Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

AY: 2025-26

Semester: V

Subject: DevOps Laboratory (DJS23OLOE501)

Experiment 4a

(Continuous Integration with Jenkins)

Aim: To Set Up Jenkins and Implement CI/CD Pipelines.

Theory:
Jenkins is an open-source automation server widely used to implement CI/CD pipelines.

Written in Java, it provides more than 1,800 plugins to integrate with various tools in the software development lifecycle.

Jenkins automates tasks like:

- Building the code.
- Running tests.
- Deploying applications.

Introduction to CI/CD

- **Continuous Integration (CI):** A development practice where developers frequently integrate their code into a shared repository, followed by automated builds and tests to detect errors early.
- **Continuous Delivery (CD):** Extends CI by automatically preparing code changes for release into production.
- **Continuous Deployment (CD):** Fully automates the release process so that every change that passes tests is deployed to production.

CI/CD Pipeline in Jenkins

A **pipeline** is a set of automated steps defined to build, test, and deploy applications.

- **Pipeline as Code:** Defined using a Jenkins file written in **Groovy syntax**.
- **Pipeline Stages:**
  - **Source:** Pull code from GitHub/GitLab.
  - **Build:** Compile the code using tools like Maven or Gradle.
  - **Test:** Run unit/integration tests.
  - **Deploy:** Deploy to staging or production servers.
  -

Setting up Jenkins

Steps to set up Jenkins on a server:

1. **Install Java** (JDK).
2. **Download and Install Jenkins** (WAR or package-based installation).
3. **Start Jenkins Service** and access it via http://localhost:8080.
4. **Unlock Jenkins** using the initial admin password.
5. **Install Recommended Plugins** (Git, Pipeline, Maven, etc.).
6. **Create Admin User** and configure basic settings.

Implementing CI/CD Pipeline in Jenkins

1. **Integrate Source Code Management (SCM):** Connect Jenkins to GitHub/GitLab using HTTPS or SSH.
2. **Create a Pipeline Job:** Define the project workflow using a Jenkinsfile.
3. **Define Stages:**
   - Checkout → Build → Test → Deploy.
4. **Automate Triggers:** Configure webhook from GitHub to Jenkins (build starts automatically on each push).
5. **Monitor Pipeline Execution:** View console output, test results, and artifacts in Jenkins dashboard.

CI/CD pipeline for a Python Flask application using:

- **Google Cloud VMs** (for Jenkins + SonarQube, and for Flask App hosting)
- **Jenkins** (automation server)
- **GitHub** (source code repository)
- **SonarQube** (code quality analysis)
- **SSH Deployment** (Flask app auto-deployment on App VM)

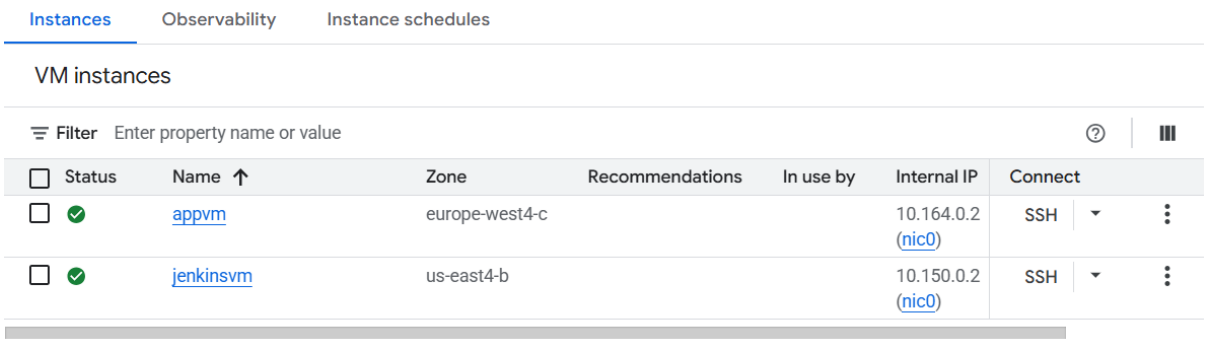
Lab experiment to be performed in this session:

1. Attache screenshots of above task performed in the lab session.
2. Create a basic Jenkins pipeline for a Java (Maven) application that performs the following steps:

- Checkout the source code from GitHub.
- Build the project using Maven.

Run the pipeline and show the output of each stage.

Step 0: VM Setup and Prerequisites



On both VMs:

```
sudo apt update -y
sudo apt install -y python3 python3-pip git unzip curl
```

On Jenkins VM only (Java for Jenkins & SonarQube):

```
sudo apt install -y openjdk-17-jdk
java --version   # Expected output: openjdk version "17.x"
```

```
hello-python/
├─ app.py
├─ requirements.txt
├─ sonar-project.properties
├─ tests/
│  └─ test_app.py
├─ .github/
│  └─ workflows/
│     └─ ci.yml
└─ Jenkinsfile
```

app.py

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

@app.route("/")
def hello():
    return "Hello from App VM!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

requirements.txt

```
Flask==2.3.2
pytest==7.3.2
Werkzeug==2.3.7
```

tests/test_app.py

```
from app import app

def test_home():
    client = app.test_client()
    res = client.get("/")
    assert res.status_code == 200
    assert b"Hello from App VM!" in res.data
```

sonar-project.properties

```
sonar.projectKey=hello-python
sonar.projectName=hello-python
sonar.projectVersion=1.0
sonar.sources=.
sonar.sourceEncoding=UTF-8
```

.github/workflows/ci.yml

```
name: Python CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]


jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - run: |
          pytest -q
```

Jekinsfile

```
pipeline {
    agent any

    environment {
        SONARQUBE = 'sonarqube'
        SCANNER = 'SonarScanner'
        SONAR_PROJECT_KEY = 'hello-python'
        SONAR_API_TOKEN = credentials('sonar-token')  // Jenkins secret
        SONAR_HOST_URL = 'http://34.6.90.93:9000' //Jenkin VM IP
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/ParthSavla2345/hello-python.git'  //github username
            }
        }

        stage('Install & Run Tests') {
            steps {
                sh '''
                  echo "Installing dependencies..."
                  python3 -m pip install --upgrade pip
                  pip3 install --user -r requirements.txt
```

```
        echo " Running tests..."
        python3 -m pytest -q
      '''
    }
}

stage('SonarQube Analysis (Async)') {
    steps {
      withSonarQubeEnv("${SONARQUBE}") {
        withEnv(["PATH+SONAR=${tool SCANNER}/bin"]) {
          sh '''
            echo "Running SonarQube scan asynchronously..."
            sonar-scanner \
              -Dsonar.projectKey=$SONAR_PROJECT_KEY \
              -Dsonar.sources=. \
              -Dsonar.host.url=$SONAR_HOST_URL \
              -Dsonar.login=$SONAR_API_TOKEN \
              -Dsonar.python.version=3.10
          '''
        }
      }
    }
}

stage('Deploy to App VM') {
    steps {
      sshagent(credentials: ['gce-ssh']) {
```

```
            sh '''
                echo "Deploying with systemd..."

                ssh -o StrictHostKeyChecking=no ParthSavla2345@34.147.64.223 "mkdir -p /home/ParthSavla2345/app"    //appvm (to check IP address whoami)
                scp -o StrictHostKeyChecking=no -r * ParthSavla2345@34.147.64.223:/home/ParthSavla2345/app/

                ssh -o StrictHostKeyChecking=no ParthSavla2345@34.147.64.223 "
                  sudo systemctl daemon-reload &&
                  sudo systemctl restart flaskapp &&
                  sudo systemctl enable flaskapp
                "

                echo "Deployment complete. Check: curl http:// 34.147.64.223:8080"
            '''
        }
    }
}

    stage('Optional: Check SonarQube Quality Gate') {
        steps {
            script {
                echo "Fetching SonarQube Quality Gate result (non-blocking)..."
                sh """
                  curl -s -u $SONAR_API_TOKEN: \
```

```
"$SONAR_HOST_URL/api/qualitygates/project_status?projectKey=$SONAR_PROJECT_
KEY" \
                    | jq '.projectStatus.status'
                """
                echo "You can manually inspect the SonarQube dashboard for full details."
            }
        }
    }
}

post {
    success {
        echo " Pipeline Succeeded"
    }
    failure {
        echo " Pipeline Failed"
    }
}
}
```

Step 3: Initialize Git repo and push to GitHub

On Jenkin-VM

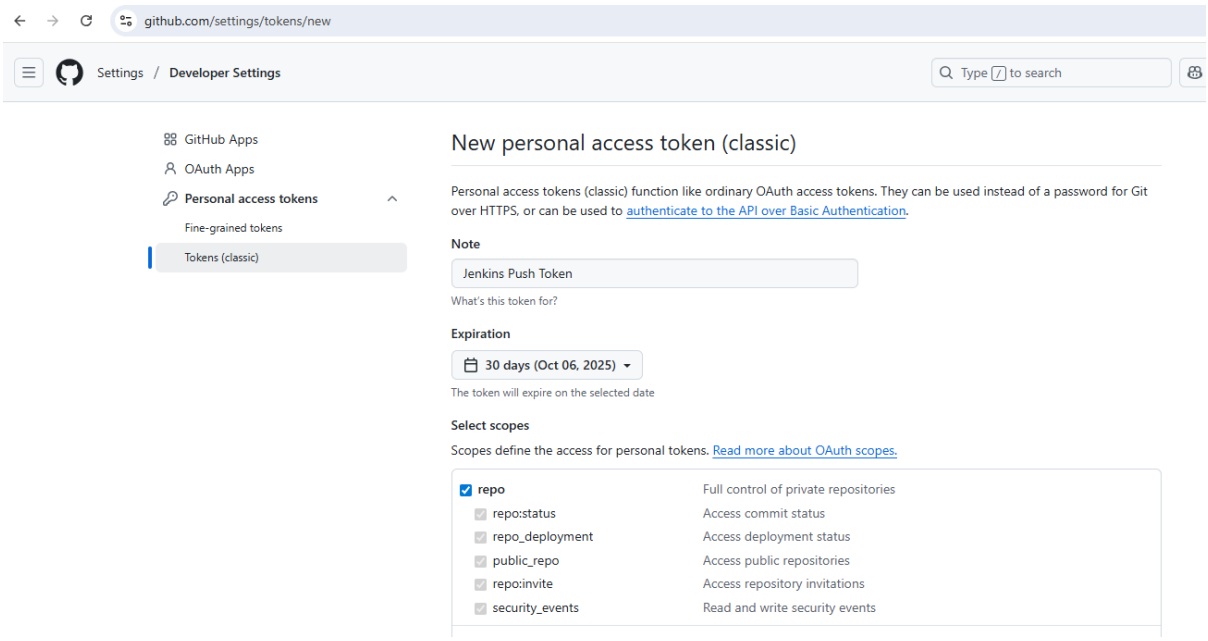
git config --global user.name " ParthSavla2345" //github username

git config --global user.email parthwork045@gmail.com //github gmail

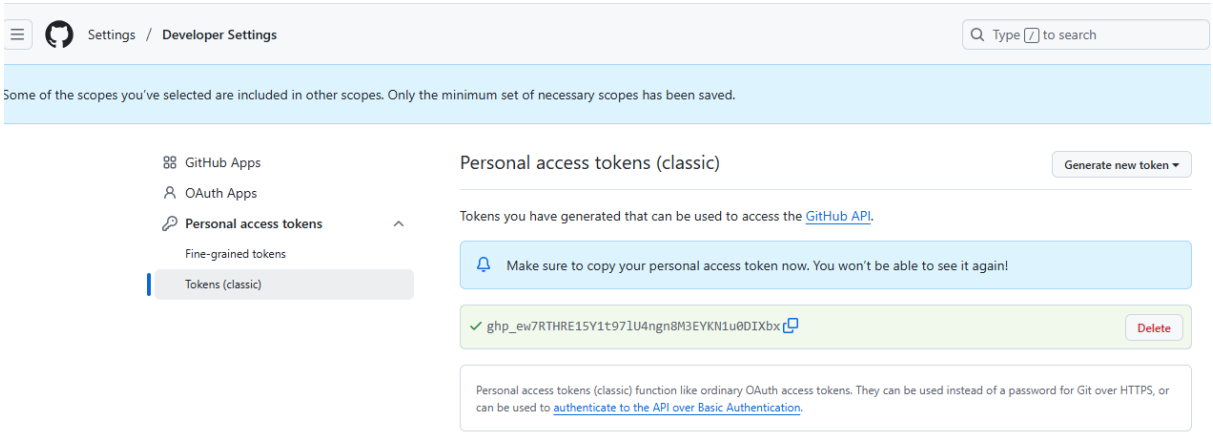
Create repository name hello-python

Generate a Personal Access Token (PAT)

1. **Go to GitHub → Settings → Developer settings → Personal access tokens → Tokens (classic) → Generate new token.**
2. Set:
   - **Note**: e.g., Jenkins Push Token
   - **Expiration**: your choice (e.g., 90 days)
   - **Scopes**: check repo and workflow (full control of private repositories)
3. Click **Generate token**.



4. Copy the token **immediately** (you won't see it again).



Copy token:
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

On Jenkin-VM Push

cd ~/hello-python

git init

git branch -M main

git add .

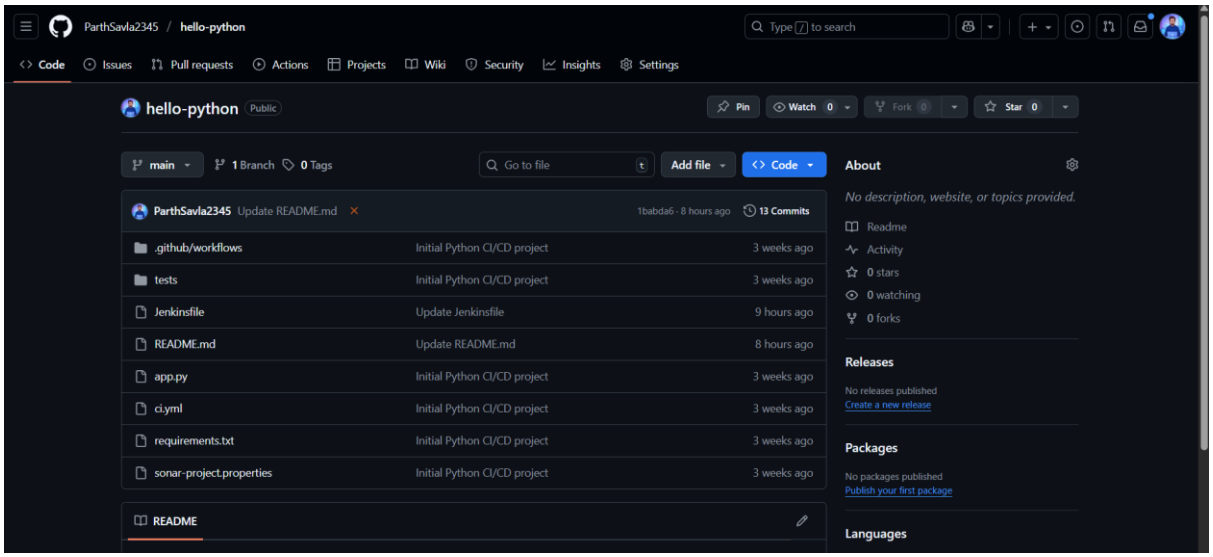
git commit -m "Initial Python CI/CD project"

git remote add origin https://github.com/ParthSavla2345/hello-python.git

When Git prompts:

Username for 'https://github.com': ParthSavla2345
Password for **PAT(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)**
git push -u origin main

Expected Output: commit pushed, GitHub repo shows all files.
Verify: GitHub Actions tab → workflow runs Python CI.



Generate SSH Key on Jenkins VM

On Jenkins VM

ssh-keygen -t ed25519 -C "student@jenkins-vm"

Press Enter to accept default location (~/.ssh/id_ed25519)

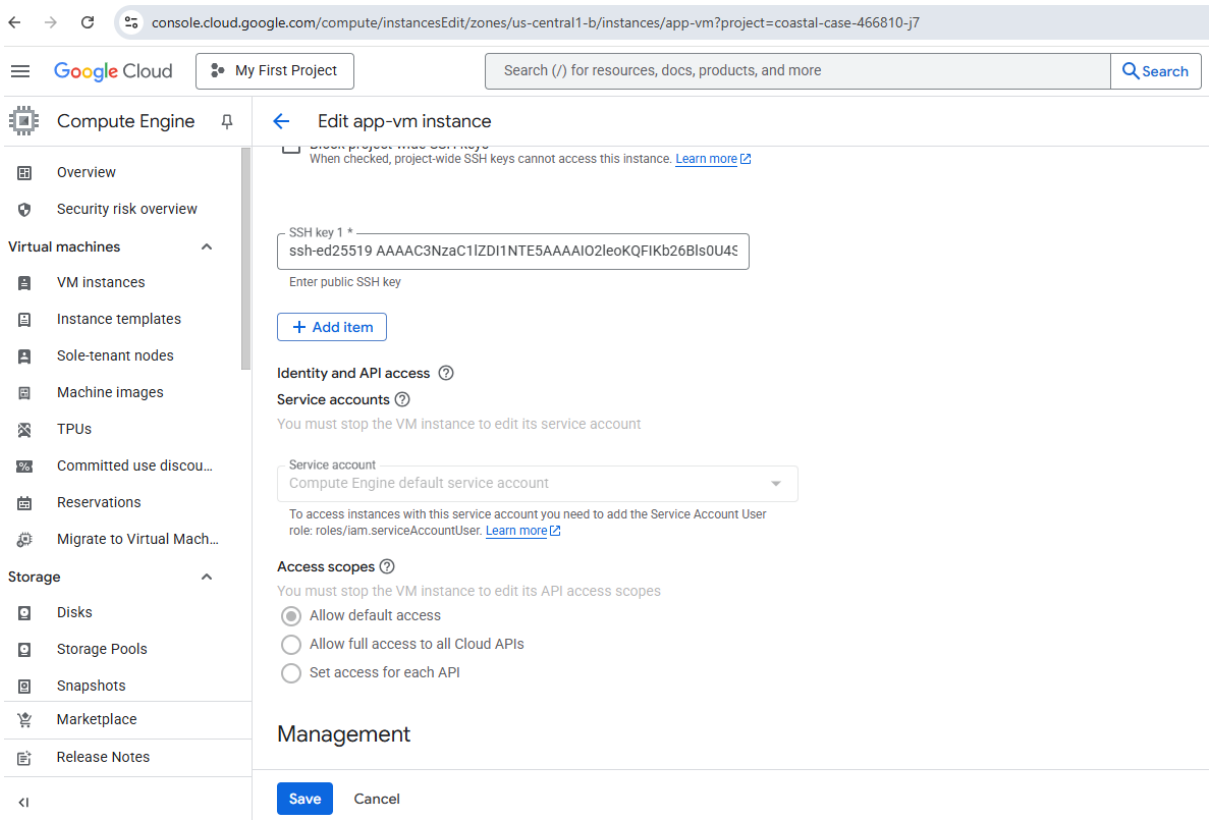
Press Enter again to leave passphrase empty

Copy the public key to App VM

1. On Jenkins VM:

cat ~/.ssh/id_ed25519.pub

2. Copy the output.
3. On App-Vm paste that SSh key.



Verify: ssh –i...

4. On App VM:

mkdir -p ~/.ssh

echo "SHA256:am2TtgPMNuCoggqSReUM2YCrQlAEvaw9ao1iiR1s9qY ParthSavla2345@jenkins-vm" >> ~/.ssh/authorized_keys

chmod 600 ~/.ssh/authorized_keys

chmod 700 ~/.ssh

Step 3: Test SSH connection

On Jenkins VM:

ssh ParthSavla2345@ 34.6.90.93 //APP IP address

- You should **connect without being asked for a password**.

Step 3: Prepare App VM

On app-vm

```
mkdir -p ~/app
cd ~/app

# Optional manual run to test
python3 -m pip install --upgrade pip
pip3 install Flask pytest
echo "from app import app" > dummy.py  # sanity check
```

Test that you can run the app manually:

python3 app.py

curl http://34.30.82.60:8080/

Docker Installation for SonarQube

Step 4: Prepare Jenkins VM

On jenkin vm

sudo apt update -y

sudo apt install -y openjdk-17-jdk python3 python3-pip git curl docker.io

sudo systemctl enable --now docker

Install Jenkins:

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb https://pkg.jenkins.io/debian binary/ > /etc/apt/sources.list.d/jenkins.list'

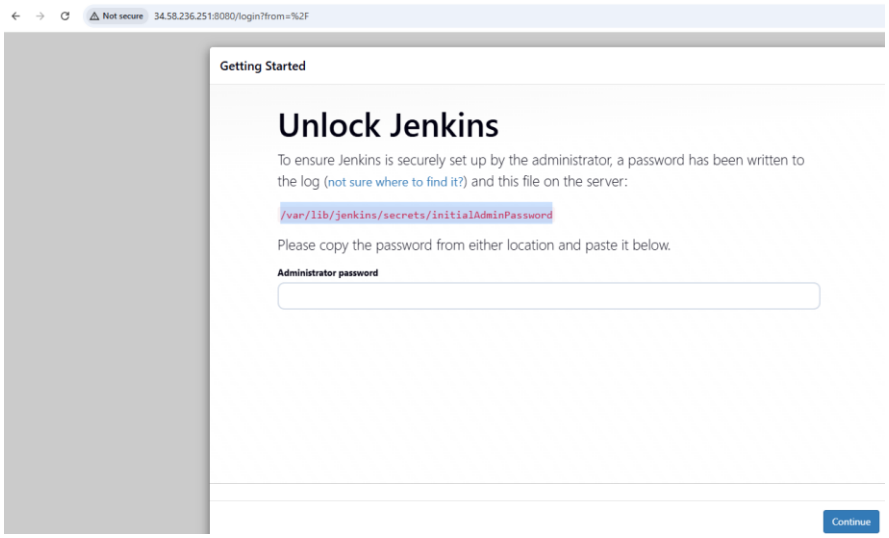
sudo apt update

sudo apt install -y jenkins

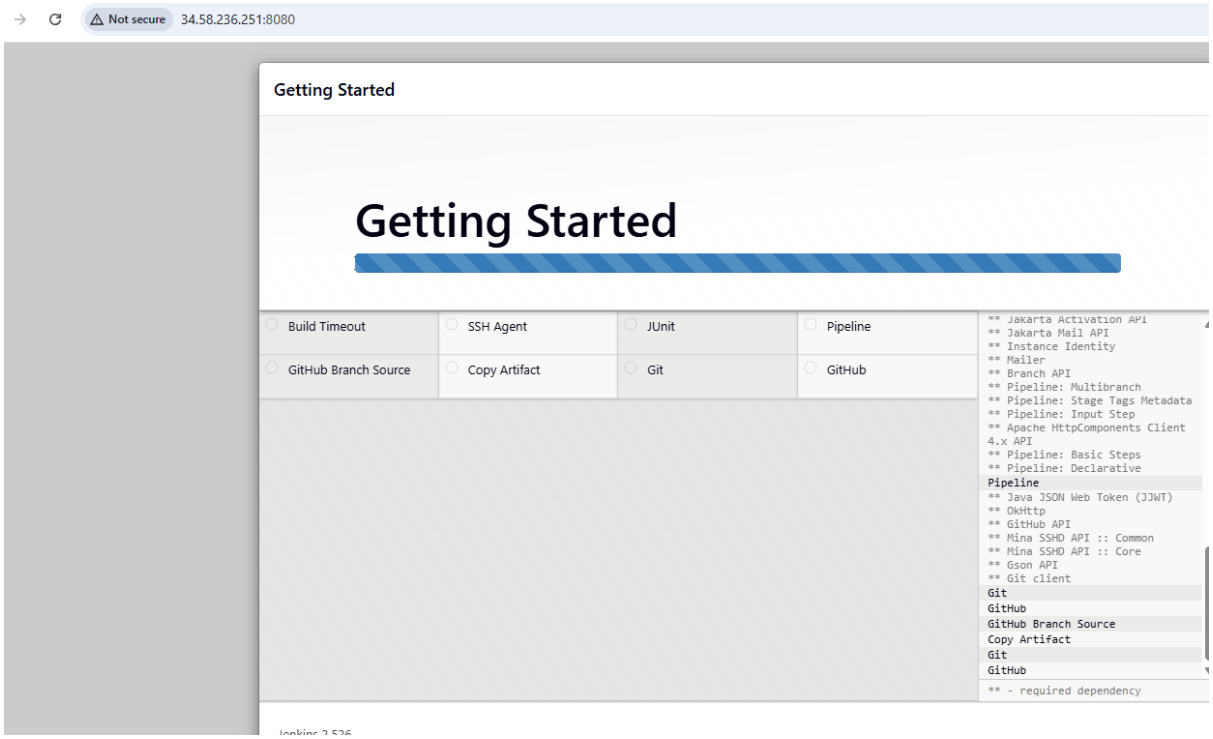
sudo systemctl enable --now jenkins

sudo systemctl status Jenkins

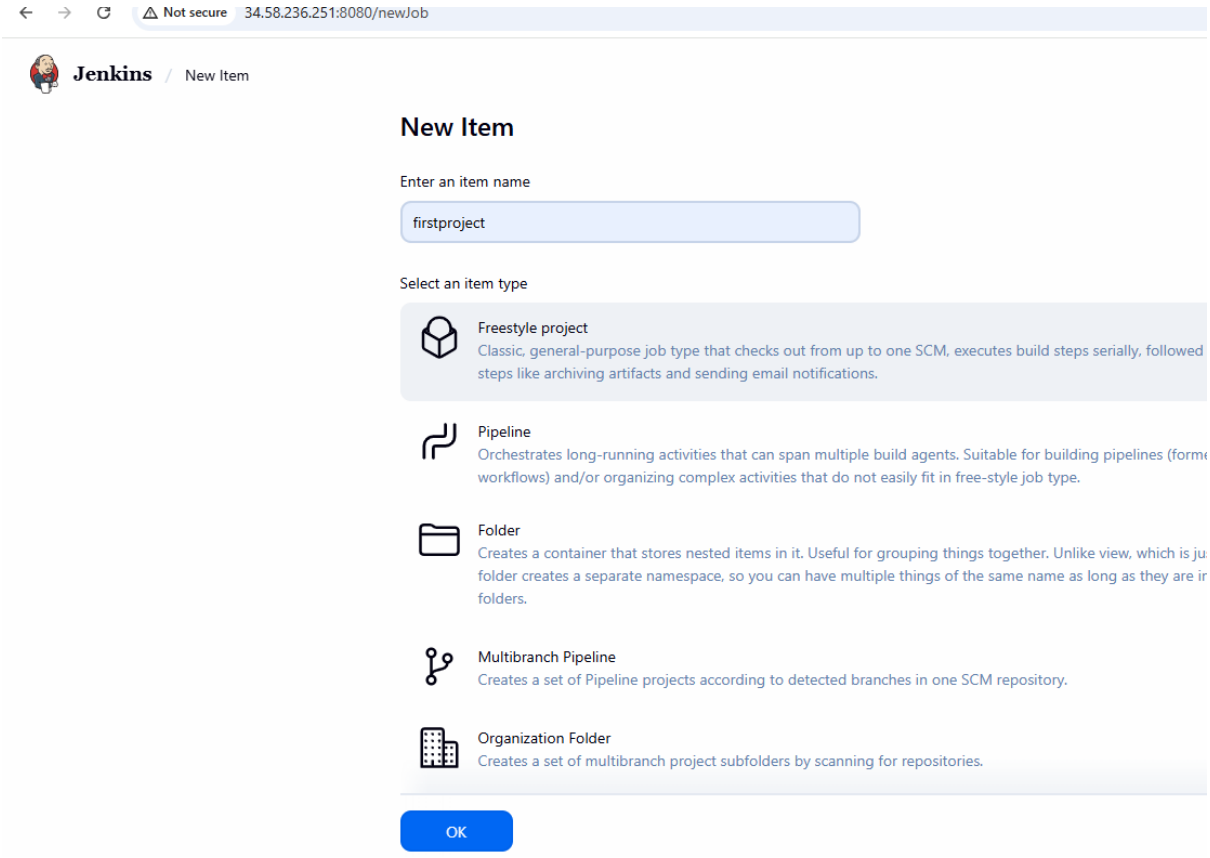
Open Jenkins UI: http://<JENKINS_VM_IP>:8080

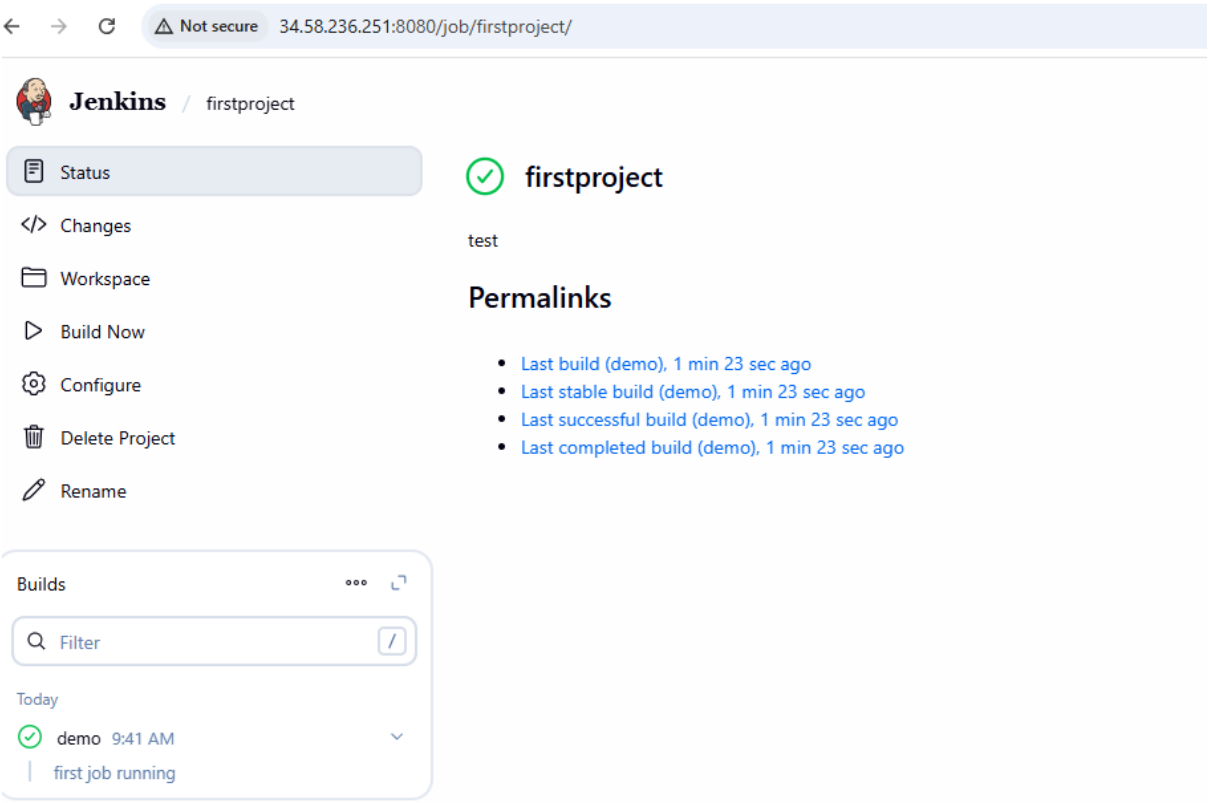


Unlock with sudo cat /var/lib/jenkins/secrets/initialAdminPassword
Install **Suggested plugins**.



Username and password create for own.


Not secure 34.58.236.251:8080/newJob
Jenkins / New Item
New Item
Enter an item name
firstproject
Select an item type
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed
steps like archiving artifacts and sending email notifications.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (forme
workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is ju
folder creates a separate namespace, so you can have multiple things of the same name as long as they are ir
folders.
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.
Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.
OK


Not secure 34.58.236.251:8080/job/firstproject/
Jenkins / firstproject
Status
Changes
Workspace
Build Now
Configure
Delete Project
Rename
firstproject
test
Permalinks
Last build (demo), 1 min 23 sec ago
Last stable build (demo), 1 min 23 sec ago
Last successful build (demo), 1 min 23 sec ago
Last completed build (demo), 1 min 23 sec ago
Builds
Filter
Today
demo 9:41 AM
first job running