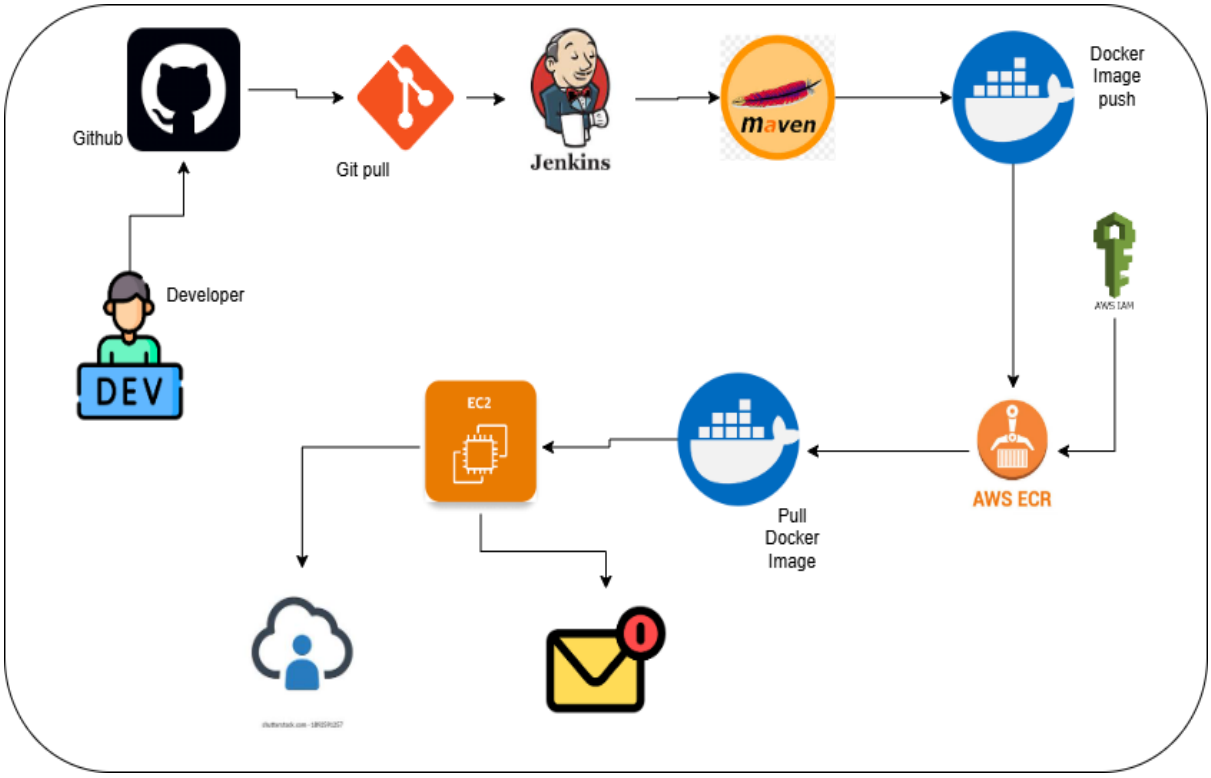
Jenkins CI/CD Pipelines

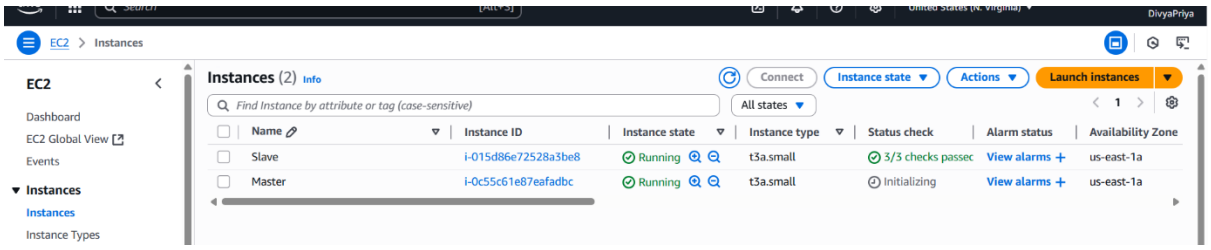
A CI/CD pipeline in Jenkins refers to the automation of the software delivery process, from code integration to deployment, using Jenkins as the orchestrator. CI/C D stands for Continuous Integration and Continuous Delivery/Deployment.



Creating Master Server

🔷 Step 1: Launch EC2 Instances

- **Master**: Ubuntu 22.04 LTS (t3a.small instance type).
- **Slave**: Amazon Linux (t3a.small instance type).



Allow **ports in security group**:

- 22 (SSH)
- 8080 (Jenkins UI)

🔷 Step 2: Connect to Master (Ubuntu)

Connect to putty,

ssh -i your-key.pem ubuntu@<MASTER_PUBLIC_IP>

switch to root user -> sudo su

🔷 Step 3: Update & Upgrade

apt update -y

apt upgrade -y

🔷 Step 4: Install Docker

apt install docker.io -y

systemctl enable docker

systemctl start docker

usermod -aG docker ubuntu

🔷 Step 5: Install Java (OpenJDK 21)

apt update -y

apt install fontconfig openjdk-21-jre -y

java -version

Expected output:

openjdk version "21.0.3" 2024-04-16

OpenJDK Runtime Environment (build 21.0.3+11-Debian-2)

OpenJDK 64-Bit Server VM (build 21.0.3+11-Debian-2, mixed mode, sharing)

🔷 Step 6: Install Git

apt install git -y

git --version

🔷 Step 7: Install Jenkins

1st Approach:

```
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" https://pkg.jenkins.io/debian-stable binary/ | sudo tee   /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update

sudo apt install Jenkins
```

2nd Approach:

```
# Move to keyrings directory

mkdir -p /etc/apt/keyrings

cd /etc/apt/keyrings

# Download Jenkins key

wget -O /etc/apt/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# Add Jenkins repo

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" \

    | tee /etc/apt/sources.list.d/jenkins.list > /dev/null

# Update and install Jenkins

apt update -y

apt install jenkins -y
```

🔷 Step 8: Start Jenkins

```
systemctl enable jenkins

systemctl start jenkins

systemctl status jenkins
```

🔷 Step 9: Access Jenkins

Open browser → http://<MASTER_PUBLIC_IP>:8080

Get initial admin password:

cat /var/lib/jenkins/secrets/initialAdminPassword

Paste it in Jenkins UI → Install suggested plugins → Create admin user.

```
ubuntu@ip-172-31-44-190: ~
ubuntu@ip-172-31-44-190:~$ docker -v
Docker version 27.5.1, build 27.5.1-0ubuntu3~24.04.2
ubuntu@ip-172-31-44-190:~$ java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)
ubuntu@ip-172-31-44-190:~$ git --version
git version 2.43.0
ubuntu@ip-172-31-44-190:~$ jenkins --version
2.516.3
ubuntu@ip-172-31-44-190:~$
```

Creating Slave server

🔷 Step 1: Connect to Slave (Amazon Linux)

ssh -i your-key.pem ec2-user@<SLAVE_PUBLIC_IP>

switch to root user -> sudo su

🔷 Step 2: Update Packages

yum update -y

🔷 Step 3: Install Java (required for Jenkins agent)

amazon-linux-extras enable corretto8

yum install java-1.8.0-amazon-corretto -y

java -version

👉 If you want Java 11 or 17 instead:

amazon-linux-extras enable corretto17

yum install java-17-amazon-corretto -y

java -version

🔷 Step 4: Install Git & Docker

yum install git -y

yum install docker -y

systemctl start docker

systemctl enable docker

usermod -aG docker ec2-user

🔷 Step 5: Configure Security Groups

On the Slave EC2 security group, ensure port 22 (SSH) is open for the Master's IP.

🔷 Step 6: Configure Jenkins Master to Connect Slave

Go to Jenkins UI → http://<MASTER_PUBLIC_IP>:8080

Login as Admin

Navigate: Manage Jenkins → Nodes & Clouds → New Node

Enter node name → Select Permanent Agent

Configure: mkdir Pipeline

Remote root directory: /home/ec2-user/Pipeline

Labels: Myslave (you'll use this in pipelines)

Launch method: "Launch agents via SSH"

Add credentials: Kind: SSH Username with private key

Username: ec2-user

Private Key: Paste your .pem file content (same one you use for SSH).

Save → Jenkins will try to connect via SSH.

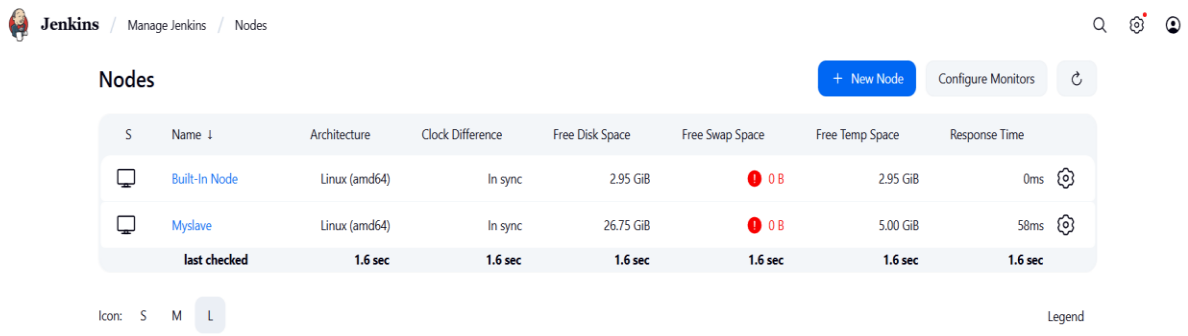
🔷 Step 7: Verify Slave Connection

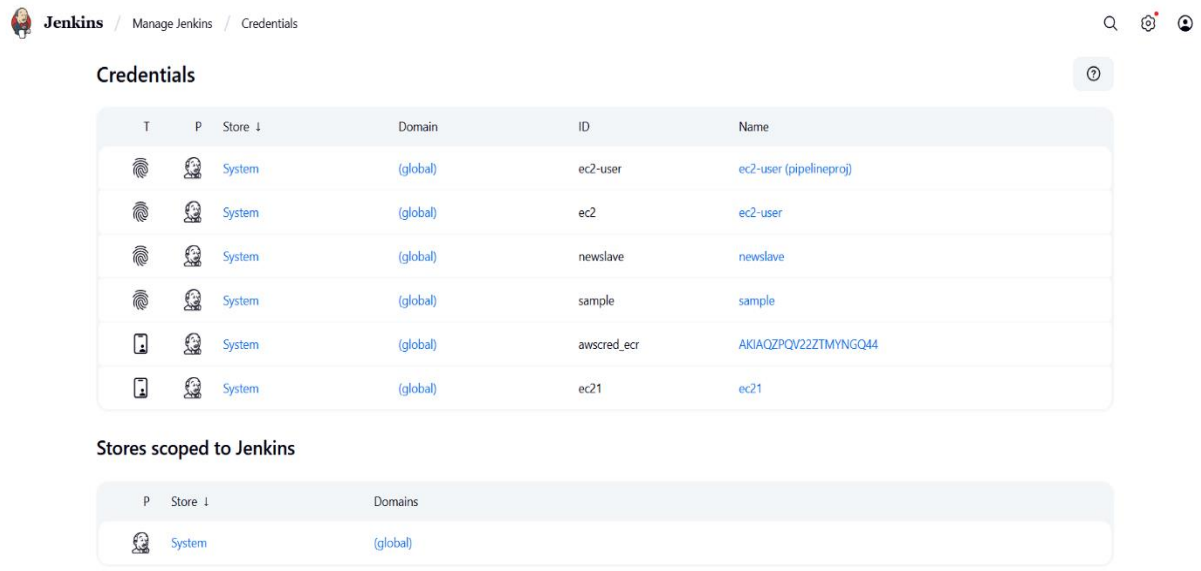
Node status should show online.

On Slave, Jenkins will create a .jenkins folder inside /home/ec2-user/Pipeline.

```
login as: ec2-user
Authenticating with public key "imported-openssh-key"

A newer release of "Amazon Linux" is available.
  Version 2023.9.20250929:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sun Sep 28 11:49:44 2025 from 27.60.175.225
[ec2-user@ip-172-31-32-28 ~]$ ls
aws  awscliv2.zip  pipeline
[ec2-user@ip-172-31-32-28 ~]$ cd pipeline
[ec2-user@ip-172-31-32-28 pipeline]$ ls
caches  remoting  remoting.jar  tools  workspace
[ec2-user@ip-172-31-32-28 pipeline]$ cd workspace
[ec2-user@ip-172-31-32-28 workspace]$ ls
pipelineproject  pipelineproject@tmp
[ec2-user@ip-172-31-32-28 workspace]$ cd pipelineproject
[ec2-user@ip-172-31-32-28 pipelineproject]$ ls
Dockerfile  Jenkinsfile  README.md  pom.xml  src  target
[ec2-user@ip-172-31-32-28 pipelineproject]$
```





🔷 Step 8: Test Slave with a Job

Create a simple job/pipeline and restrict it to run on Myslave:

```
pipeline {
  agent { label 'Myslave' }
  stages {
    stage('Test') {
      steps {
        sh 'echo "Hello from Jenkins Slave!"'
        sh 'hostname'
      }
    }
  }
}
```

Fetch Application from Github repository

Fetch the application from the GitHub repository, pull it into the Jenkins workspace, and build the Java application using the Maven installation tool.

Pushing Docker image in ECR Repository

🔷 Step 1: Create an ECR Repository,

Install AWS Credentials and AWS Pipeline plugins in Jenkins.

On AWS console:

Go to ECR (Elastic Container Registry) → Create Repository.

Name it, e.g. mypipelinerepo.

Copy repository URI:

<AWS_ACCOUNT_ID>.dkr.ecr.<REGION>.amazonaws.com/mypipelinerepo

🔷 Step 2: Create the user and Configure IAM policy

Attach IAM role with AmazonEC2ContainerRegistryFullAccess.

➔ Goto Security Credentials
➔ Access key & Secret key.

- Store AWS credentials in Jenkins → Manage Jenkins → Credentials → Add Credentials:

  ➔ Goto Jenkins, Credentials-> System -> Global Credentials
  ➔ New credentials
  ➔ Username & Password → Enter AWS Access Key ID and Secret Access Key.

🔷 Step 3: Install AWS CLI on Slave

sudo yum install awscli -y

aws --version

🔷 Step 4: Docker Login to ECR

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 054728709811.dkr.ecr.us-east-1.amazonaws.com

🔷 Step 5: Create Jenkins Pipeline for Docker Build & Push

In Jenkins → Create a Pipeline job

🔷 Step 6: Run Pipeline

Jenkins pulls code → builds Docker image → tags it → pushes to ECR.

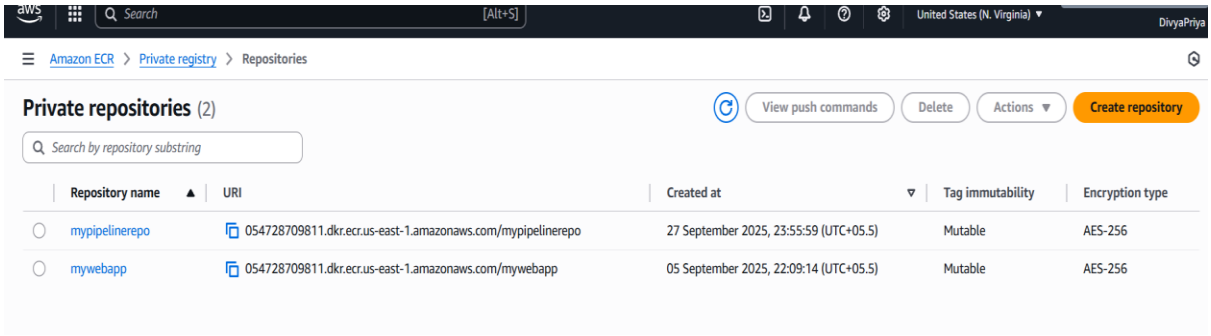
Check in AWS Console → ECR → Images → You should see the pushed image.

```
stage('Docker-Build') {
    steps {
        script {
            echo "Building Docker image and pushing to ECR..."
            withAWS(region: "${env.AWS_REGION}", credentials: 'awscred_ecr') {
                // Build image
                sh 'docker build -t myimage1:latest .'

                // Login to ECR
                sh "aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --password-stdin ${ECR_REPO

                // Tag image
                sh "docker tag myimage1:latest ${ECR_REPO}:latest"

                // Push image
                sh "docker push ${ECR_REPO}:latest"
            }
        }
    }
}
```



Pull Docker image from ECR and Deploy to an EC2 Instance

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker ec2-user

- Add Deployment Stage in Jenkins Pipeline
- Pull the image from ECR and run as docker container.

```
    stage('Deploy') {
        steps {
            script {
                echo "Pulling image from ECR for deployment...."
                withAWS(region: "${env.AWS_REGION}", credentials: 'awscred_ecr') {
                    // Login to ECR
                    sh "aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --password-stdin ${ECR_REPO}"

                    // Pull image
                    sh "docker pull ${ECR_REPO}:latest"

                    // run the container
                    sh 'docker stop mycontainer1 || true'
                    sh 'docker rm mycontainer1 || true'
                    sh 'docker run -d -p "80:8080" --name mycontainer1 ${ECR_REPO}:latest'
                }
            }
        }
    }
}
```

Add Email Notification

🔷 **Step 1: Install Email Extension Plugin**

1. Go to **Jenkins Dashboard → Manage Jenkins → Plugins → Available Plugins**.
2. Search for **Email Extension Plugin** → Install.
3. Restart Jenkins if needed.

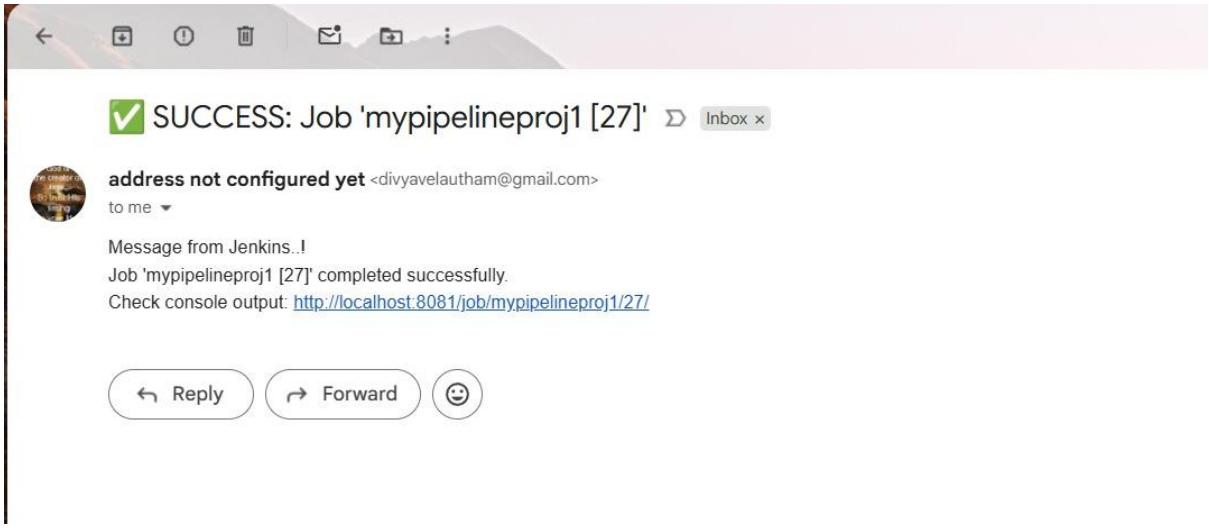
🔷 **Step 2: Configure SMTP in Jenkins**

1. Go to **Manage Jenkins → System**.
2. Scroll to **Extended E-mail Notification** and **E-mail Notification**.
3. Example (using Gmail as SMTP):
    - SMTP server: smtp.gmail.com
    - Use SSL: ✅
    - Port: 465

- Credentials: Add your Gmail App Password (not the normal password).
- Default user e-mail suffix: @gmail.com

4. Add Post stage script in Jenkins pipelines

```
    post {
        success {
            emailext (
                to: 'divyavelautham@gmail.com',
                subject: "✅ SUCCESS: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
                body: """Jenkins Pipeline!
Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' completed successfully.
Check console output: ${env.BUILD_URL}"""
            )
        }
        failure {
            emailext (
                to: 'divyavelautham@gmail.com',
                subject: "❌ FAILURE: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
                body: """Jenkins Pipeline!
Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' failed.
Check console output: ${env.BUILD_URL}"""
            )
        }
        always {
            echo "Pipeline finished, email notification sent"
        }
    }
}
```



Email notification has successfully sent from Jenkins.

Jenkins pipeline Output:

```
[Pipeline] }
[Pipeline] // withAWS
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline finished, email notification sent
[Pipeline] emailext
Sending email to: divyavelautham@gmail.com
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

pipelineproject - Stage View

	Declarative: Tool Install	Checkout	Code-Build	Docker-Build	Deploy	Declarative: Post Actions
Average stage times: (full run time: ~23s)	204ms	1s	7s	4s	3s	10s
#29 Sept 30 18:30 No Changes	264ms	719ms	8s	5s	5s	219ms
#28 Sept 30 18:11 No Changes	446ms	4s	9s	7s	4s	20s

Successful build the java application CI/CD flow using Jenkins pipeline.