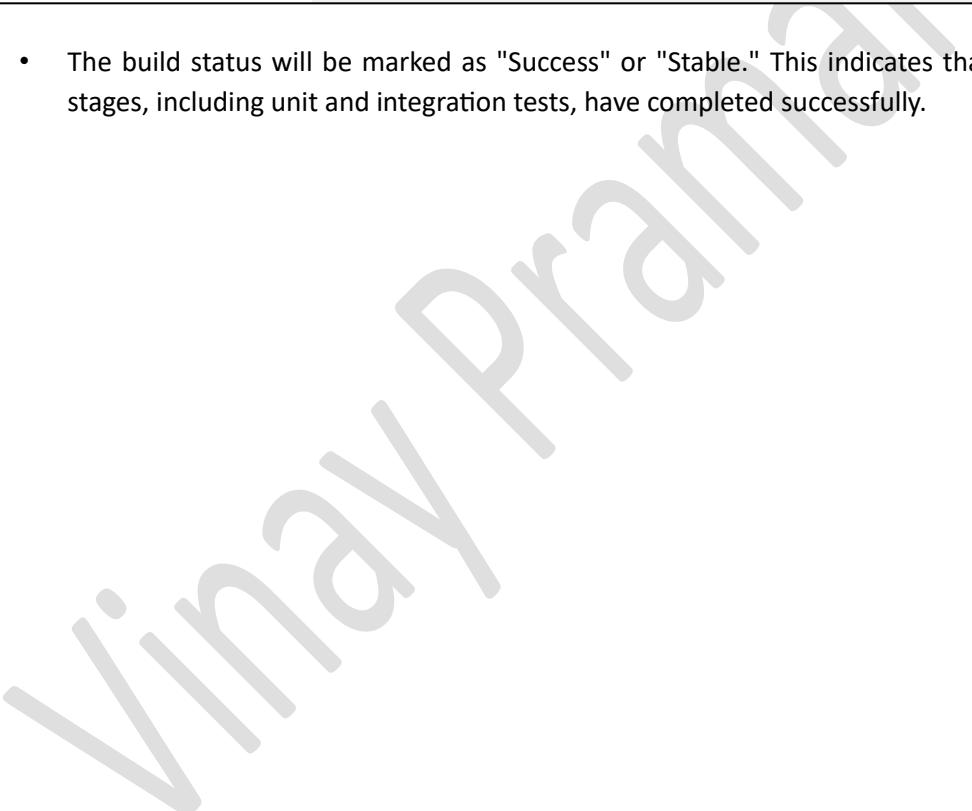
Dashboard > feature_branch > #27
+ unset VIRTUAL_ENV
+ unset VIRTUAL_ENV_PROMPT
+ [ ! nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/jenkins_home/workspace/feature_branch/venv
+ export VIRTUAL_ENV
+ _OLD_VIRTUAL_PATH=/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
+
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 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



A screenshot of a Jenkins job console output. The URL in the browser bar is [Not secure | 43.205.118.190:8080/job/feature_branch/28/console](http://43.205.118.190:8080/job/feature_branch/28/console). The console output shows a pipeline script running tests. It includes environment variable setup, directory navigation, and command execution. A failure occurs during a test run, followed by a summary of the results.

```
+ 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.

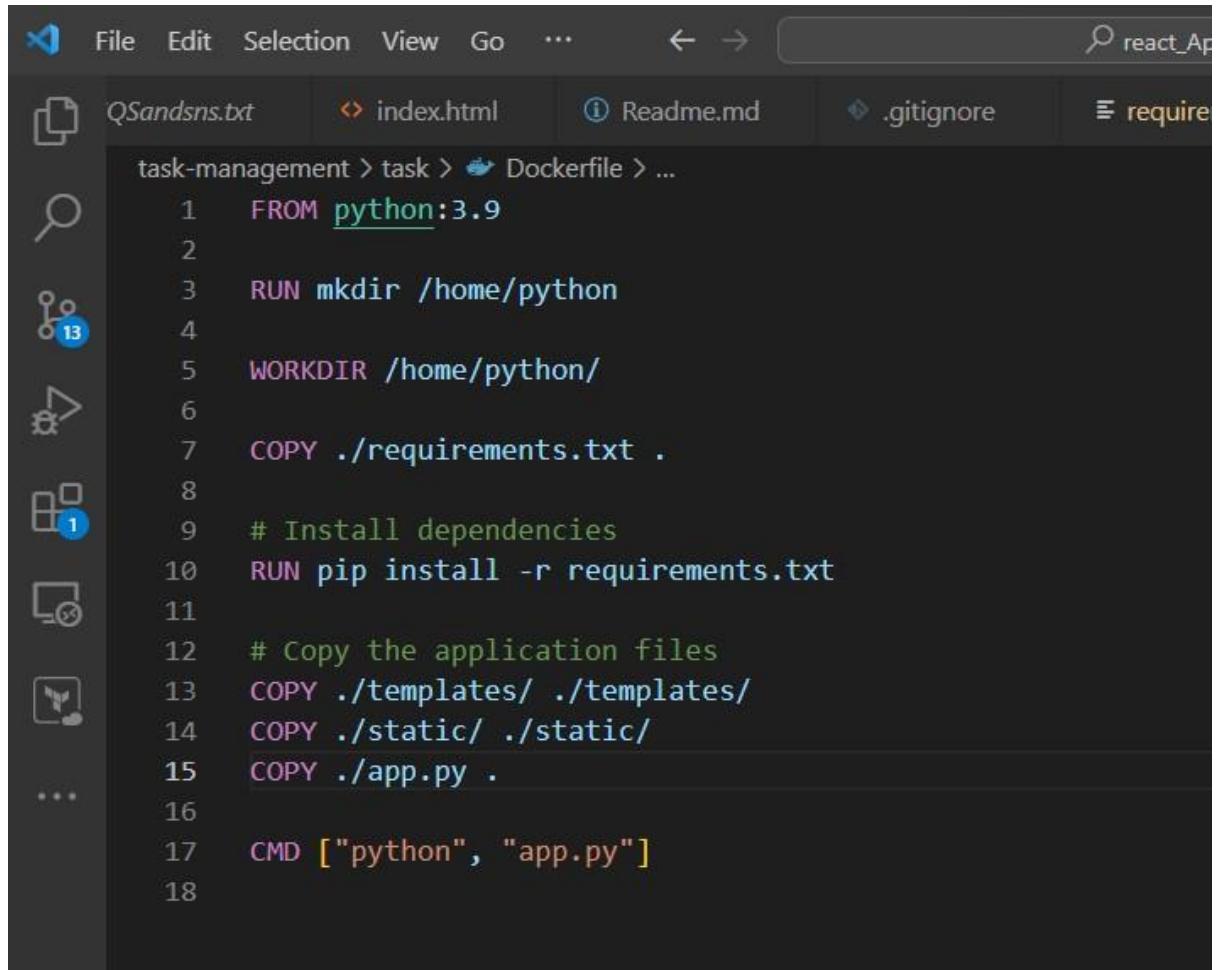
Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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. The top menu bar includes File, Edit, Selection, View, Go, and a three-dot menu. A search bar on the right contains the text "react_Ap". Below the menu is a toolbar with various icons: a file icon, a magnifying glass, a circular progress bar with the number 13, a refresh arrow, a square icon with a 1, a monitor icon, and a gear icon. The main workspace 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"]
```

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
20                script{
21                    dir('/var/jenkins_home/workspace/feature_branch')

```

```

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 'echo $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
d7b1ba4619b1: Layer already exists
86e50e0709ee: Layer already exists
12b956927ba2: Layer already exists
266def75d28e: Layer already exists
29e49b59e9ddaa: 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

```

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

Status **develop**

</> Changes **Build Now** **Configure** **Delete Pipeline**

Full Stage View **Rename** **Pipeline Syntax** **GitHub Hook Log**

Stage View

Average stage times:
(Average full run time: ~11s)

Declarative: Checkout SCM	Intro	Build the images
1s	79ms	9s

#21 Nov 20 15:11 1 commit

1s 79ms 9s

Build History **trend**

Filter builds... /

#21 Nov 20, 2023, 9:41 AM

#20

Permalinks

- Last build (#20), 48 sec ago
- Last stable build (#20), 48 sec ago
- Last successful build (#20), 48 sec ago
- Last failed build (#5), 3 hr 41 min ago

- 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.

shekhar123reddy / **General**

shekhar123reddy / **Repositories** / **ganesh** / **General**

Using 0 of 1 private repositories. [Get more](#)

General **Tags** **Builds** **Collaborators** **Webhooks** **Settings**

Add a short description for this repository [Update](#)

shekhar123reddy / ganesh

Description
This repository does not have a description [Edit](#)

Last pushed: 9 minutes ago

Docker commands
To push a new tag to this repository:

```
docker push shekhar123reddy/ganesh:tagname
```

Tags
This repository contains 1 tag(s).

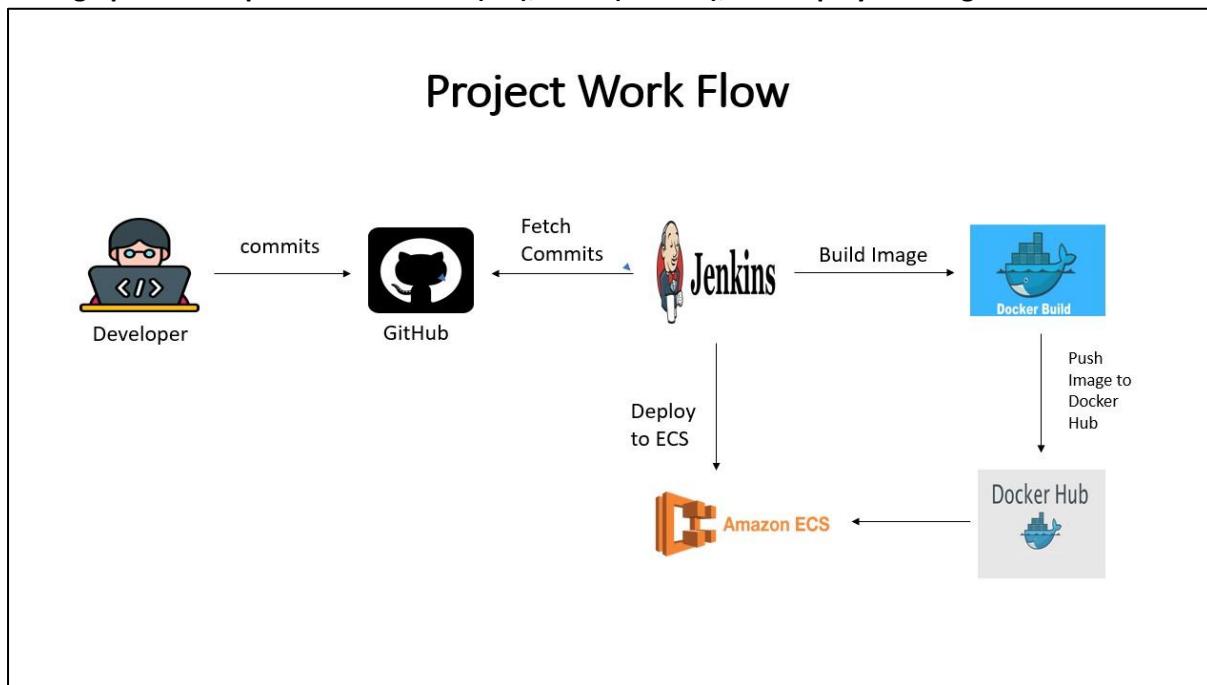
Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

Sub Task 4:

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



Prerequisites:

2. AWS Account:

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

3. Git Repository:

- Have a Git repository containing your application code.

4. Jenkins Server:

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

5. Docker and ECS:

- Ensure you have Docker installed for building containers.
- Here 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 menu includes 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'User groups', 'Users', and 'Roles' selected), 'Policies', 'Identity providers', 'Account settings', 'Access reports', 'Access analyzer' (with 'Archive roles', 'Analyzers', and 'Settings'), 'Credential report', 'Organization activity', and 'Service control policies (SCPs)'. Below these are 'Related consoles' for 'IAM Identity Center' and 'AWS Organizations'. The main content area displays the 'JenkinsRole' details. The 'Summary' tab shows the role was created on November 11, 2023, at 17:21 UTC+05:30, with an ARN of arn:aws:iam::862547479026:role/JenkinsRole. It has a maximum session duration of 1 hour and is associated with the instance profile ARN arn:aws:iam::862547479026:instance-profile/JenkinsRole. The 'Permissions' tab lists six managed policies attached to the role: 'AmazonEC2FullAccess', 'AmazonEC2FullAccess', 'AmazonS3FullAccess', 'AWSCodeDeployFullAccess', 'AWSCodeDeployRoleForECS', and 'AWSCodePipeline_FullAccess'. The 'Trust relationships' and 'Tags' tabs are also visible. At the bottom, there are buttons for 'Edit', 'Delete', 'Simulate', 'Remove', and 'Add permissions'.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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 search bar, a table lists the plugin. The first row shows the 'AWS CodePipeline Plugin' with the name 'AWS CodePipeline' and version '0.27'. Below the table are three buttons: 'Install without restart' (highlighted in blue), 'Download now and install after restart', and 'Check now'. A status message at the bottom indicates 'Update information obtained: 1 hr 17 min ago'.

- 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. The left sidebar has 'Configure' selected. Under 'Source Code Management', the 'AWS CodePipeline' option is chosen. The configuration includes:

- AWS Region: ap-south-1
- Proxy Host: (empty)
- Proxy Port: (empty)
- Credentials:
 - AWS Access Key: AKIA4RU6WBHZANTW3G4S
 - AWS Secret Key: (redacted)
- A checkbox 'Clear workspace before copying' is checked.

- 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

Configure

CodePipeline Action Type

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

Category

This value must match the Provider field that is on the Custom Action in your corresponding Pipeline.

Provider

This value must match the Version field that is on the Custom Action in your corresponding Pipeline.

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)”

Configure

Build Environment

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
shekhar123reddy/*****

+ Add

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.

Follow for more: Vinay Pramanik (LinkedIn)

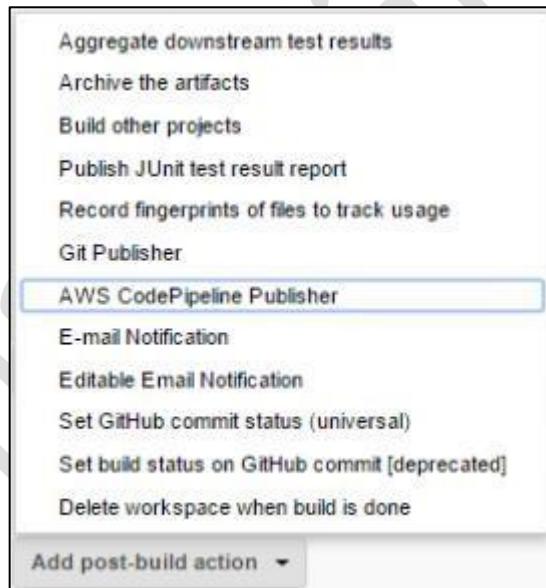
[AWS Cloud Architect]

The screenshot shows a Jenkins configuration interface for a job named 'NARASIMHA'. The left sidebar has tabs for General, Source Code Management, Build Triggers, Build Environment, Build Steps (which is selected), and Post-build Actions. The main area is titled 'Build Steps' and contains a single step named 'Execute shell'. The command entered is:

```
docker build -t simple:2.0 .
echo "$PASS" | docker login -u "$USER" --password-stdin
docker tag simple:1.0 shekhar123reddy/simple:2.0
docker push shekhar123reddy/simple:2.0
printf '[{"name":"demoworld","imageUri":"'%"'"'}]' "shekhar123reddy/simple:2.0" >| imagedefinitions.json
```

Below the command is an 'Advanced' dropdown and a 'Save' button.

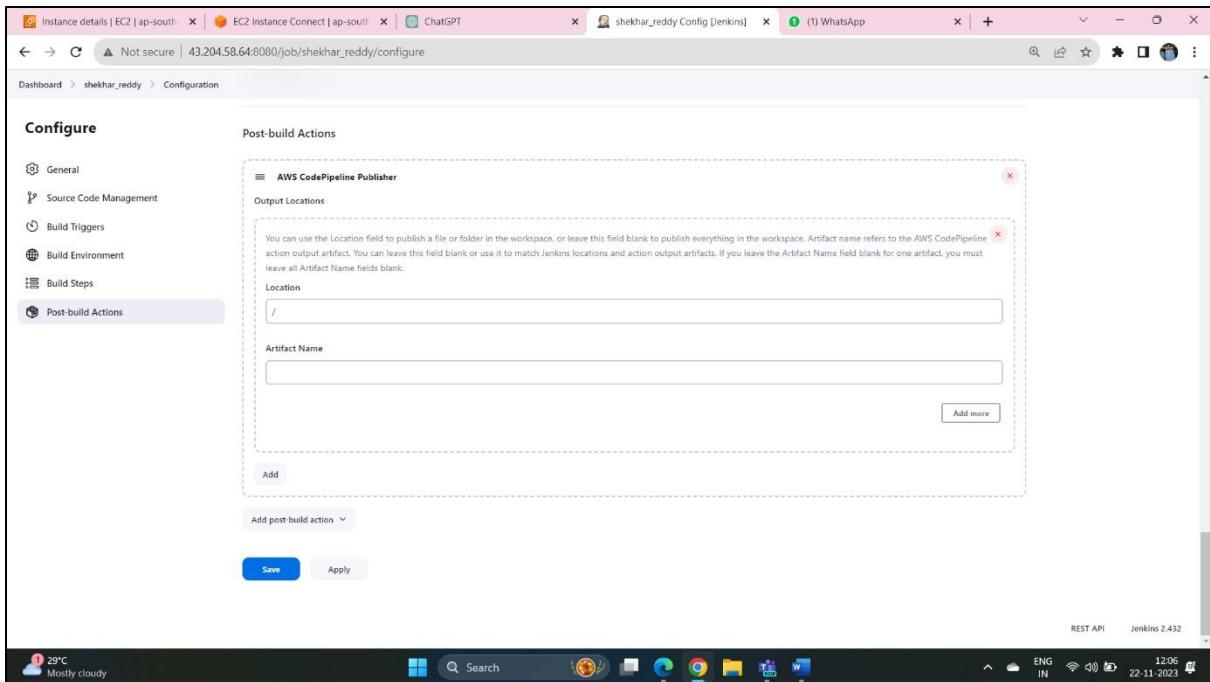
- On the **Post-build Actions** tab, add a post-build action.
- Choose **AWS CodePipeline Publisher**.



- Do not add any output locations.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



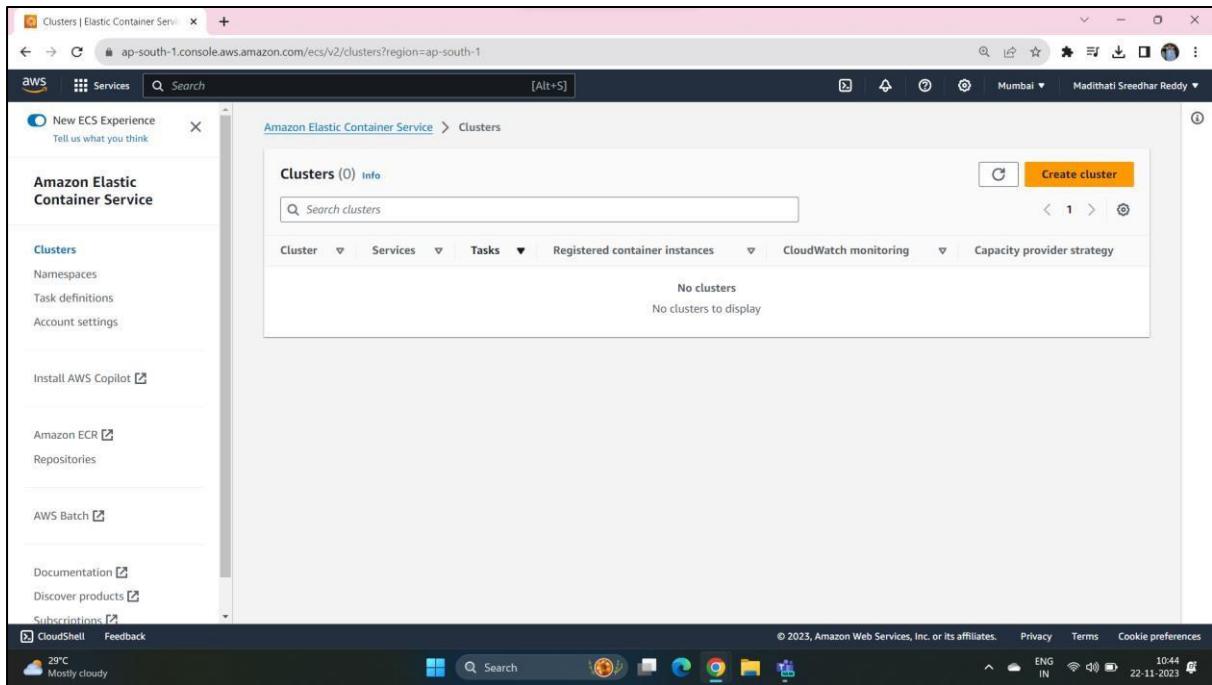
- Click **Save**.

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

1. Creating ECS Cluster:

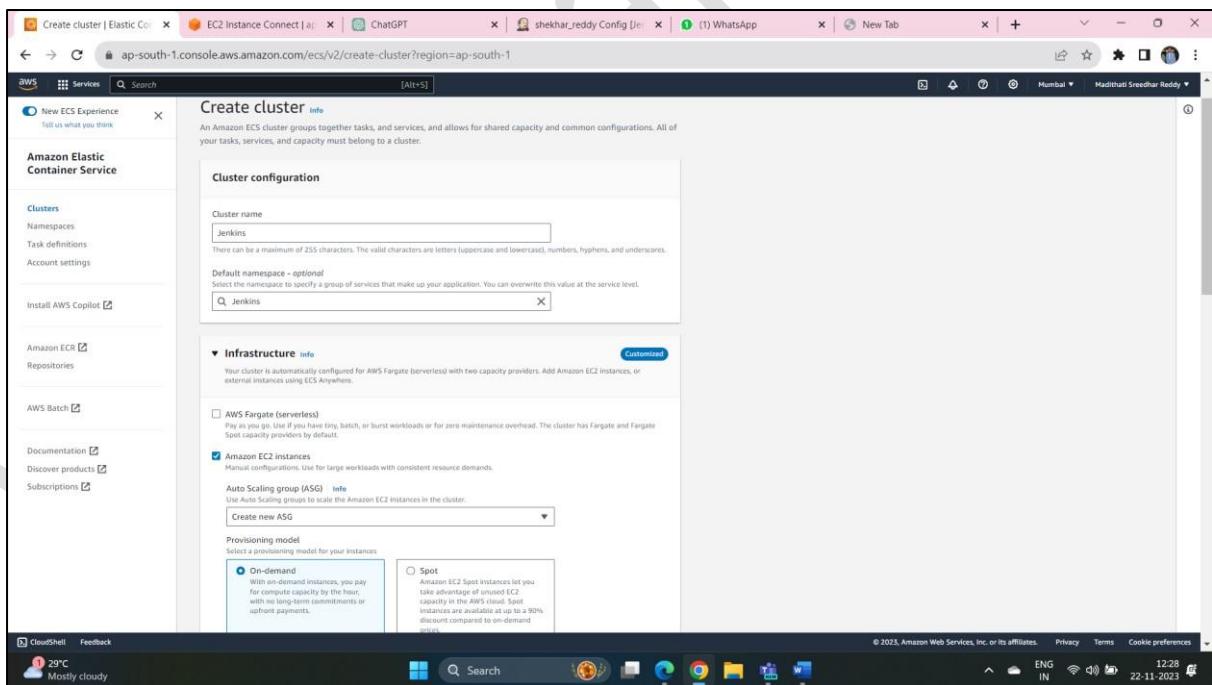
Creating an Amazon ECS (Elastic Container Service) cluster involves several steps. ECS is a fully managed container orchestration service that allows you to run, stop, and manage Docker containers on a cluster. Here are the general steps to create an ECS cluster:

- **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.

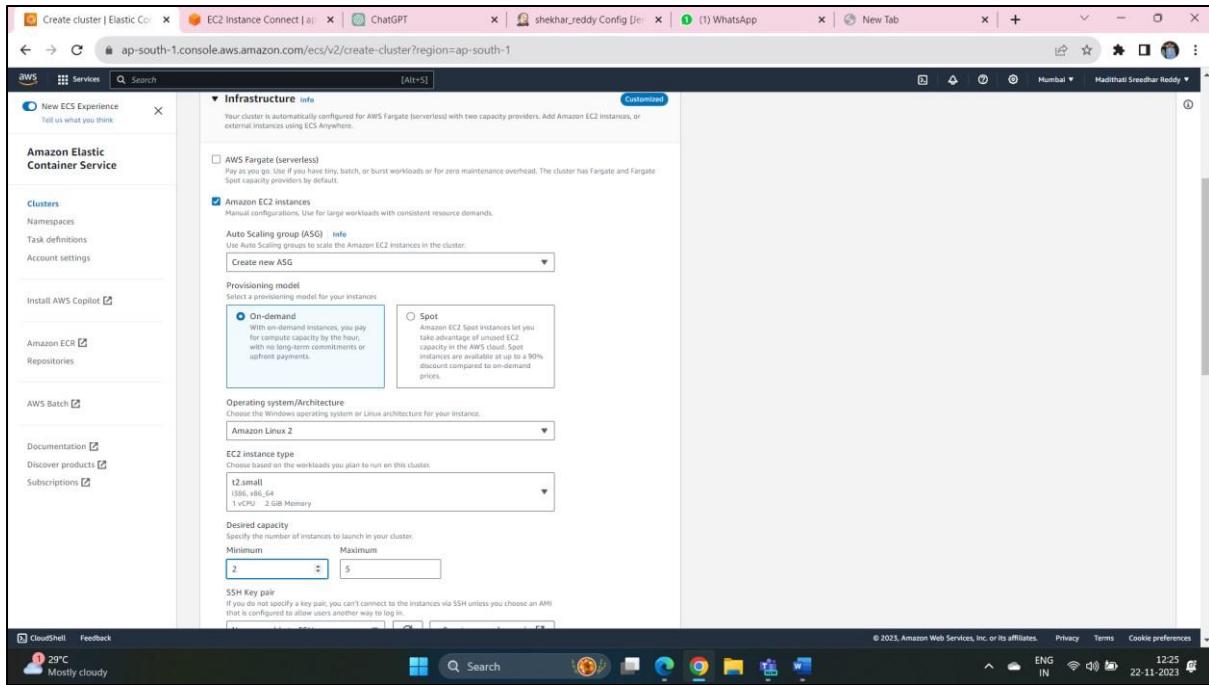


- Choose a cluster Infrastructure based on your requirements:
- **Amazon EC2 Instance:** For EC2 instances using Linux and manual networking.
- **Fargate:** For serverless containers without the need to manage EC2 instances.

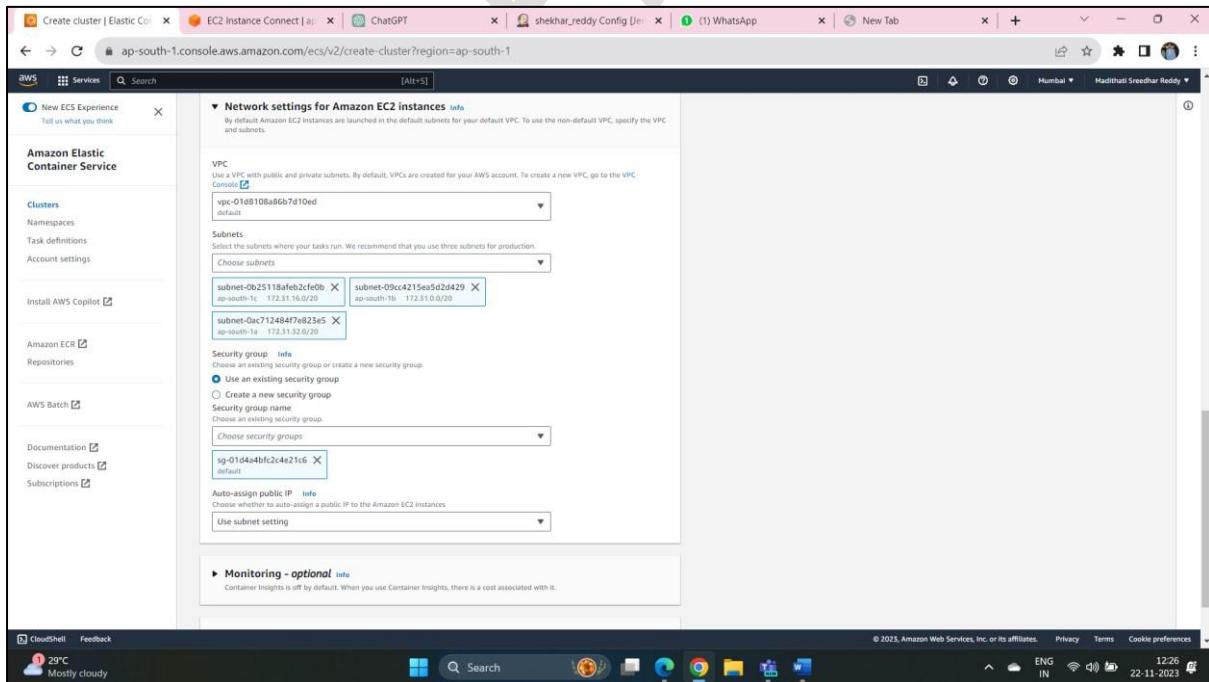
Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- Create an EC2 Instance based on your requirements.



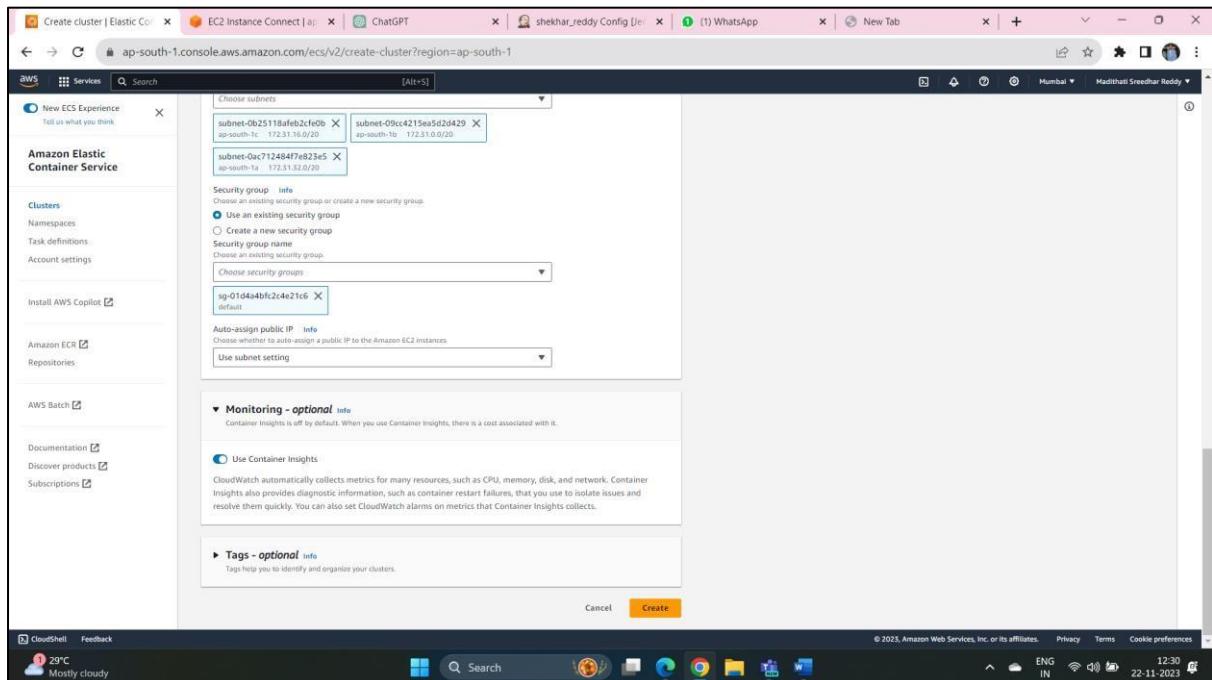
- Number of Instances (if using EC2 instances): Set the initial number of instances in your cluster.
- VPC, Subnets, and other networking settings.



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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



- 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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), and AWS Batch. The main area displays a table of task definitions with columns for Task definition name and Status of last revision. All four entries (NARASIMHA, TaskApp, ganesh-1, ramesh-1) are listed as ACTIVE. A prominent orange button at the top right says "Create new task definition". The browser address bar shows the URL: <https://ap-south-1.console.aws.amazon.com/ecs/v2/task-definitions?region=ap-south-1>.

- 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.

This screenshot shows the "Create task definition" wizard, Step 1: Infrastructure requirements. The "Task definition family" field is filled with "Jenkins". Under "Launch type", the "Amazon EC2 instances" checkbox is checked. In the "Network mode" section, "awsvpc" is selected. The "Task size" section shows "1 vCPU" and "1 GB" for Memory. The browser address bar shows the URL: <https://ap-south-1.console.aws.amazon.com/ecs/v2/create-task-definition?region=ap-south-1>.

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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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 'Create task definition' page in the AWS ECS console. Under the 'Task definitions' section, a new task definition is being created. In the 'Container - 1' section, the container is named 'ganesh' and the image URI is 'shekhar123reddy/ganesh:latest'. The 'Essential container' dropdown is set to 'Yes'. Under 'Port mappings', a single mapping is defined: Host port 5004 to Container port 5004, using protocol TCP. The AWS navigation bar and footer are visible at the bottom.

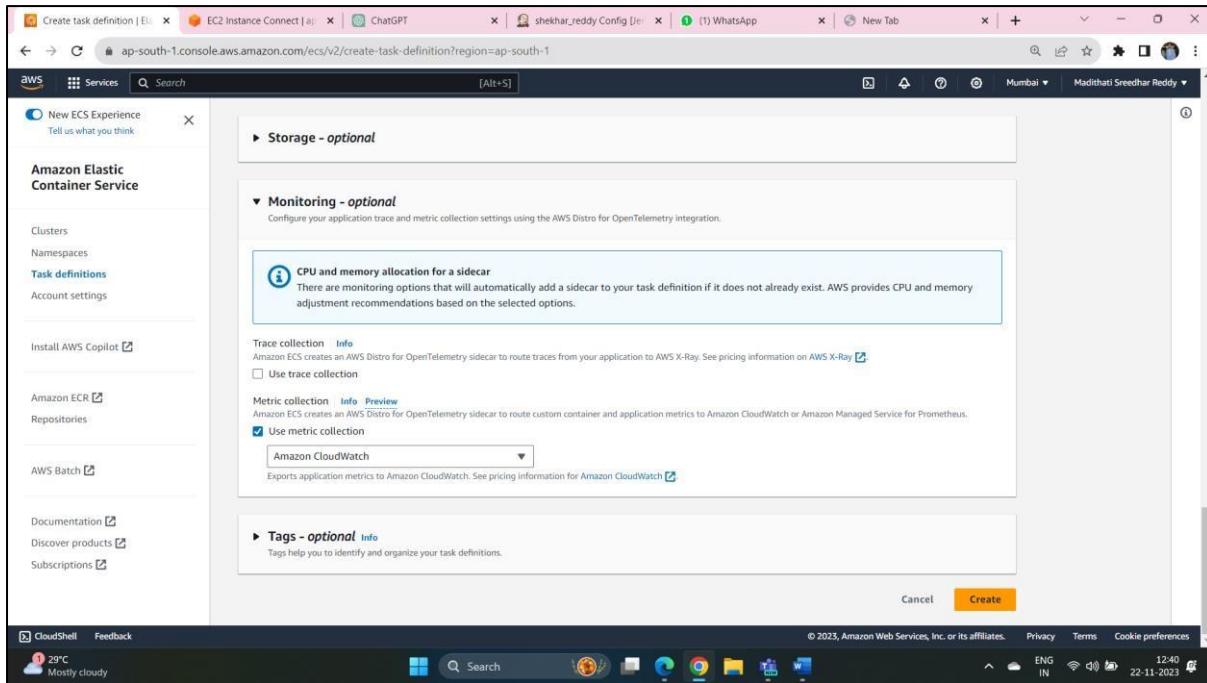
- **Environment:** Set environment variables if needed
- Configure Logging, Health Check, Docker configuration and resource limits depends on your need.

The screenshot shows the continuation of the 'Create task definition' page. In the 'Environment' section, there is a 'Add environment variable' button. Below it, a list of optional configurations is shown: 'Add from file', 'Logging - optional', 'HealthCheck - optional', 'Container timeouts - optional', 'Container network settings - optional', 'Docker configuration - optional', 'Resource limits (Ulimits) - optional', and 'Docker labels - optional'. At the bottom of the page, there is a '+ Add container' button and a 'Storage - optional' section. The AWS navigation bar and footer are visible at the bottom.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

- 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."

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the AWS ECS console with the Jenkins cluster selected. The cluster overview section displays the ARN (arn:aws:ecs:ap-south-1:862547479026:custer/Jenkins), status (Active), CloudWatch monitoring (Container Insights), and registered container instances (2). The Services tab shows 0 services listed. The bottom navigation bar includes links for Services, Tasks, Infrastructure, Metrics, Scheduled tasks, and Tags.

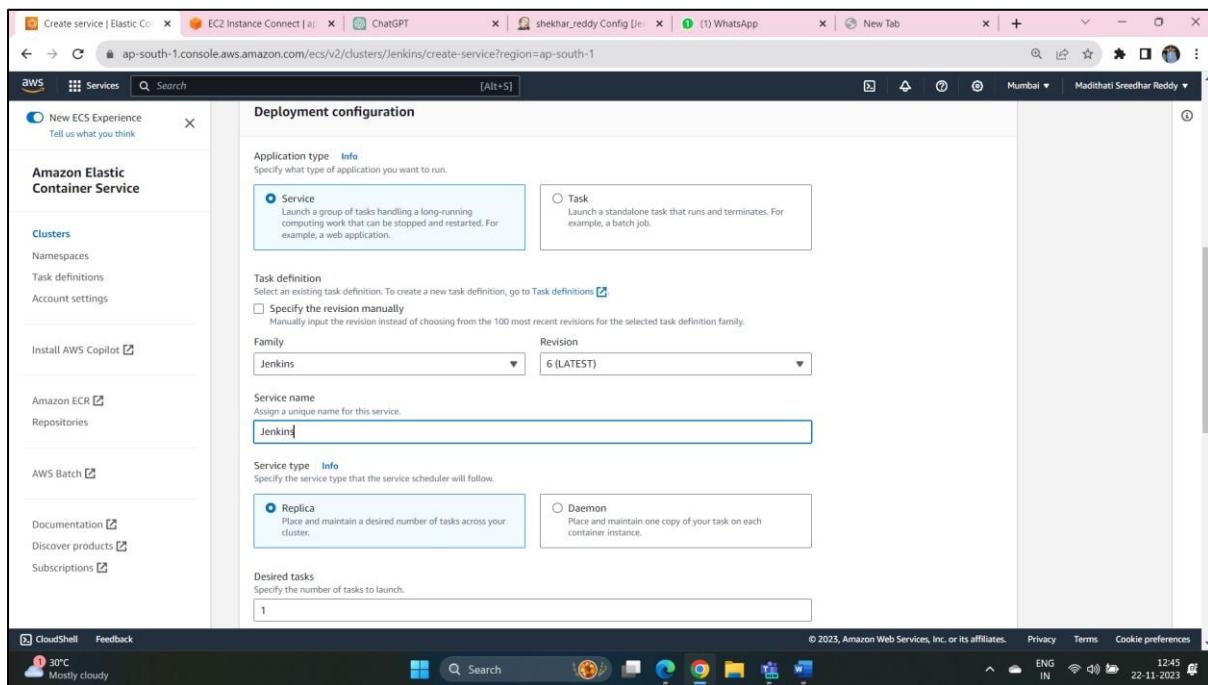
- Select the Configuration.

The screenshot shows the 'Create' configuration page for a new service. Under 'Compute configuration (advanced)', the 'Launch type' is selected as 'Launch tasks directly without the use of a capacity provider strategy' (EC2). The 'Deployment configuration' section shows 'Application type' set to 'Service'. The bottom navigation bar includes links for CloudShell and Feedback.

- Configure as below.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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:

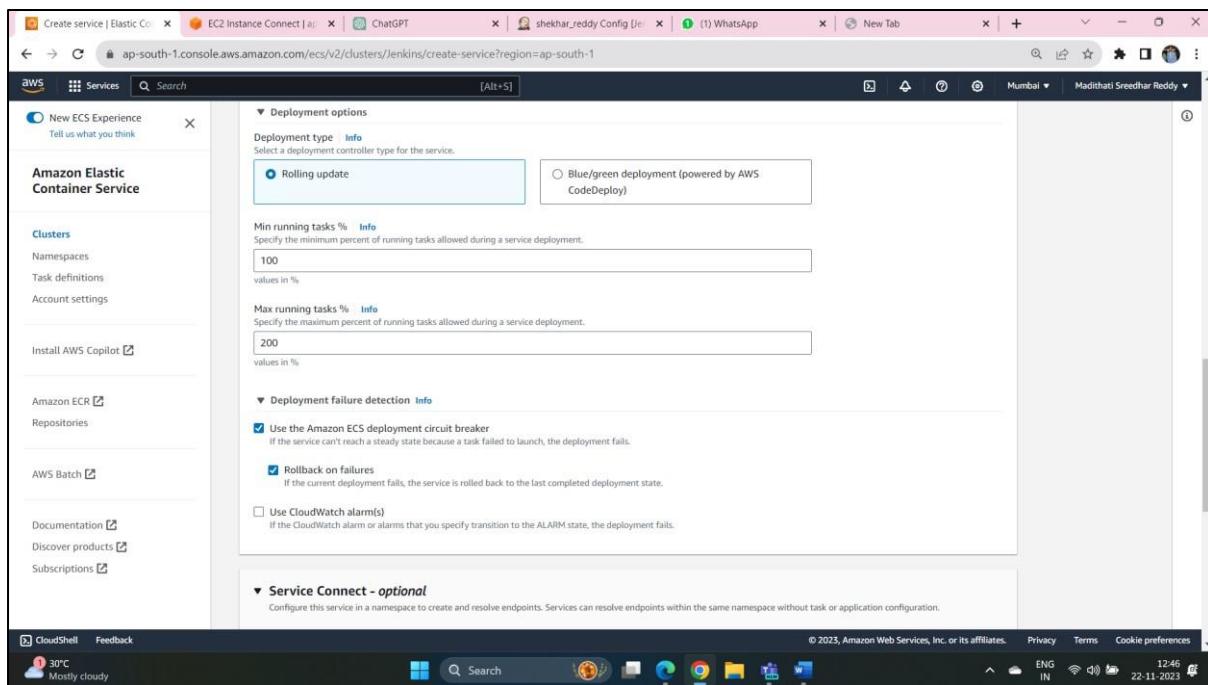
Updates are applied incrementally to a subset of tasks while maintaining a specified number of healthy tasks. Provides a continuous deployment process, gradually updating tasks without downtime. Allows for a controlled rollout of the new version across the ECS cluster. Optimizes resource usage by replacing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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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."

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the AWS Cloud Console with the URL [ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LoadBalancersv3:\\$case=tags:false%5Cclient:false\\$regex=tags:false%5Cclient:false](https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LoadBalancersv3:$case=tags:false%5Cclient:false$regex=tags:false%5Cclient:false). The left sidebar is expanded to show 'Load Balancing' under 'Network & Security'. The main content area displays a table titled 'Load balancers (1)'. The table has columns for Name, DNS name, State, VPC ID, Availability Zones, and Type. One row is present with the values: Name '5004', DNS name '5004-845676065.ap-sout...', State 'Active', VPC ID 'vpc-01d8108a86b7d1...', Availability Zones 'ap', and Type 'Application Load Balancer'. A context menu is open over the '5004' entry, with the 'Create Application Load Balancer' option highlighted.

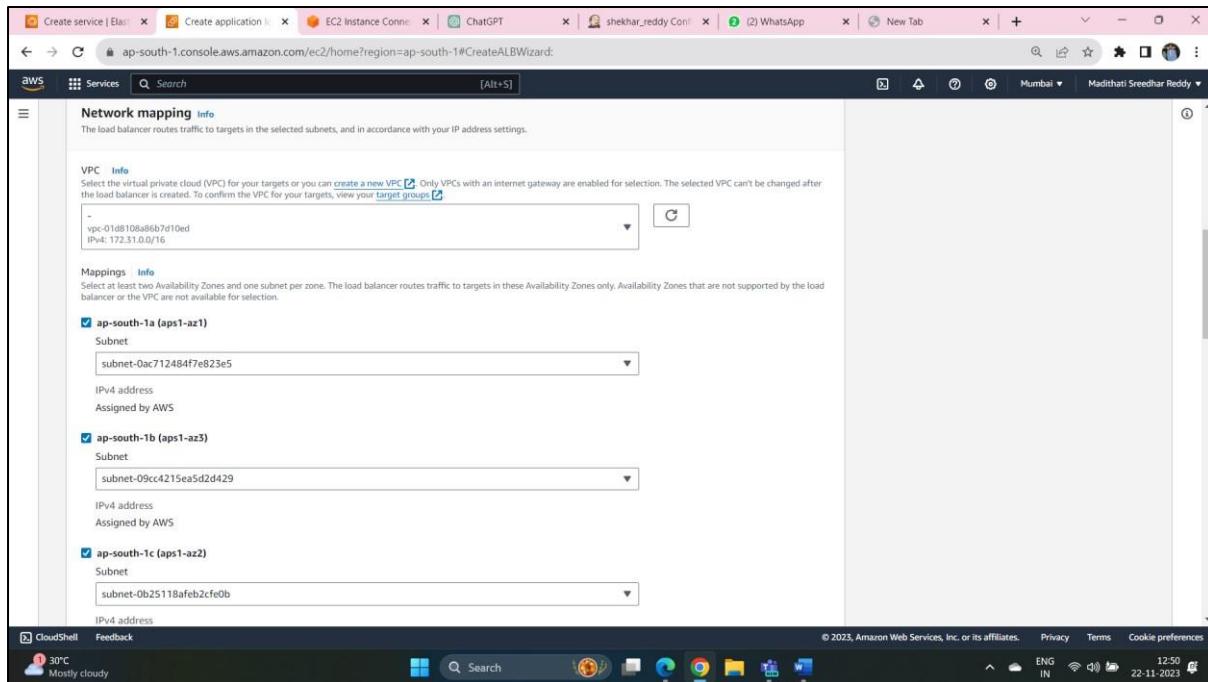
- Give a name for it.

The screenshot shows the AWS Cloud Console with the URL ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#CreateALBWizard. The left sidebar is collapsed. The main content area is titled 'Create Application Load Balancer' with an 'Info' link. It contains a section titled 'How Elastic Load Balancing works' and a 'Basic configuration' section. In the 'Basic configuration' section, there is a 'Load balancer name' input field containing '5004', a 'Scheme' dropdown set to 'Internet-facing', and an 'IP address type' dropdown set to 'IPv4'.

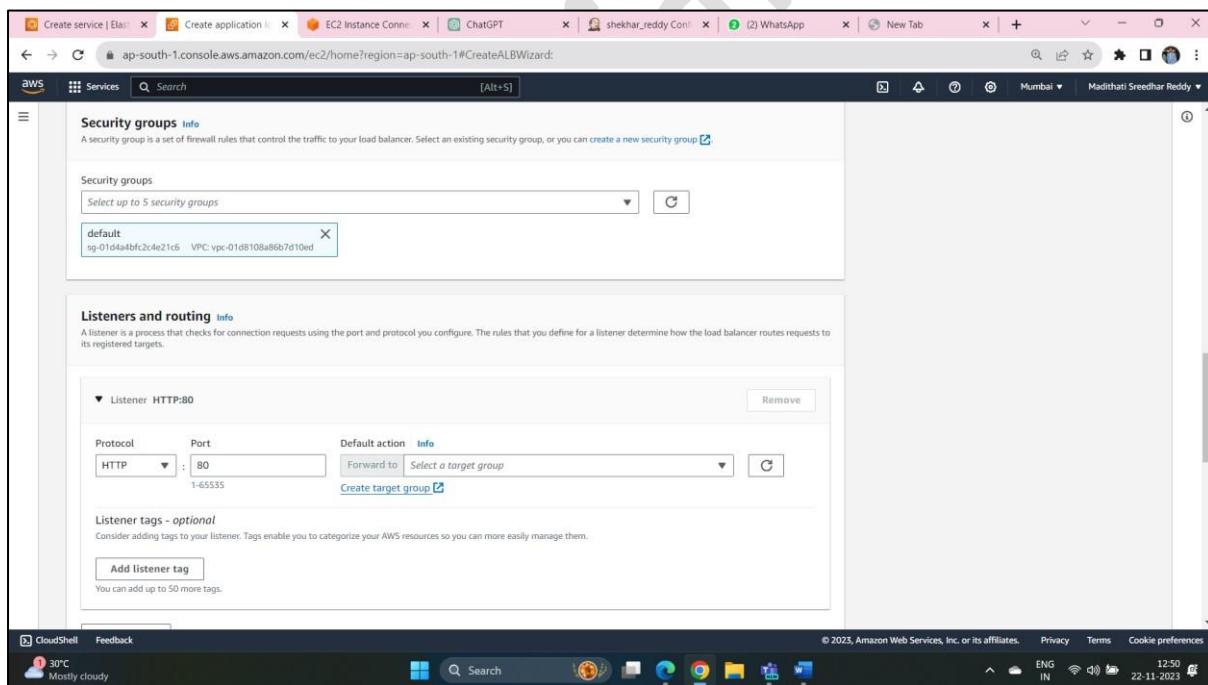
- Configure the load balancer settings (VPC, Subnets, Security Groups).

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



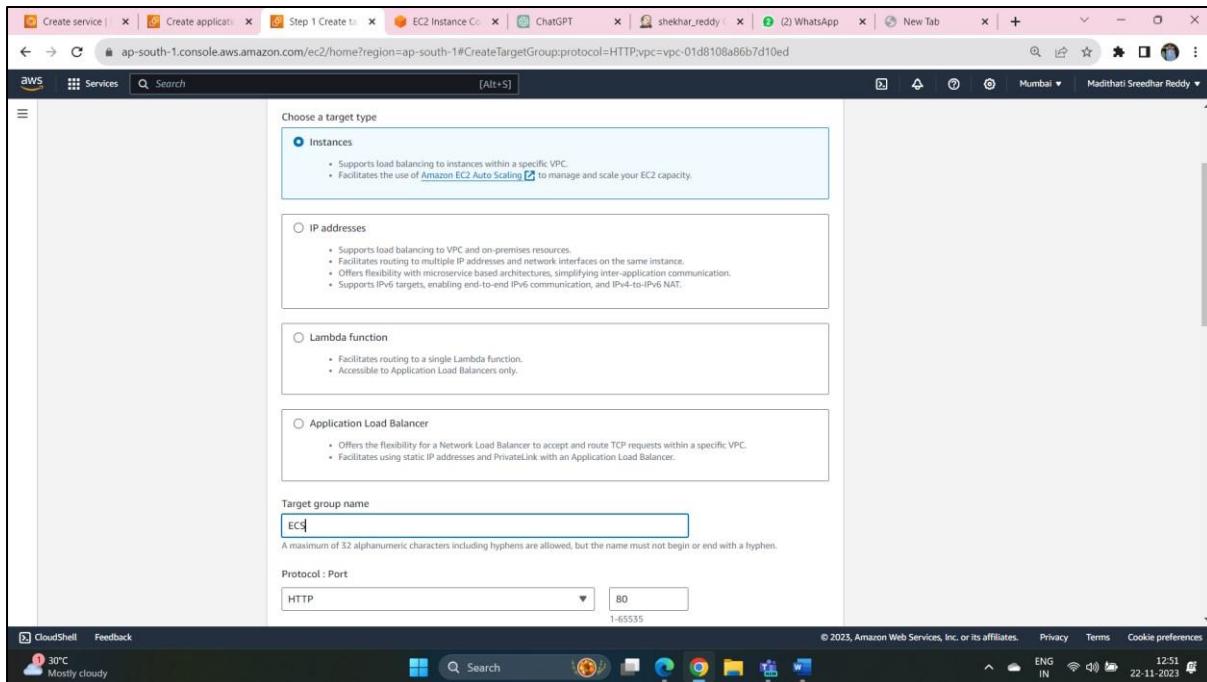
- Configure routing by adding listeners and target groups.



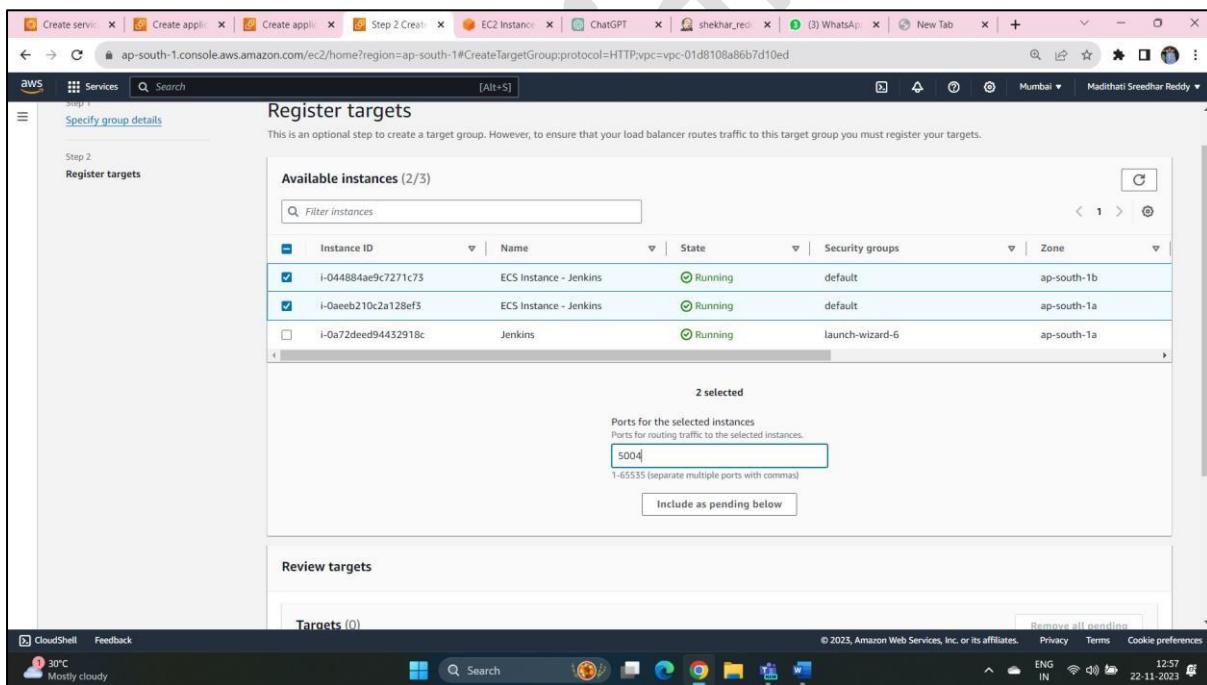
- For this Create a Target Group by clicking on the Blue Option present there

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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



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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the AWS CloudFormation Create Service wizard Step 2: Create Listener. The configuration for a Listener HTTP:80 is displayed. The protocol is set to HTTP, port is 80, and it is forwarded to an ECS target type. Listener tags and optional add-on services like AWS Global Accelerator are also shown.

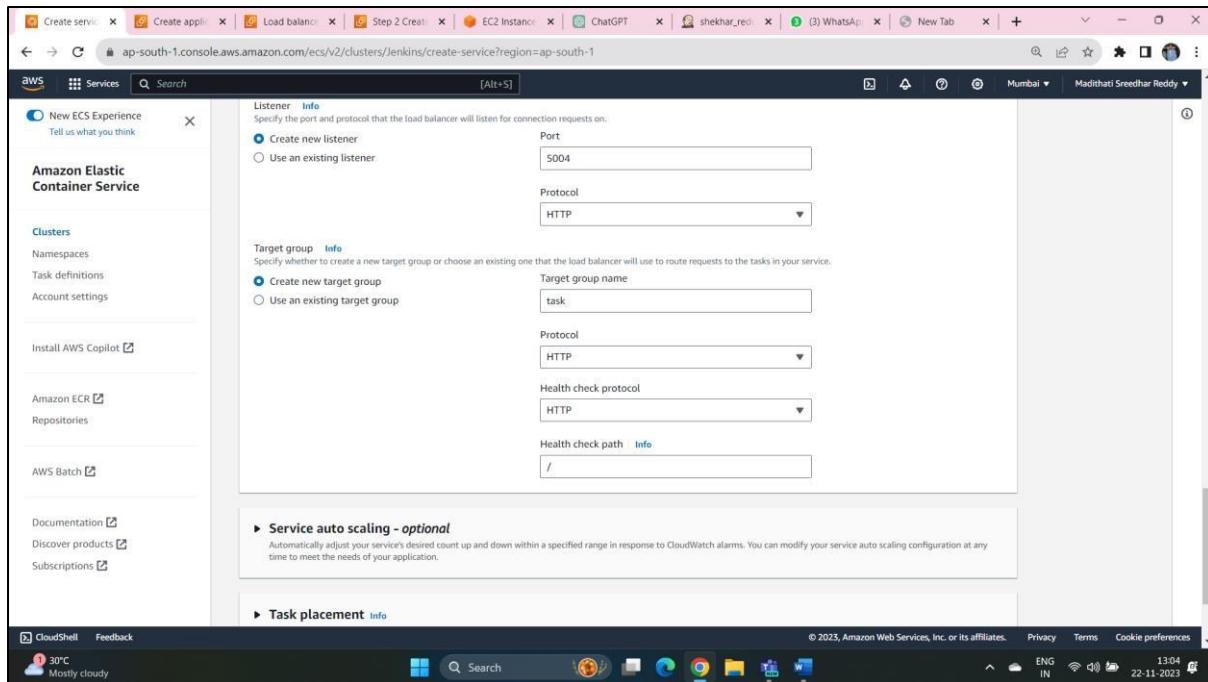
- Give that LoadBalancer under the LoadBalancing Section.

The screenshot shows the AWS CloudFormation Create Service wizard Step 2: Create Listener. The Load balancing - optional section is displayed, showing the Application Load Balancer selected for the load balancer type.

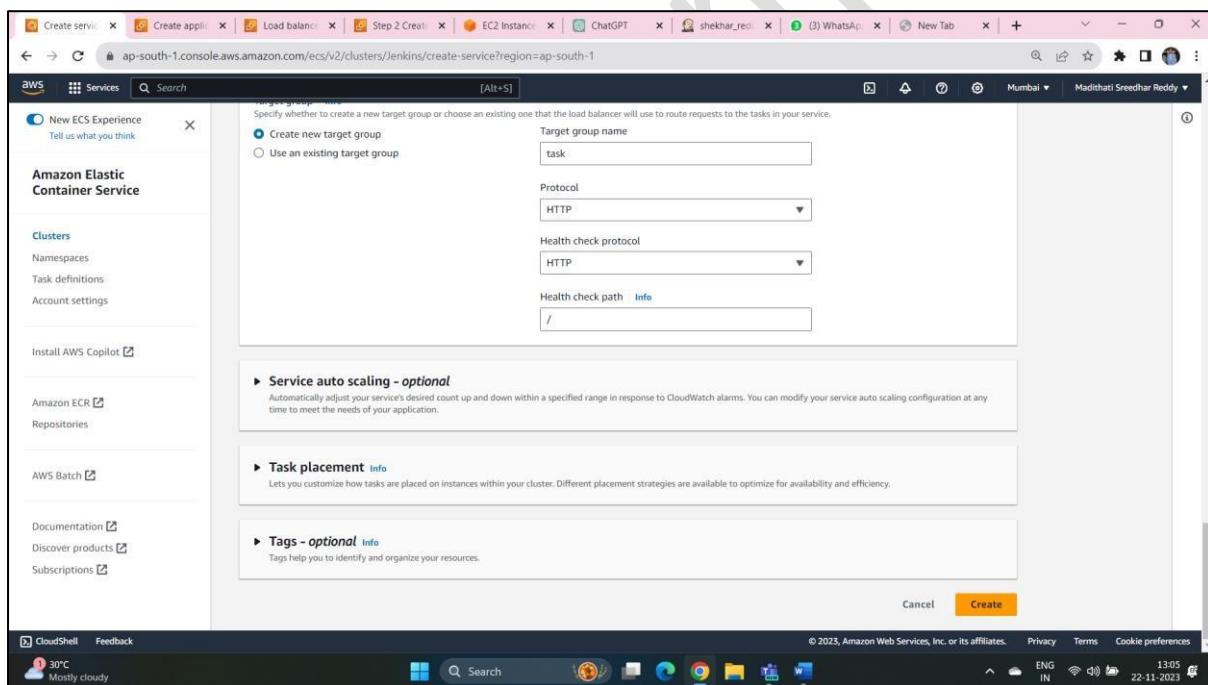
- Configure other Options as Below.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



□ 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.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

Cluster overview

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:ap-south-1:862547479026:cluster/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 ShekharReddy/feature	1 hour ago
TaskWebApp	V2	In progress	Source - e173df9c: as	19 hours ago

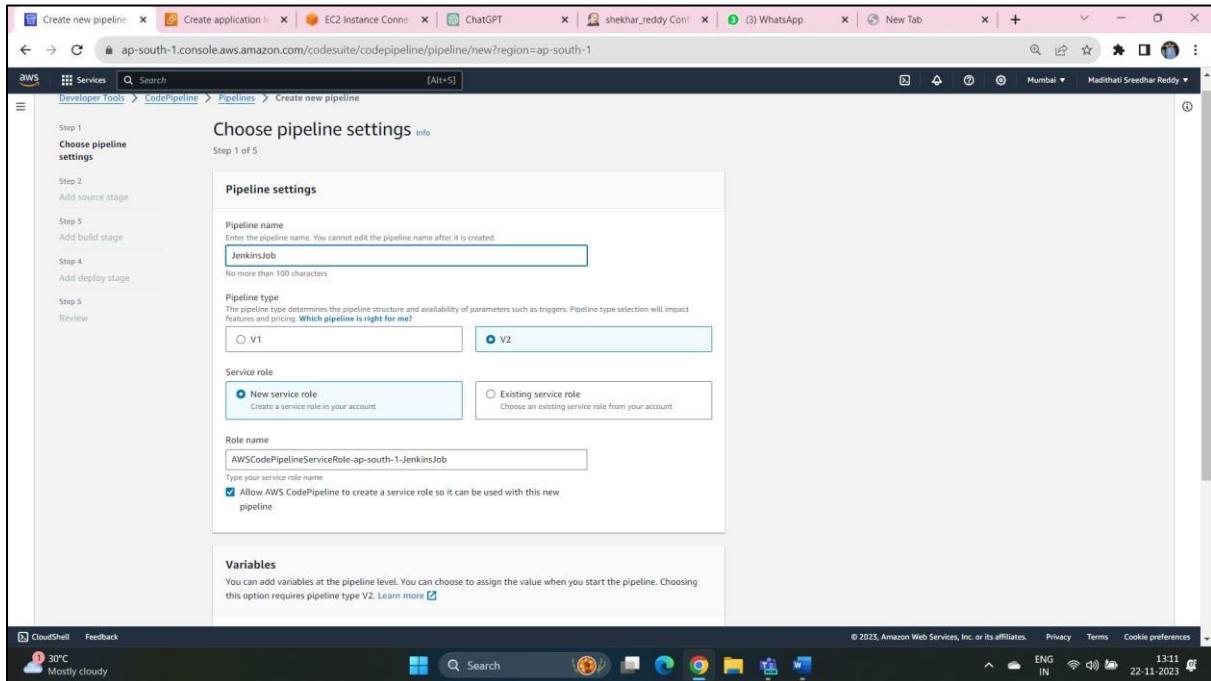
- Configure your pipeline with the following stages:
- **Source Stage:** Connect to your “GitHub” repository.
- **Build Stage:** Use “Jenkins” as the build provider.

Follow for more: Vinay Pramanik (LinkedIn)

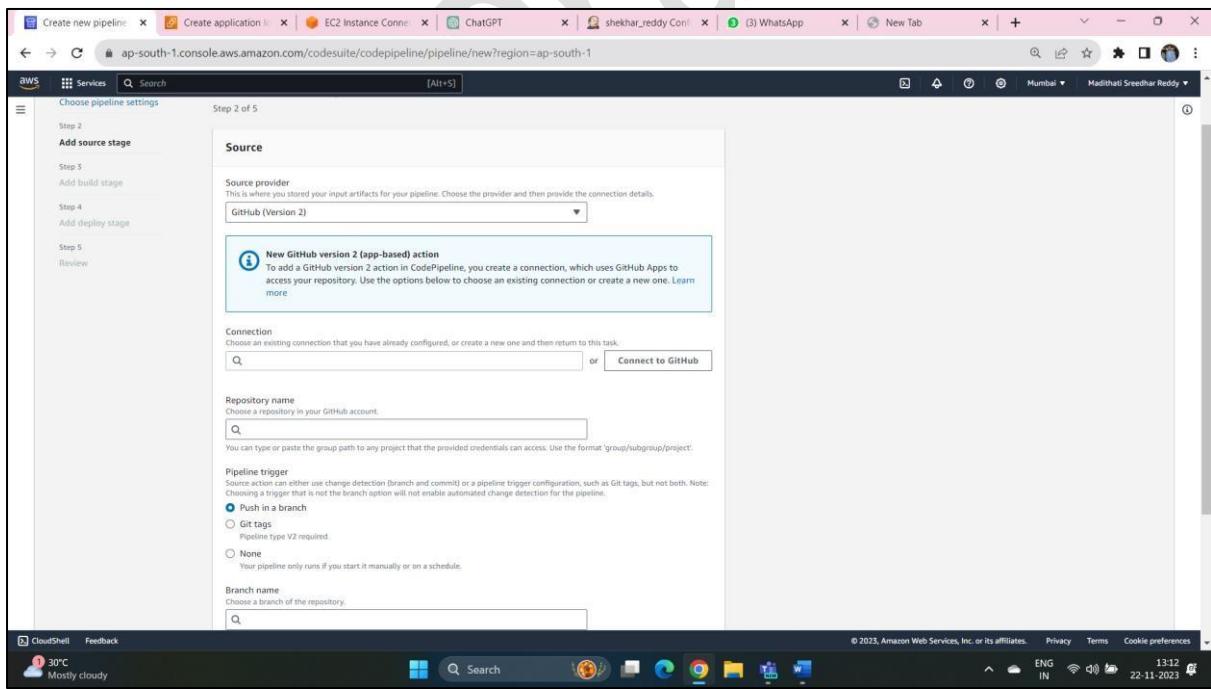
[AWS Cloud Architect]

- **Deploy Stage:** Choose “ECS” as the deployment provider.

- Enter the Unique Pipeline name.



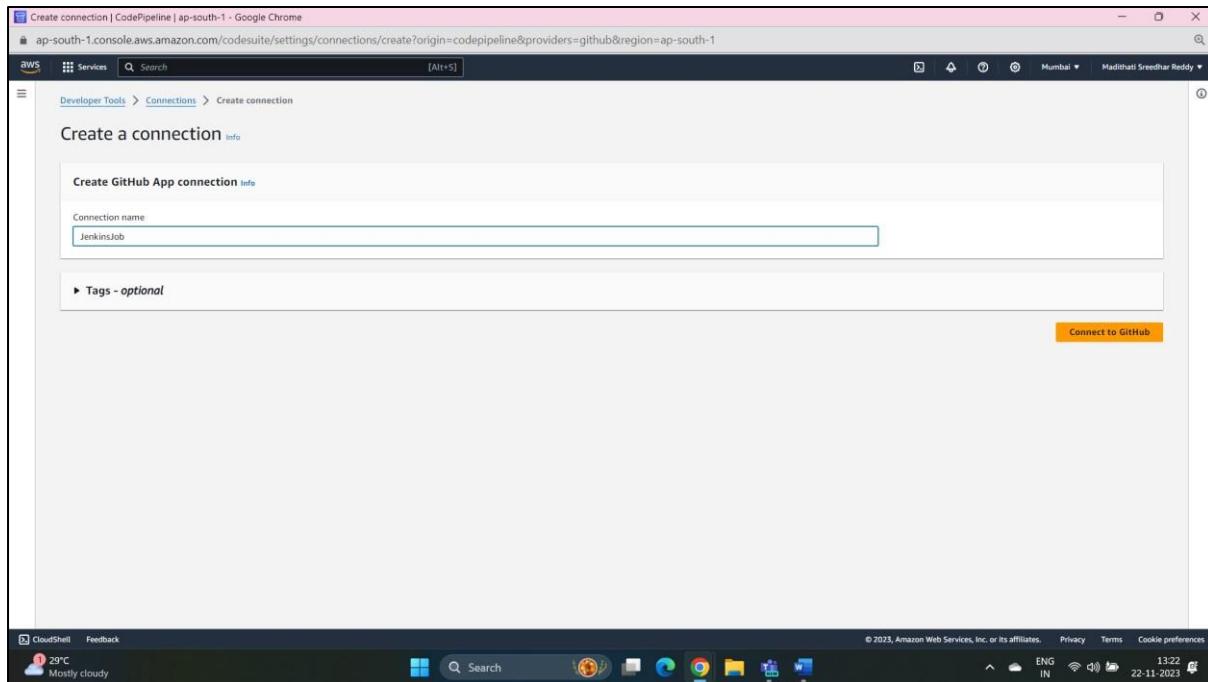
- Click next
- Choose the source provider (e.g., GitHub)



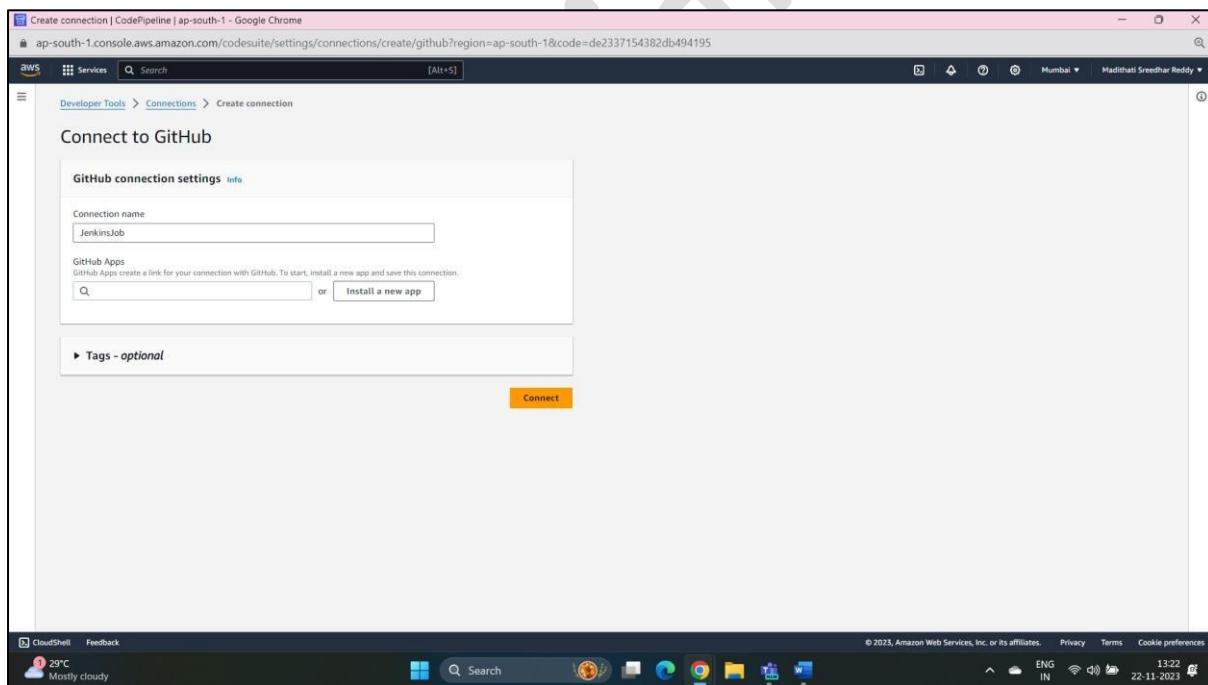
- Click on “Connect to GitHub”

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



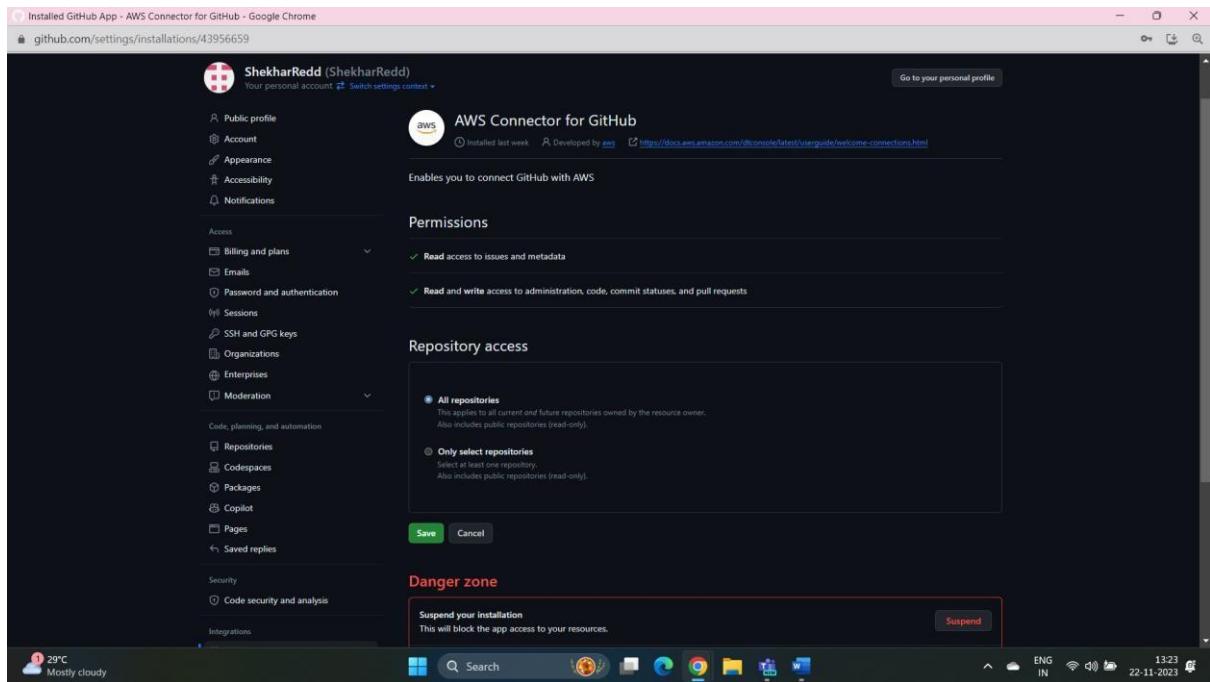
- Click on the tab mentioned there.



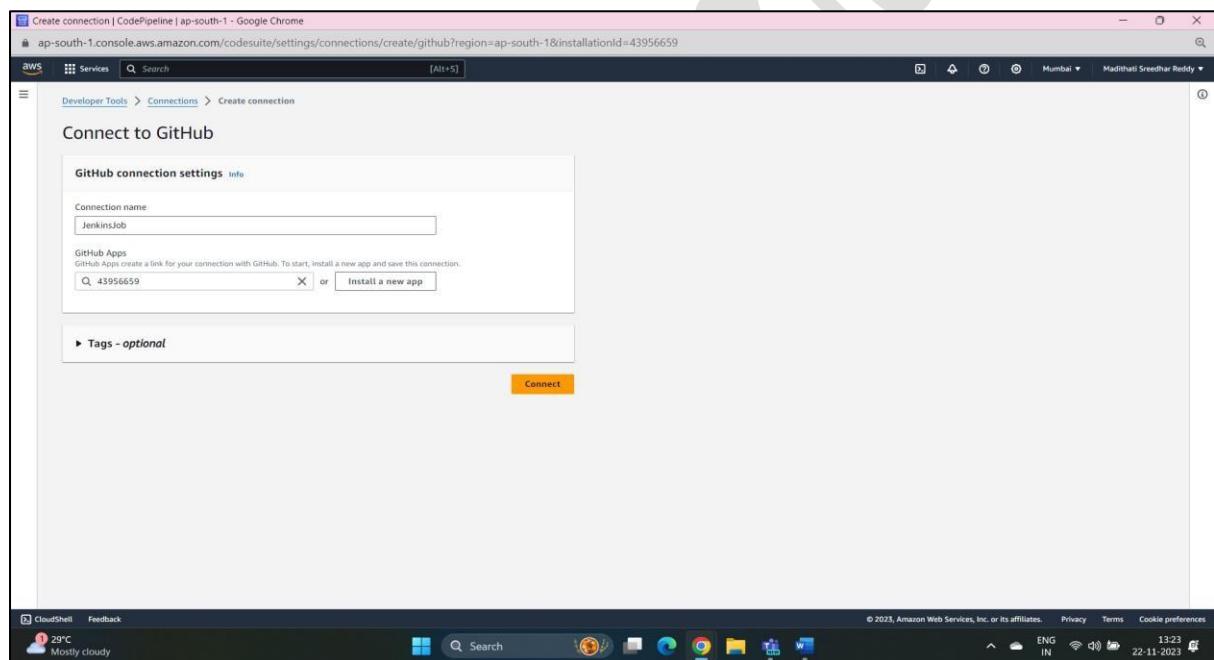
- Click on Install New App
- Next Enter the credentials of your GitHub
- Then Choose the Repo where your SourceCode relies.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



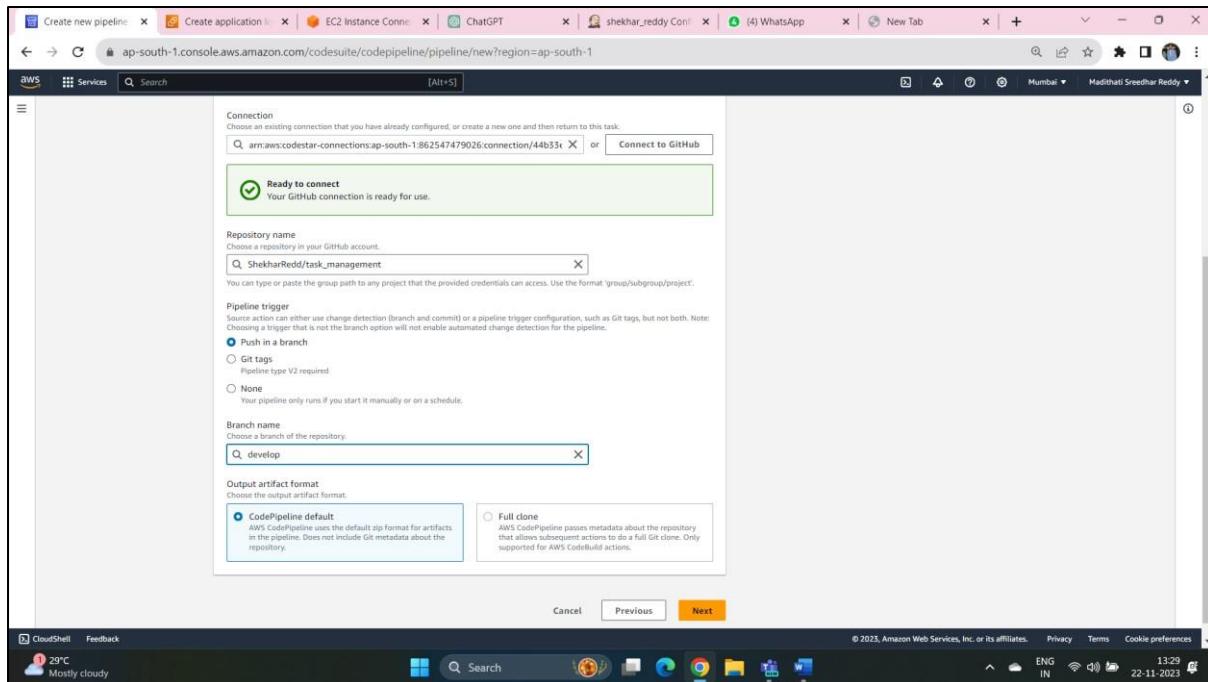
- Click on save and then return to the AWS Console.



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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

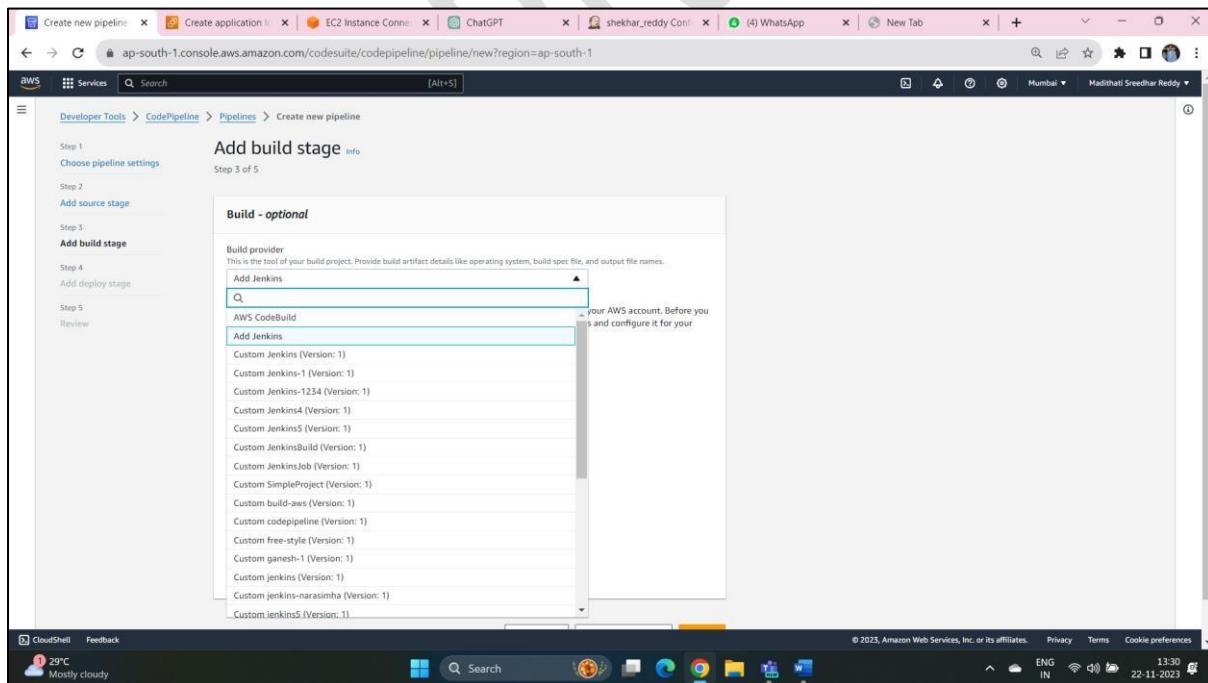


□ Click on next

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

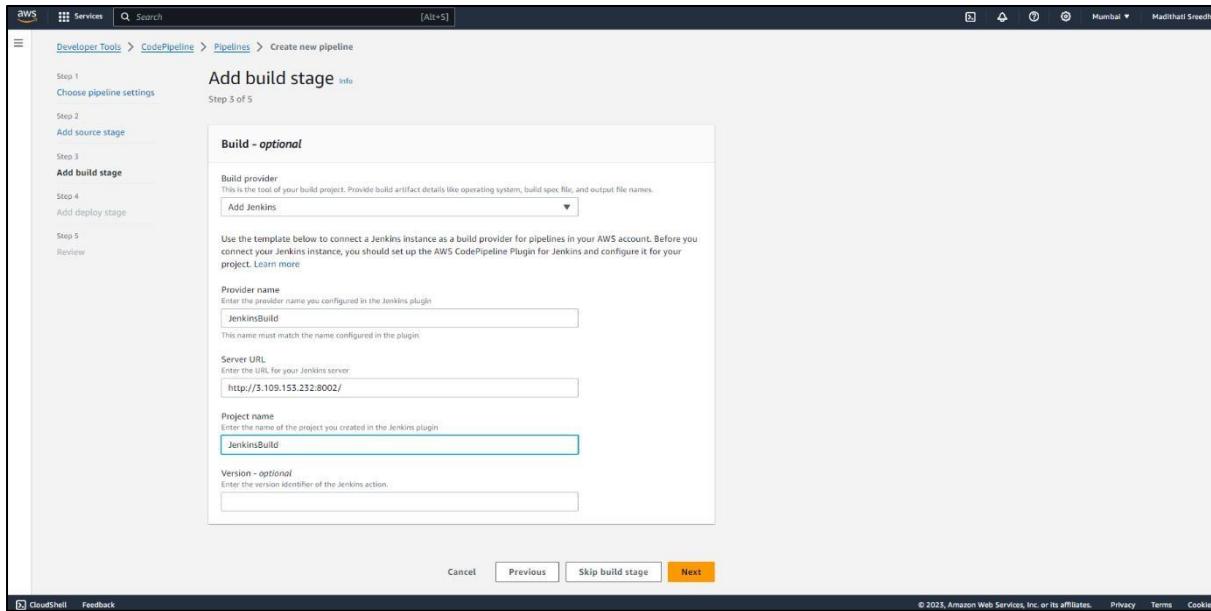
(Note: This name must match the name configured in the Pipeline)

- Server URL: Enter the URL of your Jenkins server

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

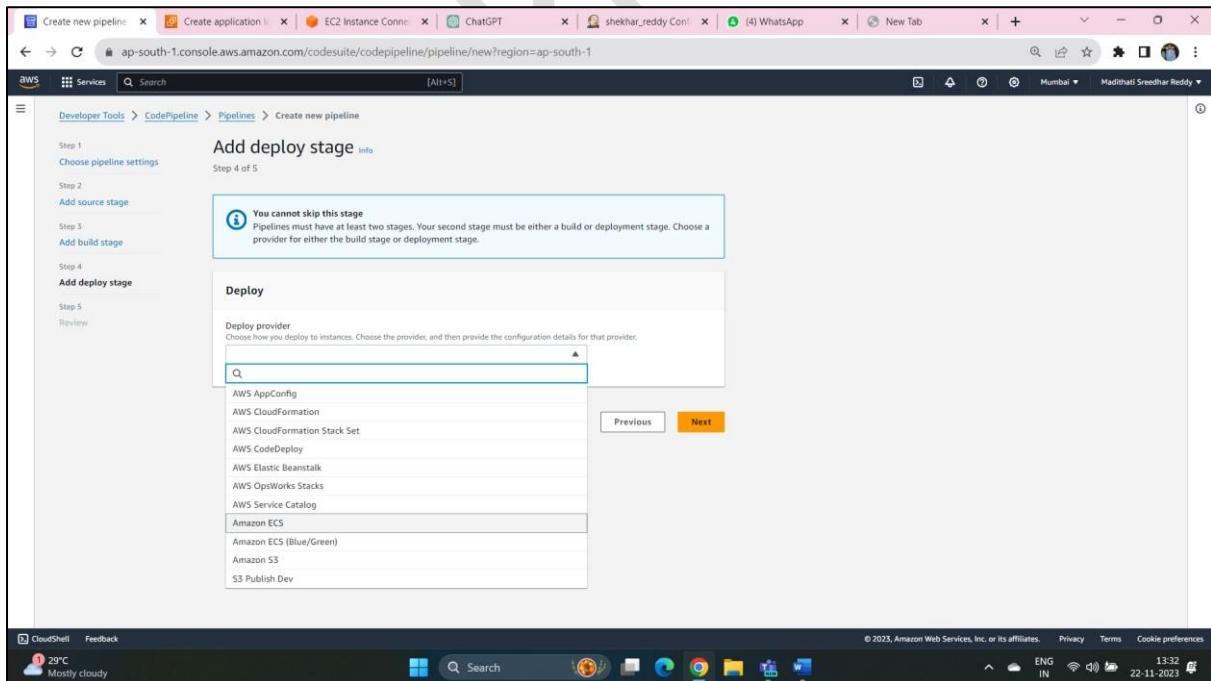
- Project name: Enter the project name you created in the Jenkins plugin



- Click next

Configure Deploy Stage:

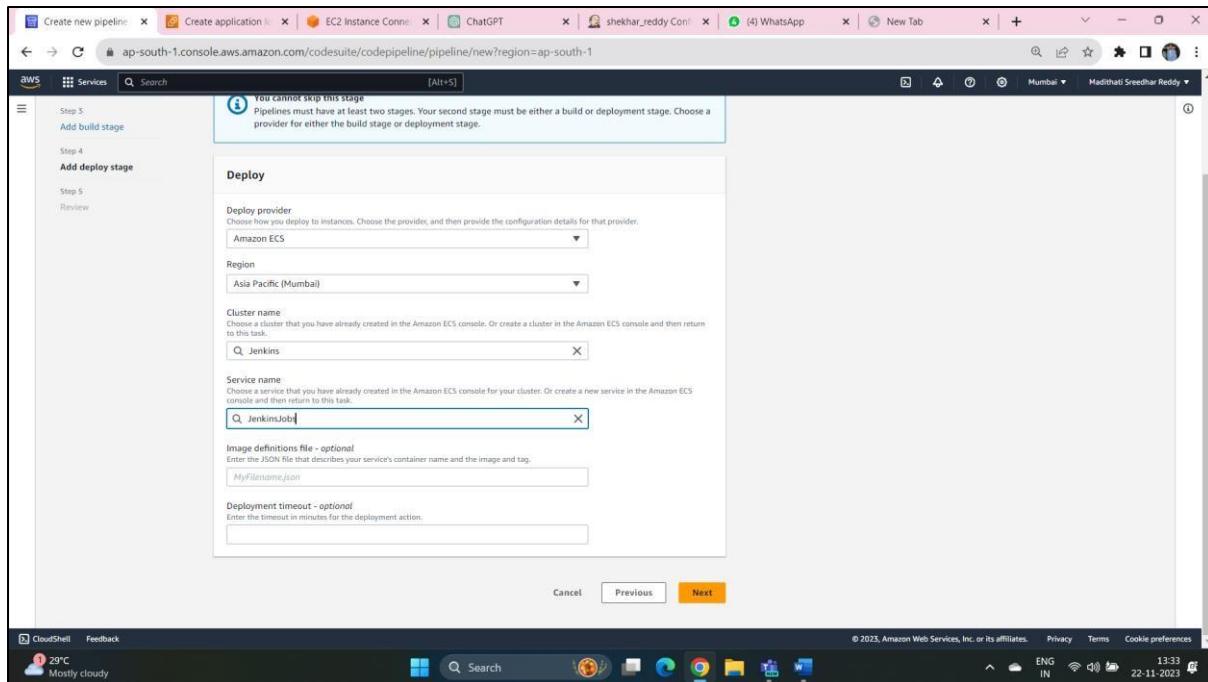
- Add a deployment stage based on your deploy provider (ECS).



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

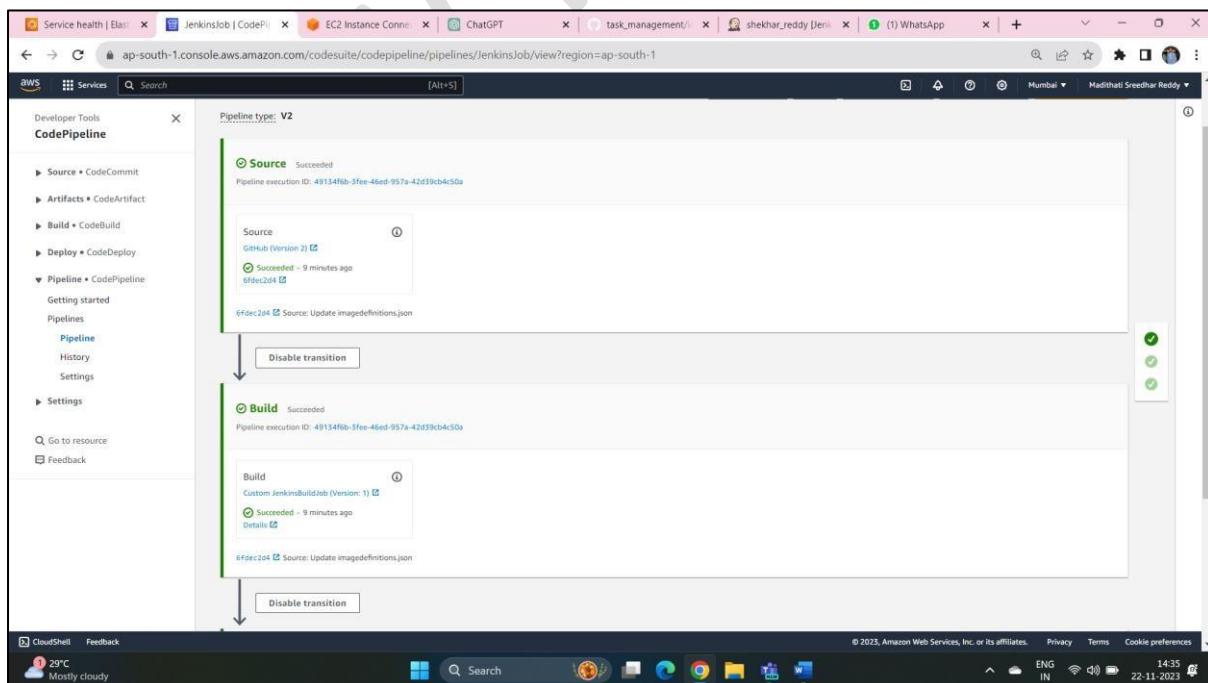
Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



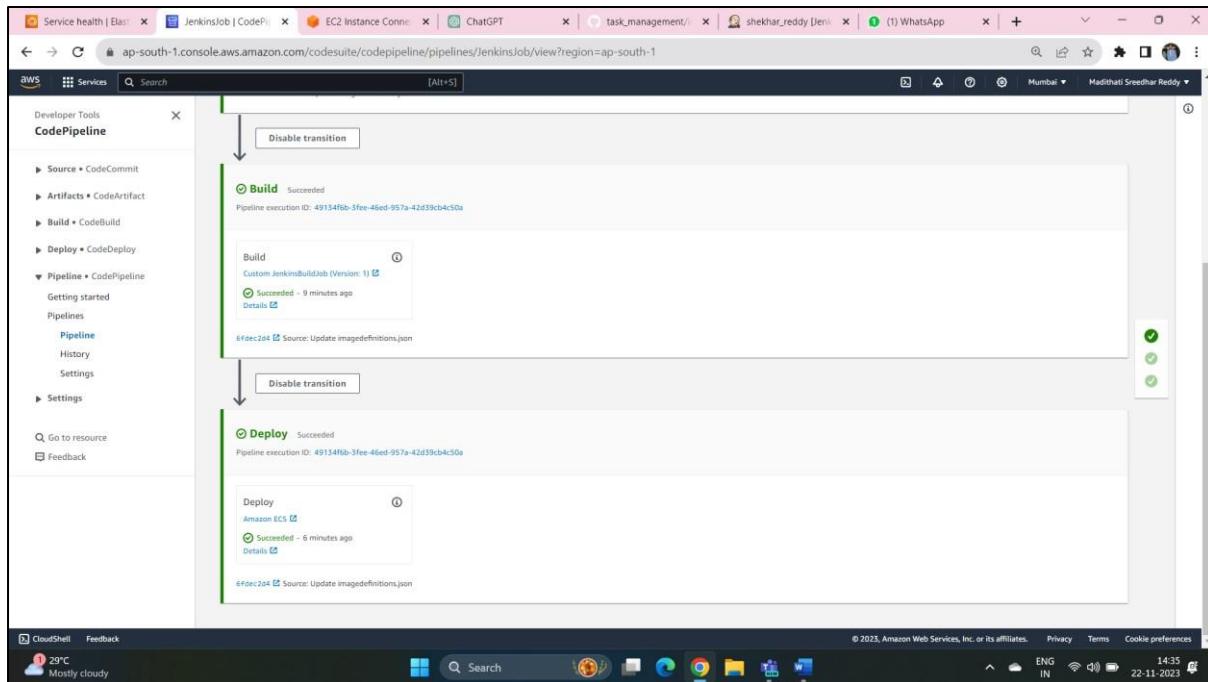
- Click on Next.
- Add additional stages if you have more steps in your pipeline (e.g., testing, approval).
- Review the pipeline configuration.
- Click the "Next" button to create the pipeline.

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.



Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



- 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 'd47f3181-0036-4bff-afbd-5fc6c7e4254' received
[AWS CodePipeline Plugin] Acknowledged job with ID: d47f3181-0036-4bff-afbd-5fc6c7e4254
[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#MoA7z?' to '/var/jenkins_home/workspace/shekhar_reddy'
[AWS CodePipeline Plugin] Artifact uncompression successfully
[shekhar_reddy] $ /bin/sh -xe /tmp/jenkins5092329920073020503.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: 2B 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:b6cc878074fdcc6aff44867f42bf4cc6d10f4e71ed44027649856355b0e23dbe4
#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 /home/python
#7 CACHED

```

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the Jenkins job console output for the build #8. The output is a long list of log entries from a Docker container. It includes messages like 'Preparing', 'Waiting', and 'Layer already exists' for various Docker layers. It also shows the publishing of artifacts, compressing the workspace directory into a zip archive named 'jenkins_home/workspace/shekhar_reddy'. The artifact is uploaded to the AWS S3 bucket 'codepipeline-ap-south-1-725747133598'. The build concludes with a success message: '[AWS CodePipeline Plugin] Build succeeded, calling PutJobSuccessResult' and 'Finished: SUCCESS'.

```

Not secure | 43.204.58.64:8080/job/shekhar_reddy/8/console

Dashboard > shekhar_reddy > #8 > Console Output

2bb0d7f7d92e: Preparing
29e4b059e6da: Preparing
1777ac7d3d97b: Preparing
5f70bf18a086: Waiting
07b0be403b01: Waiting
f8a3a37145d5: Waiting
70866f64c093: Waiting
2d788bc474240: Waiting
06e50e0709ee: Waiting
12b95692792a21: Waiting
266edf75d28e1: Waiting
29e4b059e6da1: Waiting
1777ac7d3d97b: Waiting
e27b52d85899: Layer already exists
0a3ca610895c: Layer already exists
44b549f5: Layer already exists
15a170a5454d1: Layer already exists
e7939e15c6b: Layer already exists
5f70bf18a086: Layer already exists
70866f64c093: Layer already exists
f8a3a37145d5: Layer already exists
07b0be403b01: Layer already exists
2d788bc474240: Layer already exists
266edf75d28e1: Layer already exists
1777ac7d3d97b: Layer already exists
12b95692792a21: Layer already exists
06e50e0709ee: Layer already exists
29e4b059e6da1: Layer already exists
latest, digest: sha256:97b4114fa25588d173f5e42fc17cbe37d4814f402a127cc8d81dbaf6c2425 size: 3461
[AWS CodePipeline Plugin] Publishing artifacts
[AWS CodePipeline Plugin] Compressing directory '/var/jenkins_home/workspace/shekhar_reddy' as a 'zip' archive
[AWS CodePipeline Plugin] Uploading artifact: {Name: BuildArtifact, Location: {Type: S3, S3Location: {BucketName: codepipeline-ap-south-1-725747133598, ObjectKey: Jenkins/BuildArtif/yOkaTxY}}}, file: /tmp/shekhar_reddy-12623279406983855616.zip
[AWS CodePipeline Plugin] Upload successful
[AWS CodePipeline Plugin] 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.

The screenshot shows the AWS S3 console interface. The left sidebar shows navigation options like Buckets, Storage Lens, and Feature spotlight. The main area displays the contents of the 'Jenkins' folder within the 'codepipeline-ap-south-1-725747133598' bucket. The 'Objects' tab is selected, showing two items: 'BuildArtif/' and 'SourceArtif/'. The 'Actions' menu is open, with the 'upload' option highlighted.

Name	Type	Last modified	Size	Storage class
BuildArtif/	Folder	-	-	-
SourceArtif/	Folder	-	-	-

- 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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the AWS ECS Service Health interface. On the left, there's a sidebar with navigation links like Clusters, Namespaces, Task definitions, Account settings, and more. The main area has tabs for Health and metrics, Tasks, Logs, Deployments, Events, Configuration, Networking, and Tags. Under the Status tab, it shows ARN (arnaws:ecs:ap-south-1:862547479026:service/Jenkins/jenkinsJobs), Status (Active), and a completed deployment. It also displays the Application Load Balancer (EC2) configuration, including Listener protocol (HTTP:5004), Target group name (protocol:taskHTTP), and Health check path (/). Below this, there's a section for CPU utilization and Memory utilization over time. At the bottom, there are links for CloudShell and Feedback.

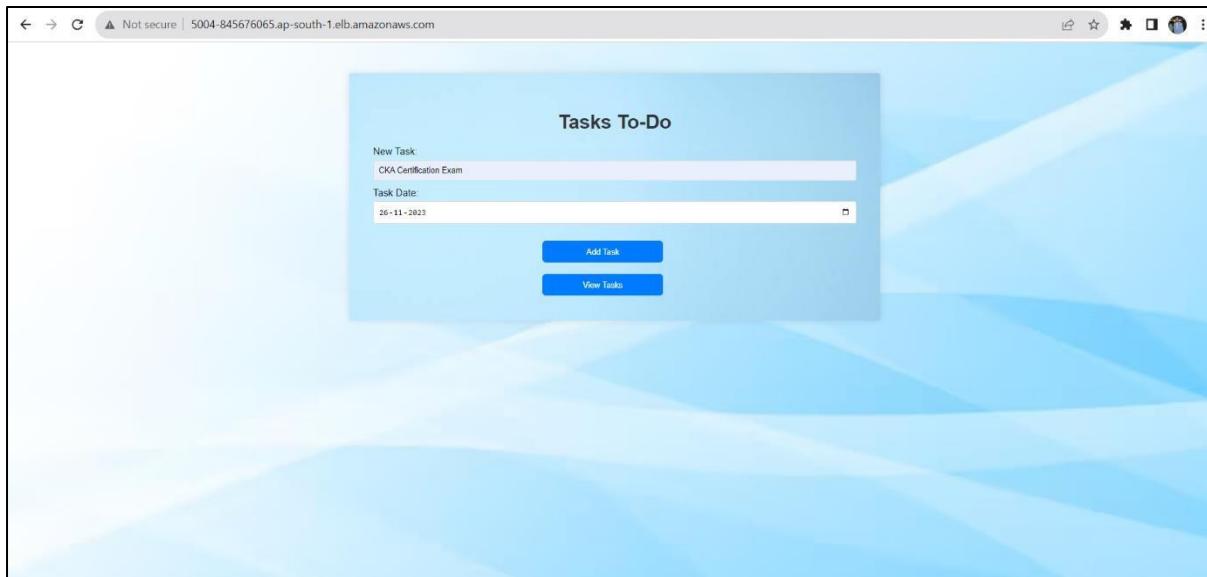
- Click on View Load Balancer.
- Then Copy the DNS Name of it and paste it in your browser

The screenshot shows the AWS EC2 Load Balancer Details page. The left sidebar includes EC2 Dashboard, Instances, Images, and Network & Security sections. The main area shows the load balancer details: Type (Application), Status (Active), VPC (arn:aws:vpc:ap-south-1:01d8108a6b7d10ed), and Availability Zones (subnets across aps1-az1, az2, az3). The Load balancer ARN is listed as arn:aws:elasticloadbalancing:ap-south-1:862547479026:loadbalancer/app/ECS/e11c533dc25719ef. A callout box highlights the DNS name info, which is ECS-542256558.ap-south-1.elb.amazonaws.com (A Record). Below this, theListeners and rules section shows two rules: one for HTTP:80 forwarding to target group ECS (1 rule) and another for HTTP:5004 forwarding to target group task (1 rule). The bottom of the screen shows the standard Windows taskbar.

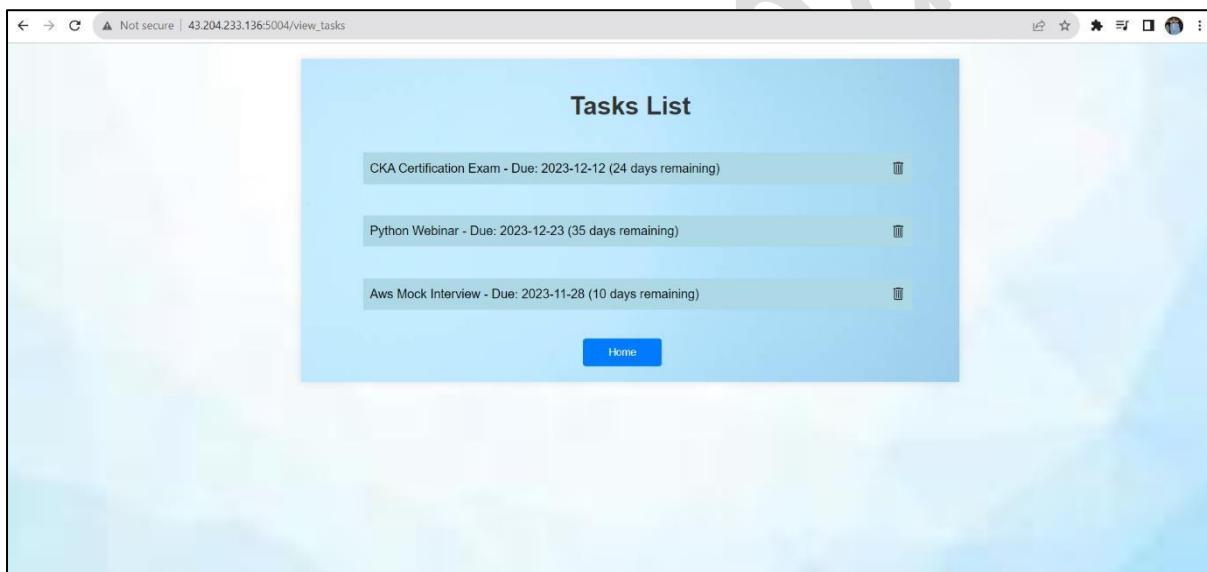
- We can access our Web application using DNS name of the Load balancer.
- The web UI will Look Like below.
- Now add Some Tasks as per your Requirement.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



- 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:

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

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.

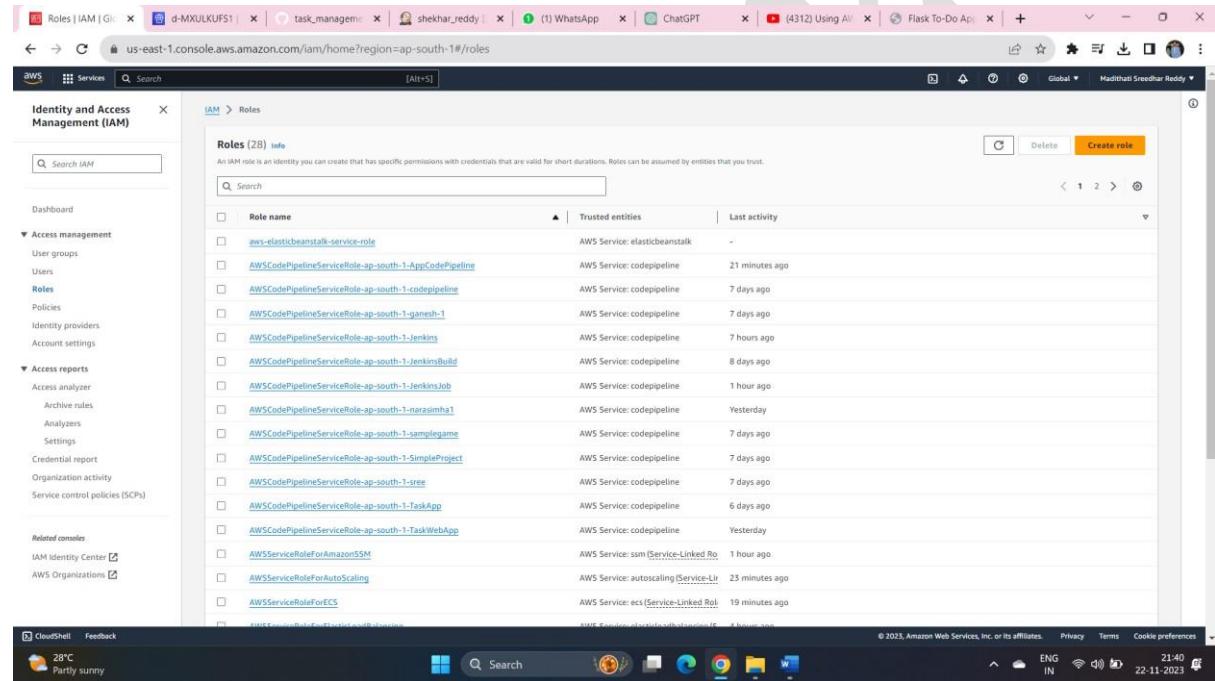
Creation of the IAM Role

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:

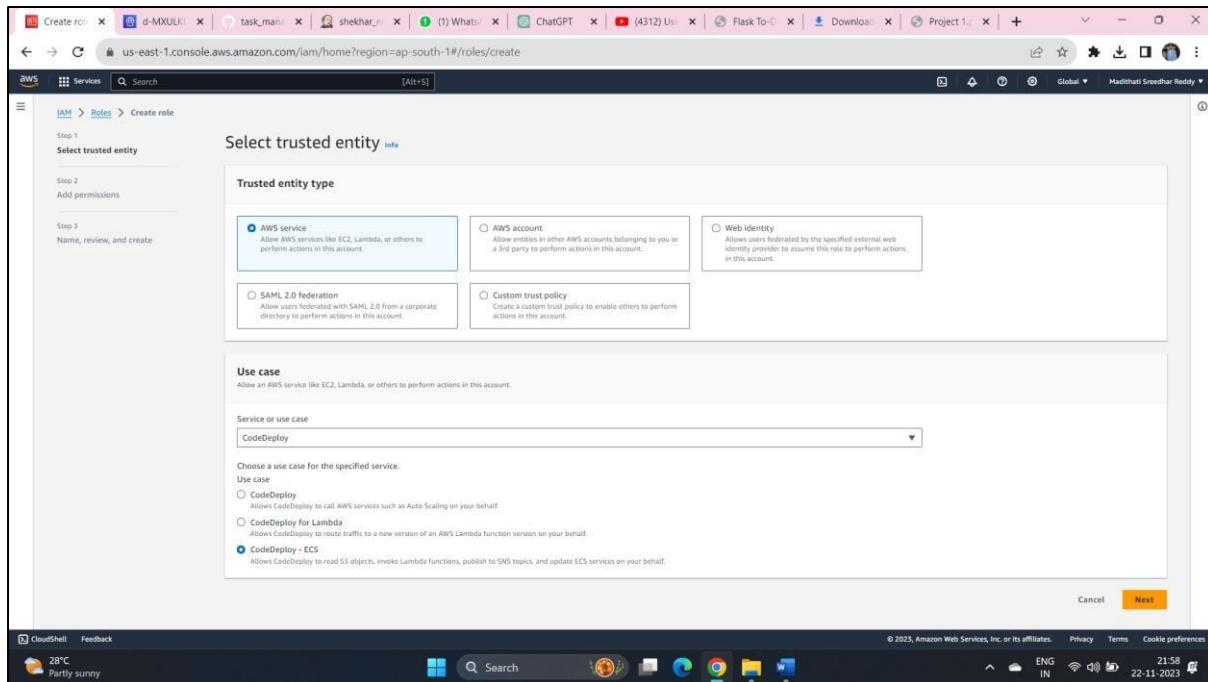
In the IAM dashboard, select "Roles" from the left navigation pane.



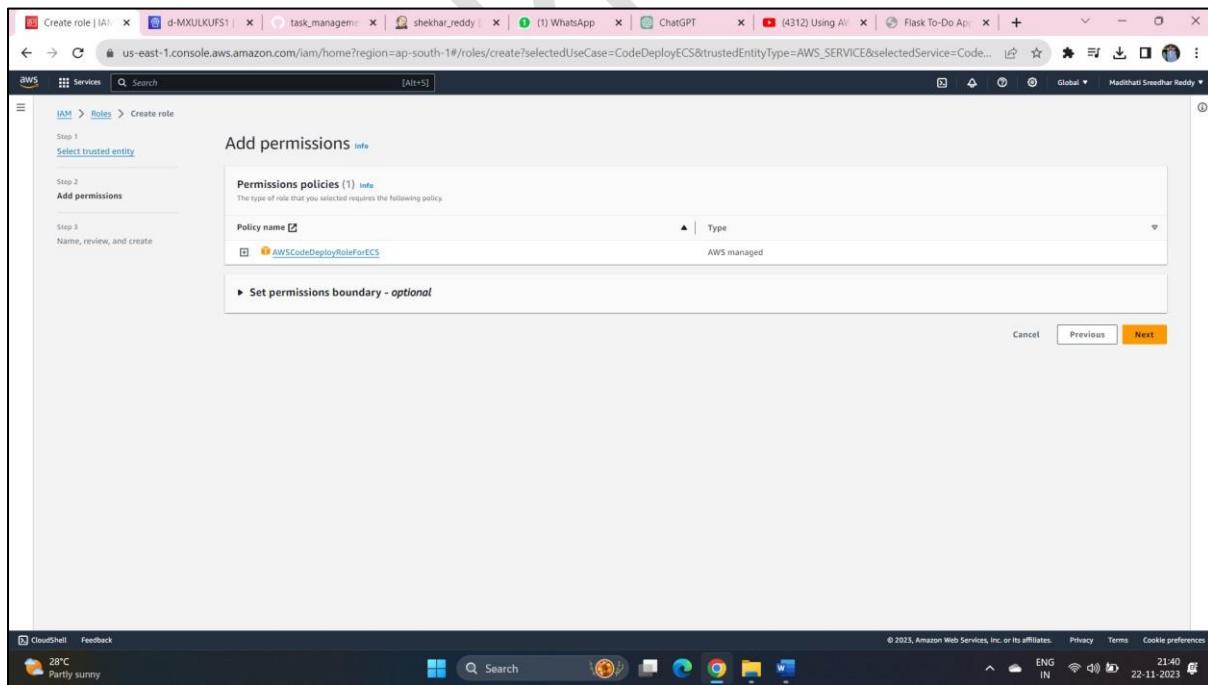
- Click "Create role":
- Click the "Create role" button to start the role creation process.

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



- 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

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]

The screenshot shows the 'Create role' wizard in the AWS IAM console. The current step is 'Step 1: Select trusted entities'. The 'Trust policy' section displays the following JSON code:

```

1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Effect": "Allow",
5         "Principal": "*",
6         "Service": "codedeploy.amazonaws.com"
7     },
8     {
9         "Action": "sts:AssumeRole"
10    }
11 ]
12
13

```

The 'Permissions policy summary' table shows one managed policy attached:

Policy name	Type	Attached as
ANSCodeDeployRoleForECS	AWS managed	Permissions policy

Give the Name for the role .

The screenshot shows the 'Create role' wizard in the AWS IAM console. The current step is 'Step 2: Add permissions'. The 'Permissions policy summary' table shows one managed policy attached:

Policy name	Type	Attached as
ANSCodeDeployRoleForECS	AWS managed	Permissions policy

The 'Step 3: Add tags' section is visible at the bottom.

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:

Follow for more: Vinay Pramanik (LinkedIn)

[AWS Cloud Architect]



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:

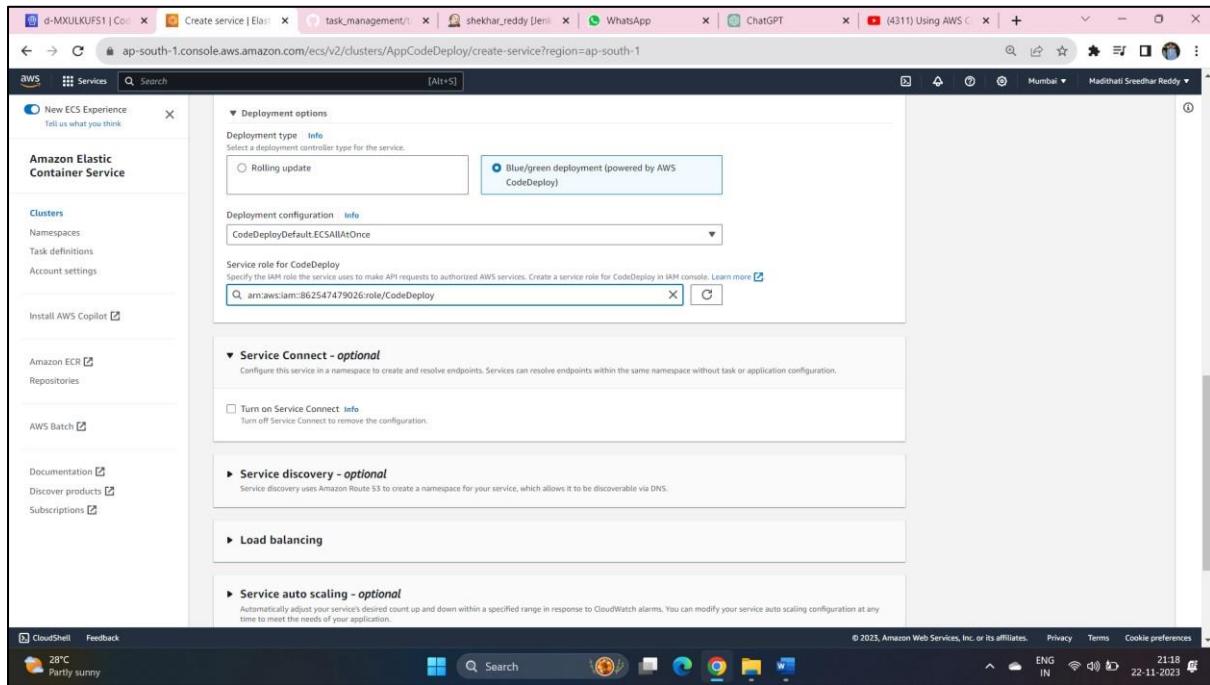
Launch type: Choose whether your tasks will run on EC2 instances or as AWS Fargate tasks.

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.



• **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

The screenshot shows the AWS CloudWatch Metrics console with the URL <https://ap-south-1.console.aws.amazon.com/metrics/home?region=ap-south-1>. The interface includes a search bar, navigation tabs, and a sidebar with links like 'Clusters', 'Namespaces', 'Task definitions', and 'Account settings'. The main content area displays a table of metrics data, with columns for 'Metric Name', 'Dimensions', 'Unit', 'Value', and 'Last Value'. A specific row is highlighted, showing 'aws.ecs.serviceMetrics' with dimensions 'ClusterName:task-management', 'ServiceName:task-management', and 'ContainerName:task-management'. The value is 1.0 and the last value is also 1.0.

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 CloudWatch Metrics console with the URL <https://ap-south-1.console.aws.amazon.com/metrics/home?region=ap-south-1>. The interface includes a search bar, navigation tabs, and a sidebar with links like 'Clusters', 'Namespaces', 'Task definitions', and 'Account settings'. The main content area displays a table of metrics data, with columns for 'Metric Name', 'Dimensions', 'Unit', 'Value', and 'Last Value'. A specific row is highlighted, showing 'aws.ecs.serviceMetrics' with dimensions 'ClusterName:task-management', 'ServiceName:task-management', and 'ContainerName:task-management'. The value is 1.0 and the last value is also 1.0.

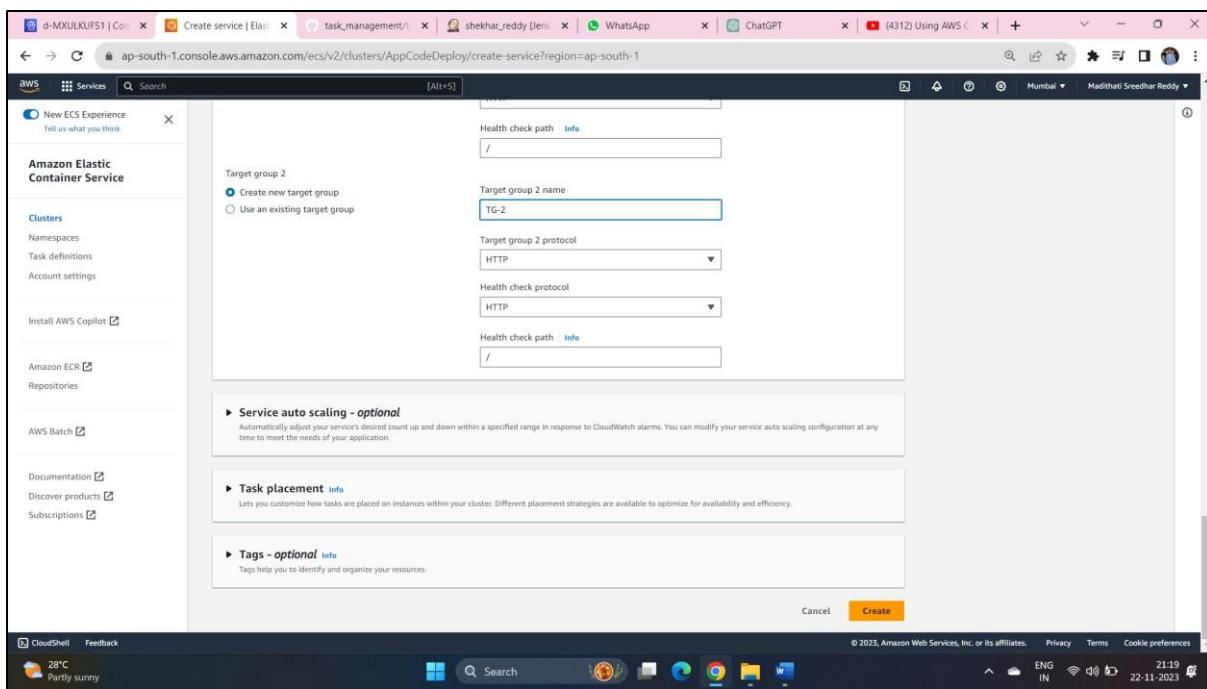


□ Create the Listeners as below

The screenshot shows the AWS Elastic Load Balancing service configuration page. On the left, there's a sidebar with various AWS services like New ECS Experience, Clusters, Task definitions, Account settings, Install AWS Copilot, Amazon ECR, Repositories, AWS Batch, Documentation, Discover products, and Subscriptions. The main area is titled 'Listeners' and contains the following fields:

- Production listener:** A radio button for 'Create new listener' is selected. The 'Production listener port' is set to 5004 and the 'Production listener protocol' is set to HTTP.
- Test listener:** A checkbox for 'Add a test listener' is unchecked. A note says 'An optional test listener is used to test the new application revision before rolling traffic to it.'
- Target groups:** A section for creating new target groups or choosing existing ones. It includes:
 - Target group 1:** A radio button for 'Create new target group' is selected. The 'Target group 1 name' is TG-1, 'Protocol' is set to HTTP, and the 'Health check protocol' is also set to HTTP. The 'Health check path' is set to '/'.
 - Target group 2:** A radio button for 'Create new target group' is selected. The 'Target group 2 name' is TG-2.

- 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.

The screenshot shows the AWS CodeDeploy console with the URL ap-south-1.console.aws.amazon.com/codesuite/codedeploy/applications?region=ap-south-1. The left sidebar is collapsed, showing the navigation menu for CodeDeploy. The main content area displays a table titled 'Applications' with the following data:

Application name	Compute platform	Created
Jenkins	AWS Lambda	6 hours ago
AppECS-AppCodeDeploy-AppCodeDeploy	Amazon ECS	4 hours ago
sample-code	Amazon ECS	8 days ago
webapp	Amazon ECS	8 days ago

At the top right of the main content area, there are buttons for 'Notify', 'View details', 'Deploy application', and a prominent orange 'Create application' button.

- 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. The left sidebar is collapsed, showing the navigation menu for CodeDeploy. The main content area displays the 'Application details' for 'AppECS-AppCodeDeploy-AppCodeDeploy'. The 'Deployment groups' tab is selected, showing a table with the following data:

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

At the top right of the main content area, there are buttons for 'Notify', 'Delete application', and a 'Create deployment group' button.

□ Then Click on the Create Deployment Option

DgpECS-AppCodeDeploy-AppCodeDeploy

Deployment group details

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	Deployment configuration	CodeDeployDefault.ECSAllAtOnce
Rollback enabled	True				

Environment configuration

ECS cluster name	AppCodeDeploy	ECS service name	AppCodeDeploy
------------------	---------------	------------------	---------------

Load Balancing

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	Test listener ARN	-

- Under it Choose the Deployment group

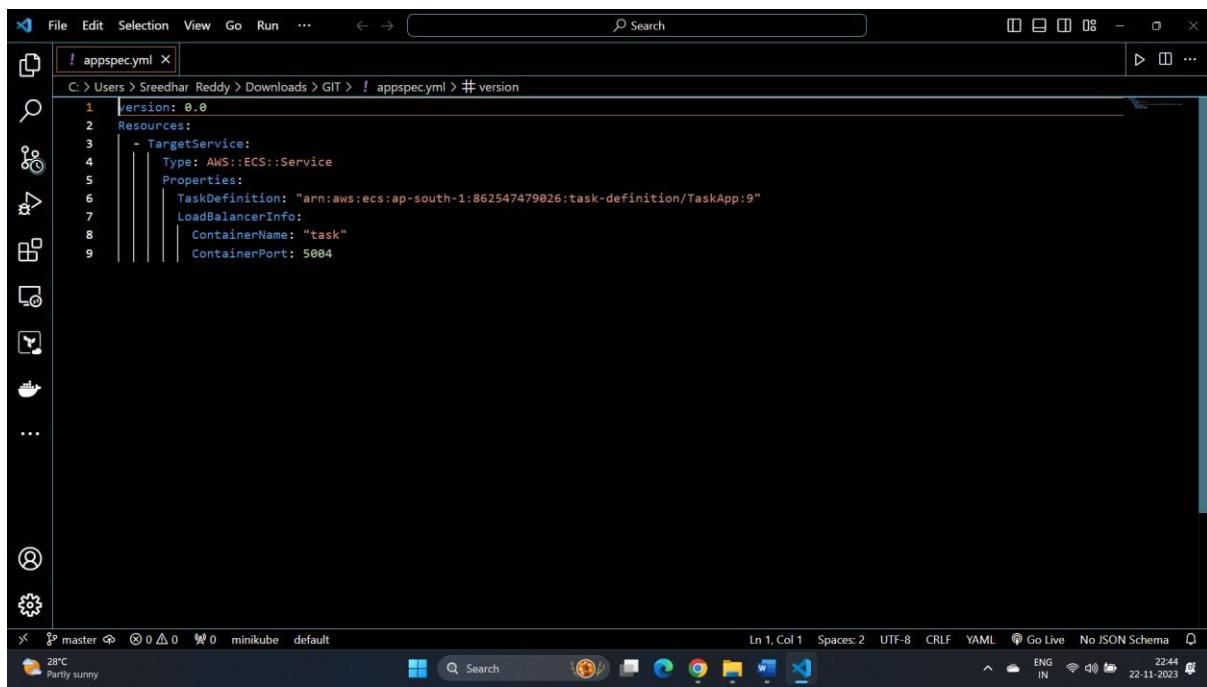
Create deployment

Deployment settings

Application	AppECS-AppCodeDeploy-AppCodeDeploy
Deployment group	DgpECS-AppCodeDeploy-AppCodeDeploy
Compute platform	Amazon ECS
Deployment type	Blue/green
Revision type	<input checked="" type="radio"/> My application is stored in Amazon S3
Revision location	s3://bucket-name/folder/object.[json yaml tgz zip]

Deployment description

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



The screenshot shows a terminal window with the following content:

```
! appspec.yml x
C:\Users\Sreedhar Reddy\Downloads\GIT\! appspec.yml> # version
1 Version: 0.0
2 Resources:
3   - TargetService:
4     Type: AWS::ECS::Service
5       Properties:
6         TaskDefinition: "arn:aws:ecs:ap-south-1:862547479026:task-definition/TaskApp:9"
7         LoadBalancerInfo:
8           ContainerName: "task"
9           ContainerPort: 5004
```

The terminal also displays system information at the bottom:

- git master
- 0 0 0 minikube default
- 28°C Partly sunny
- Search bar
- Icons for various applications (File Explorer, Task View, File History, Task Scheduler, Task Manager, File Explorer, File History, Task Scheduler, Task Manager)
- Ln 1, Col 1 Spaces: 2 UTF-8 CRLF YAML Go Live No JSON Schema
- 22:44 ENG IN 22-11-2023

- Create a new Bucket with Unique name in Amazon S3

The screenshot shows the AWS S3 console with the 'Buckets' tab selected. There are four buckets listed:

Name	AWS Region	Access	Creation date
sree-1110	Asia Pacific (Mumbai) ap-south-1	Public	October 11, 2023, 15:42:35 (UTC+05:30)
sree-1110-staging	Asia Pacific (Mumbai) ap-south-1	Public	October 12, 2023, 12:21:15 (UTC+05:30)
codepipeline-ap-south-1-725747133598	Asia Pacific (Mumbai) ap-south-1	Public	November 11, 2023, 17:55:03 (UTC+05:30)
appspecbucket	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	November 22, 2023, 17:05:25 (UTC+05:30)

□ Click on Create Bucket

- Give an Unique name and click create

The screenshot shows the 'Create bucket' wizard. Step 1: General configuration. Bucket name: appspecbucket, AWS Region: Asia Pacific (Mumbai) ap-south-1. Step 2: Object Ownership. Option: ACUs disabled (recommended). Step 3: Public Access settings. Option: Bucket owner enforced.

Upload the file created in the local to it.

The screenshot shows the AWS S3 console interface for uploading files to a bucket named 'appspecbucket'. The 'Upload' page is displayed, showing a single file 'appspec.yml' selected for upload. The file details are: Name: appspec.yml, Type: file, Size: 274.0 B. The destination is set to 's3://appspecbucket'. There are sections for 'Destination details' (Bucket settings) and 'Properties' (Storage class, encryption, tags). At the bottom right, there is a prominent orange 'Upload' button.

- Once Uploaded S3 URI of it



The screenshot shows the AWS S3 console with the URL <https://s3.console.aws.amazon.com/s3/object/appspecbucket?region=ap-south-1&prefix=appspec.yml>. The object name is `appspec.yml`. The properties tab is selected. Key details shown include:

- Object overview:**
 - Owner: 5742ad64e956aa276e999744631dd585bdb7c24d9ed2a9df05a1a92d2fb246
 - AWS Region: Asia Pacific (Mumbai) ap-south-1
 - Last modified: November 22, 2023, 17:07:19 (UTC+05:30)
 - Size: 274.0 B
 - Type: yaml
 - Key: `appspec.yml`
- Object management overview:** The following bucket properties and object management configurations impact the behavior of this object.
- Bucket properties:** Bucket Versioning (Enabled), Versioning allows multiple variants of an object can be stored in the bucket to easily recover from unintended user actions and application failures.
- Management configurations:** Replication status: When a replication rule is applied to an object the replication status indicates the progress of the operation.

At the bottom, the status bar shows CloudShell, Feedback, 28°C Partly sunny, ENG IN, 22-11-2023, and a battery icon.

Paste it under the Revision location

Choose the Revision File Type as .yaml Click on Create Deployment

Revision type

My application is stored in Amazon S3 Use AppSpec editor

Revision location

Copy and paste the Amazon S3 bucket where your revision is stored
s3://appspecbucket/appspec.yml

Revision file type

yaml

Deployment description

Deployment description - optional
Add a brief description about the deployment

▶ Deployment group overrides

▶ Rollback configuration overrides

Create deployment

- Under Deployments Section The one created will be in progress state

Developer Tools > CodeDeploy > Deployments

Deployment history

	Deployment Id	Status	Deployment ...	Compute pla...	Application	Deployment ...	Revision loca...	Initiating event	Start time	End time
1	d-MXULKUF51	In progress	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://codepipe...	User action	Nov 22, 2023...	-
2	d-B1WN45BS1	Succeeded	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://appspec...	User action	Nov 22, 2023...	Nov 22, 2023...
3	d-NEIF2K3GE	Failed	Blue/green	Amazon ECS	AppECS-App...	DgpECS-App...	s3://appspec...	User action	Nov 22, 2023...	Nov 22, 2023...
4	d-JOR7SECS1	Failed	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 (Routing production traffic to replacement task set) completed at 100% success; Step 3 (Wait 1 hour 0 minutes) waiting 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 group: DgppECS-AppCodeDeploy-AppCodeDeploy, and Status: In progress.

- 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 four steps completed at 100% success. The 'Traffic shifting progress' section shows 'Original' at 0% and 'Replacement' at 100%. Deployment details show the status as Succeeded.

And all the deployment events will be marked as Succeeded

Vinay Pramanik



The screenshot shows the AWS CodeDeploy console for a deployment named d-MXULKUFS1. The left sidebar has a navigation tree under 'CodeDeploy' with sections like Source, Artifacts, Build, Deploy, Pipeline, and Settings. The main area displays 'Revision details' with a revision location of s3://codepipeline-ap-south-1-725747133598/AppCodePipeline/BuildArtif/OxWbUEk?versionId=ROT03jbVXnku1q4ECMu.zhkjWC&eTag=5bafed4964b4c2066c63241be19a6eb1-1. It also shows a 'Task set activity' table with two rows: one for 'ecs-svc/1109913773616991282' (Replacement, PRIMARY, 100% traffic, 1 desired count, 1 running count) and another for 'ecs-svc/2207025905170465844' (Original, ACTIVE, 0 traffic, 1 desired count, 1 running count). Below that is a 'Deployment lifecycle events' table listing various stages from BeforeInstall to AfterAllowTraffic, all of which succeeded.

- Navigate to the created Cluster it will resemble as below

The screenshot shows the AWS ECS console for a cluster named AppCodeDeploy. The left sidebar includes sections for New ECS Experience, Amazon Elastic Container Service, Clusters, Namespaces, Task definitions, Account settings, and other services like ECR and Batch. The main area shows the 'Infrastructure' tab for the cluster. It displays 'Capacity providers' (1) with Infra-ECS-Cluster-App... listed. There are 2 registered container instances. Under 'Container instances' (2), three instances are listed: 313550c1e26342d..., 81e1c7029e46447..., and 81e1c7029e46447... All are Active and assigned to the Infra-ECS-Clu... service in ap-south-1b and ap-south-1a. The CPU available and Memory available columns show values like 954.

- For Accessing Our Web Application Navigate to the EC2 Console

The screenshot shows the AWS CodeDeploy console with a search bar for 'ec2'. The left sidebar lists various AWS services under 'CodeDeploy'. The main search results for 'ec2' show a list of services like EC2, EC2 Image Builder, Recycle Bin, and Amazon Inspector. On the right, a detailed view of deployment history for EC2 is displayed, showing four recent deployments for different application versions (DgpECS-App... v3) initiated by user action on November 22, 2023.

□ Choose the Created Load Balancer

The screenshot shows the AWS EC2 Load Balancers console. The left sidebar has sections for Instances, Images, Elastic Block Store, Network & Security, and more. The main area displays a table for 'Load balancers (1)'. The table includes columns for Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created. One entry is listed: 'CodeDeploy' with a DNS name of 'CodeDeploy-20207529.ap...', state 'Active', VPC ID 'vpc-01d8108a86b7d1...', 3 Availability Zones, type 'application', and a creation date of November 22, 2023, 16:46 (UTC+05:30).

- Copy the DNS Name of it

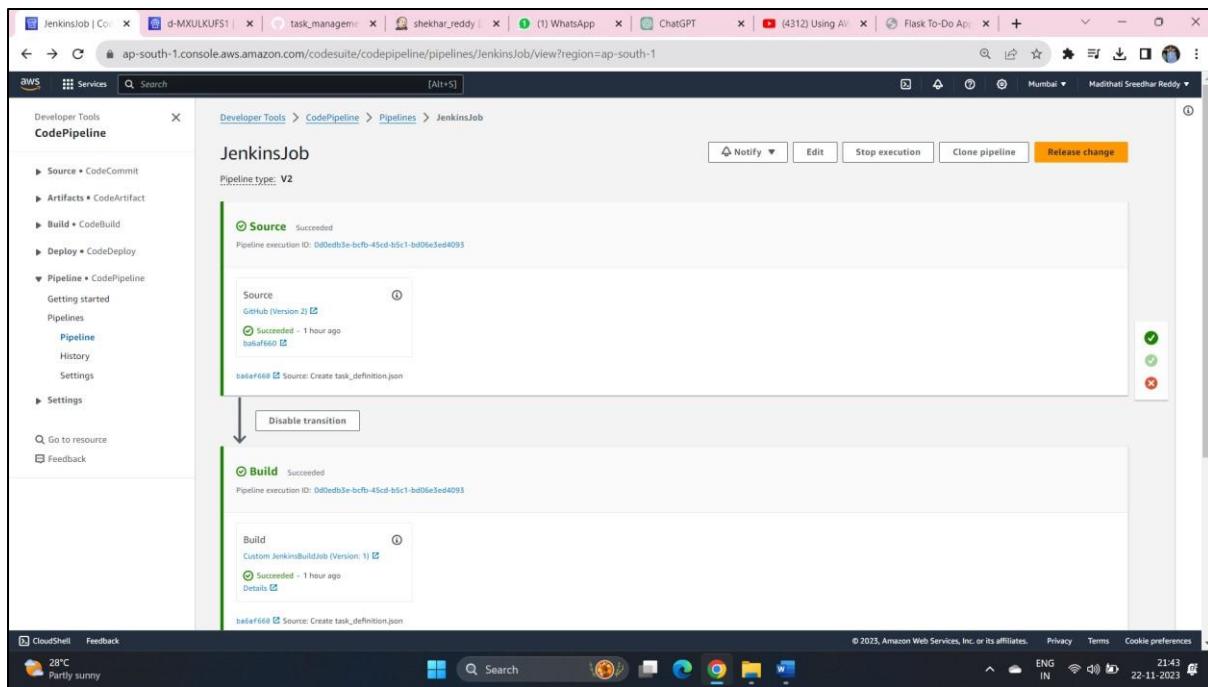
The screenshot shows the AWS EC2 Load Balancer configuration page. The main panel displays the 'CodeDeploy' load balancer details, including its type (Application), status (Active), VPC (vpc-01d8108a86b7d10ed), and IP address type (IPv4). It also lists the availability zones (ap-south-1c, ap-south-1b, ap-south-1a) and the DNS name (codeDeploy-20207529.ap-south-1.elb.amazonaws.com). The left sidebar shows navigation links for EC2 Dashboard, Services, Instances, Images, Elastic Block Store, Network & Security, and CloudShell.

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

The screenshot shows a web browser window displaying the 'Tasks To-Do' application. The interface includes fields for 'New Task:' and 'Task Date:' (dd-mm-yyyy), along with 'Add Task' and 'View Tasks' buttons. The background features a blue abstract design. The browser's address bar shows the URL: Not secure | codedeploy-20207529.ap-south-1.elb.amazonaws.com.

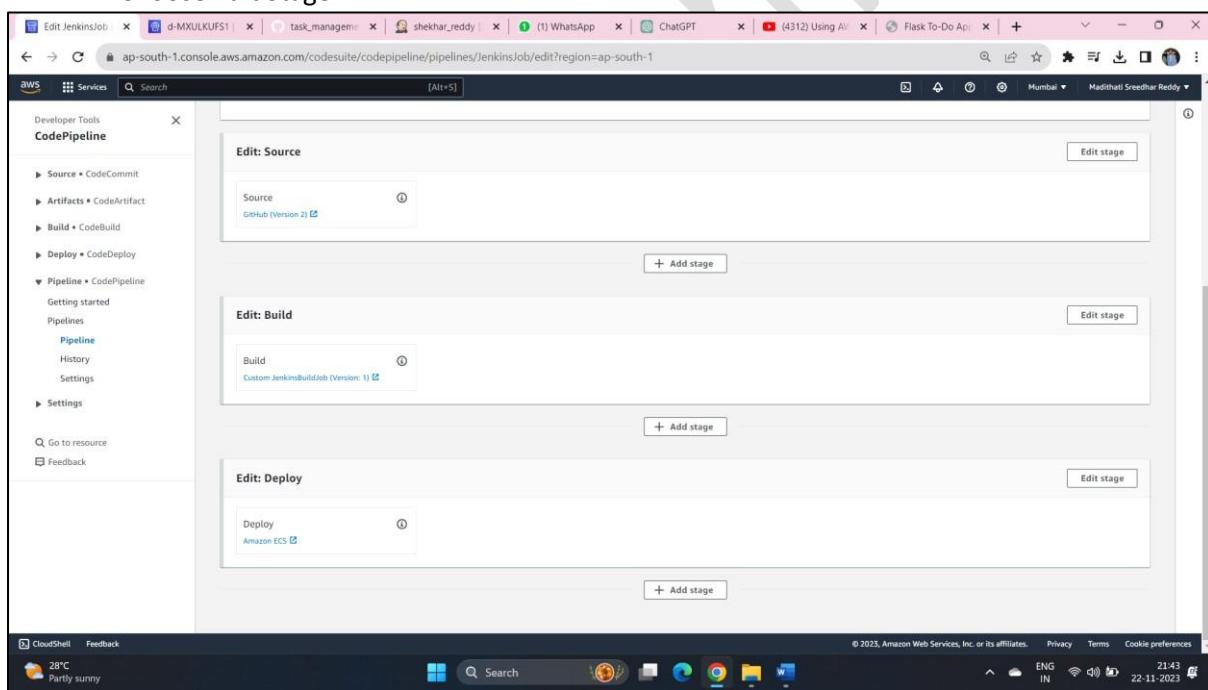
Configuring CodePipeline For Updates:

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

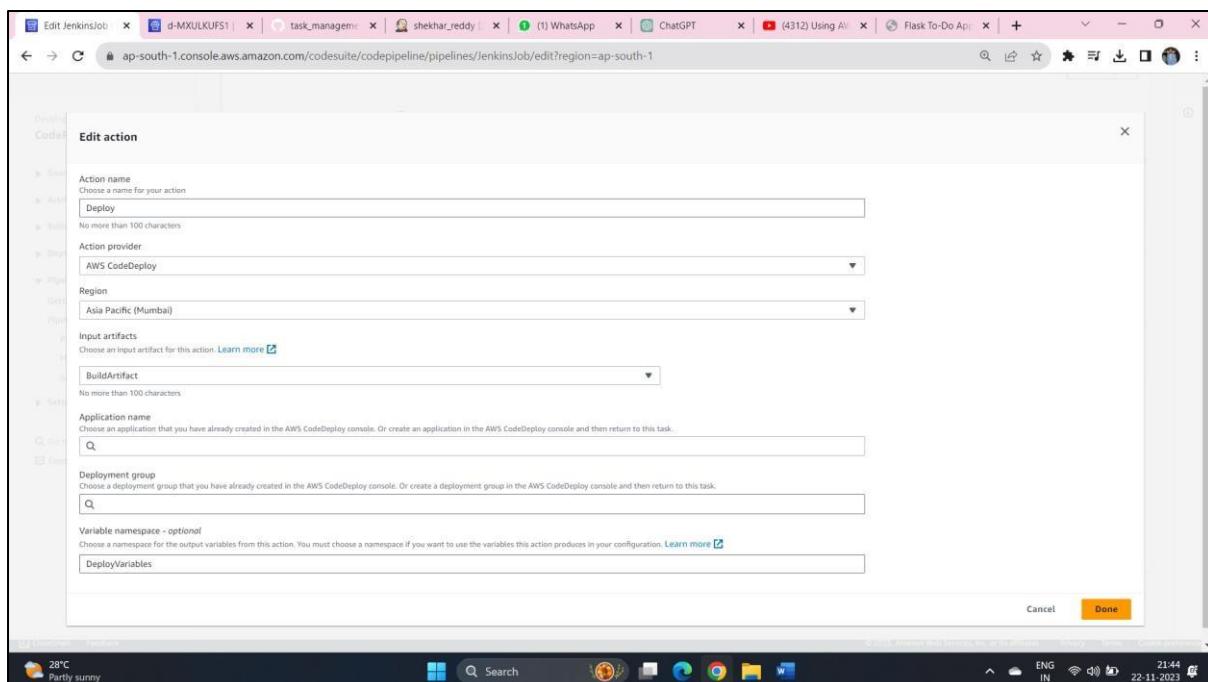


□ Navigate to the Deploy Stage

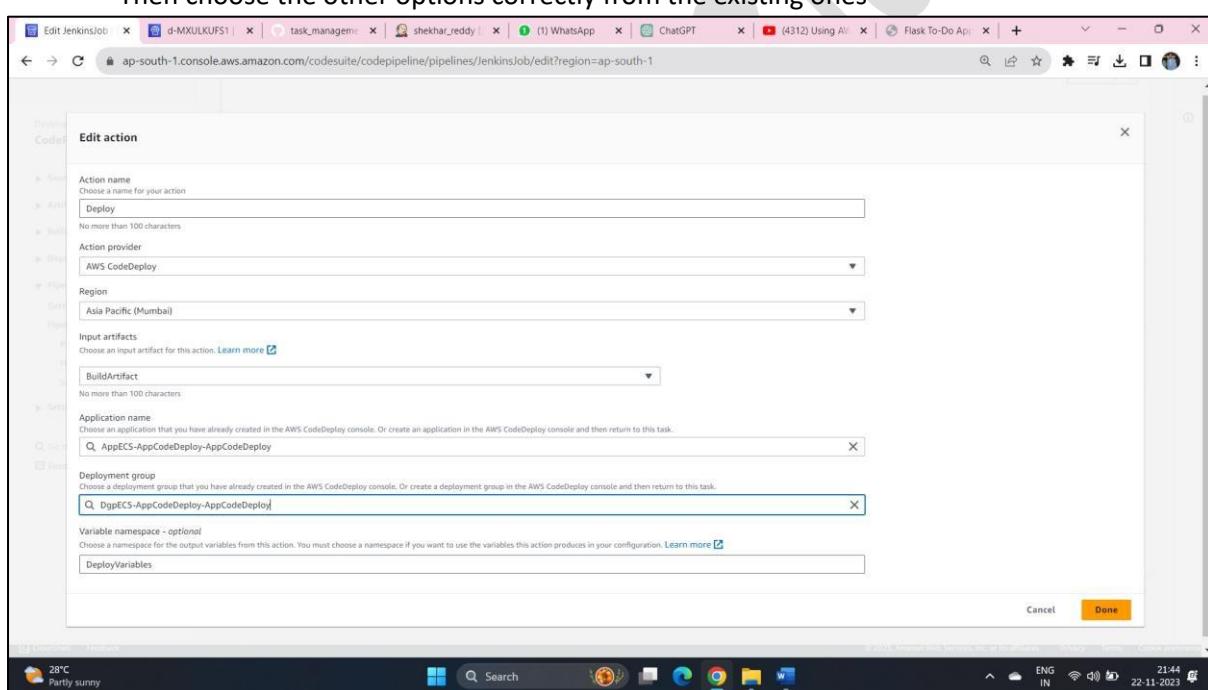
- Choose Edit Stage



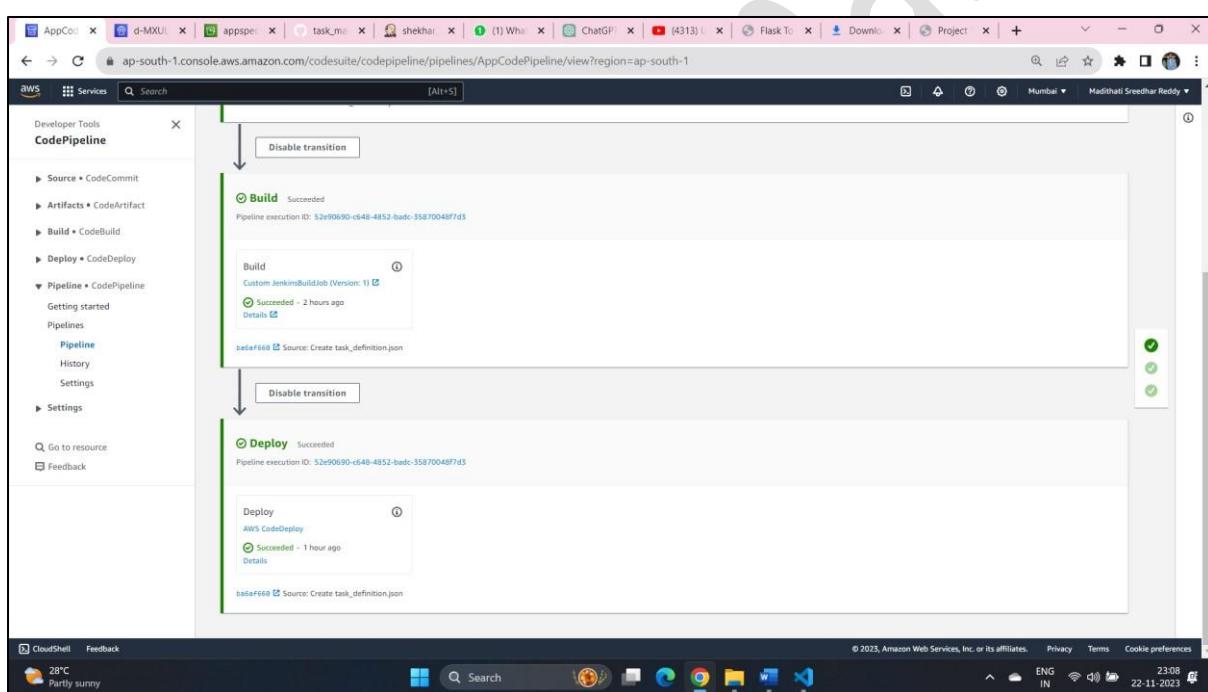
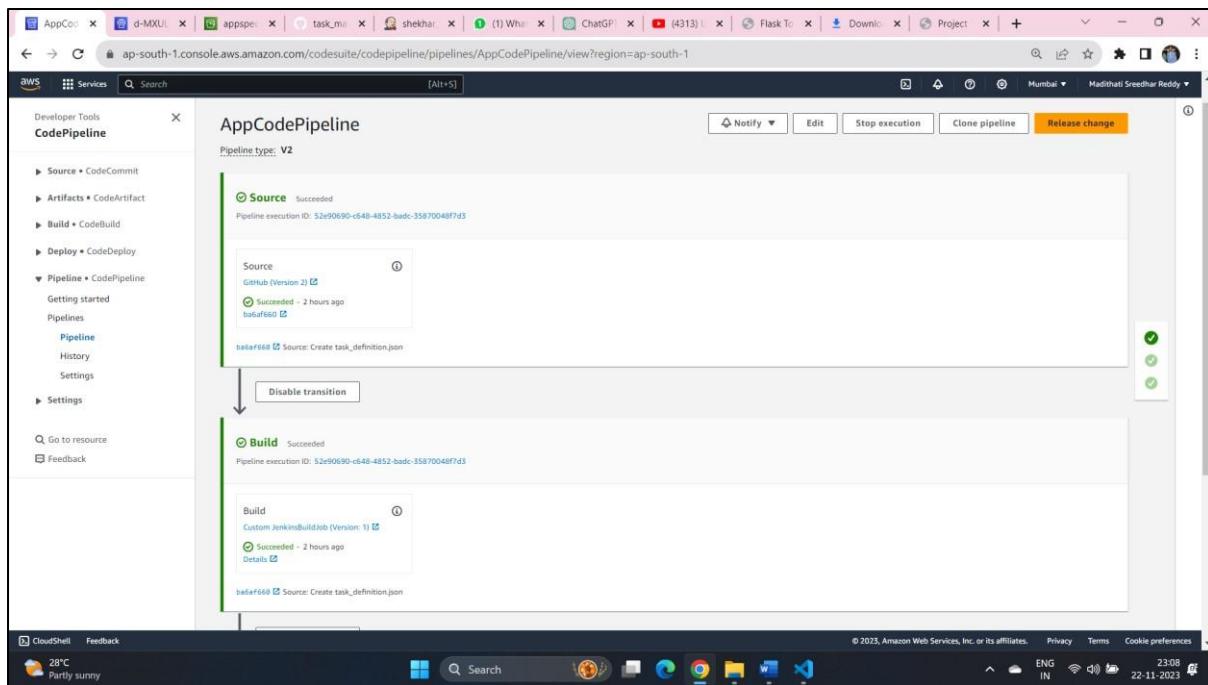
- Choose the Action Provider as AWS CodeDeploy



- 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.

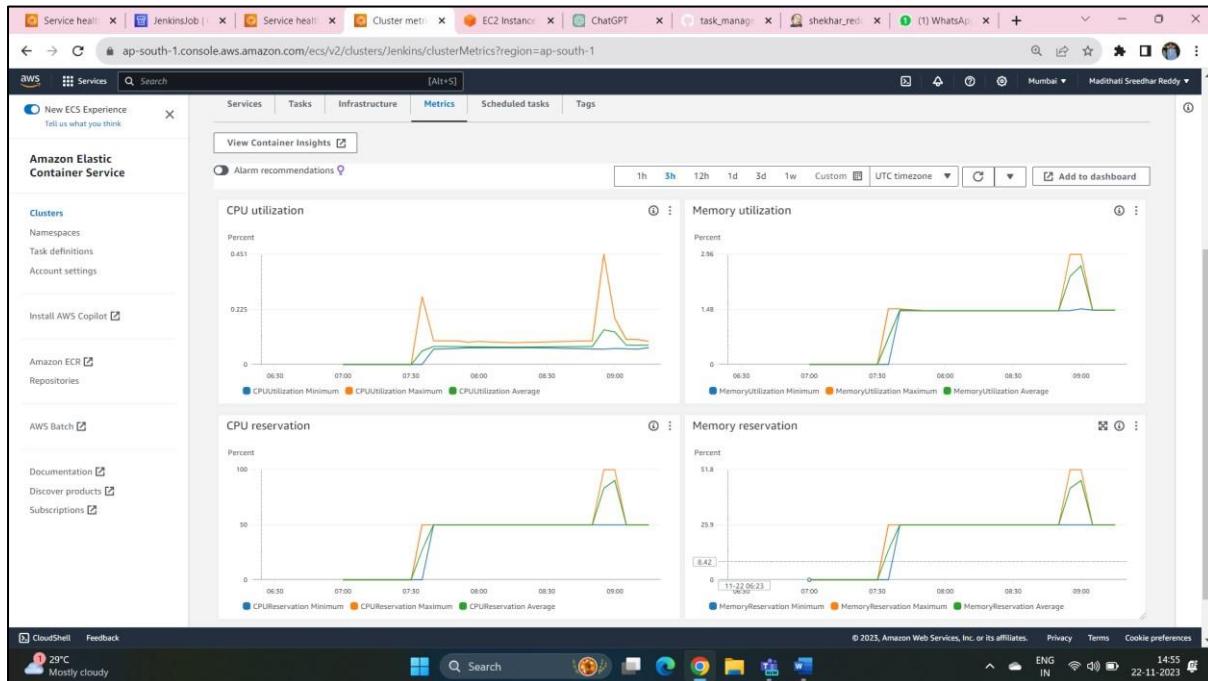


- Everytime we Release the changes It will be running in one instance that is named as Blue with target group TG-1
- Another will be the old Version of our Application running in another instance that is named as Green with Target group TG-2
- If anything goes wrong the loadbalancer will reroute the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

Monitoring:

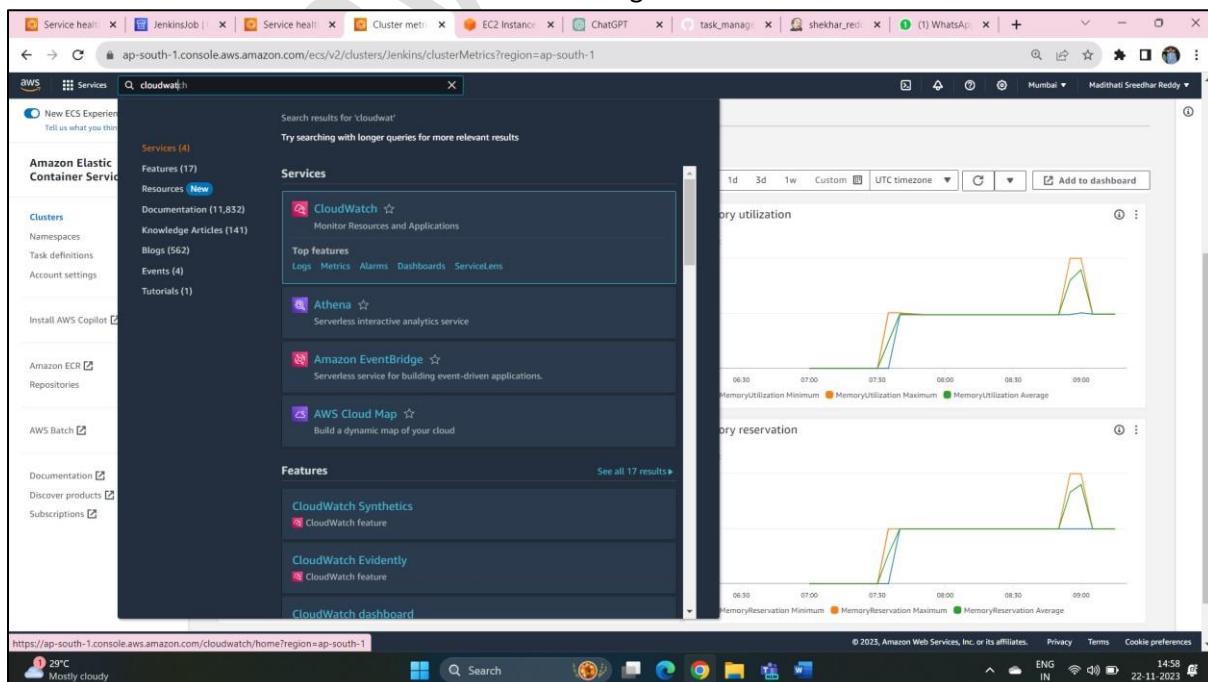
1. View the metrics

- 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 Metrics console. On the left sidebar, under the 'Logs' section, 'Log groups' is selected. A single log group is listed: 'arn:aws:logs:ap-south-1:862547479026:log-group:/aws/ecs/containerinsights/jenkins/performance;'. Below this, the 'Log streams' tab is active, showing 8 log streams. Each stream is represented by a row with a checkbox, the stream name (e.g., 'AgentTelemetry-66e1817e9a5147d58f7d97172c8aa2ac'), and the last event time (e.g., '2023-11-22 14:53:00 (UTC+05:30)').

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

The screenshot shows the AWS CloudWatch Metrics console with a more detailed view. The path 'CloudWatch > Log groups > /aws/ecs/containerinsights/jenkins/performance > AgentTelemetry-66e1817e9a5147d58f7d97172c8aa2ac' is visible in the breadcrumb navigation. Under the 'Log events' section, there are three entries. The first entry is timestamped '2023-11-22T14:26:00.000-05:30' and contains a large JSON object representing log data. The second entry is '2023-11-22T14:26:00.000-05:30' and the third is '2023-11-22T14:27:00.000-05:30'. The log data includes fields like 'ContainerName', 'Image', 'ContainerDefinitionArn', 'Memory', 'CpuReserved', 'StorageReadBytes', 'StorageWrittenBytes', and various network metrics.

- 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 services, Dashboards, Alarms (with 2 items), Logs, Metrics (with All metrics, Explorer, Streams), X-Ray traces, Events, Rules, Event Buses, Application monitoring, ServiceLens Map, Resource Health, Internet Monitor, and Synthetic APIs. The main content area displays a table of alarms:

Name	State	Last state update	Conditions	Actions
TargetTracking-Infra-ECS-Cluster-Jenkins-a2feef9- ECSAutoScalingGroup-y1eHvKScnJ-AlarmLow-d078120c- 7997-45b6-acf1-417569a6fd67	In alarm	2023-11-22 09:19:02	CapacityProviderReservation < 100 for 15 datapoints within 15 minutes	Actions enabled
ecs	In alarm	2023-11-22 07:39:20	ContainerInstanceCount > 1 for 2 datapoints within 2 minutes	Actions enabled

At the bottom of the page, there is a link to 'Create alarm'.

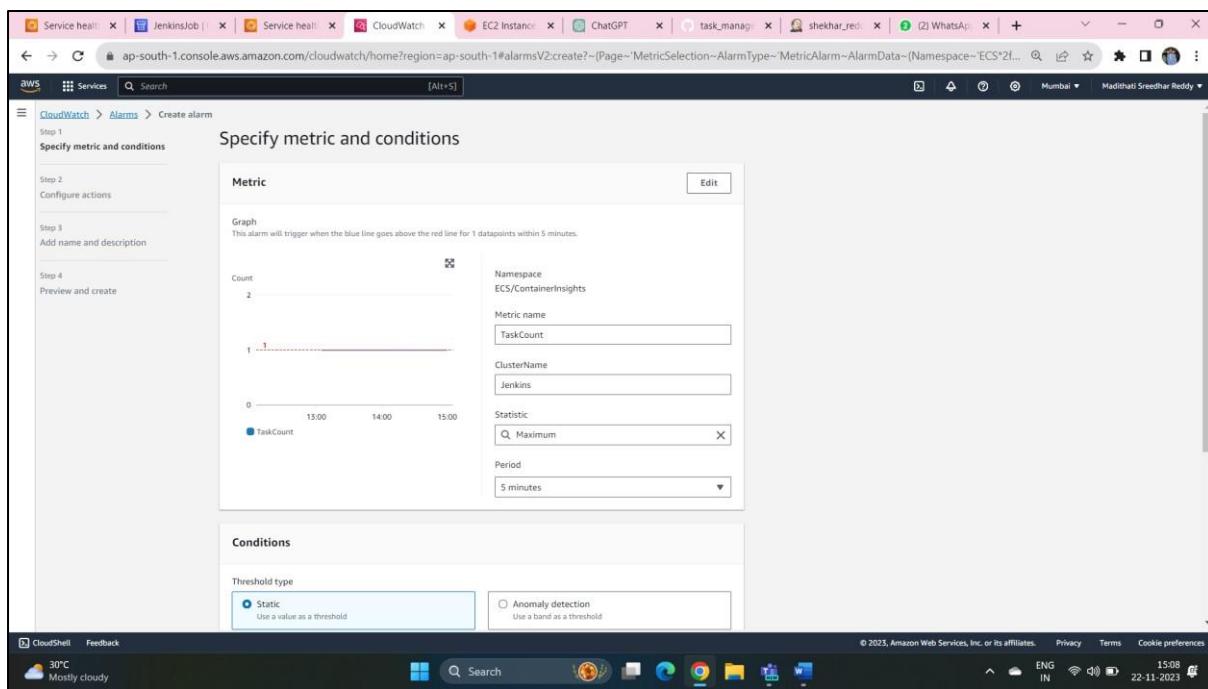
- Click on Create Alarm
- Then select the metric you want to create alarm

The screenshot shows the 'Select metric' dialog box. It has tabs for Browse, Query, Graphed metrics (1), Options, and Source. The 'Browse' tab is active. The list of metrics includes:

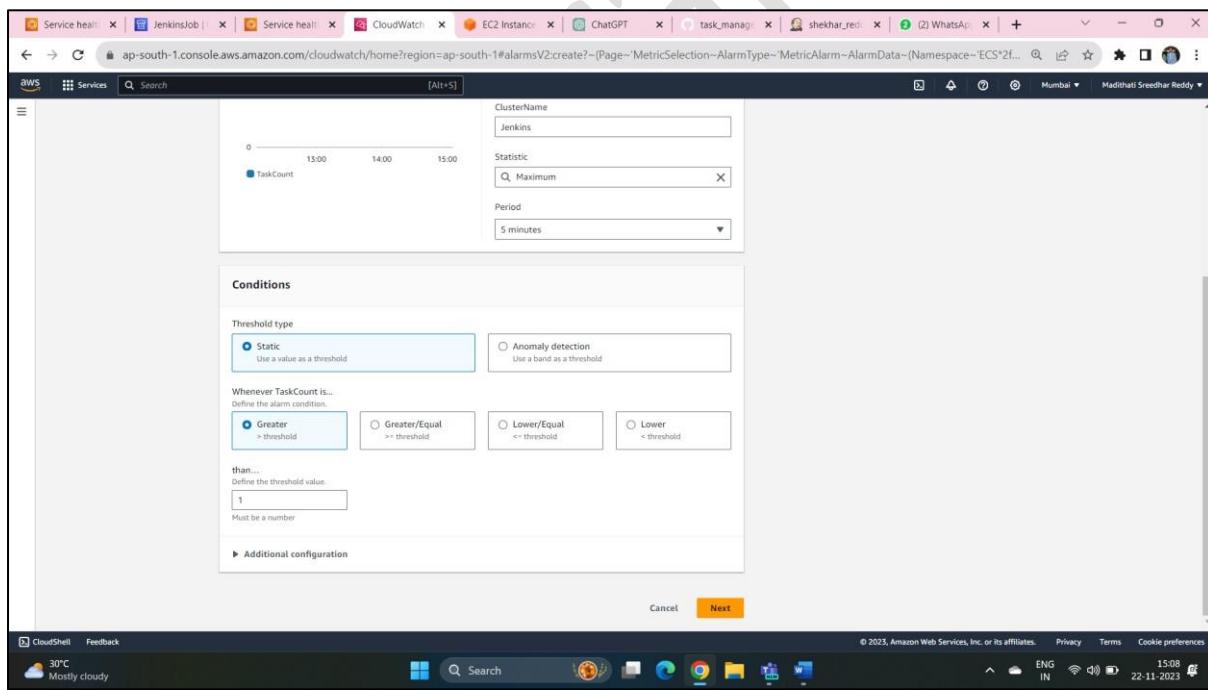
- build-aws.ServiceCount
- build-aws.ContainerInstanceCount
- Jenkins.StorageWriteBytes
- Jenkins.NetworkTxBytes
- Jenkins.TaskCount** (selected)
- Jenkins.CpuUtilized
- Jenkins.StorageReadBytes
- Jenkins.MemoryReserved
- Jenkins.MemoryUtilized
- Jenkins.NetworkRxBytes
- Jenkins.CpuReserved
- Jenkins.ServiceCount

At the bottom right of the dialog 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 URL [ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#alarmsV2:create?~\[Page=~Actions~AlarmType~MetricAlarm~AlarmData~\(Namespace~'ECS*2fContain...](https://ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#alarmsV2:create?~[Page=~Actions~AlarmType~MetricAlarm~AlarmData~(Namespace~'ECS*2fContain...). The page is titled "Configure actions". On the left, there's a sidebar with steps: Step 1 (Specify metric and conditions), Step 2 (Configure actions), Step 3 (Add name and description), and Step 4 (Preview and create). The main area is titled "Notification". It defines an alarm state trigger ("Define the alarm state that will trigger this action") with three options: "In alarm" (selected), "OK" (The metric or expression is outside of the defined threshold), and "Insufficient data" (The alarm has just started or not enough data is available). Below this, it says "Send a notification to the following SNS topic" and provides options to "Select an existing SNS topic", "Create new topic" (which is selected), and "Use topic ARN to notify other accounts". A text input field contains "Default_CloudWatch_Alarms_Topic". It also says "Create a new topic..." and "Email endpoints that will receive the notification...". The email endpoint listed is "medhithasreedhar123@gmail.com". There are two other entries: "user1@example.com, user2@example.com". At the bottom are "Create topic" and "Add notification" buttons.

- It will send the Subscription mail Accept it

The screenshot shows a Gmail inbox with the URL mail.google.com/mail/u/0/#inbox/1MfcgGwHgPGicqFKHvdjWKQQLkxMS. The subject of the email is "AWS Notification - Subscription Confirmation" from "AWS Notifications <no-reply@sns.amazonaws.com>". The email body starts with "You have chosen to subscribe to the topic: arn:aws:sns:ap-south-1:862547479026:ecs". It continues with "To confirm this subscription, click or visit the link below (if this was in error no action is necessary):" followed by a blue "Confirm subscription" link. Below this, it says "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](#)". At the bottom are "Reply" and "Forward" buttons.

- Click on Create topic

The screenshot shows the AWS SNS 'Topics' page. A new topic named 'ecs' has been created. The 'Subscriptions' tab is active, displaying one subscription:

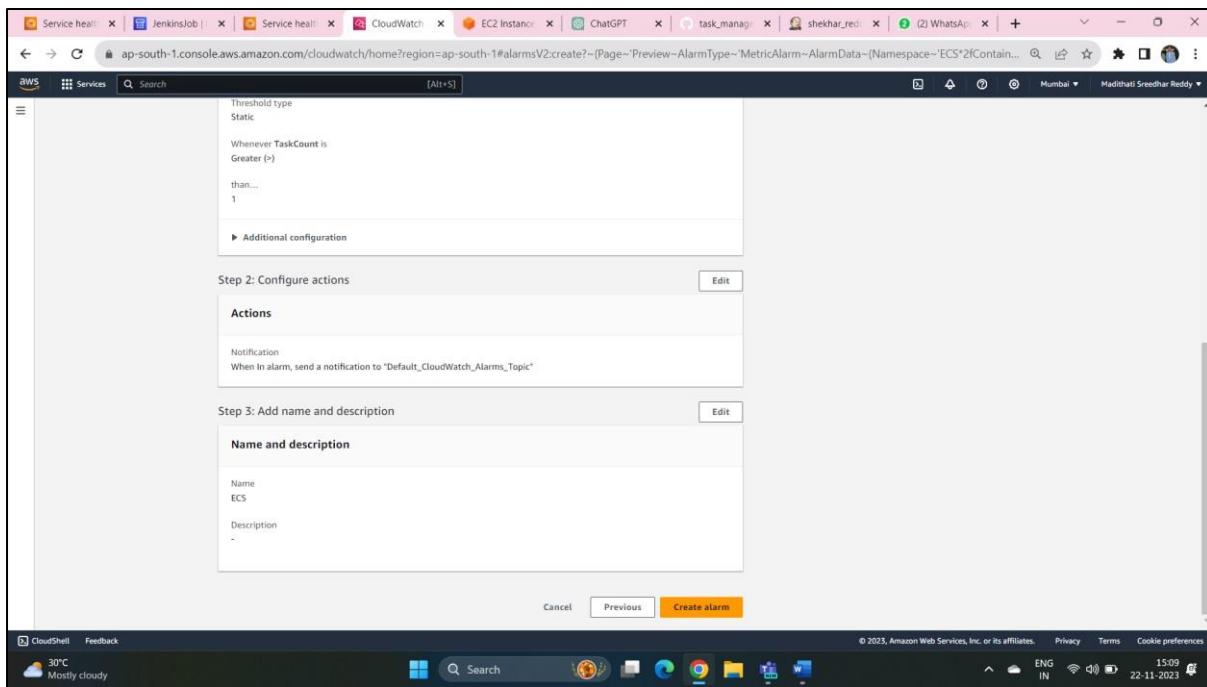
ID	Endpoint	Status	Protocol
bffd5497-46be-4f8d-aace-2d564a313a0c	madithatisreedhar123@gmail.com	Confirmed	EMAIL

The screenshot shows the AWS CloudWatch Metrics 'Create alarm' wizard. Step 3, 'Add name and description', is selected. The 'Name and description' section contains:

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

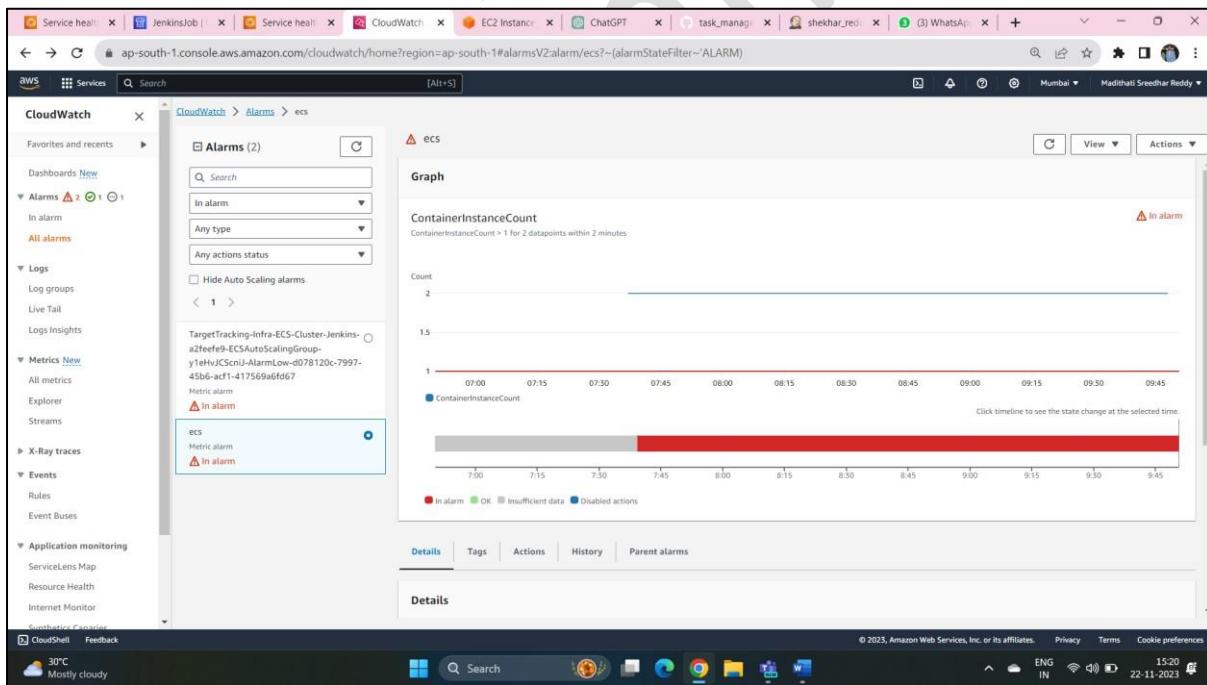
At the bottom, there is a note: "Markdown formatting is only applied when viewing your alarm in the console. The description will remain in plain text in the alarm notifications." Navigation buttons include 'Cancel', 'Previous', and 'Next'.

- Click on Next for Preview

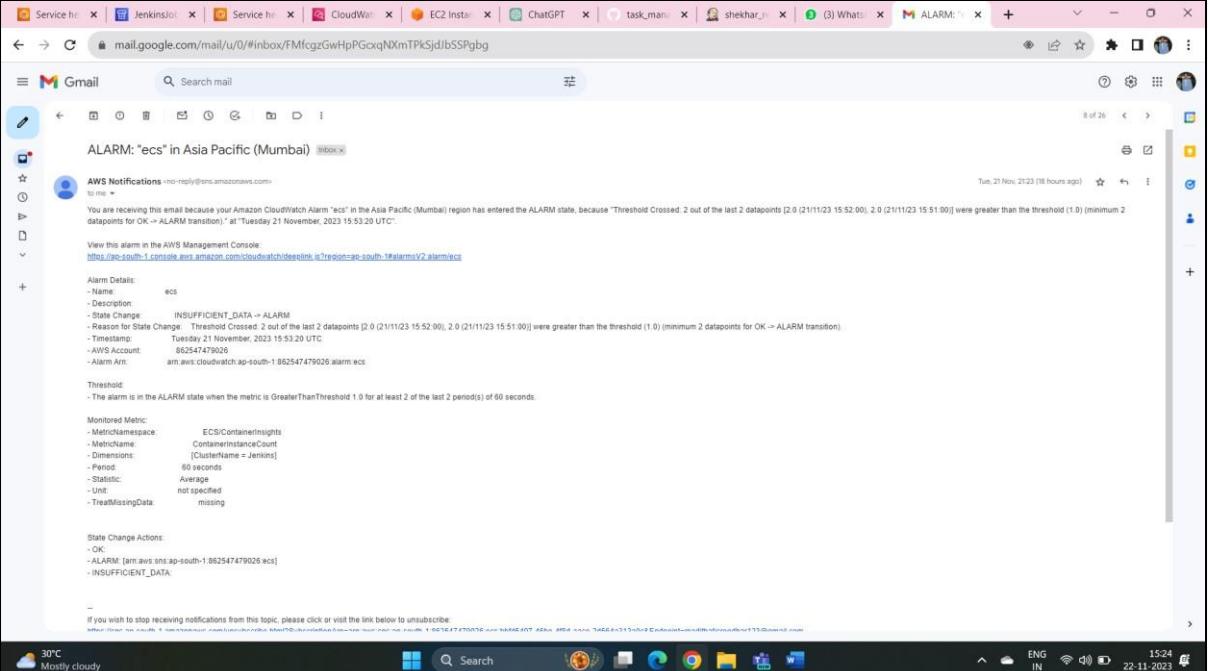


□ Click on Create Alarm

- 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.



ALARM: "ecs" in Asia Pacific (Mumbai)

AWS Notifications no-reply@amazonaws.com

to me

You are receiving this email because your Amazon CloudWatch Alarm "ecs" in the Asia Pacific (Mumbai) region has entered the ALARM state, because "Threshold Crossed: 2 out of the last 2 datapoints [2.0 (21/11/23 15:52:00), 2.0 (21/11/23 15:51:00)] were greater than the threshold (1.0) (minimum 2 datapoints for OK->ALARM transition)" at "Tuesday 21 November, 2023 15:53:20 UTC".

View this alarm in the AWS Management Console
https://ap-south-1.console.aws.amazon.com/cloudwatch/alarms/region-ap-south-1#alarm/V2_alarm/ecs

Alarm Details:

- Name: ecs
- Description: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 2 out of the last 2 datapoints [2.0 (21/11/23 15:52:00), 2.0 (21/11/23 15:51:00)] were greater than the threshold (1.0) (minimum 2 datapoints for OK->ALARM transition).
- Timestamp: Tuesday 21 November, 2023 15:53:20 UTC
- AWS Account: 862547479026
- Alarm Arn: arn:aws:cloudwatch:ap-south-1:862547479026:alarm:ecs

Threshold:

The alarm is in the ALARM state when the metric is GreaterThanThreshold 1.0 for at least 2 of the last 2 period(s) of 60 seconds.

Monitored Metric:

- MetricNamespace: ECS/ContainerInsights
- MetricName: ContainerInstanceCount
- Dimensions: [ClusterName = Jenkins]
- Period: 60 seconds
- Statistic: Average
- Unit: not specified
- TreatMissingData: missing

State Change Actions:

- OK
- ALARM: [arn:aws:sns:ap-south-1:862547479026:ecs]
- INSUFFICIENT_DATA

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe.

30°C Mostly cloudy

ENG IN 15:24 22-11-2023