Expert Cloud Consulting

Enhance Optimise & Scale

ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting” -

CI-CD Fundamentals [Title,18, Arial]

15.Jan.2025 [Subtitle,14, Arial]

version 1.0

–

Contributed by Shraddha Chaudhari [Normal text,14, Arial]

Approved by (In Review)

Expert Cloud Consulting

Office #811, Gera Imperium Rise,

Hinjewadi Phase-II Rd, Pune, India – 411057

“Expert Cloud Consulting”

CI-CD Fundamentals [Title,18, Arial]

1.0 Contents [Heading3,14, Arial]

2.0 General Information: [Heading3,14, Arial]

2.1 Document Jira/ Github Ticket(s) [Heading4,12, Arial]

Ticket(s) Name	Url
CI-CD Fundamentals [Normal text,10, Arial]	https://github.com/shaanicha/CI-CD-pipeline/tree/main

2.2 Document Purpose

The purpose of this documentation is to provide a clear guide for setting up a **CI/CD pipeline using Jenkins**. It covers the steps to automate the process of:

1. **Continuous Integration (CI)**: Automatically pulling code from GitHub, running unit tests, and building Docker images.
2. **Continuous Deployment (CD)**: Automatically pushing the Docker image to Docker Hub and deploying it to a staging environment.
3. **Email Notifications**: Sending email alerts for both successful and failed pipeline runs.
4. **Troubleshooting**: Addressing common issues such as permission errors or failed builds.

Overall, this documentation helps automate the entire development workflow, ensuring faster, reliable, and consistent deployments.

4o mini

[Normal text,10, Arial, Justify Alignment]

2.3 Document Revisions

Date	Version	Contributor(s)	Approver(s)	Section(s)	Change(s)
15/Jan/2025	1.0	Shraddha Chaudhari	Akshay Shinde	All Sections	New Document Created

2.4 Document References

The following artifacts are referenced within this document. Please refer to the original documents for additional information.

Date	Document	Filename / Url
2017	Jenkins Documentation - Pipelines	https://www.jenkins.io/doc/book/pipeline/
2025	Jenkins User Handbook	https://www.jenkins.io/user-handbook.pdf
2025	Docker Documentation	https://docs.docker.com/
2025	GitHub Documentation	https://docs.github.com/en

3.0 Document Overview:

The goal is to streamline development workflows, enabling faster, reliable, and automated testing, building, and deployment processes. This guide includes setup instructions, best practices, and references to relevant tools and documentation.

1. Build a CI/CD pipeline using Jenkins to:
   - Pull code from GitHub.
   - Run unit tests automatically.
   - Build and deploy a Docker container to a staging environment.

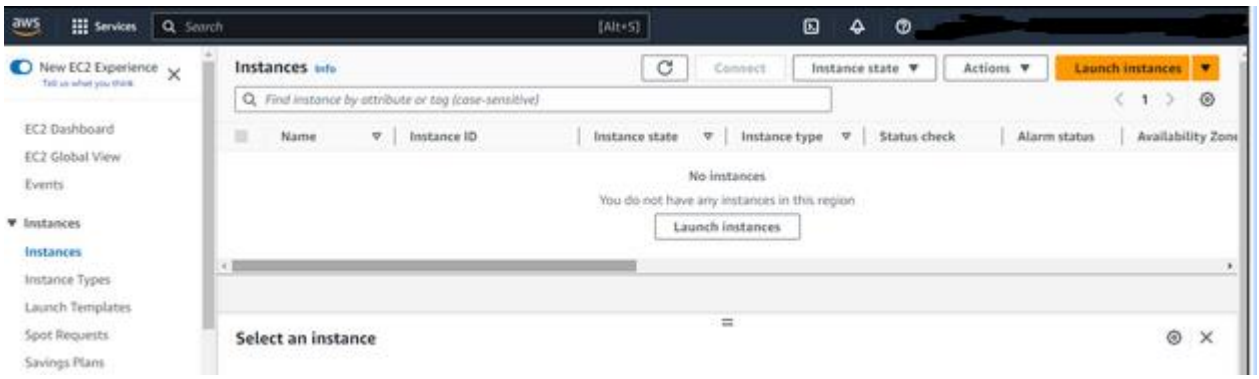
4.0 Steps / Procedure

4.1 : Setup the ubuntu server for Jenkins application

The following procedure has been done during the installation of ubuntu server:

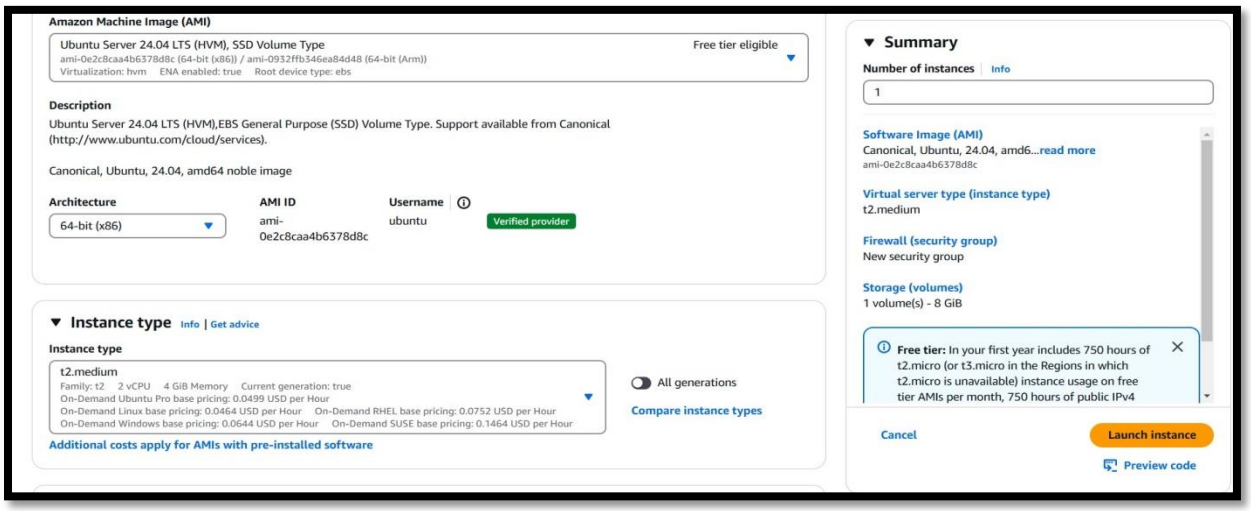
Log in to the AWS Management Console (Company-SandBox) and navigate to the EC2 dashboard.

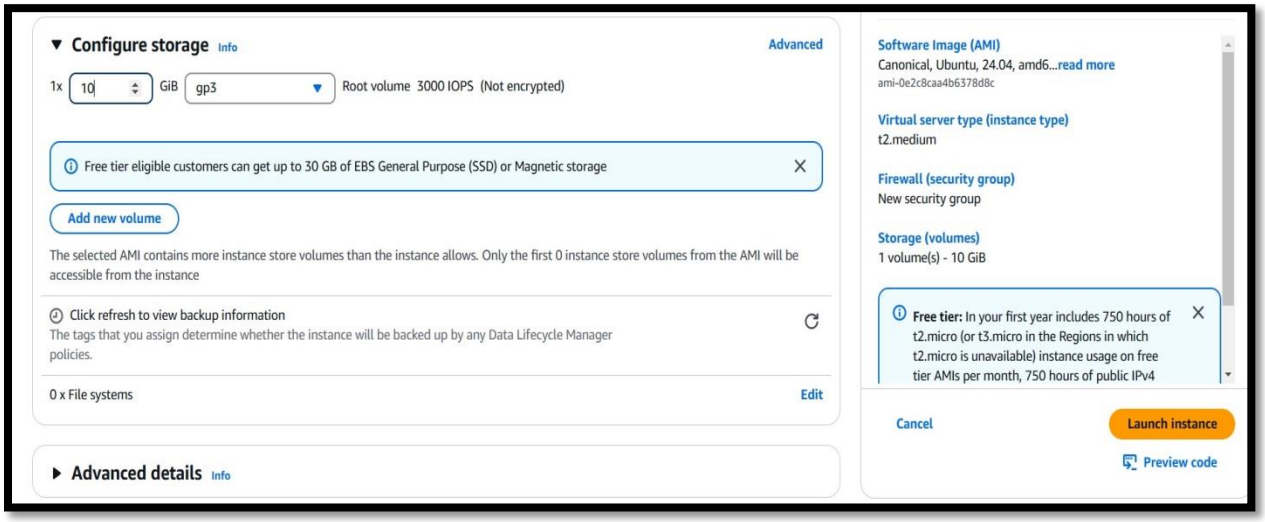
Click on the "Launch Instance" button to start the process of launching a new EC2 instance.



4.2: Choose an Amazon Machine Image (AMI)

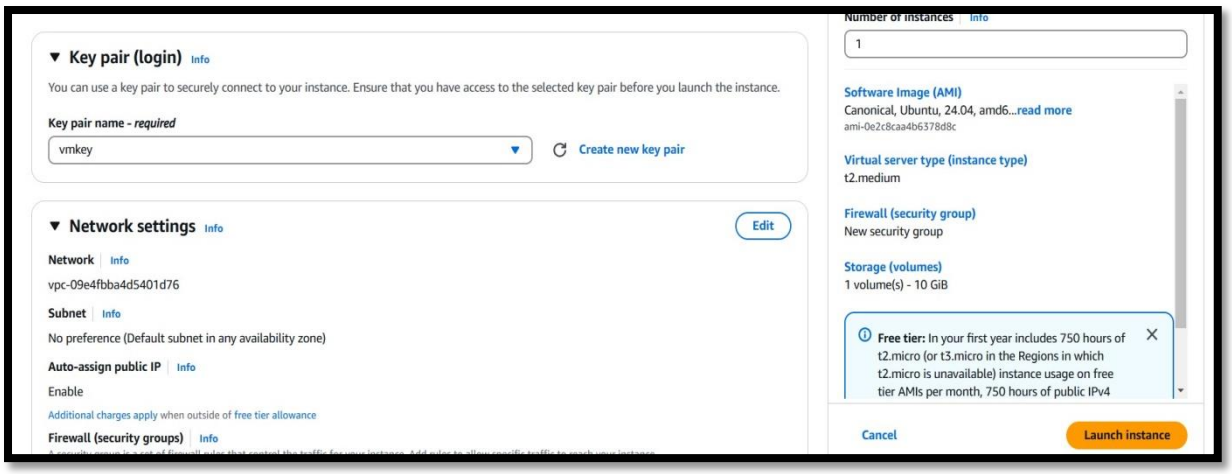
Select an instance type, configure your instance details (such as the number of instances, and storage)





4.3: Key-Pair Configuration

Select an instance type and create a new key-pair as name is vmkey.



4.4: Network settings

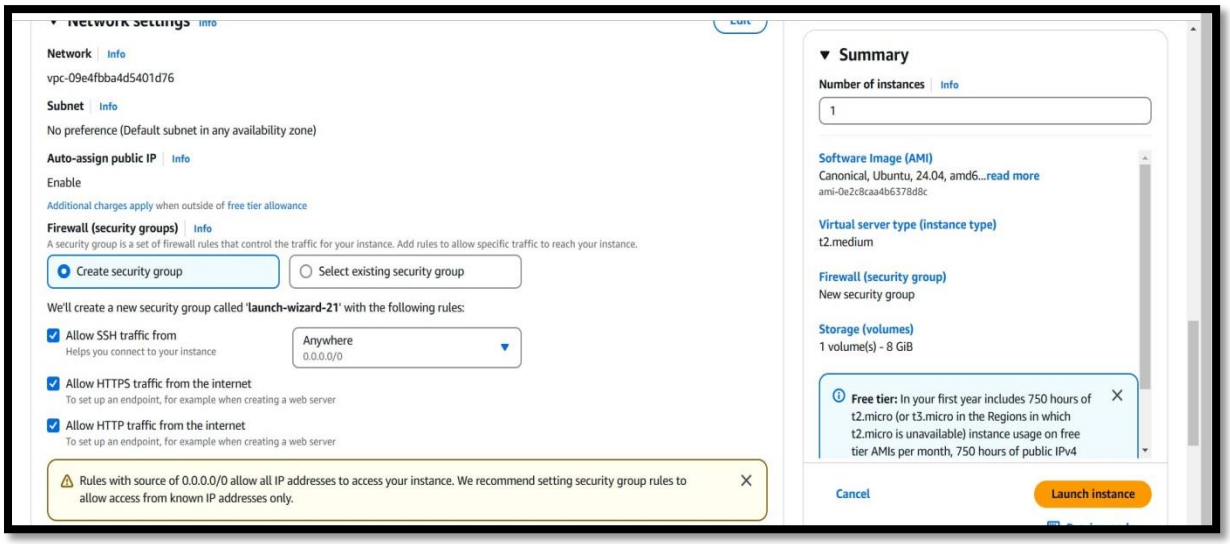
4.4.1: VPC Configuration:

Select the vpc and subnet for the ec2 instance.

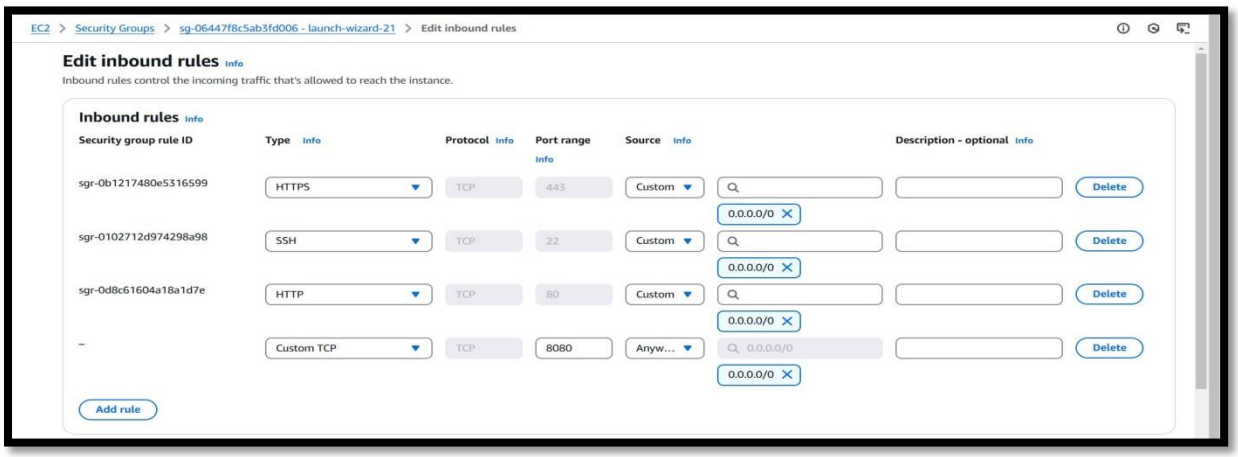
4.4.2: Security Group Configuration

Create a new security group with required security rules

Specified Security group rule for this ec2 instances are shown below:

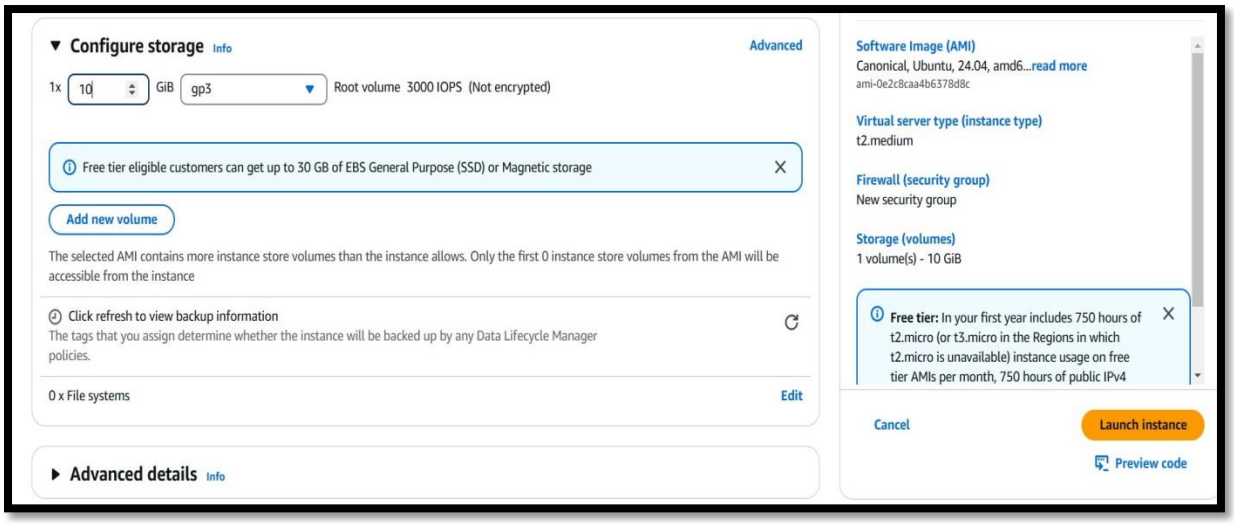


Allow port 8080 for Jenkins server

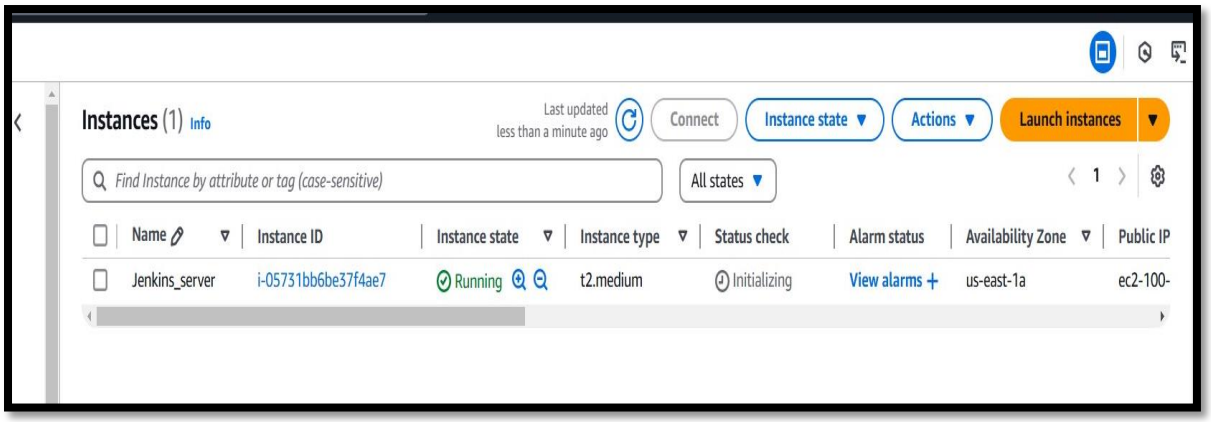


4.5: Launch Instance

Click on Launch instance

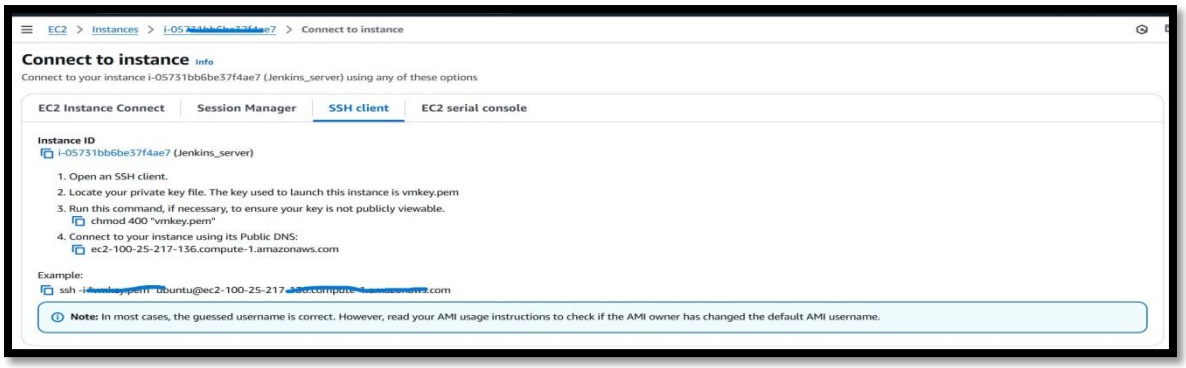


Instances are created and they are ready to use.

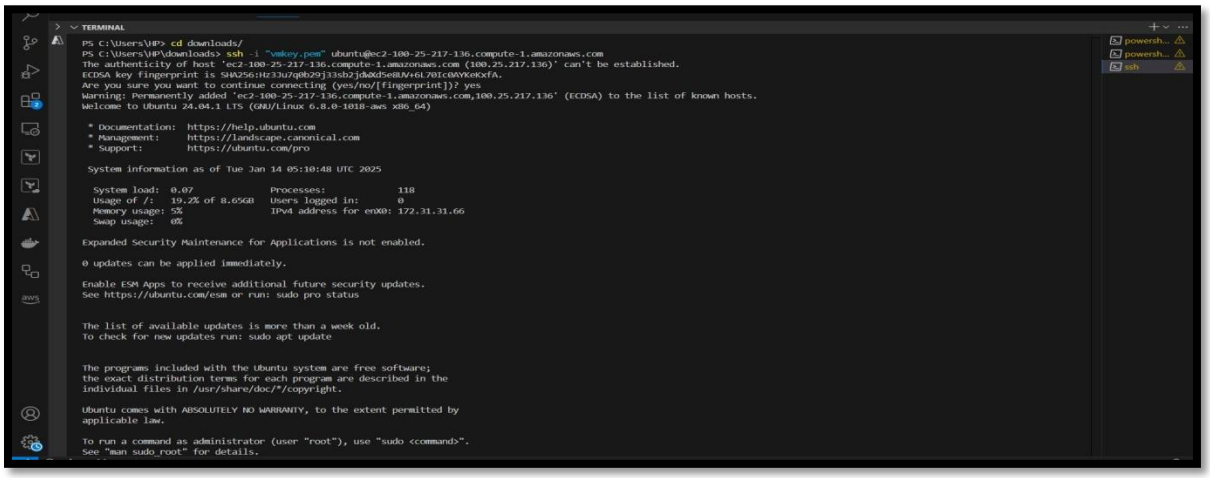


4.5: SSH Configuration

Log in ec2 instance using SSH client.



Successfully able to connect the ec2-Instance by using ssh client.



4.6: Setting up the CI/CD pipeline with Jenkins

Here’s a step-by-step guide to setting up the CI/CD pipeline with Jenkins:

So, first need to update package and install other dependencies which need for Jenkins.

```
sudo apt update
sudo apt install openjdk-11-jdk -y
wget -q -O https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt update
sudo apt install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

```
root@ip-172-31-20-30:/home/ubuntu# history
    1  apt update
    2  apt upgrade -y
    3  sudo apt install openjdk-17-jdk -y
    4  java -version
    5  wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
    6  sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
    7  apt update
    8  sudo apt-key del 5BA31D57EF5975CA
    9  curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
   10  echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list
   11  apt update
   12  sudo apt install jenkins -y
   13  clear
   14  systemctl status jenkins
```

This above mentioned all cmds need to setting up Jenkins.

```
TERMINAL
root@ip-172-31-20-30:/home/ubuntu# systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-01-14 06:21:16 UTC; 24s ago
   Main PID: 15684 (java)
      Tasks: 52 (limit: 4676)
     Memory: 686.6M (peak: 695.1M)
        CPU: 15.803s
     CGroup: /system.slice/jenkins.service
             └─15684 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jan 14 06:21:12 ip-172-31-20-30 jenkins[15684]: 8915e8be11c749e2974296a34560f9ae
Jan 14 06:21:12 ip-172-31-20-30 jenkins[15684]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Jan 14 06:21:12 ip-172-31-20-30 jenkins[15684]: *************************************************************
Jan 14 06:21:12 ip-172-31-20-30 jenkins[15684]: *************************************************************
Jan 14 06:21:12 ip-172-31-20-30 jenkins[15684]: *************************************************************
Jan 14 06:21:16 ip-172-31-20-30 jenkins[15684]: 2025-01-14 06:21:16.318+0000 [id=32]	INFO	jenkins.InitReactorRunner$1#onAttained: Completed initialization
Jan 14 06:21:16 ip-172-31-20-30 jenkins[15684]: 2025-01-14 06:21:16.332+0000 [id=23]	INFO	hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
Jan 14 06:21:16 ip-172-31-20-30 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Jan 14 06:21:16 ip-172-31-20-30 jenkins[15684]: 2025-01-14 06:21:16.468+0000 [id=48]	INFO	h.m.DownloadService$Downloadable#load: Obtained the updated data file for hud
Jan 14 06:21:16 ip-172-31-20-30 jenkins[15684]: 2025-01-14 06:21:16.472+0000 [id=48]	INFO	hudson.util.Retrier#start: Performed the action check updates server successf
```

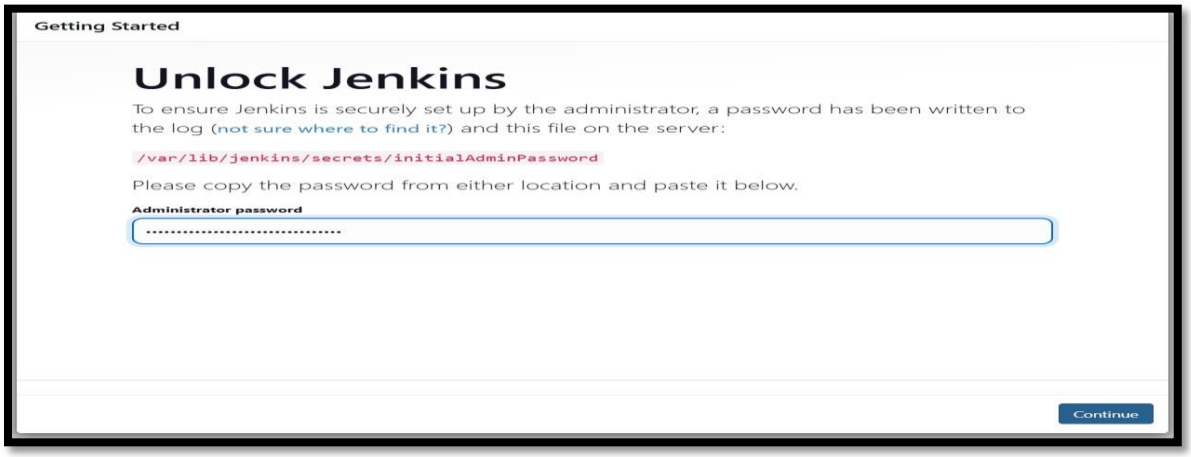
4.7: Access Jenkins

Once Jenkins installation done next take access of jenkins.

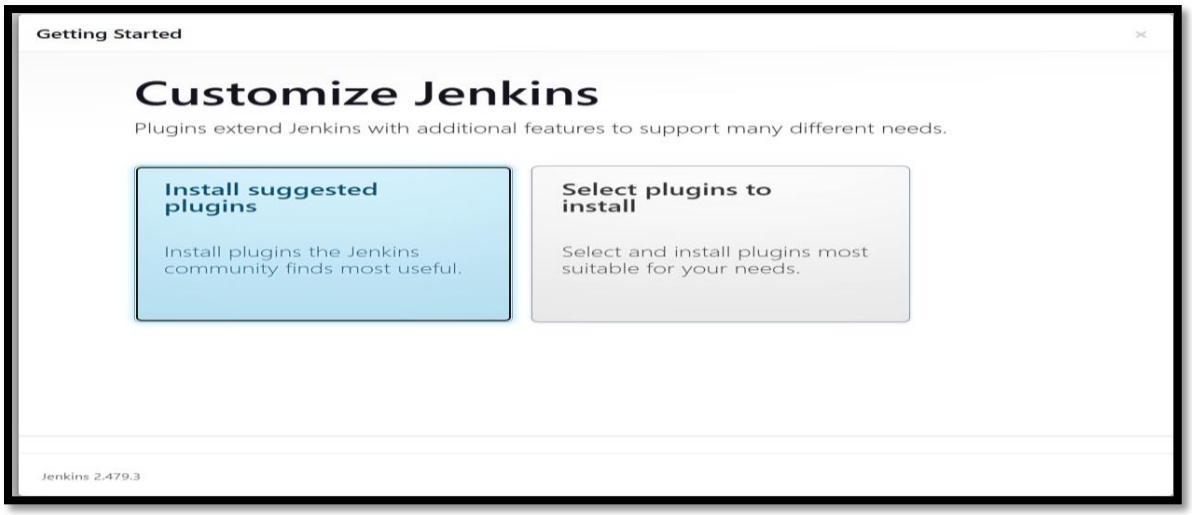
- Open http://<your_server_ip>:8080 in your browser.

Retrieve the initial admin password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword



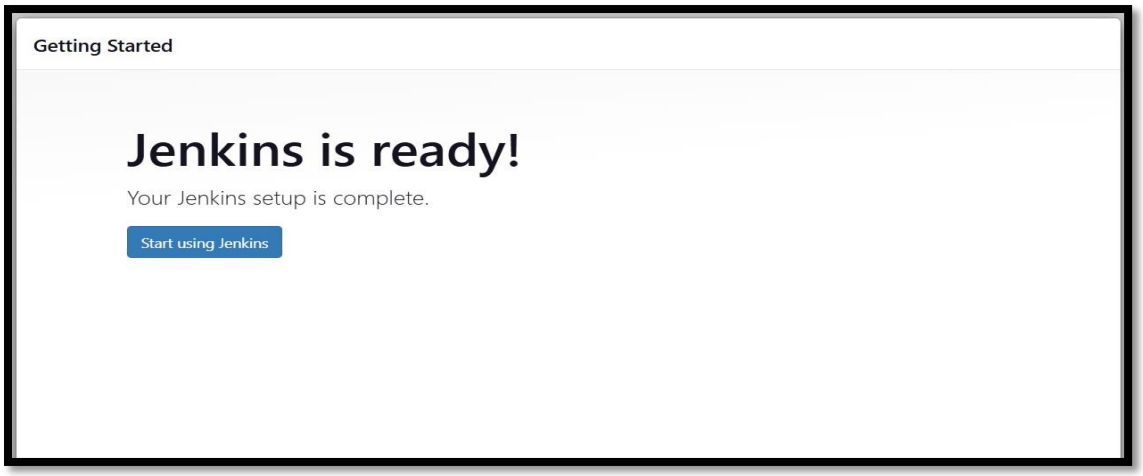
After this it showing this page click on install suggested plugins.

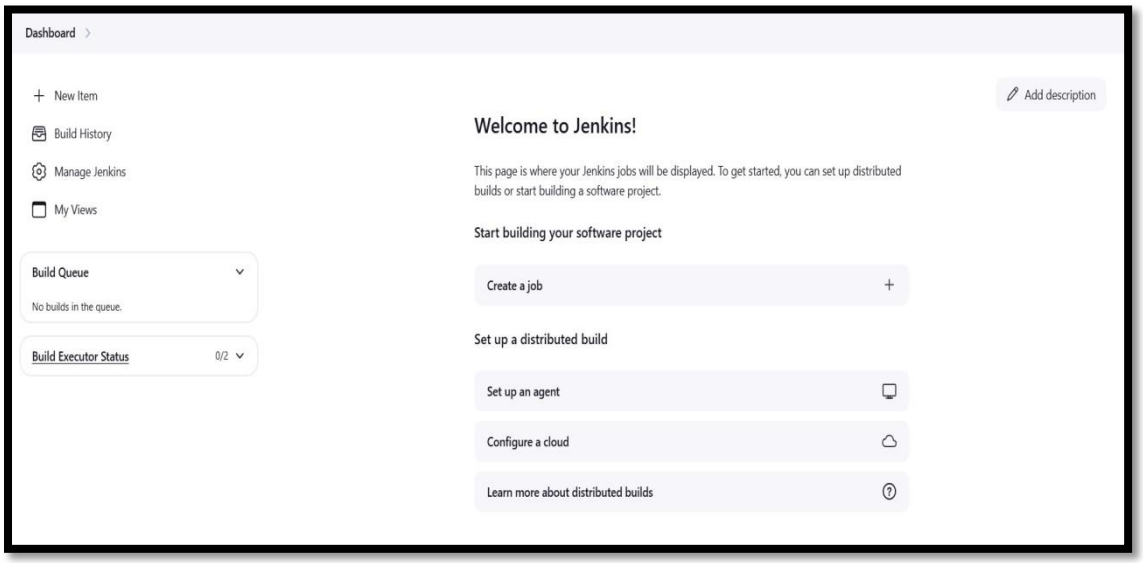


Next, we have to create admin user for login to Jenkins.

Once filled all needed information then click to save and continue.

So then it will show that setup is ready.

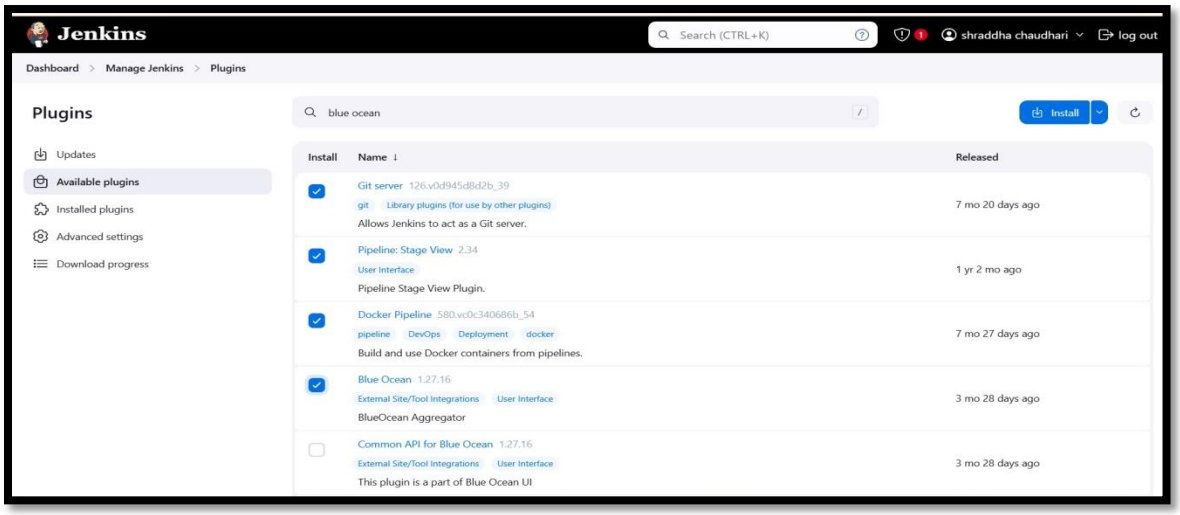


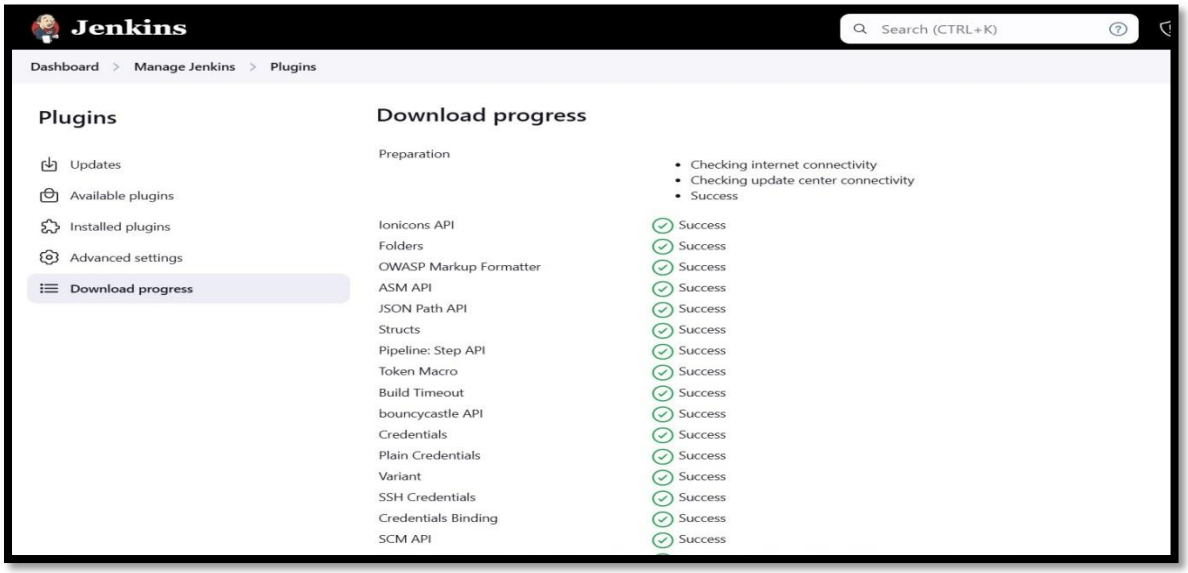


Once done with all setup next have to install required plugins.

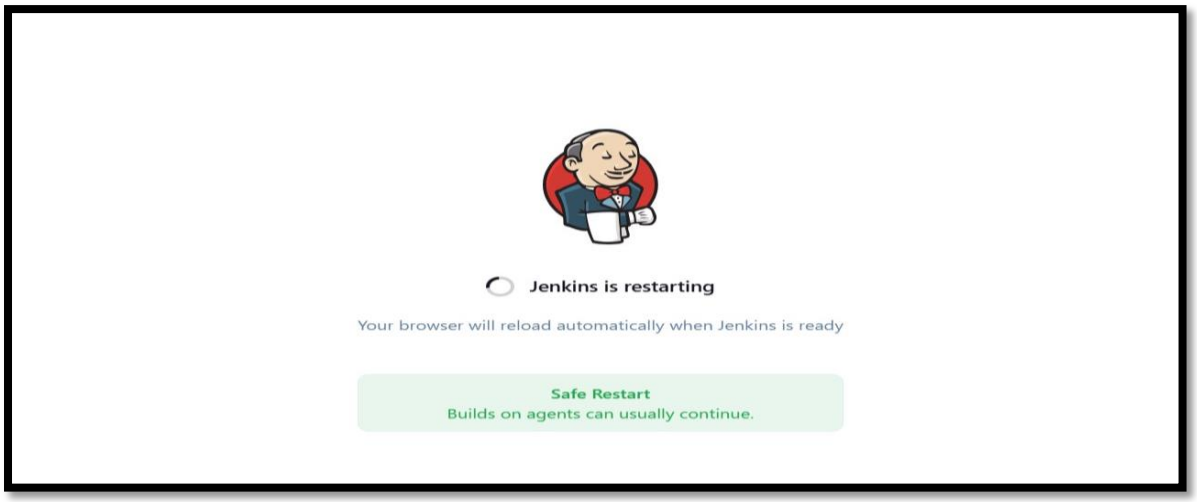
4.8: Install required Jenkins plugins:

- Go to **Manage Jenkins** → **Manage Plugins** → **Available**.
- Install:
  - GitHub Integration Plugin
  - Pipeline
  - Docker Pipeline
  - Docker Commons





Once install all plugins then next have to restart jenkins.



It's done with setup and plugins.

4.9: Configured Jenkins Credentials :

For that we need to follow below steps:

- Go to **Manage Jenkins** → **Manage Credentials**.
- Add GitHub credentials (username and personal access token).
- Add DockerHub credentials if pushing images.

5.0: Install Docker on Jenkins server:

Follow below cmds to install docker on jenkins server

```
sudo apt install docker.io -y
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

5.1: Project Structure:

We need to follow this below project structure

```
project-root/
├── app/
│ ├── __init__.py
│ ├── main.py
│ ├── ... (other app files)
├── tests/
```

```
│ ├── test_main.py
│ ├── ... (other test files)
├── requirements.txt
├── Dockerfile
├── Jenkinsfile
├── README.md
```

5.2: Create app directory

mkdir app/

Then navigate to app directory

cd app/

Once navigate to app/ directory so in this directory we have to create main.py file

nano main.py

In this file setup flask application

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

5.3: Create tests directory

mkdir tests/

Navigate to tests/ directory

cd tests/

We have to create test_main.py created this file for run unit test

nano test_main.py

```
def test_sample():
    assert 1 + 1 == 2
```

5.4: Create requirements.txt

Used in python project to list all the dependencies required for the project to run.

nano requirements.txt

```
flask
pytest
```

5.5: Create Dockerfile

Used to create docker image all necessary configurations includes in dockerfile required to build image

nano Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app/main.py"]
```

5.6: Create Jenkinsfile

It is script that defines a Jenkins pipeline which automates the build, test, and deployment process of software.

nano Jenkinsfile

```
enkinsfile
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'shaani/ci-cd-pipeline-image'
        STAGING_ENV = 'staging-container'
    }
    stages {
        stage('Checkout Code') {
            steps {
                echo 'Checking out code from GitHub...'
                git credentialsId: 'github-credentials', branch: 'main', url: 'https://github.com/shaanicha/CI-CD-pipeline.git'
            }
        }
        stage('Install Dependencies') {
            steps {
                echo 'Installing Python dependencies...'
                script {
                    // Create a virtual environment
                    sh 'python3 -m venv venv'
                    // Activate the virtual environment and install dependencies
                    sh '. venv/bin/activate && pip install -r requirements.txt'
                }
            }
        }
        stage('Run Unit Tests') {
            steps {
                echo 'Running unit tests...'
                script {
                    // Activate virtual environment and run tests
                    sh '. venv/bin/activate && pytest tests/'
                }
            }
        }
        stage('Build Docker Image') {
            steps {
                echo 'Building Docker image...'
                sh 'docker build -t $DOCKER_IMAGE .'
```

```
pipeline {
    stages {
        stage('Build Docker Image') {
            }
        }
        stage('Push Docker Image') {
            steps {
                echo 'Pushing Docker image to Docker Hub...'
                withDockerRegistry([credentialsId: 'docker-credentials', url: '']) {
                    sh 'docker push $DOCKER_IMAGE'
                }
            }
        }
        stage('Deploy to Staging') {
            steps {
                echo 'Deploying Docker container to staging environment...'
                sh """
                docker stop $STAGING_ENV || true
                docker rm $STAGING_ENV || true
                docker run -d --name $STAGING_ENV -p 5000:5000 $DOCKER_IMAGE
                """
            }
        }
    }
    post {
        always {
            echo 'Pipeline execution completed.'
        }
        success {
            echo 'Pipeline executed successfully.'
        }
        failure {
            echo 'Pipeline failed. Check the logs for details.'
        }
    }
}
```

This provided files and project structure mentioned in above GitHub link.

6.0: Initialize a git repository:

We have to push all created project structure to github

So first need to install git on Ubuntu server. Once done then do git initialize

git init

git add .

Add modify changes

git commit –m “new ci-cd pipeline”

```
root@ip-172-31-20-30:/home/ubuntu# git init
Reinitialized existing Git repository in /home/ubuntu/.git/
root@ip-172-31-20-30:/home/ubuntu# git add .
root@ip-172-31-20-30:/home/ubuntu# git commit -m "ci/cd pipeline"
[master (root-commit) c2c474e] ci/cd pipeline
 Committer: root <root@ip-172-31-20-30.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 12 files changed, 243 insertions(+)
 create mode 100644 .bash_history
 create mode 100644 .bash_logout
 create mode 100644 .bashrc
 create mode 100644 .cache/motd.legal-displayed
 create mode 100644 .profile
 create mode 100644 .ssh/authorized_keys
 create mode 100644 .sudo_as_admin_successful
 create mode 100644 Dockerfile
 create mode 100644 Jenkinsfile
 create mode 100644 app/main.py
 create mode 100644 requirements.txt
 create mode 100644 tests/test_main.py
root@ip-172-31-20-30:/home/ubuntu#
```

6.1: Push to GitHub:

- Create a new repository on GitHub (e.g., my-ci-cd-pipeline).

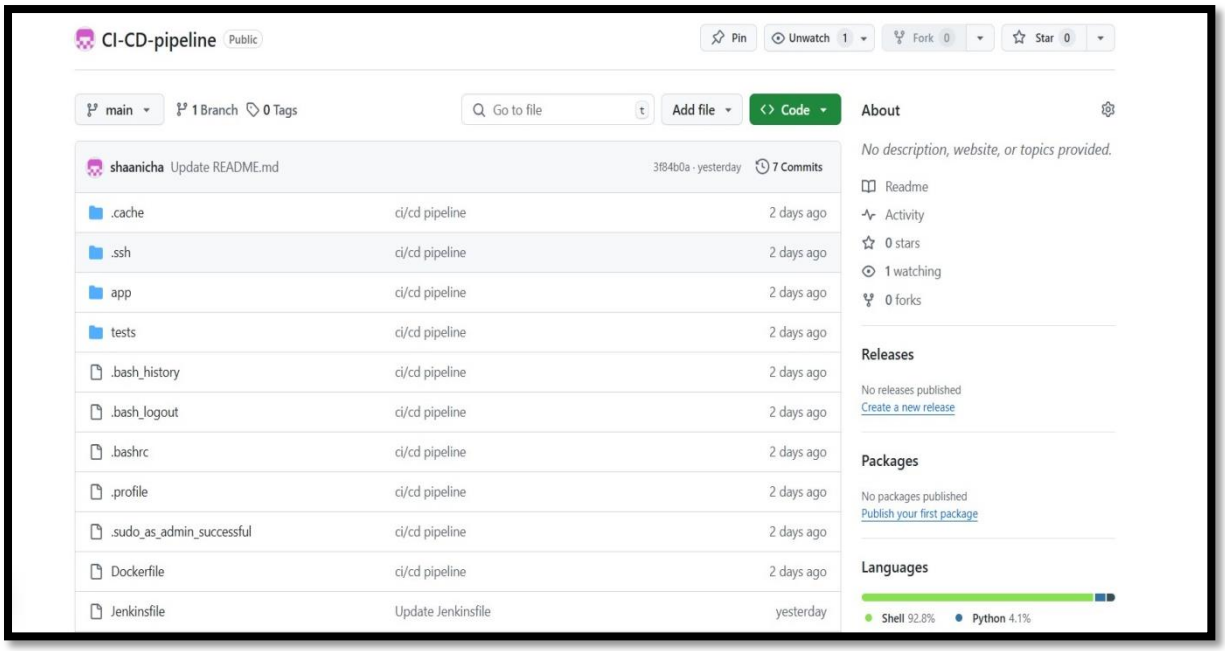
  Link the repository:

  git remote add origin https://github.com/your-username/my-ci-cd-pipeline.git

  git branch -M main

- git push -u origin main

```
root@ip-172-31-20-30:/home/ubuntu# git push origin main
Username for 'https://github.com': shaanicha
Password for 'https://shaanicha@github.com':
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 2 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (17/17), 4.40 KiB | 4.40 MiB/s, done.
Total 17 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/shaanicha/CI-CD-pipeline.git
   e3d35c0..c38fc15  main -> main
```

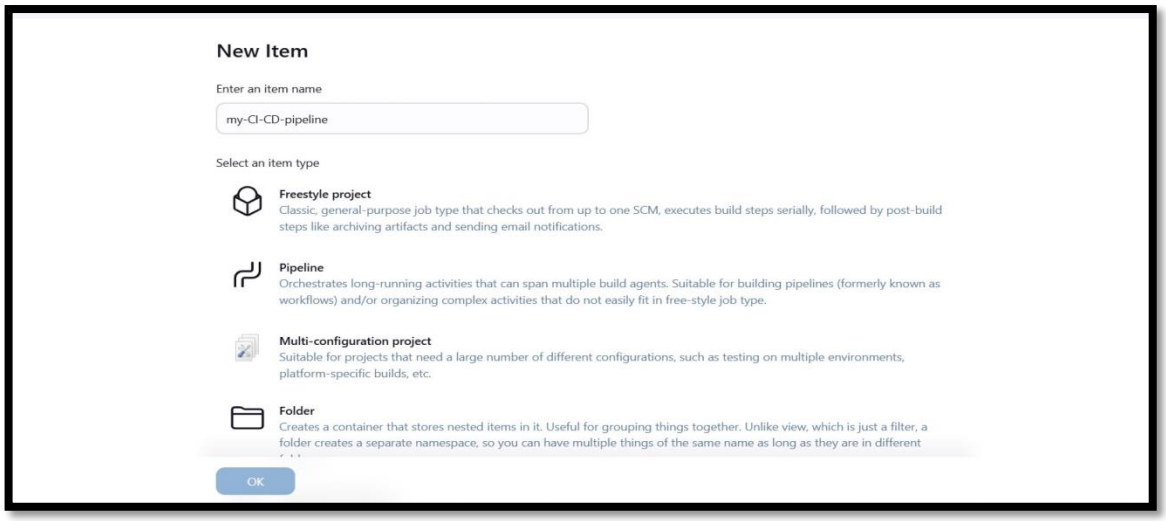


7.0: Setup Jenkins pipeline job:

Already we setup Jenkins now we have to create new pipeline so we have to create new job and pipeline

7.1: **Create a New Job:**

- Go to the Jenkins dashboard, click New Item.
- Enter a name (e.g., My-CI-CD-Pipeline), select Pipeline, and click OK.



7.2: Configure the Pipeline:

Under Pipeline, select **Pipeline script from SCM**.

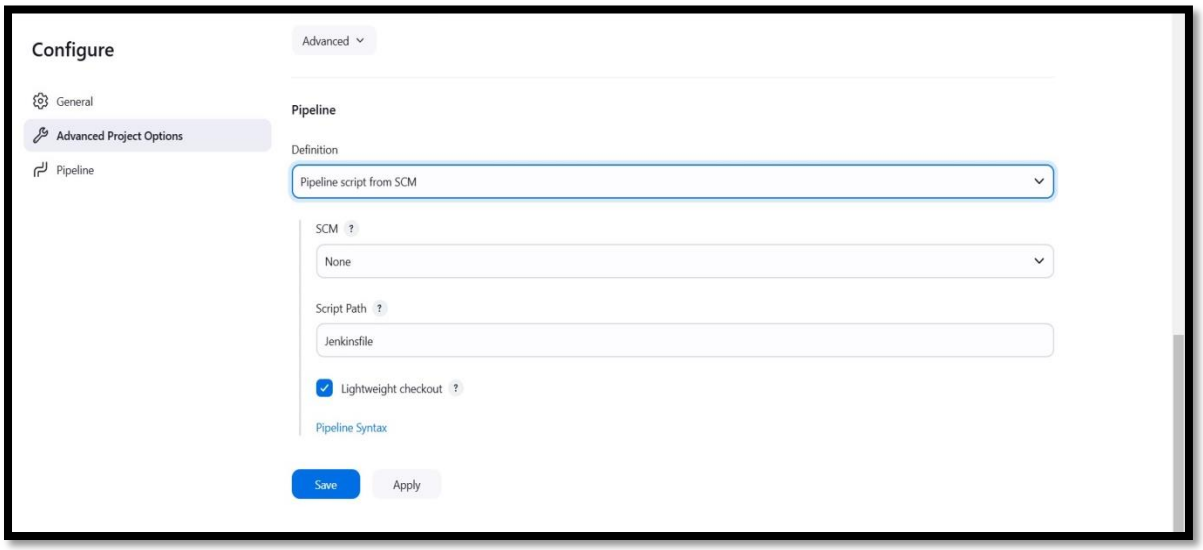
Set:

SCM: Git.

Repository URL: https://github.com/your-username/my-ci-cd-pipeline.git.

Branch: main.

7.3: Save the Job.



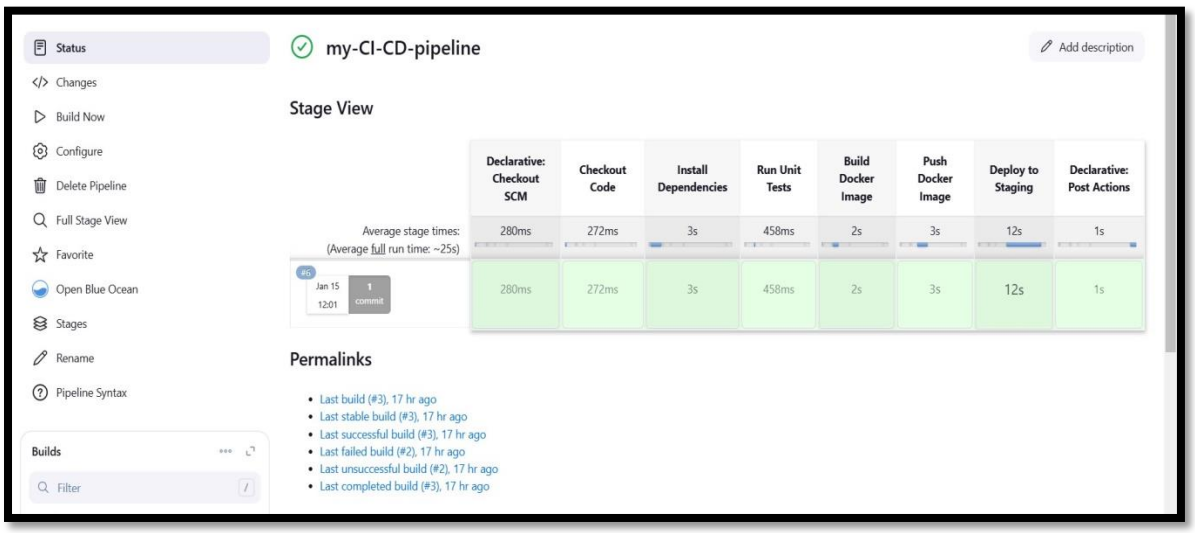
8.0: Test the pipeline:

8.1: Trigger a Build:

- Click Build Now.

8.2: Observe the Stages:

- Jenkins will:
  - Pull code from GitHub.
  - Install dependencies (pip install).
  - Run unit tests (pytest).
  - Build a Docker image.
  - Push the image to Docker Hub.
  - Deploy the container to the staging environment.



Pipeline run successfully.

9.0: Verify Deployment:

9.1: Check the Docker Container:

SSH into your EC2 instance and run:

docker ps

```
TERMINAL

root@ip-172-31-20-30:/home/ubuntu# docker ps
CONTAINER ID   IMAGE                    COMMAND              CREATED        STATUS        PORTS                                      NAMES
606c40fc089f   shaani/ci-cd-pipeline-image   "python app/main.py"   2 minutes ago   Up 2 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   staging-container
root@ip-172-31-20-30:/home/ubuntu#
```

You should see your container running.

9.2: **Access the Application:**

Visit http://<EC2-PUBLIC-IP>:5000.

You should see the message: **"Hello, World!"**.

Application access successfully.

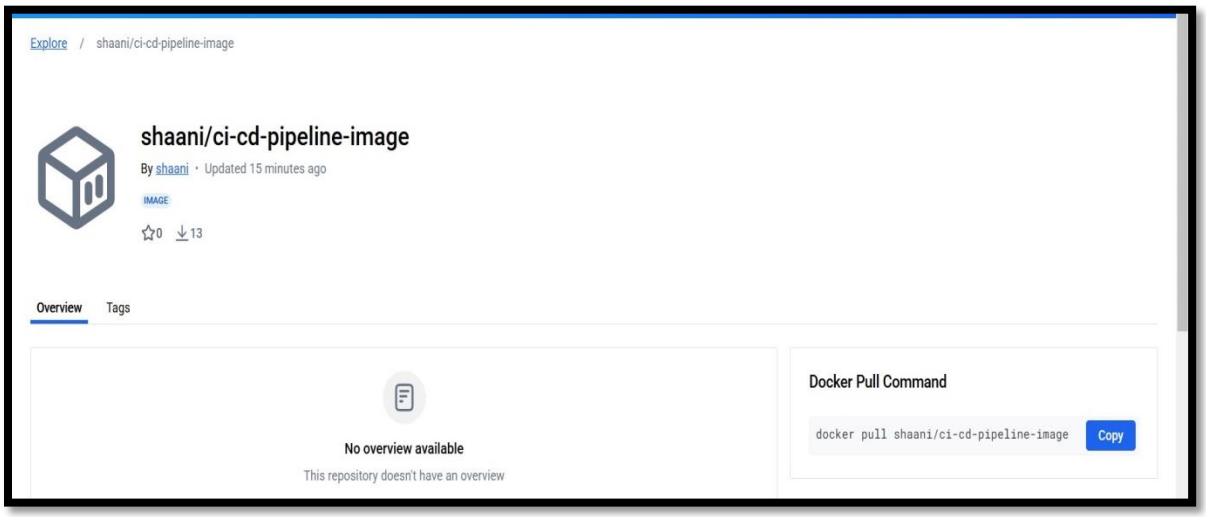
9.3: Verify Docker image:

Once our application access then check docker image correctly push to dockerhub.

Go to dockerhub

Login to dockerhub

Then It will showing our updates image



10: Integrate Notifications:

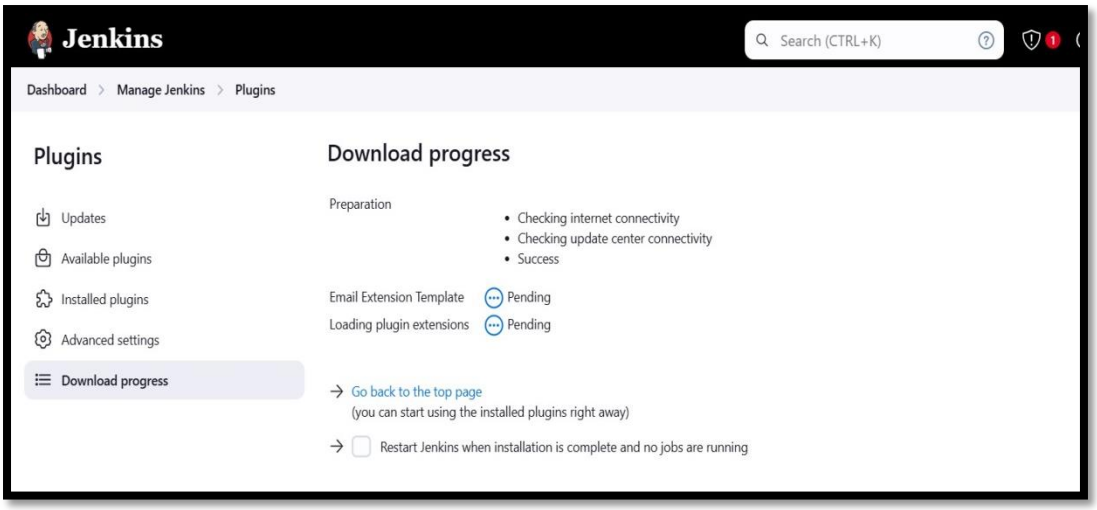
- Configure Jenkins to send Slack/email alerts on pipeline failures.

To configure Jenkins to send email notifications for pipeline failures,

Follow these steps:

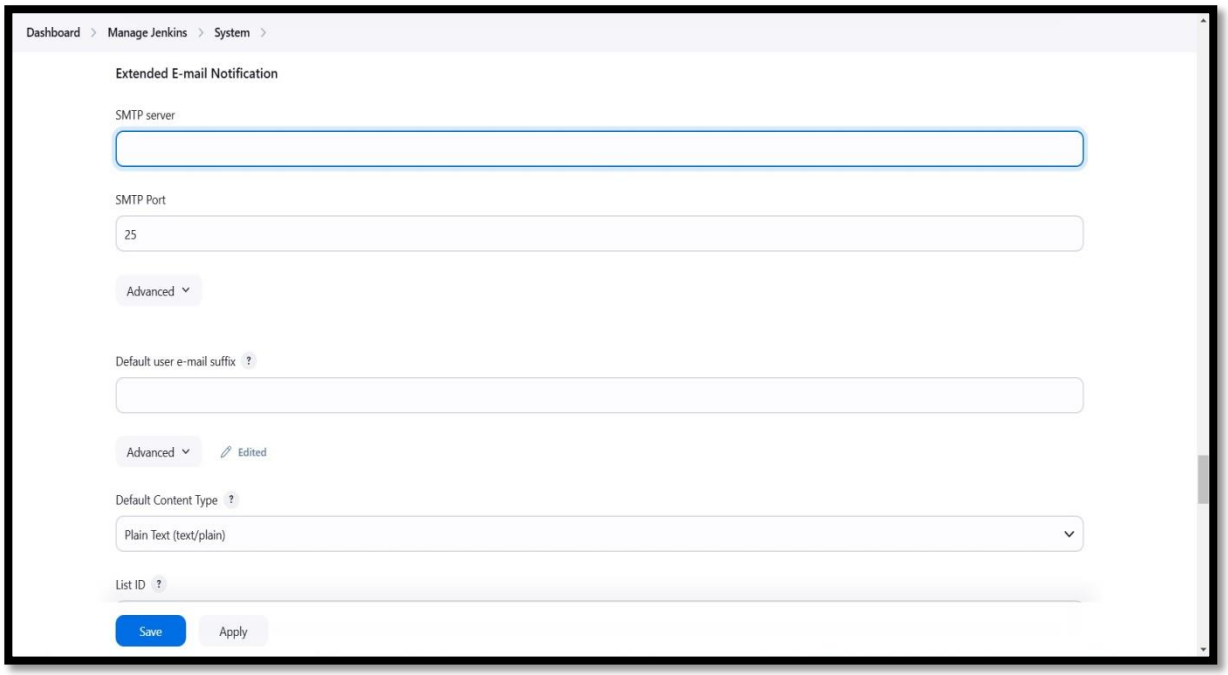
Step 1: Install Email Extension Plugin

1. Go to the **Jenkins Dashboard**.
2. Navigate to **Manage Jenkins > Manage Plugins**.
3. Under the **Available** tab, search for Email Extension Plugin (or verify it's installed under **Installed**).
4. Install the plugin and restart Jenkins if necessary.



Step 2: Configure Jenkins Email Notification Settings

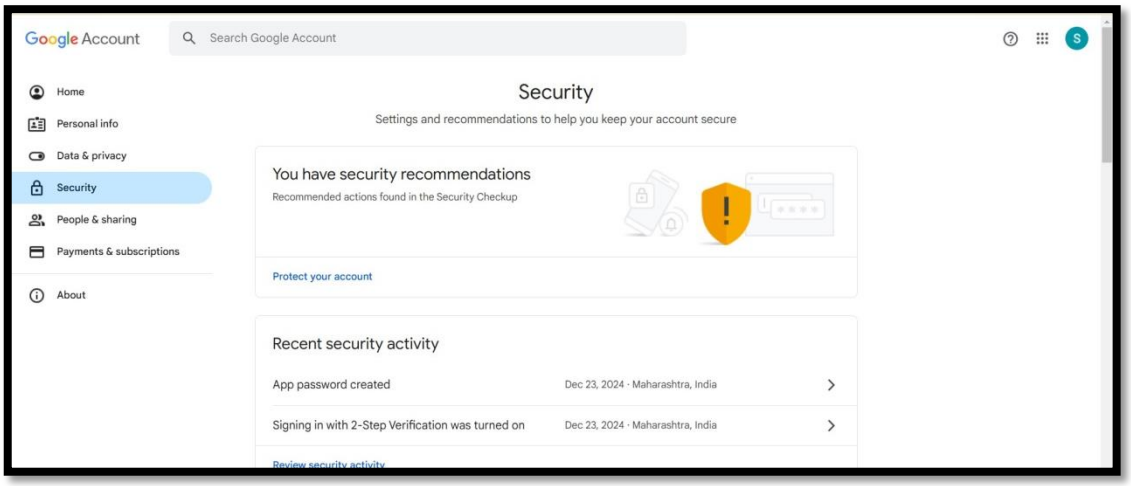
1. Navigate to **Manage Jenkins > Configure System**.
2. Scroll down to the **Extended E-mail Notification** section.



3. Fill in the following details based on your email provider:
   - **SMTP Server**: e.g., smtp.gmail.com for Gmail.
   - **Use SMTP Authentication**: Check this box.

- **User Name**: Your email address, e.g., your-email@gmail.com.
- **Password**: Your email's application-specific password or token (if required by your email provider).

4. For that, create an app password for email: Go to Google account > security > search for app password > give name “Jenkins” or anything > copy password and paste.



5. **SMTP Port**: 465 (for Gmail or any email provider using TLS).

- **Use SSL**: Uncheck (TLS will be used instead).
- **Reply-To Address**: The email address for replies, e.g., your-email@gmail.com.
- **Charset**: Leave as UTF-8.

6. Click **Test Configuration by Sending Test E-mail** to verify the settings.
    - Enter a recipient email address and ensure the test email is received.

This is test email notification. Means email integration correctly.

7. **Step 3: Modify the Jenkinsfile for Notifications**

   To modify the Jenkinsfile so that it intentionally fails (for testing the notification system), we can introduce an artificial error in one of the pipeline stages. Here's an updated Jenkinsfile with the intentional failure included:

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'shaani/ci-cd-pipeline-image'
        STAGING_ENV = 'staging-container'
    }
    stages {
        stage('Checkout Code') {
            steps {
                echo 'Checking out code from GitHub...'
                git credentialsId: 'github-credentials', branch: 'main', url: 'https://github.com/shaanicha/CI-CD-pipeline.git'
            }
        }
        stage('Install Dependencies') {
            steps {
                echo 'Installing Python dependencies...'
                script {
                    sh 'python3 -m venv venv'
                    sh '. venv/bin/activate && pip install -r requirements.txt'
                }
            }
        }
        stage('Run Unit Tests') {
            steps {
                echo 'Running unit tests...'
                script {
                    sh '. venv/bin/activate && pytest tests/'
                }
            }
        }
        stage('Build Docker Image') {
            steps {
                echo 'Building Docker image...'
                sh 'docker build -t $DOCKER_IMAGE .'
            }
        }
        stage('Push Docker Image') {
```

```
pipeline {
    stages {
        stage('Push Docker Image') {
            steps {
                echo 'Pushing Docker image to Docker Hub...'
                withDockerRegistry([credentialsId: 'docker-credentials', url: '']) {
                    sh 'docker push $DOCKER_IMAGE'
                }
            }
        }
        stage('Deploy to Staging') {
            steps {
                echo 'Deploying Docker container to staging environment...'
                sh """
                docker stop $STAGING_ENV || true
                docker rm $STAGING_ENV || true
                docker run -d --name $STAGING_ENV -p 5000:5000 $DOCKER_IMAGE
                """
            }
        }
        stage('Intentional Failure') {
            steps {
                echo 'Introducing a failure for testing notifications...'
                script {
                    error("This is an intentional failure for testing!")
                }
            }
        }
    }
    post {
        always {
            echo 'Pipeline execution completed.'
        }
        failure {
            echo 'Pipeline failed. Sending email notification...'
            emailext(
                subject: "Jenkins Pipeline Failure: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
```

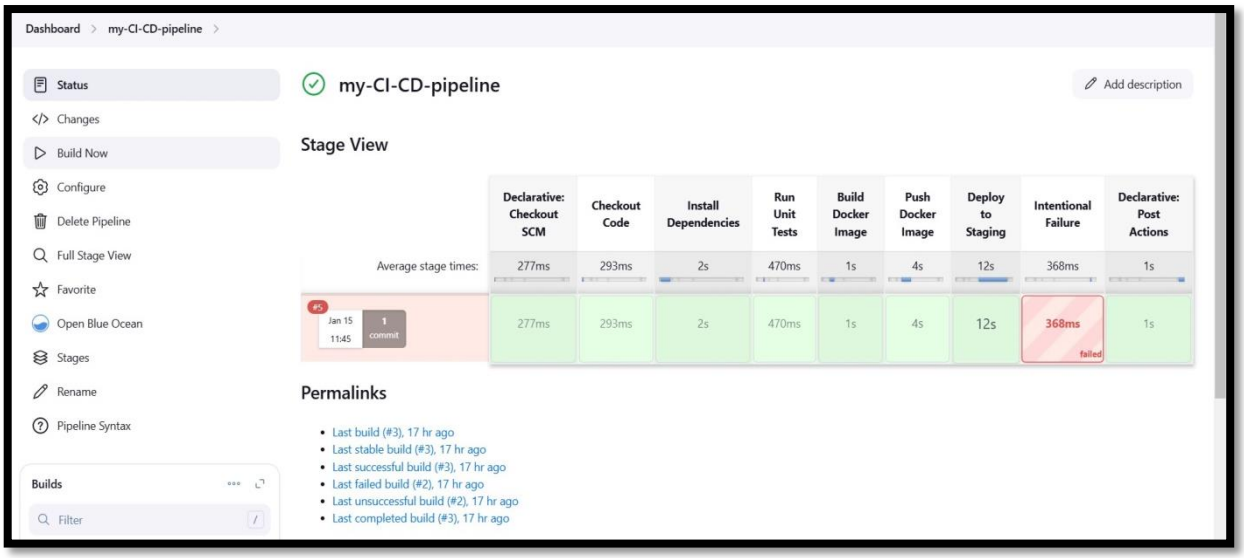
```
pipeline {
    stages {
        stage('Intentional Failure') {
            steps {
                echo 'Introducing a failure for testing notifications...'
                script {
                    error("This is an intentional failure for testing!")
                }
            }
        }
    }
    post {
        always {
            echo 'Pipeline execution completed.'
        }
        failure {
            echo 'Pipeline failed. Sending email notification...'
            emailext(
                subject: "Jenkins Pipeline Failure: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                body: """
                    <p>The pipeline has failed during execution.</p>
                    <p>Details:</p>
                    <ul>
                        <li><b>Job:</b> ${env.JOB_NAME}</li>
                        <li><b>Build Number:</b> ${env.BUILD_NUMBER}</li>
                        <li><b>URL:</b> <a href="${env.BUILD_URL}">${env.BUILD_URL}</a></li>
                    </ul>
                """,
                recipientProviders: [[$class: 'DevelopersRecipientProvider']],
                to: 'your-recipient@example.com',
                mimeType: 'text/html'
            )
        }
    }
}
```

11. Changes Made:

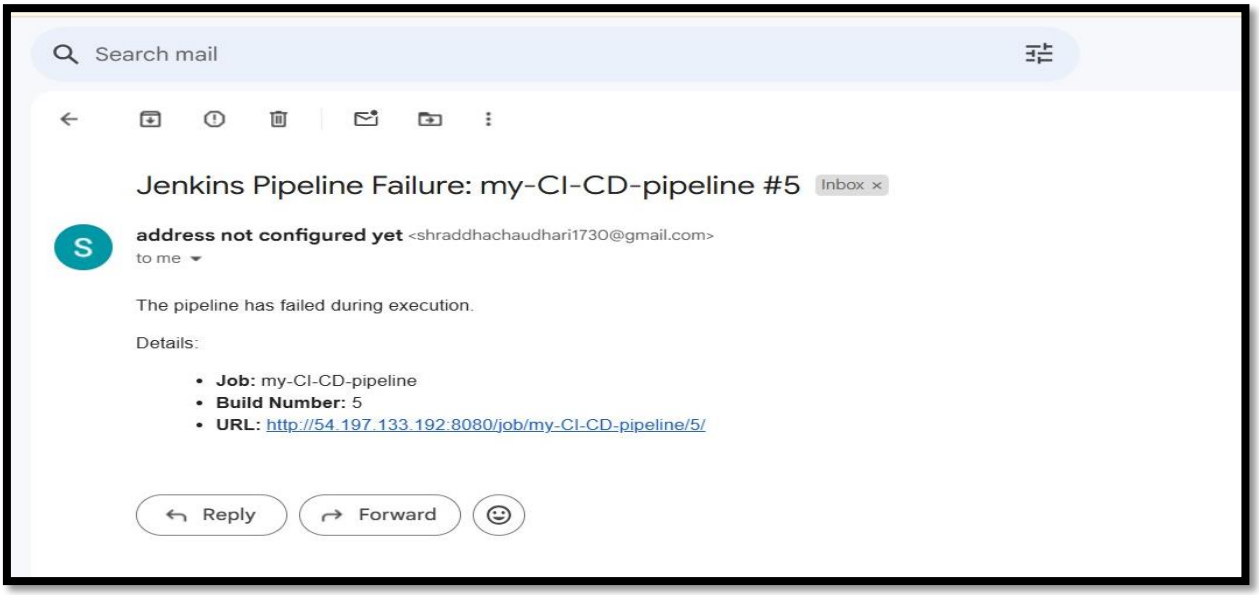
1. **Added an Intentional Failure Stage**:
    - This stage includes a deliberate error() step to make the pipeline fail for testing purposes.
2. **Post-Failure Notification**:
    - The failure block in the post section ensures an email is sent when the pipeline fails.

12. Steps to Test:

1. Replace your-recipient@example.com with the email address where notifications should be sent.
2. Commit the updated Jenkinsfile to your repository.
3. Trigger a new build in Jenkins.



4. The pipeline will fail at the Intentional Failure stage, and you should receive an email notification with failure details.



Configure Jenkins to send automated notifications on pipeline failures. Email alerts will be used to promptly inform the team about errors, ensuring timely resolution and smooth development workflows.

Note: All files related to project structure provided in above mentioned GitHub link.