Hands-On Jenkins Implementation for Real-World CI/CD Automation

Proud to showcase my practical journey working with Jenkins — a powerful CI/CD automation tool — where I executed and tested real-time tasks like installation, job configuration, Git integration, pipelines, and master-agent setups to streamline the software development lifecycle.

Introduction to Jenkins & CI/CD

What is Jenkins?

Jenkins is an **open-source automation server** that helps automate the building, testing, and deploying of applications. It supports Continuous Integration and Continuous Deployment (CI/CD) and is a widely adopted tool in the DevOps ecosystem.

What is CI/CD?

- **CI (Continuous Integration)**: Developers merge code into a shared repository several times a day. Jenkins builds and tests code automatically.
- **CD (Continuous Delivery/Deployment)**: After passing tests, the code is automatically deployed to production or staging environments.

Why Jenkins?

Jenkins is one of the most popular automation tools in the DevOps ecosystem due to its flexibility, extensibility, and strong community support. Here's why it stands out:

- **Open-Source and Free** – Available to everyone with continuous updates and community contributions
- **Platform-Independent** – Works seamlessly across Windows, Linux, and macOS
- **Extensive Plugin Ecosystem** – Over 1800 plugins available to integrate with tools like Git, Maven, Docker, Selenium, AWS, Kubernetes, and more
- **Large Community Support** – A vast community ensures better support, updates, and shared knowledge
- **Easy Integration** – Connects effortlessly with version control systems, build tools, testing frameworks, and deployment tools
- **Distributed Builds** – Master-agent architecture enables parallel execution of jobs across multiple nodes
- **User-Friendly & Flexible** – Provides both a simple web interface for configuration and powerful scripting via pipelines (Groovy)

How Jenkins Works (Behind the Scenes)

1. **Developers push code** to a version control system like **GitHub**
2. **Jenkins detects the change** (via Poll SCM or Webhook)
3. It **pulls the code** and sends it to a **build tool** (like Maven)
4. Once the code is built, Jenkins can:
    - **Run automated tests** (e.g., with Selenium)
    - **Deploy to test or production servers**
    - **Archive artifacts** or send notifications

This whole process is defined as:

- **Freestyle Jobs** (GUI-based task automation)
- **Pipelines** (script-based, using Groovy)

Example Workflow:

GitHub (Code Commit) → Jenkins → Build → Test → Deploy → Notify

Jenkins acts like a **bridge** between code and deployment, helping teams deliver faster and more reliably.

Jenkins Installation on Linux (EC2)

1. **Update packages**

   yum update -y

2. **Add Jenkins repo**

   wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

3. **Import key**

   rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

4. **Upgrade system**

   yum upgrade -y

5. **Install Java**

   yum install java-17-amazon-corretto -y

6. **Install Jenkins**

   yum install jenkins -y

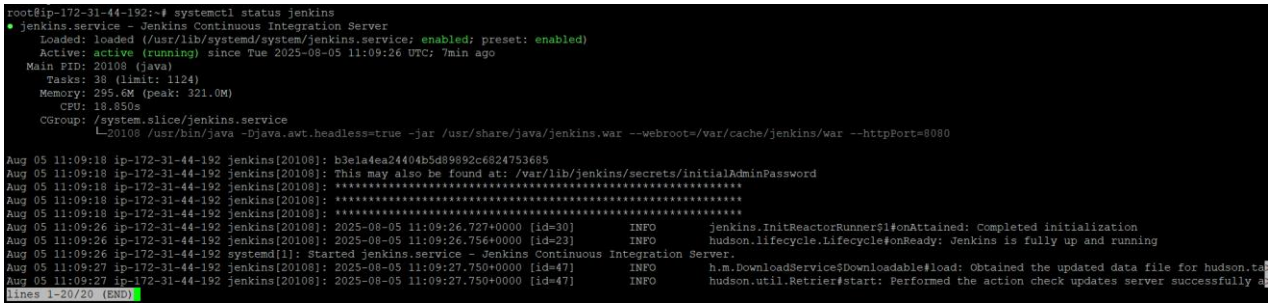
7. **Enable & Start Jenkins**

systemctl enable jenkins

systemctl start jenkins

8. **Check Jenkins status**

systemctl status jenkins



Screenshot 1: Jenkins service status

Steps to Access and Login to Jenkins via Web Interface

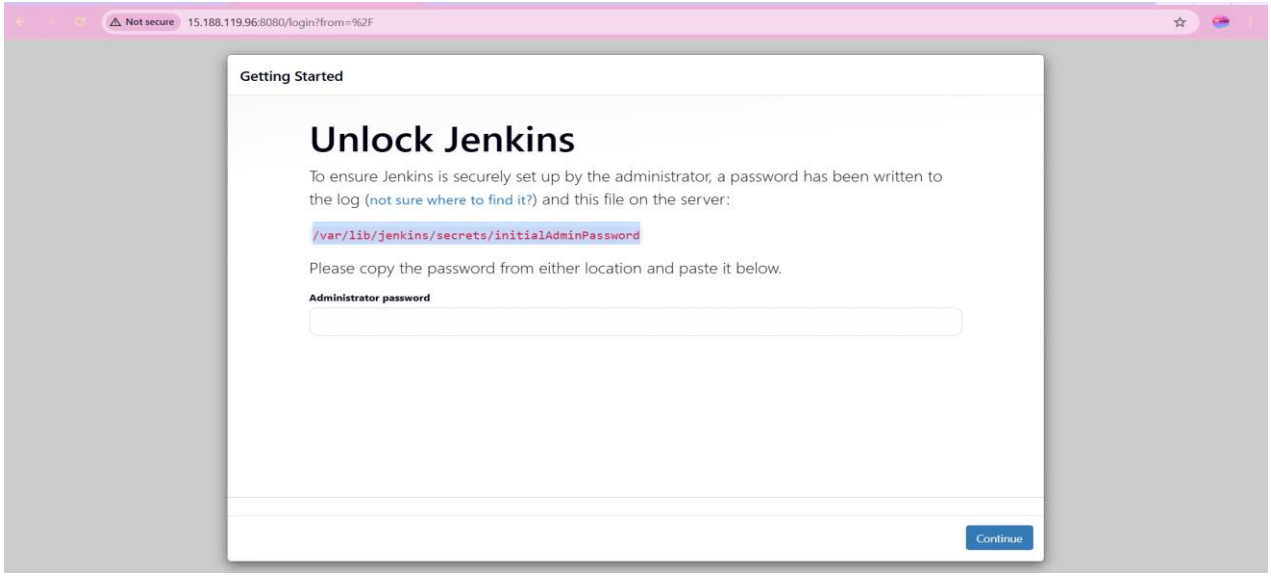
Step 1: Open Jenkins URL in Browser

- **Action**: Open your browser and enter the following URL:
- http://<your-public-ip>:8080

Example:

http://13.234.67.120:8080

- Jenkins welcome page should load.



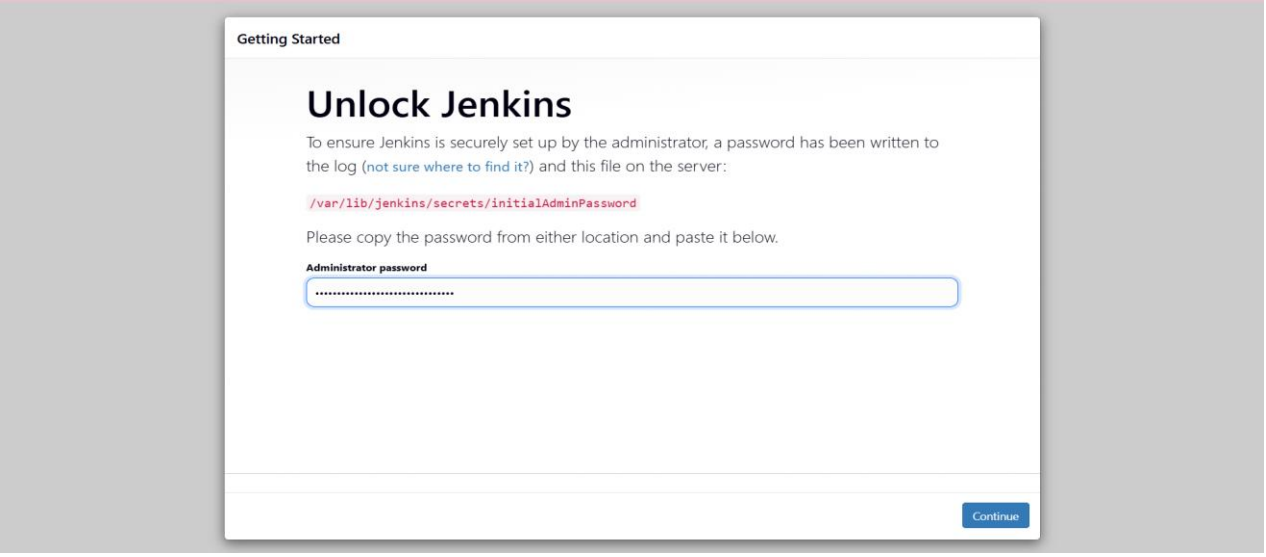
Screenshot 2: Jenkins Welcome Page

Step 2: Unlock Jenkins

- **Action**: Copy the password from the file:
- cat /var/lib/jenkins/secrets/initialAdminPassword
- Paste the password into the unlock field on the Jenkins page and click **Continue**.

```
root@ip-172-31-44-192:~# id jenkins
uid=111(jenkins) gid=113(jenkins) groups=113(jenkins)
root@ip-172-31-44-192:~# cat /var/lib/jenkins/secrets/initialAdminPassword
b3e1a4ea24404b5d89892c6824753685
```

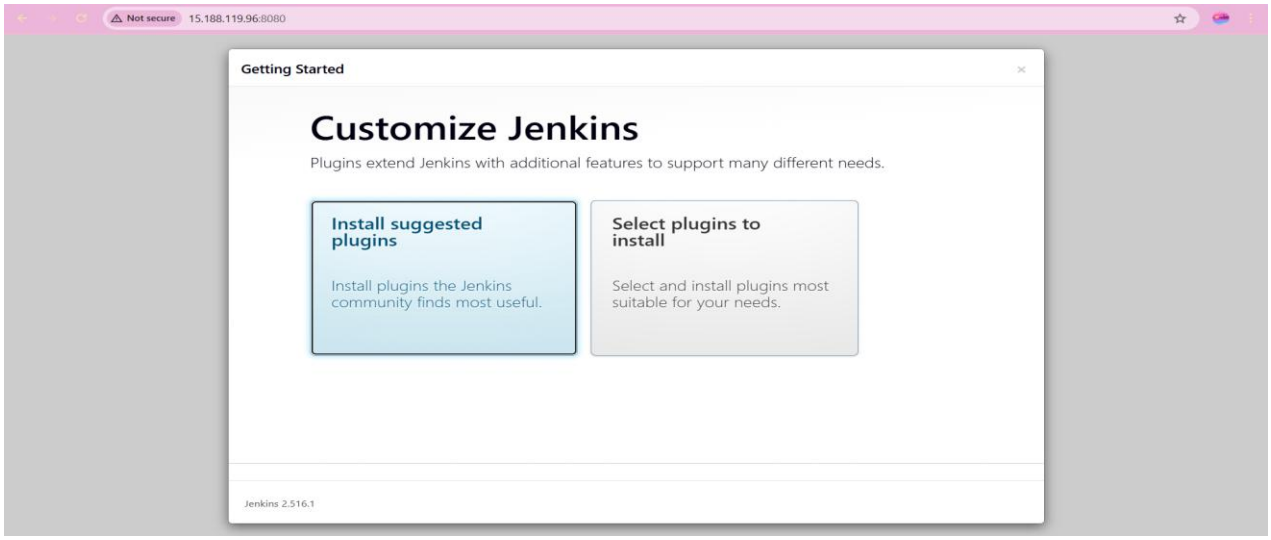
Screenshot 3: Unlock Jenkins Password Terminal



Screenshot 4: Jenkins Unlock Page

Step 3: Customize Jenkins

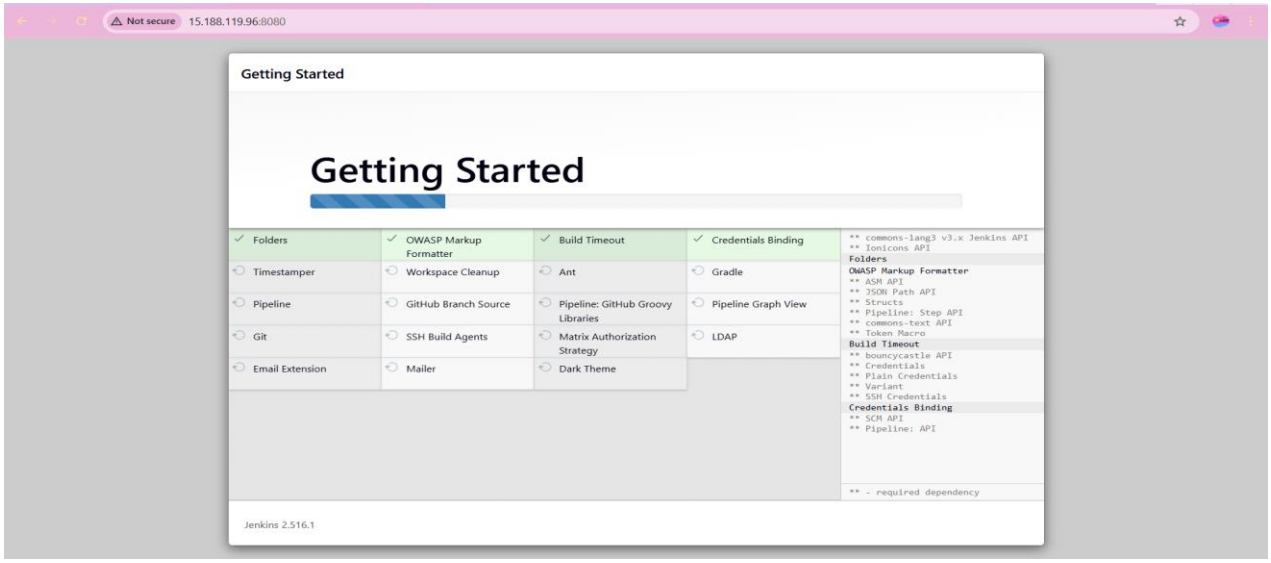
- **Action**: Choose either:
  - **Install suggested plugins** (Recommended)
  - **Select plugins to install** (Custom)
- Click on **Install suggested plugins**.



Screenshot 5: Customize Jenkins Plugins

Step 4: Plugin Installation in Progress

- **Action**: Wait while Jenkins installs the necessary plugins.

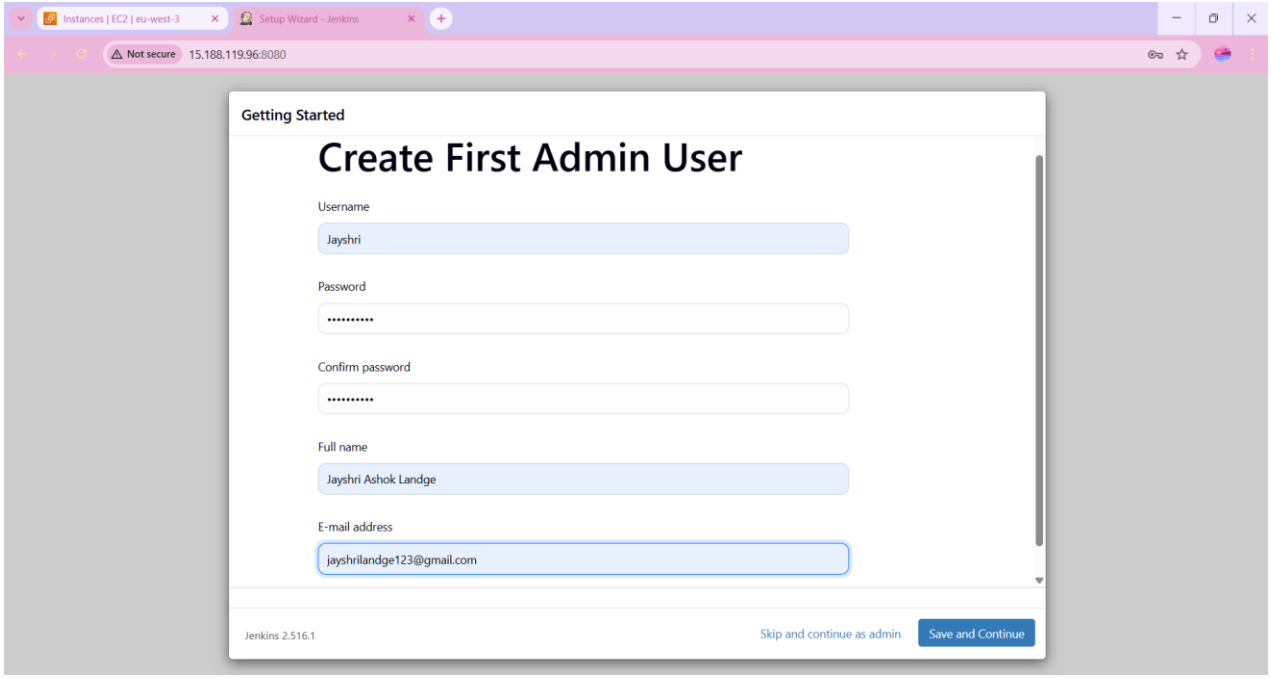


Screenshot 6: Plugin Installation Progress

Step 5: Create First Admin User

- **Action**: Fill out the admin user form:
  - Username
  - Password
  - Full name
  - Email address

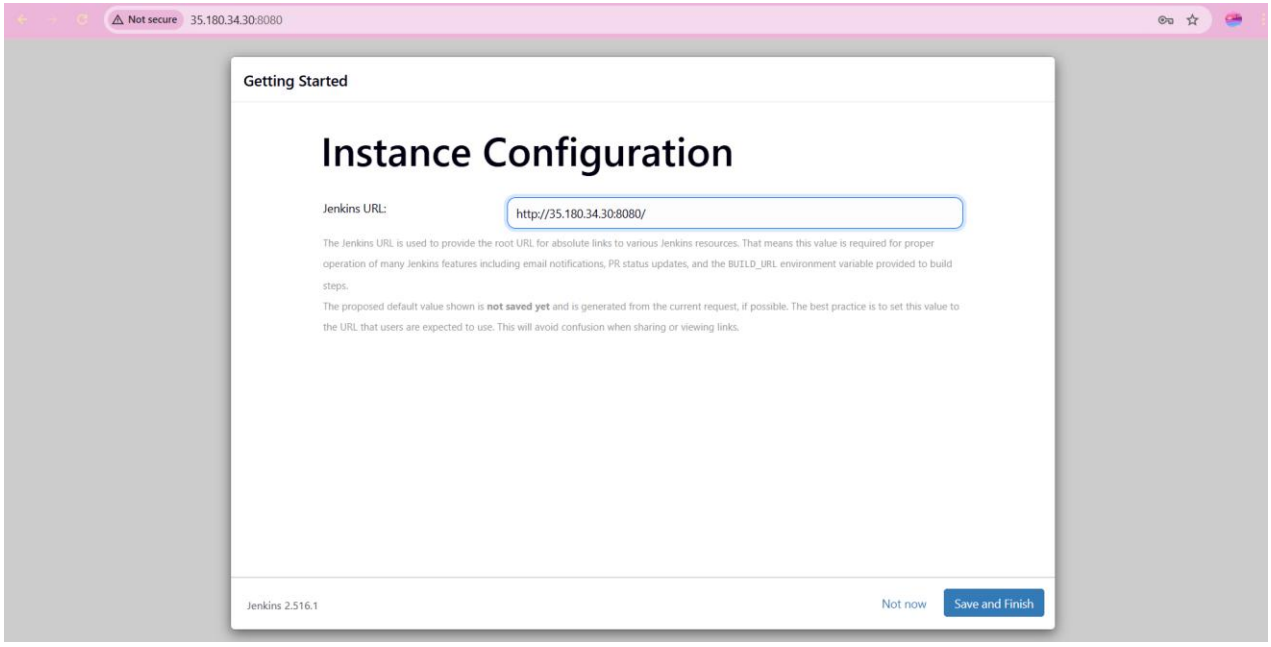
- Click **Save and Continue**.



Screenshot 7: Create Admin User

Step 6: Instance Configuration

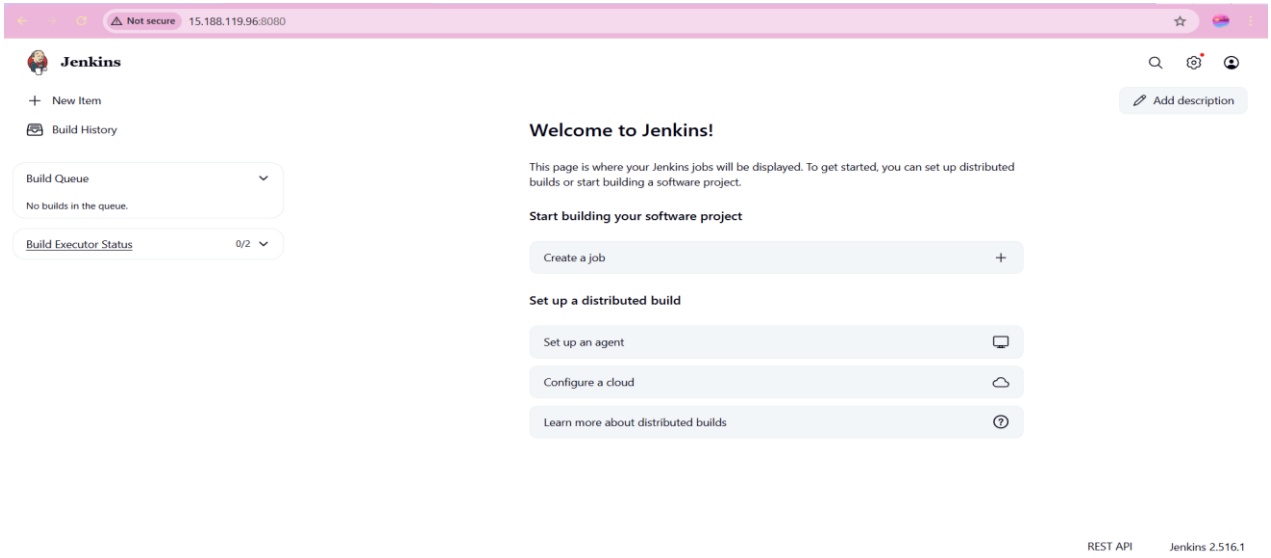
- **Action**: Jenkins will prompt to confirm the Jenkins URL. Leave it as default or update if needed.
- Click **Save and Finish**.



Screenshot 8: Configure Jenkins URL

Step 7: Jenkins is Ready!

- **Action**: Click **Start using Jenkins**.
- **Action**: You are now logged into the Jenkins Dashboard.



Screenshot 9: Jenkins Dashboard

Installing & Removing HTTPD via Jenkins

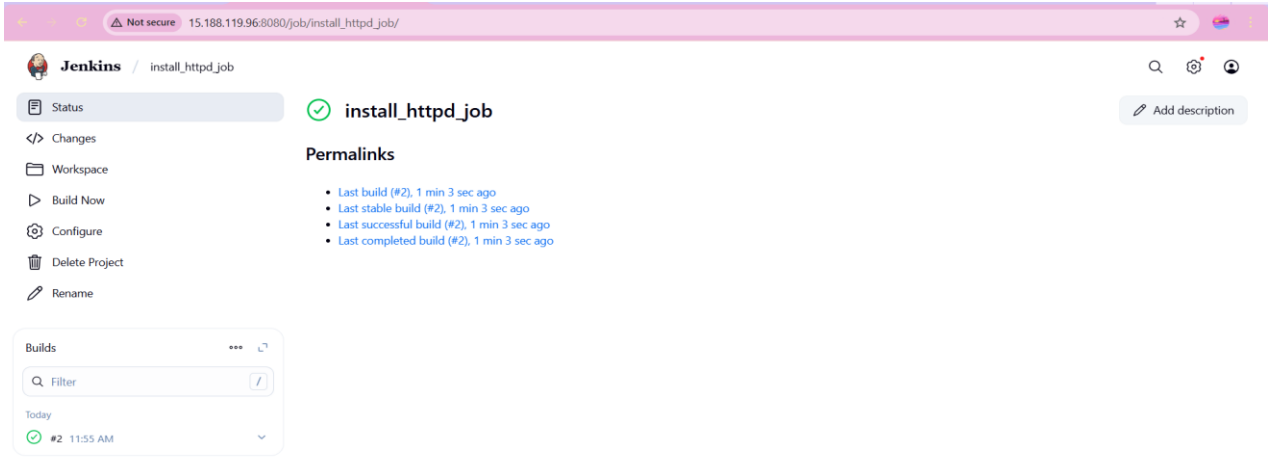
Install HTTPD:

sudo yum install -y httpd

sudo systemctl start httpd

sudo chkconfig httpd on

echo "Welcome to Jenkins" | sudo tee /var/www/html/index.html

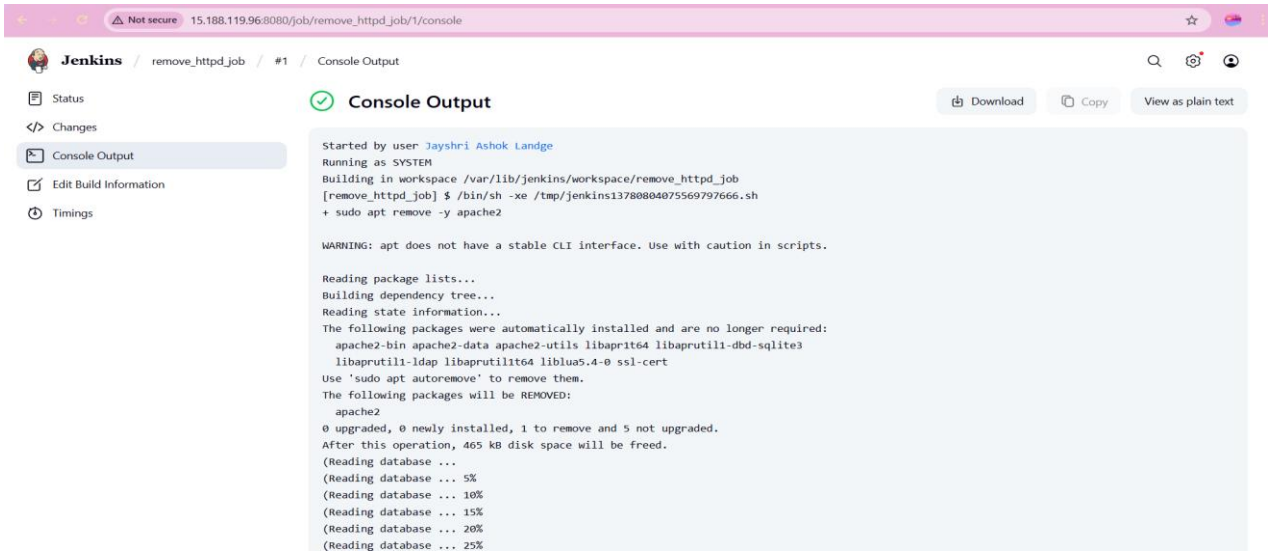


Screenshot 10: Jenkins Build Success Status Of Install httpd

Remove HTTPD:

sudo yum remove -y httpd

sudo rm -rf /var/www/html/index.html



Screenshot 11: Console output of HTTPD remove job in Jenkins

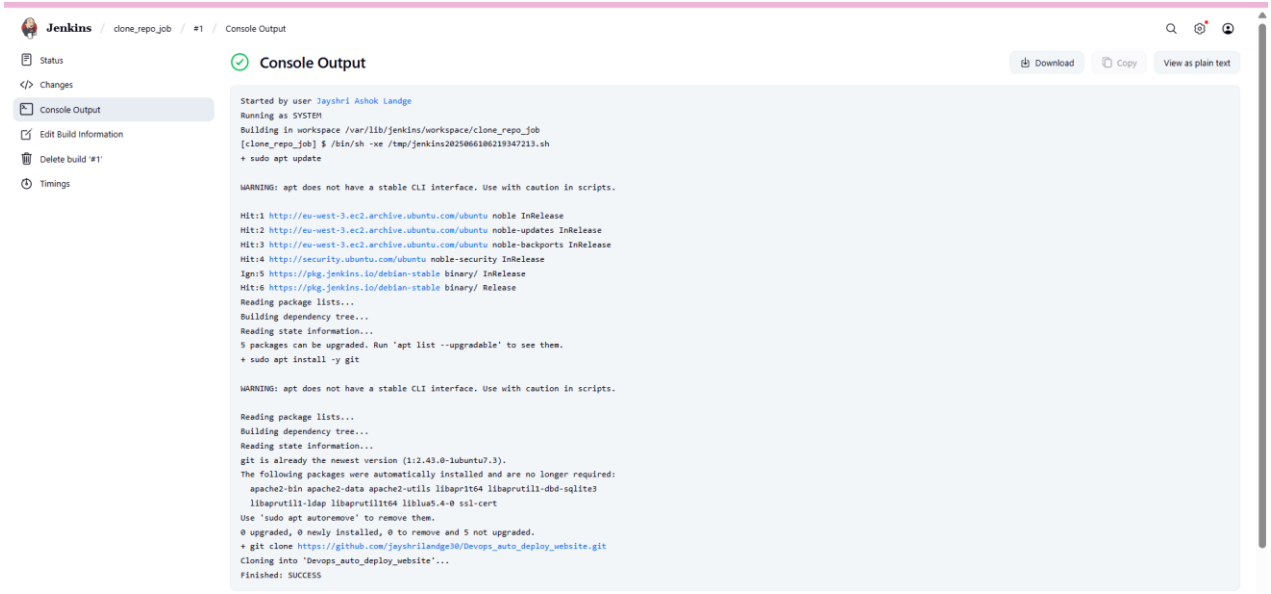
Cloning GitHub Repositories

1 Without Plugin:

Use shell in Jenkins job:

sudo yum install -y git

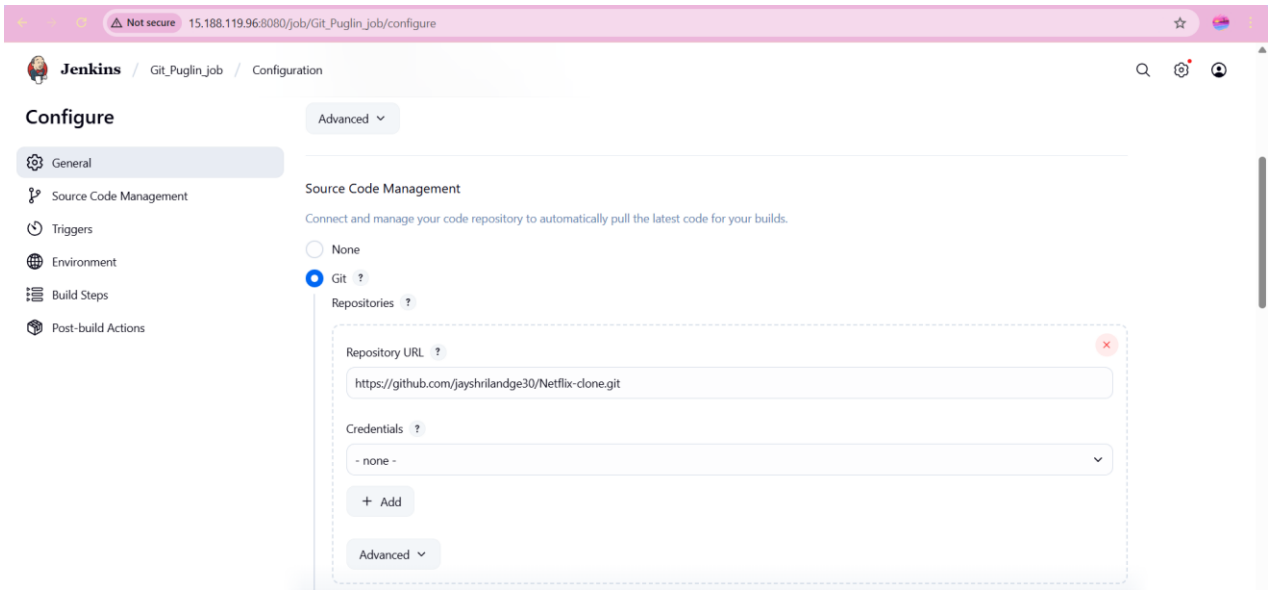
sudo git clone https://github.com/username/repo.git



Screenshot 12: Console Output showing successful git clone

2 With Plugin (Public Repo):

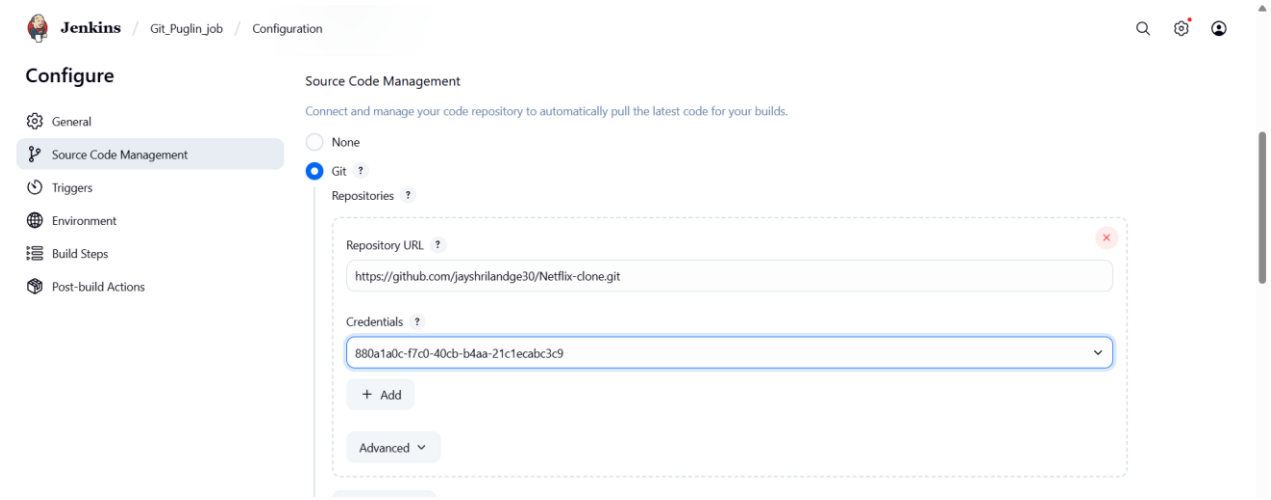
Configure Git in Source Code Management → Git URL → Save → Build



Screenshot 13: Source Code Management configuration

3 With Plugin (Private Repo):

Add Git credentials (username and PAT) → Select from dropdown → Save → Build

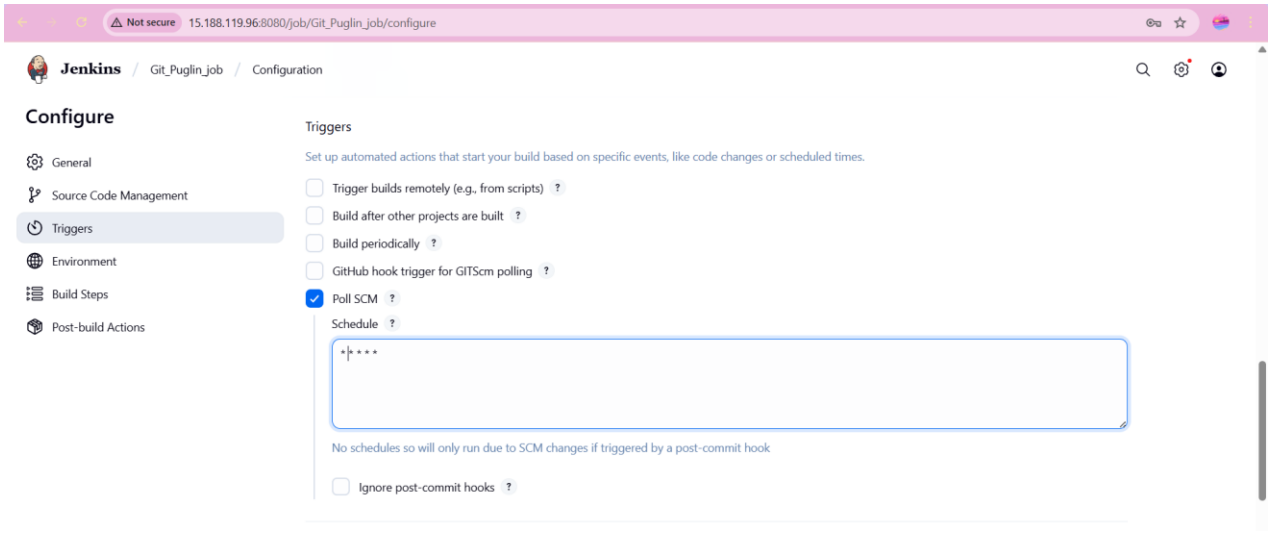


Screenshot 14: Credential configuration for Git in Jenkins

Automating Builds

Poll SCM

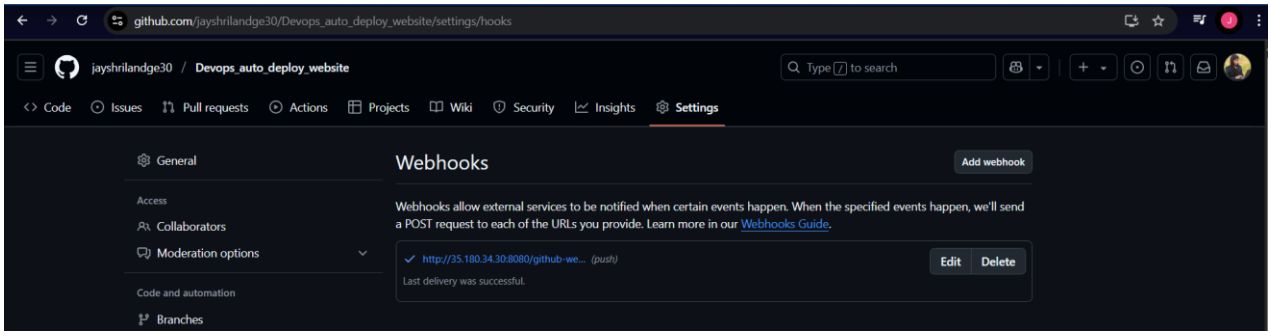
Cron schedule like * * * * * for polling → Jenkins triggers build on change



Screenshot 15: Build Triggers - Poll SCM configuration

Git Webhook

- GitHub → Webhooks → Add Jenkins URL /github-webhook/
- Jenkins → Manage Jenkins → GitHub server → Secret Token

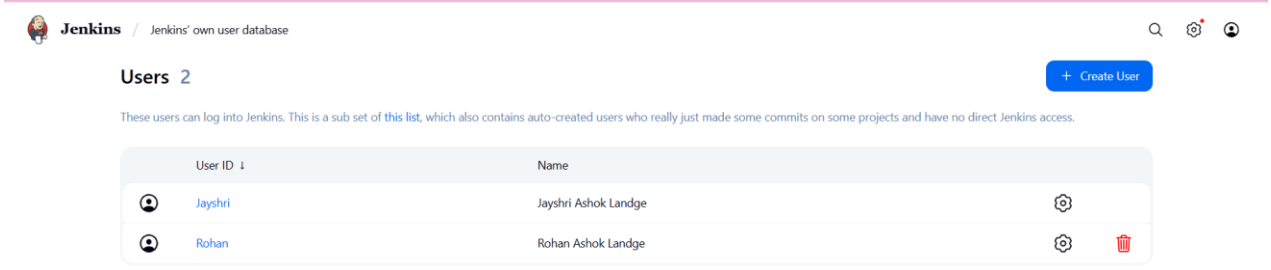


Screenshot 16: Webhook setup in GitHub

User & Access Management

Create User:

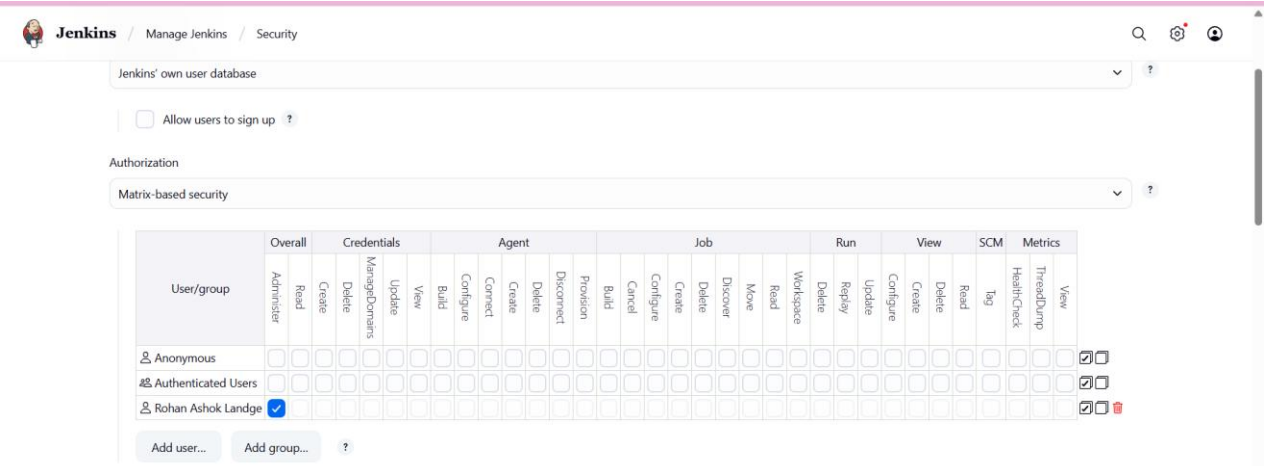
Manage Jenkins → Manage Users → Create User



Screenshot 17: User creation

Configure Permissions:

Matrix-based security → Add user → Assign permissions → Save

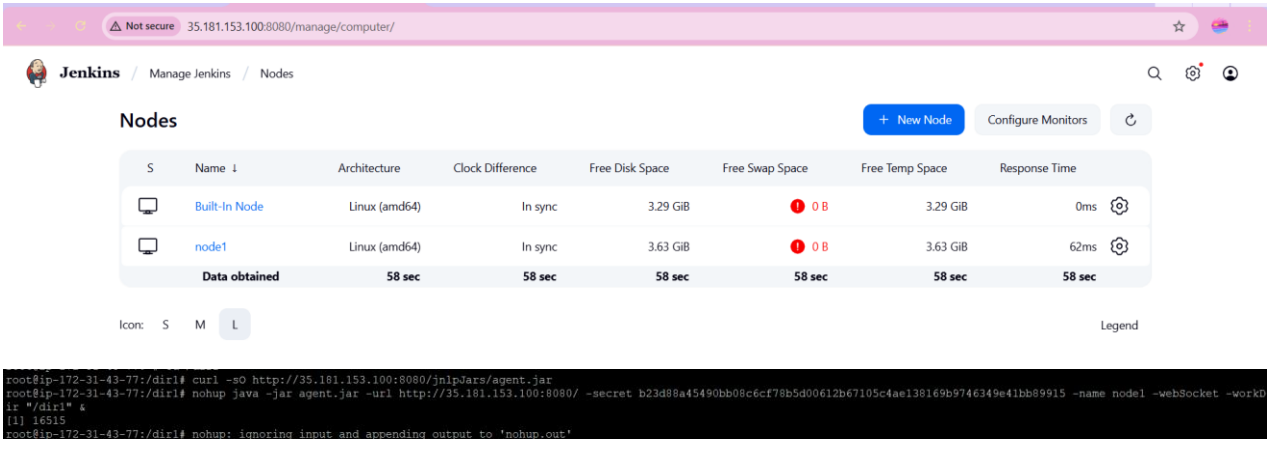


Screenshot 18: Matrix permission panel

Connecting Jenkins Master and Agent

A. Using JNLP:

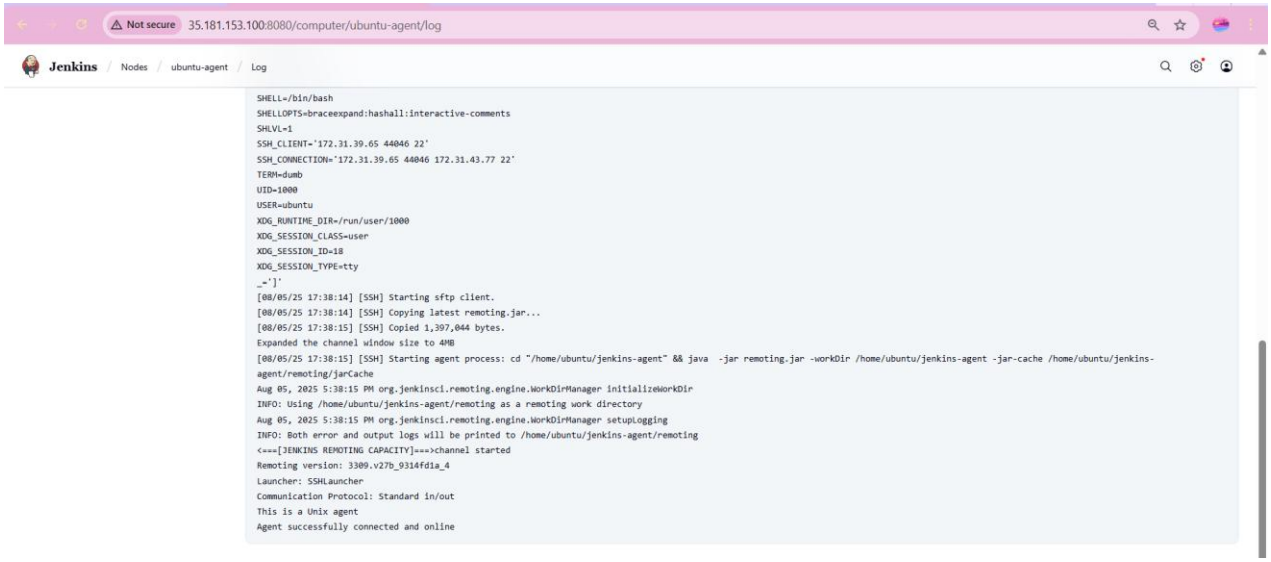
- Launch EC2 (Slave)
- Install Java
- Create workspace: mkdir /dir1
- Jenkins → New Node → Permanent Agent
- Launch using JNLP
- Run agent using nohup command



Screenshot 19: Node creation and JNLP command on Jenkins

B. Using SSH:

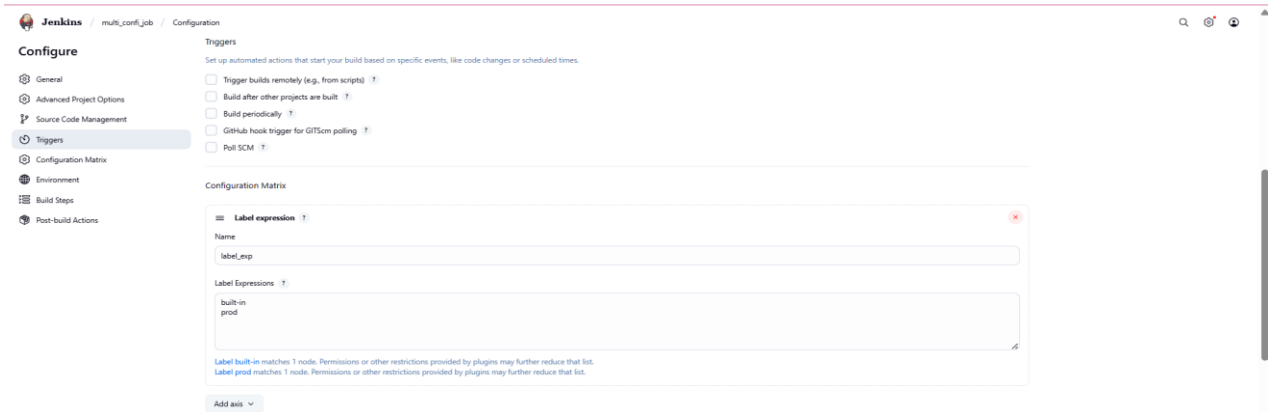
- Launch EC2 instance (Agent)
- Install Java
- Create directory /home/ec2-user/dir2
- Jenkins → New Node → Launch via SSH
- Add credentials (username + PEM key)
- Trust SSH key → Save



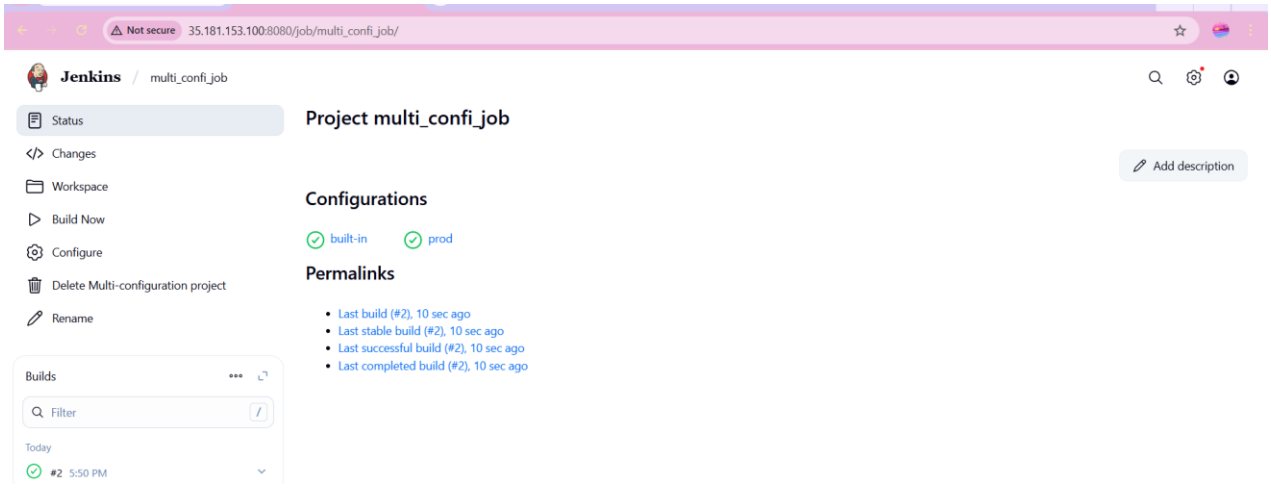
Screenshot 20: SSH Node configuration and status: Connected

Multi-configuration (Matrix) Jobs

- Jenkins → New Item → Multi-configuration project
- Add Axis: Label Expression → Values: built-in, instance1
- Configure build script → Run on both nodes in parallel



Screenshot 21: Axis configuration



Screenshot 22: Build results on different nodes

Jenkins Pipeline Scripts (Groovy)

Basic Pipeline:

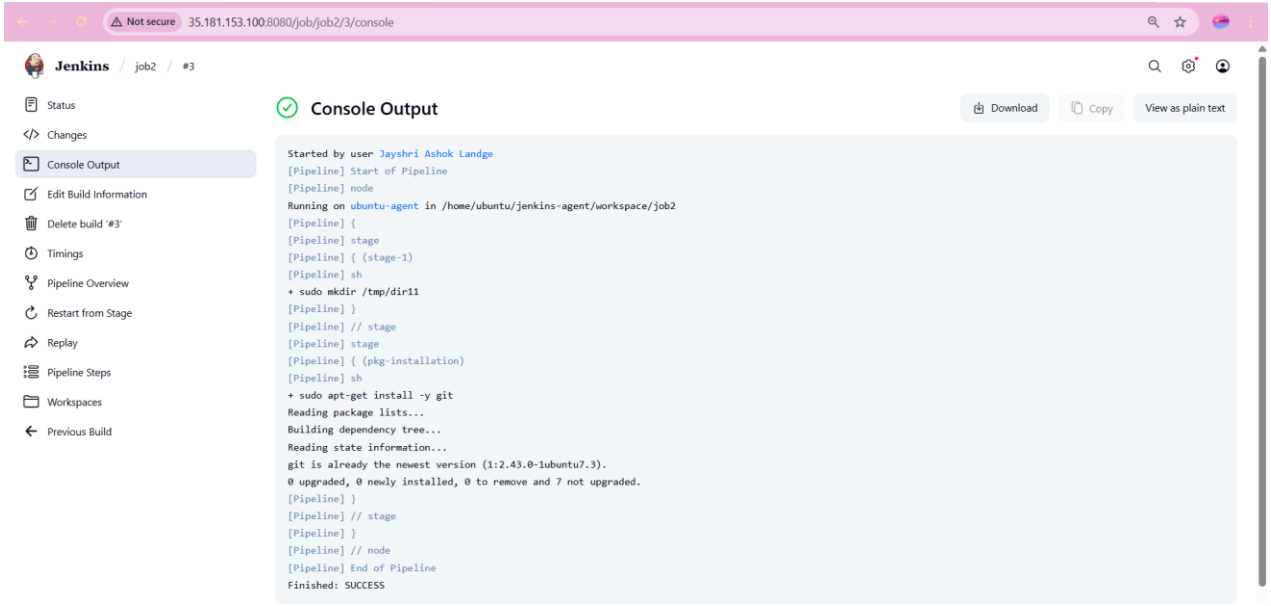
```
pipeline {
 agent any
 stages {
  stage('Build') { steps { echo 'Building...' } }
  stage('Test') { steps { echo 'Testing...' } }
  stage('Deploy') { steps { echo 'Deploying...' } }
 }
}
```

Real-Time Tasks:

- User creation: sudo useradd jayshri

Screenshot 23: Console outputs of pipeline jobs (user creation)

- Directory operations: mkdir, touch, yum install, build other jobs



Screenshot 24: Console outputs of pipeline jobs (mkdir, httpd install)

Pipeline with Agent-Specific Workspaces

Run on Master:

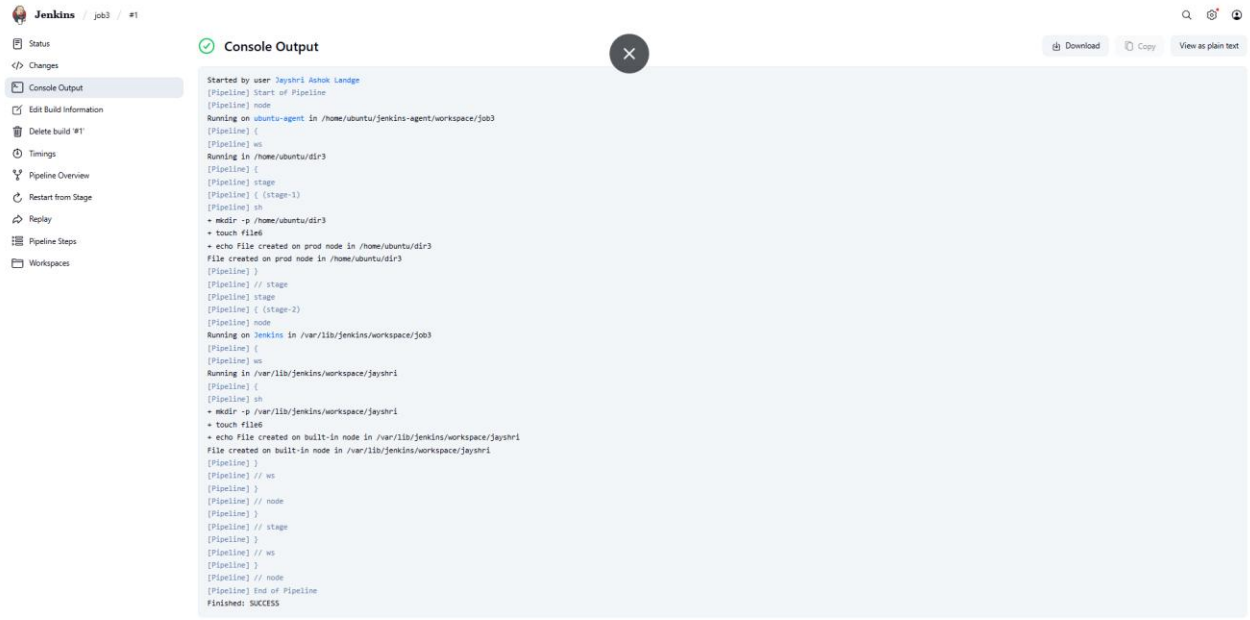
```
agent { label 'built-in' }
```

Run on Custom Workspace:

```
agent {
  node {
    label 'prod'
    customWorkspace '/home/ec2-user/dir3'
  }
}
```

Hybrid Pipeline (Stage-1 on Agent, Stage-2 on Master):

```
pipeline {
  agent {
    node {
      label "prod"
      customWorkspace "/home/ec2-user/dir3"
    }
  }
  stages {
    stage ("stage-1") {
      steps { sh "sudo touch file6" }
    }
    stage ("stage-2") {
      agent {
        node {
          label "built-in"
          customWorkspace "jayshri"
        }
      }
      steps { sh "sudo touch file6" }
    }
  }
}
```



Screenshot 25: Pipeline build output for hybrid workspace setup

Conclusion

Through this practical exploration of Jenkins, I've:

- Installed and configured Jenkins from scratch
- Cloned repositories (public & private)
- Created freestyle and matrix jobs
- Integrated Jenkins with GitHub using both polling and webhooks
- Built pipelines that span across master and agent nodes with custom workspaces

This journey reinforced my knowledge in **DevOps automation**, **pipeline scripting**, **CI/CD practices**, and **agent node management**, providing a strong foundation for tackling real-world DevOps projects.