**2) Setup a jenkins CICD pipeline using Declarative pipeline using feature-1.1 branch.**

[**https://github.com/betawins/sabear\_simplecutomerapp/tree/feature-1.1**](https://github.com/betawins/sabear_simplecutomerapp/tree/feature-1.1)

**stages:**

**1) Git Clone**

**2) Sonarqube Integration**

**3) Maven Compilation**

**4) Nexus Artifactory**

**5) Slack Notification**

**6) Deploy On tomcat**

# Jenkins Setup for Your Jenkinsfile

### 1. ****Install Required Plugins****

Go to **Manage Jenkins → Plugins** and ensure these are installed:

* **Pipeline** (Declarative + Scripted)
* **Pipeline Utility Steps** (for readMavenPom, findFiles)
* **Nexus Artifact Uploader**
* **SonarQube Scanner for Jenkins**
* **Slack Notification**

### 2. ****Configure Global Tools****

**Manage Jenkins → Global Tool Configuration**

* **Maven:**
  + Name: MVN\_HOME (matches your Jenkinsfile)
  + Install automatically or point to /usr/share/maven
* **SonarQube Scanner:**
  + Name: sonar\_scanner (matches your Jenkinsfile)
  + Provide installation path or let Jenkins auto-install

### 3. ****Add Server Integrations****

**a) SonarQube**

* Go to **Manage Jenkins → Configure System → SonarQube servers**
  + Name: sonarqube-server (matches your Jenkinsfile)
  + Add Server URL (e.g., http://<sonar-host>:9000)
  + Add authentication token (create in SonarQube UI → My Account → Security → Tokens)

**b) Slack**

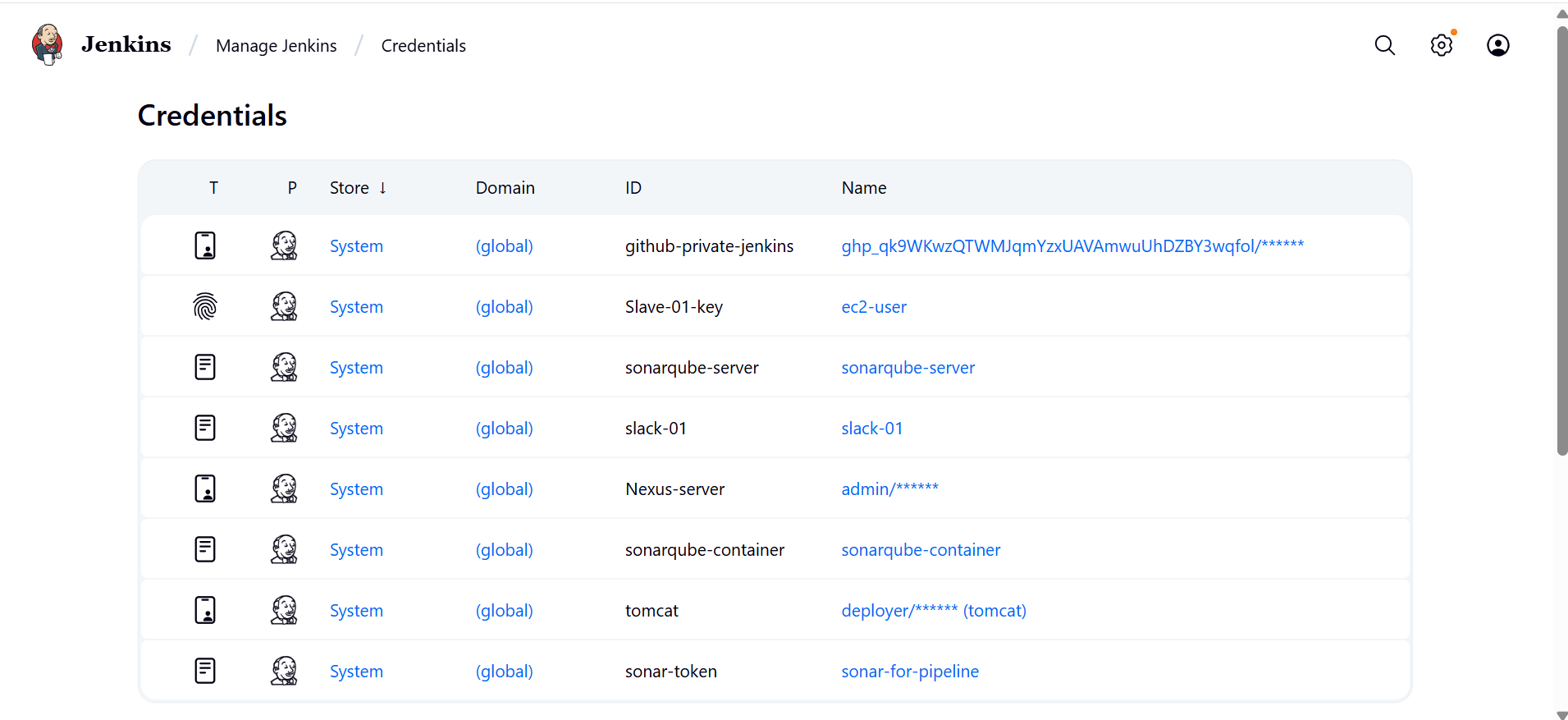
* Go to **Manage Jenkins → Configure System → Slack**
  + Workspace: your Slack team domain
  + Integration token (from Slack App)
  + Default channel: #jenkins-integration
* Test connection to ensure Jenkins can post.

### 4. ****Set Credentials****

**Manage Jenkins → Credentials → System → Global credentials**

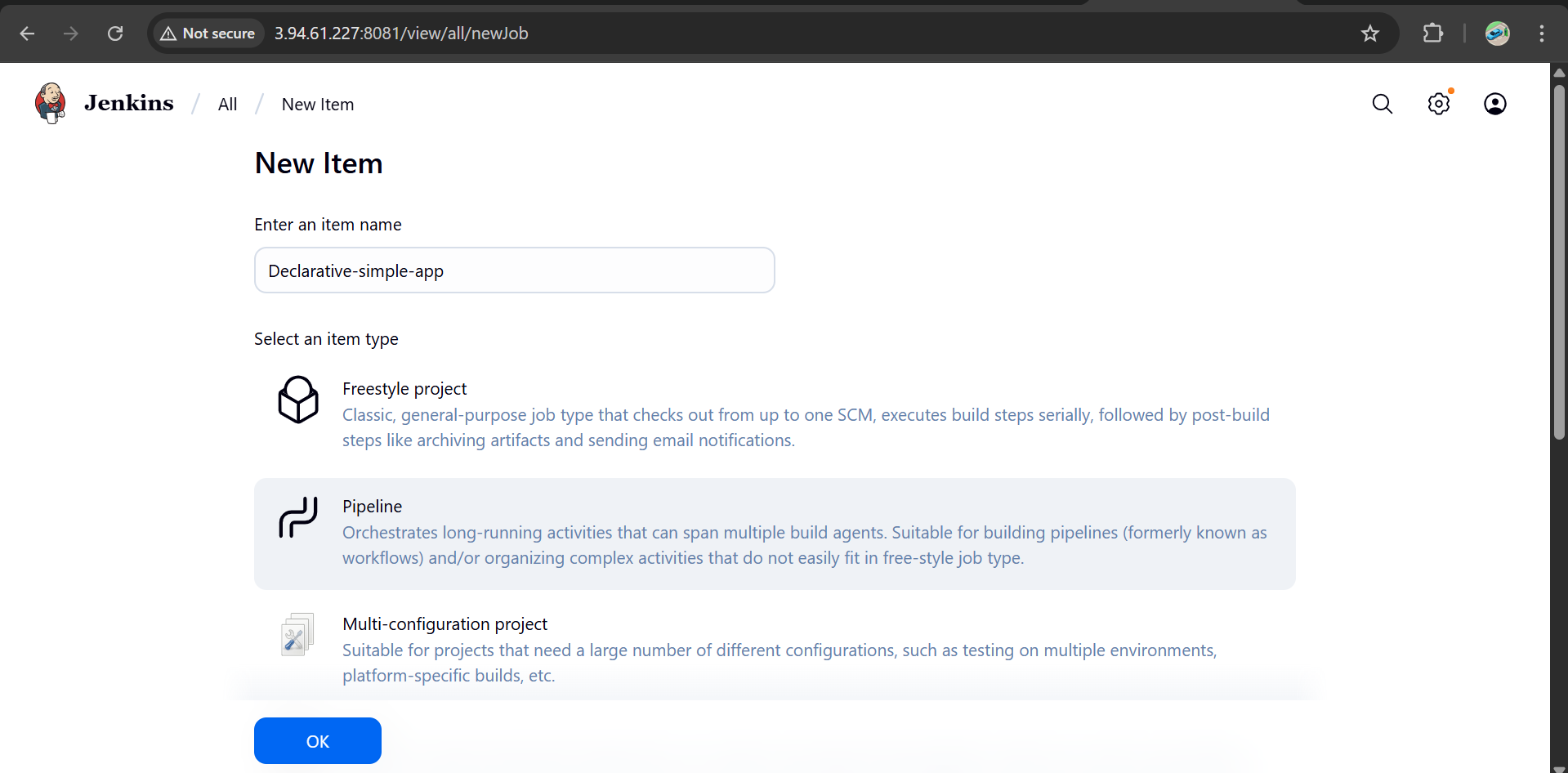
Add the following IDs (must match your Jenkinsfile):

* **Nexus-server**
  + Type: Username with password
  + Username: Nexus repo user
  + Password: Nexus repo password
* **tomcat**
  + Type: Username with password
  + Username: Tomcat manager user
  + Password: Tomcat manager password



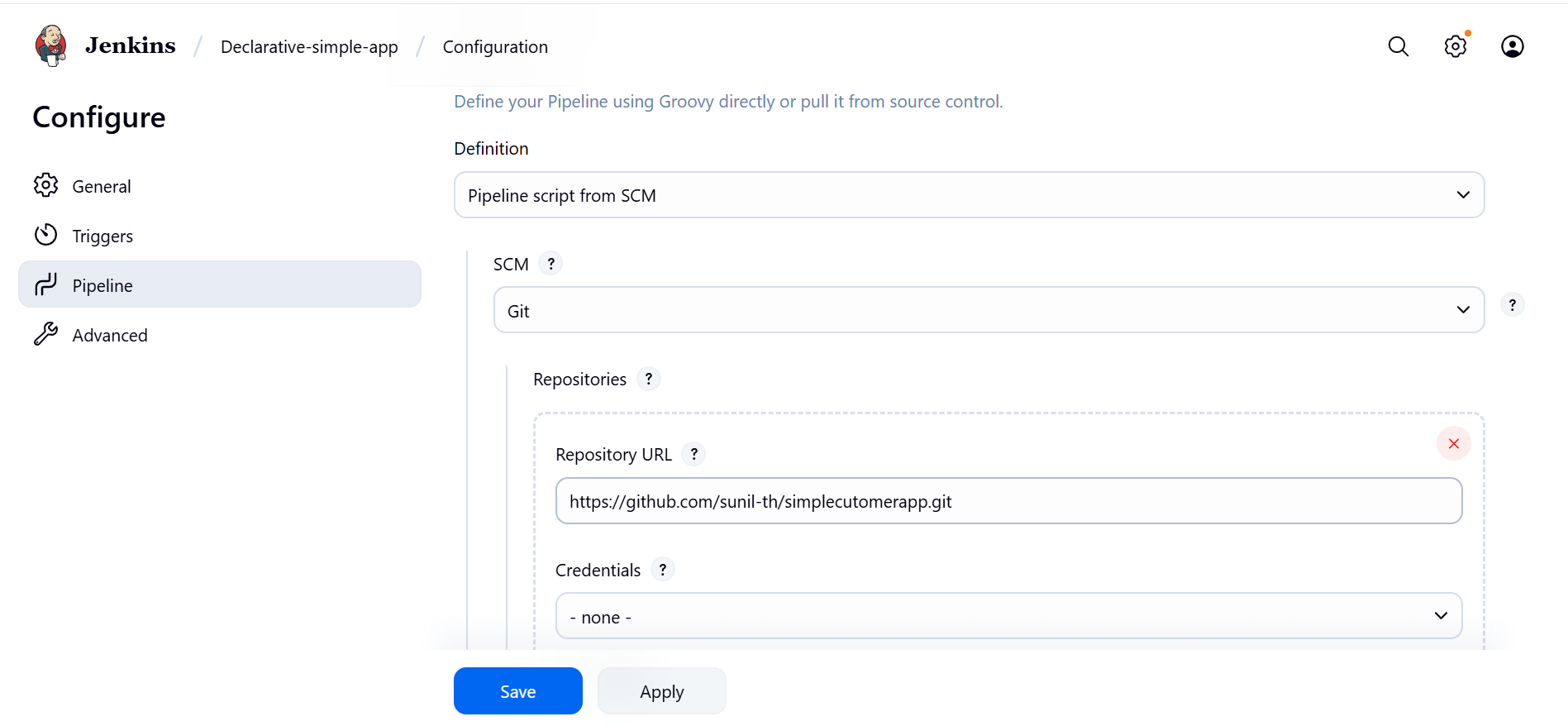
**5.Create a New Pipeline Job**

1. Open Jenkins dashboard → **New Item**.
2. Enter a job name (e.g., simple-customerapp-pipeline).
3. Select **Pipeline** → Click **OK**.

****

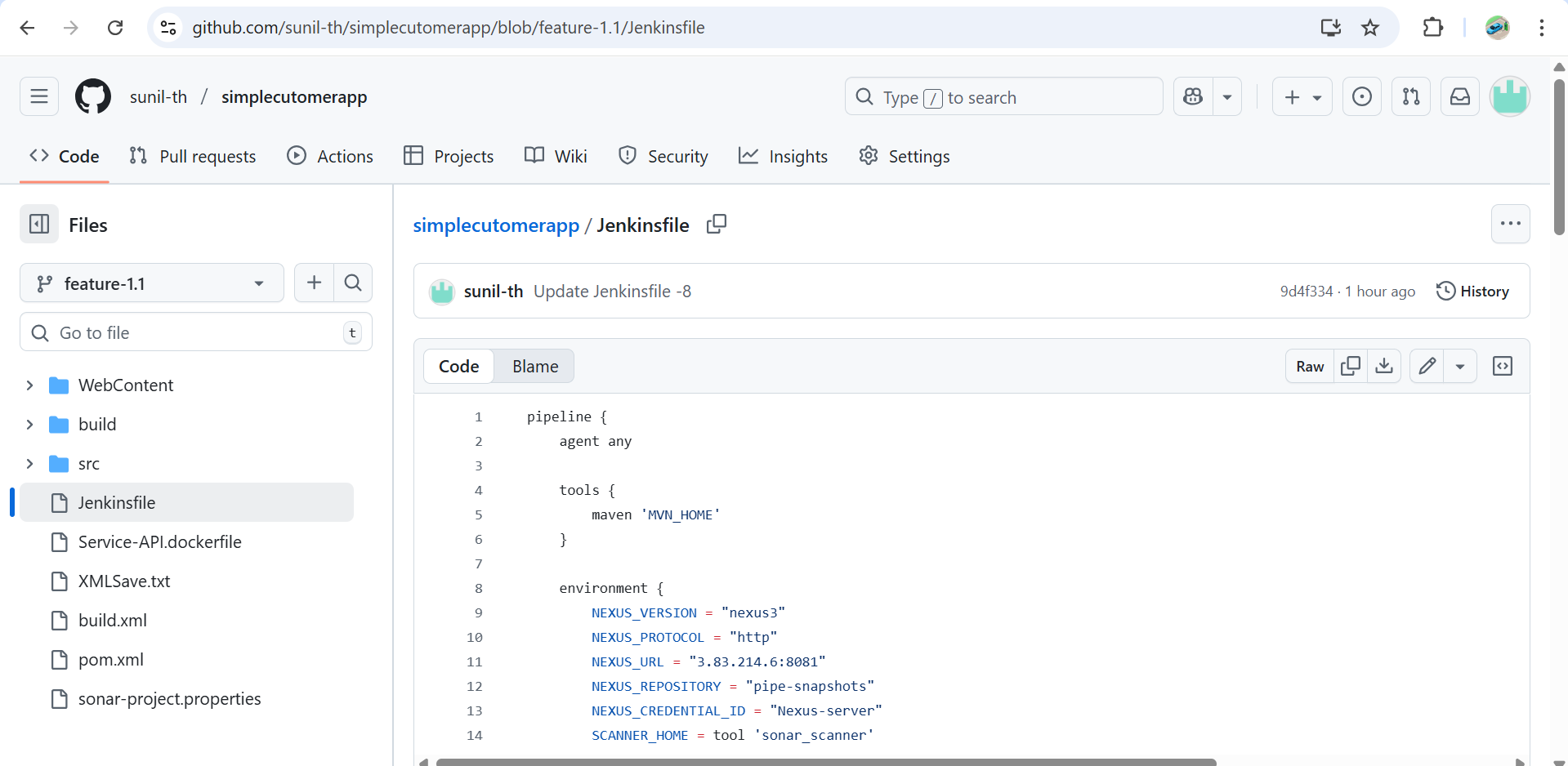
**Pipeline from SCM**

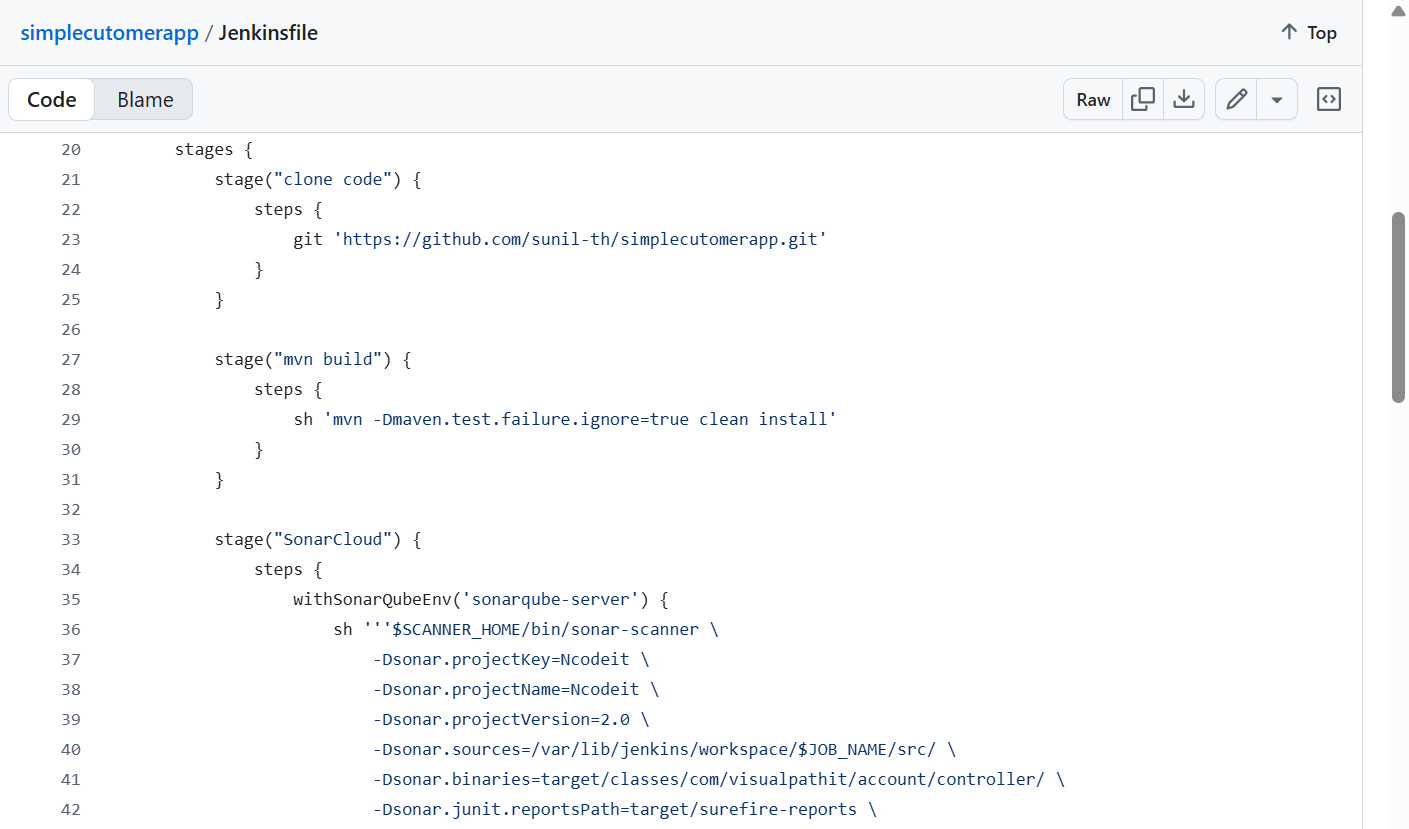
* **SCM:** Git
* **Repository URL:** https://github.com/betawins/sabear\_simplecutomerapp.git
* **Branch Specifier:** \*/feature-1.1

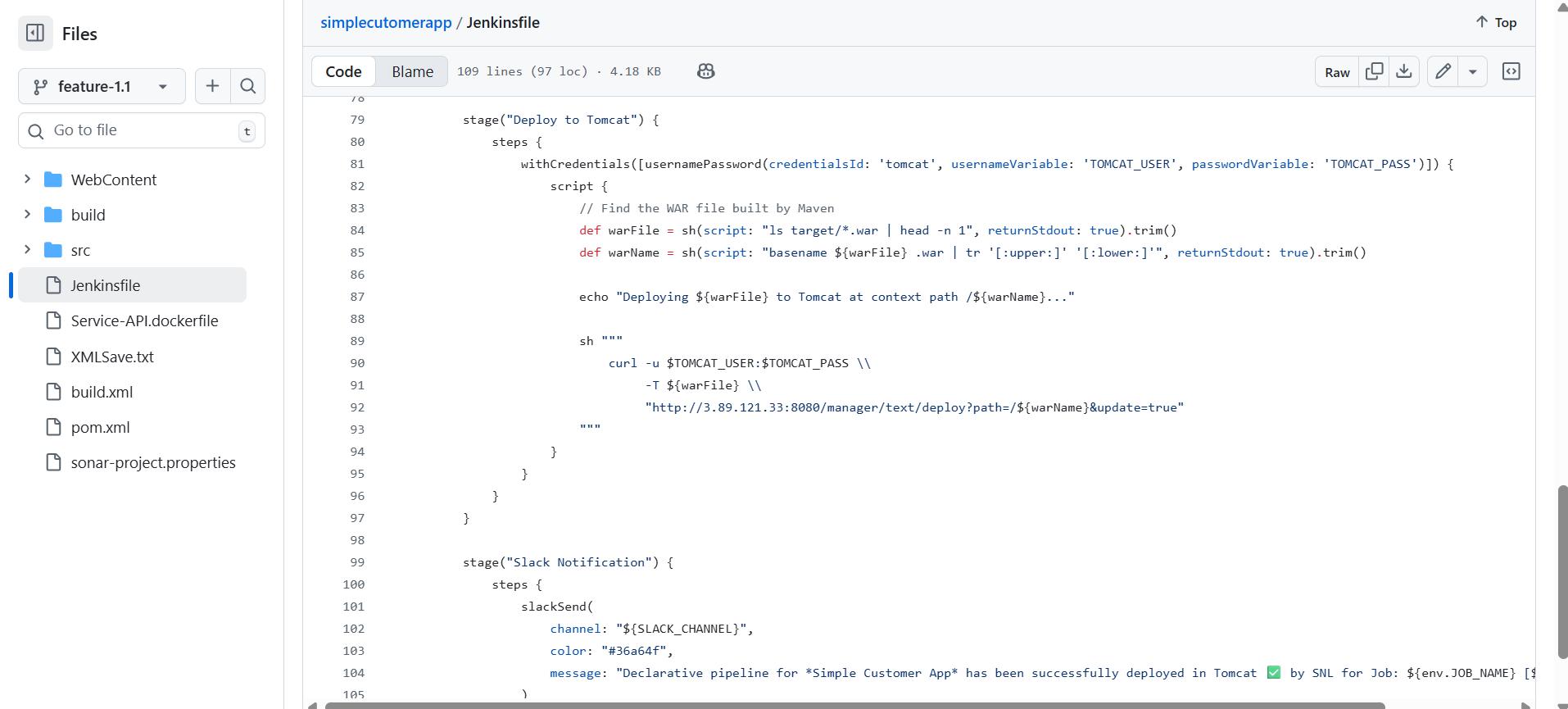


**Drop this Jenkinsfile into the repo**

Stages: **Git Clone → SonarQube → Maven Compile → Nexus Upload → Deploy on Tomcat → Slack**

****

****

****

# How the Stages Work After Setup

1. **Clone Code**
   * Uses Jenkins Git plugin, no creds required since repo is public.
2. **Maven Build**
   * Compiles + packages the WAR, ignores test failures.
3. **SonarQube Scan**
   * Runs sonar-scanner CLI with your provided parameters.
   * Uses the sonarqube-server config and scanner tool.
4. **Publish to Nexus**
   * Reads pom.xml → finds WAR in target/
   * Uploads artifact + POM to Nexus repo pipe-snapshots at http://3.83.214.6:8081
5. **Deploy to Tomcat**
   * Grabs WAR file, extracts context name from filename,
   * Deploys to Tomcat Manager (http://3.89.121.33:8080/manager/text/deploy)
6. **Slack Notification**
   * Posts green success message with Job & Build info in #jenkins-integration

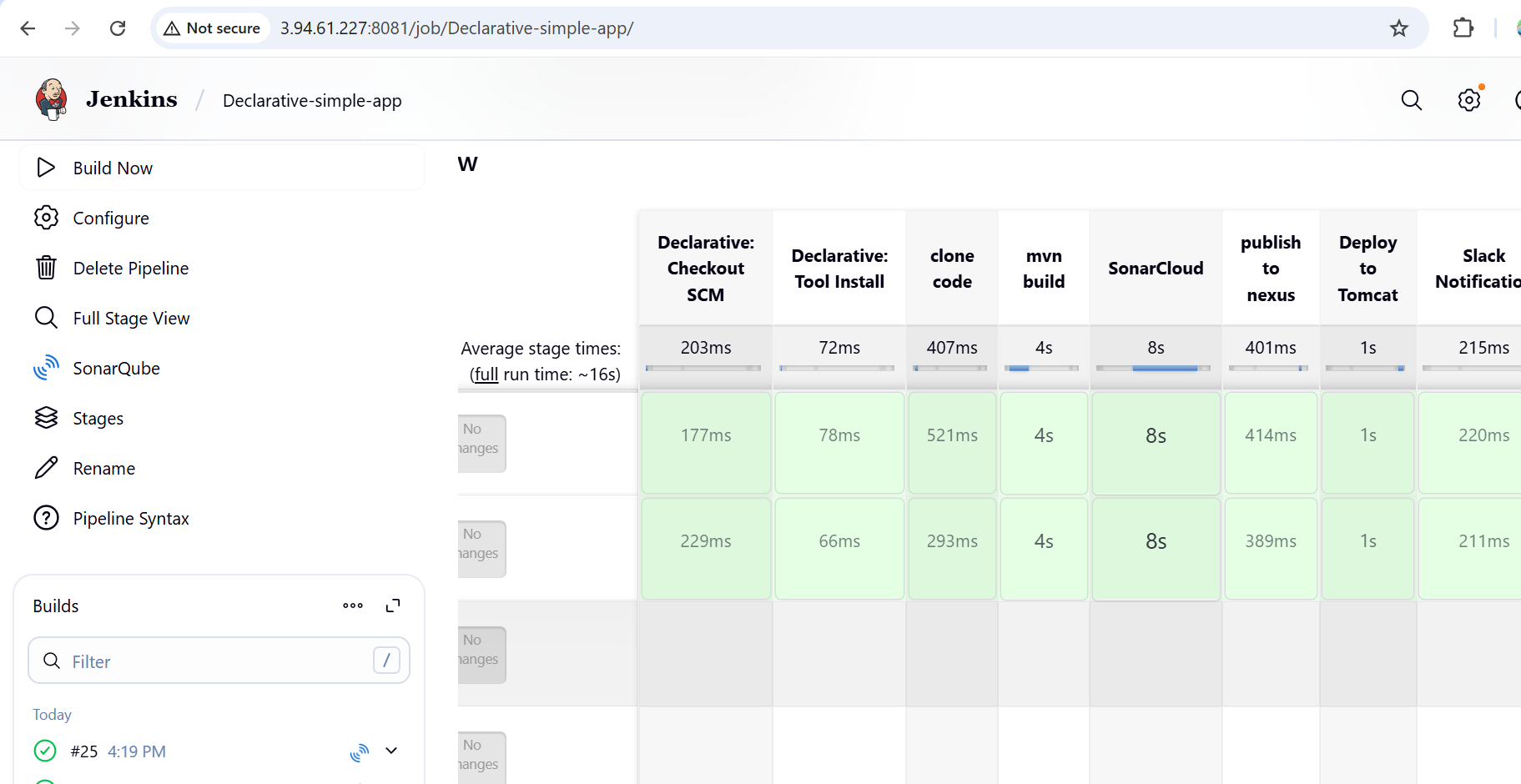
# Validation Checklist

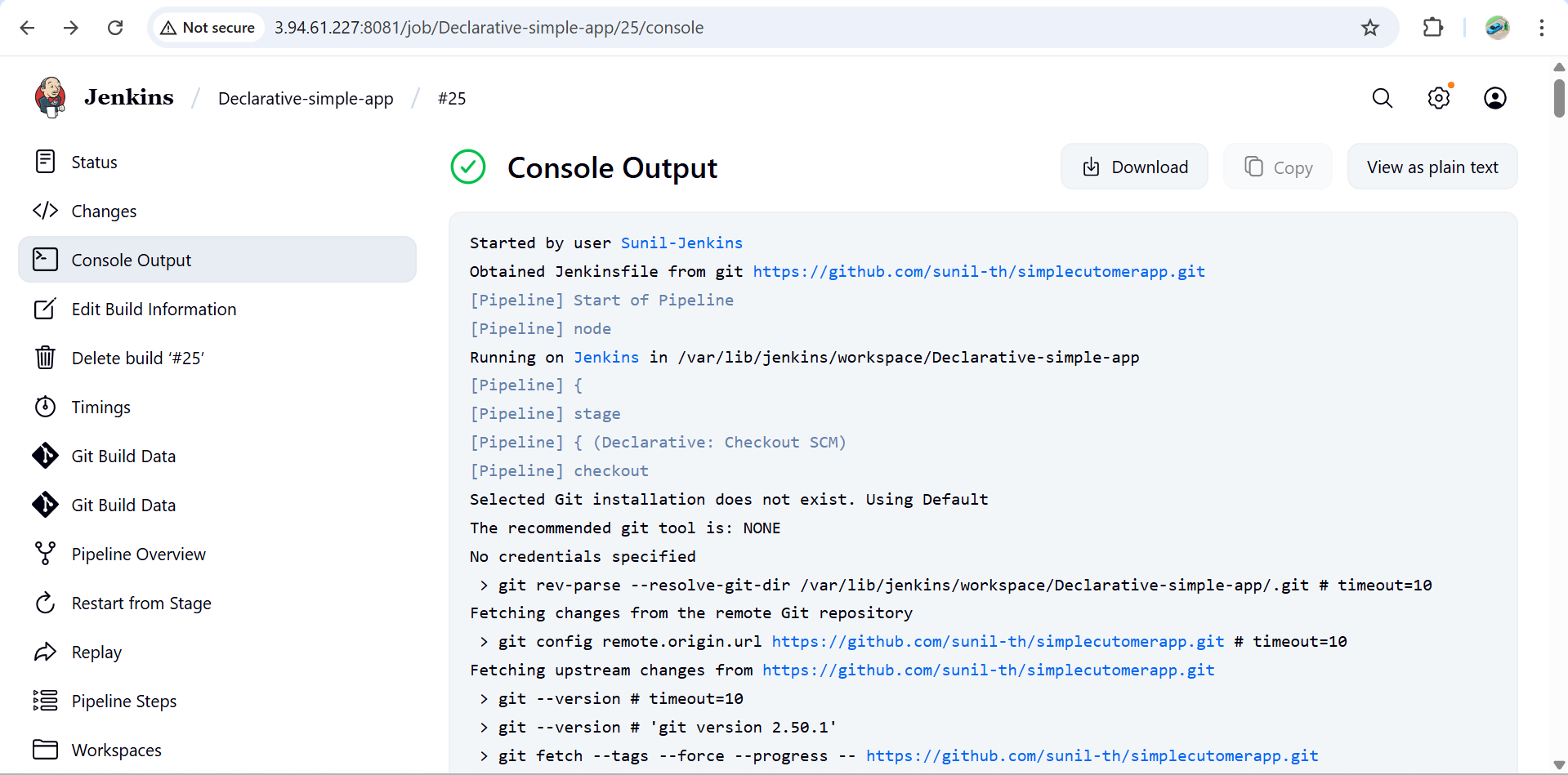
Before running:

* Can Jenkins reach **SonarQube**, **Nexus**, and **Tomcat** (firewall rules)?
* Does the **Tomcat Manager app** run at http://3.89.121.33:8080/manager/html?
* Does nexusArtifactUploader show up in Jenkins pipeline syntax generator (proof plugin installed)?
* Run mvn clean install locally to ensure it builds a WAR in target/.

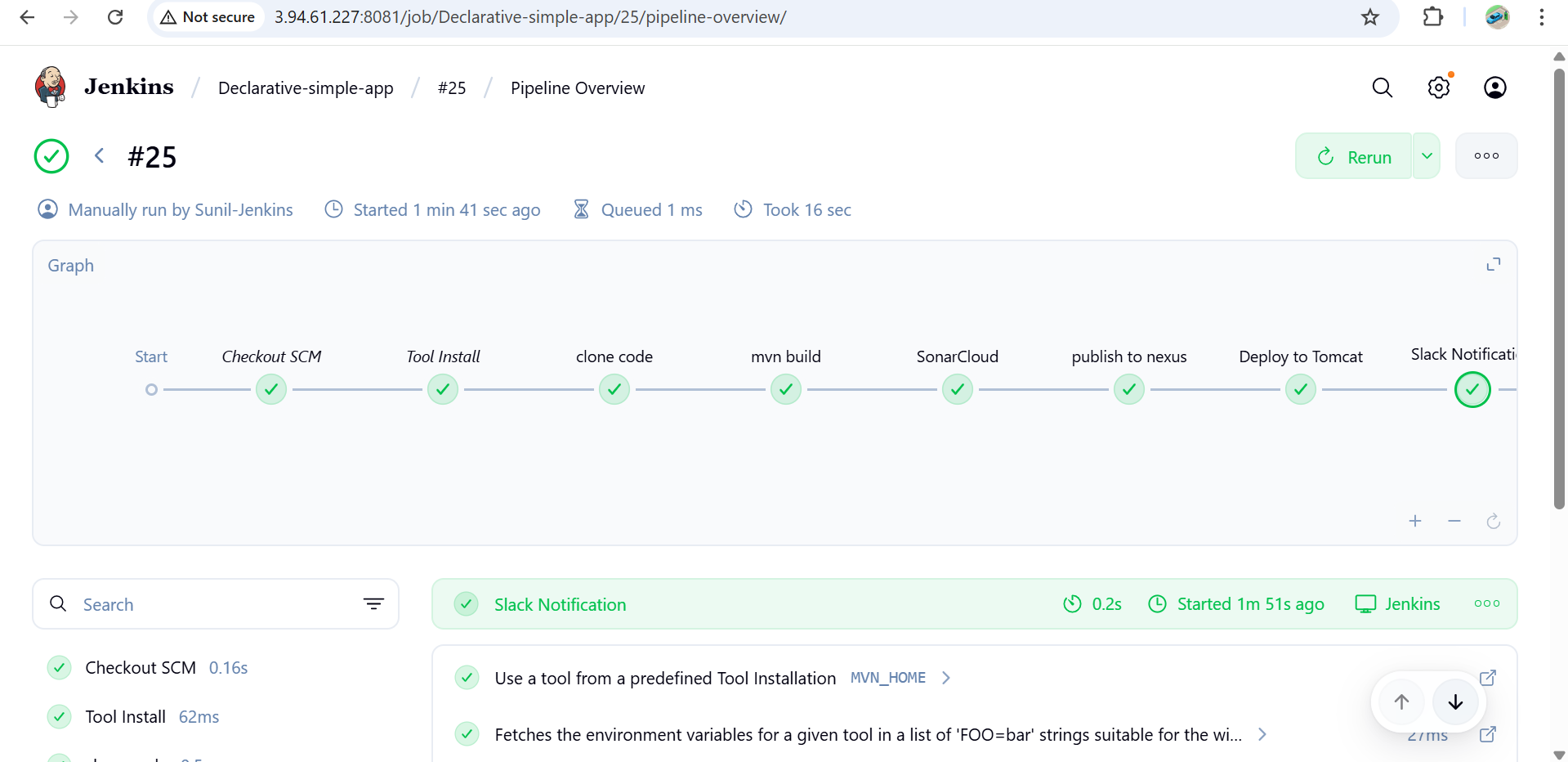
#### ****Save and Build****

* Click **Save**.
* Run the job → Click **Build Now**.

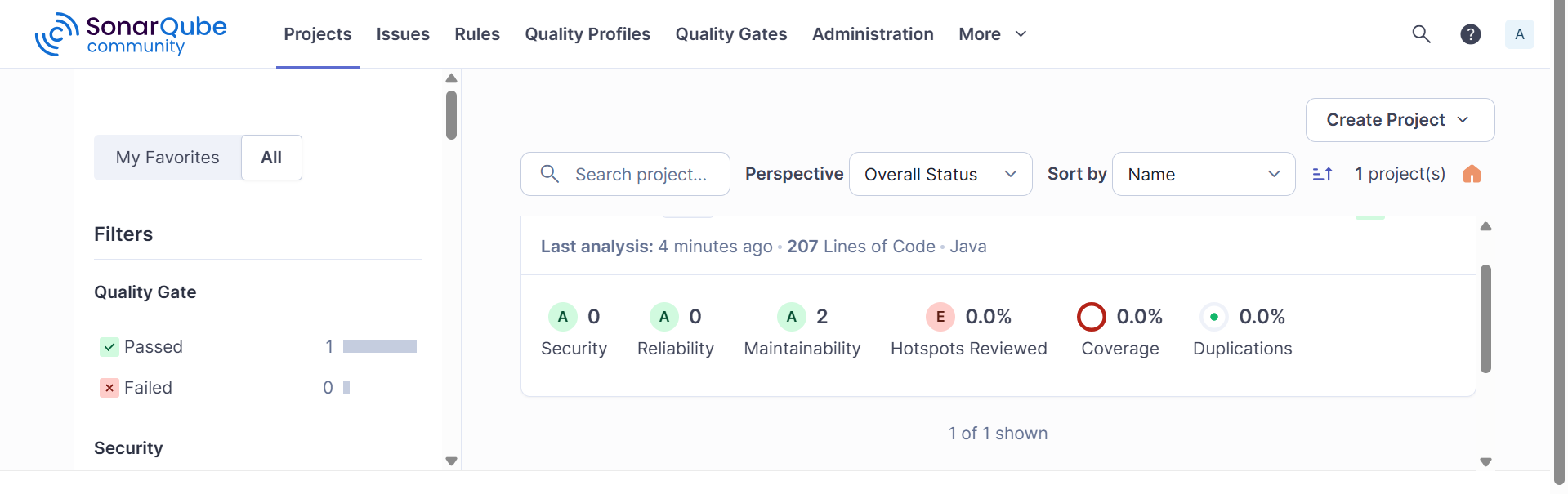
****

****

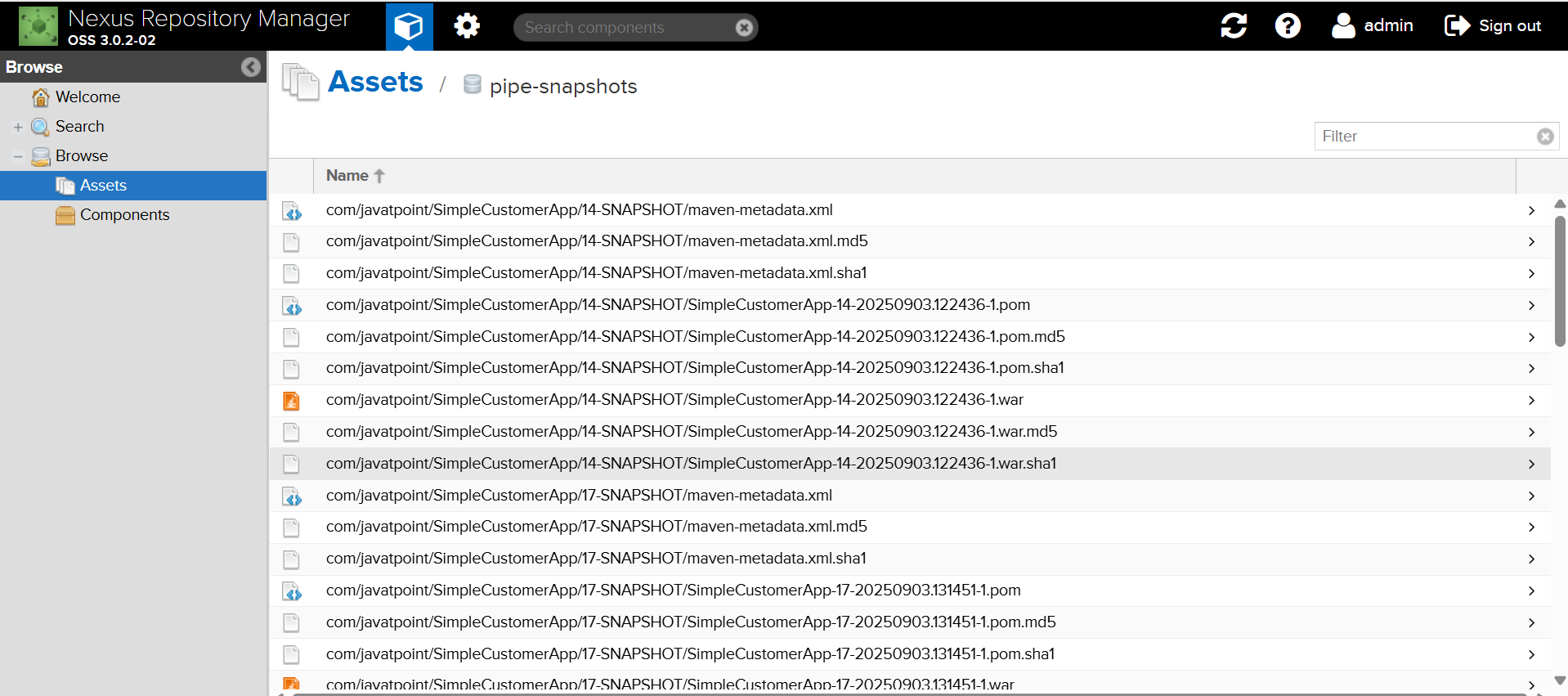
**Pipeline Flow**

****

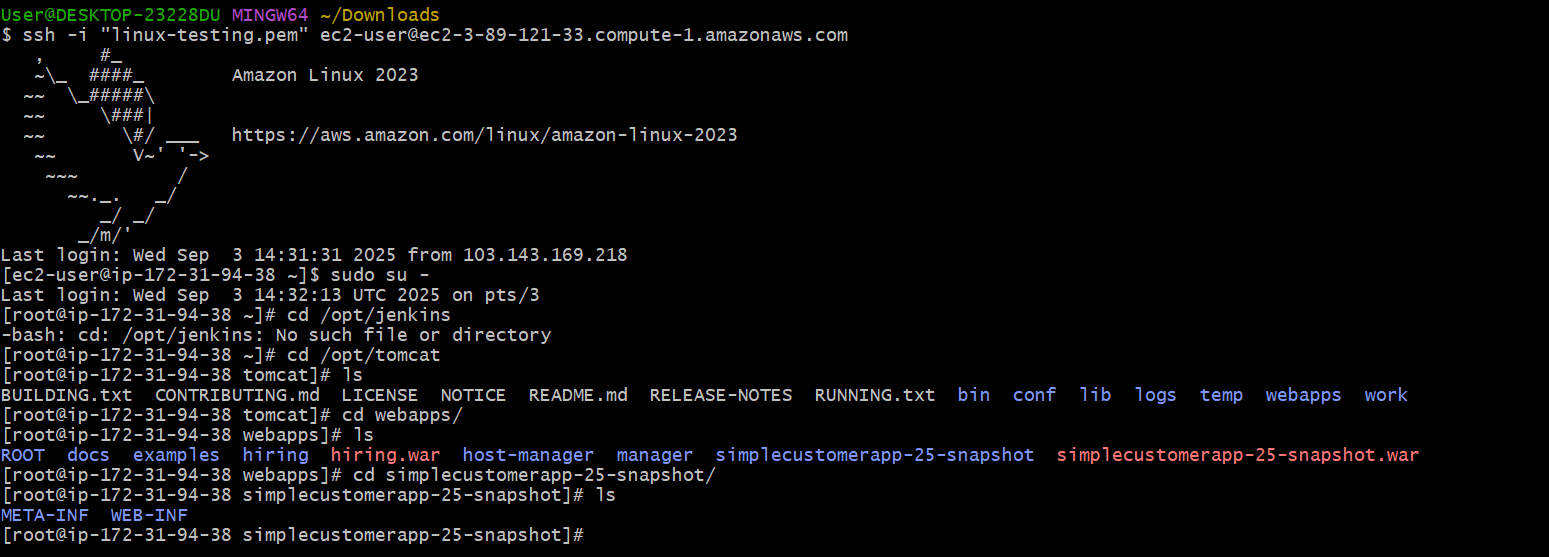
**SonarQube**

****

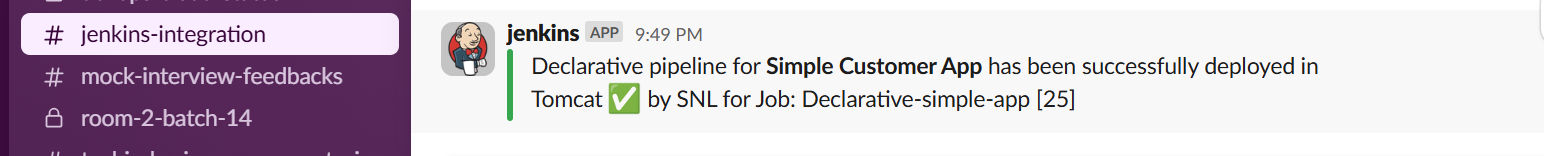
**Nexus**

****

**Tomcat Deploy**

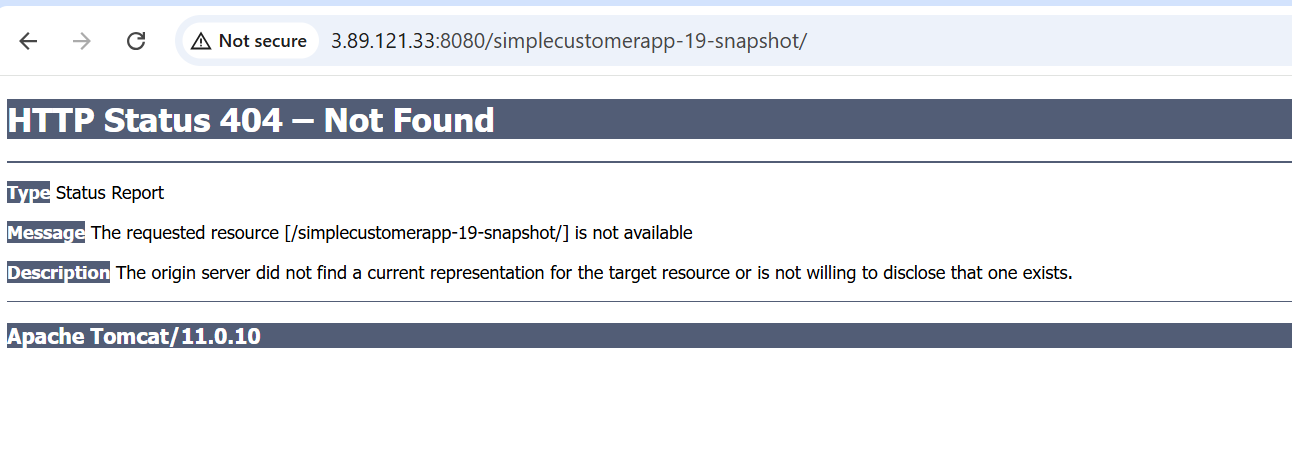
****

**Check the slack**

****

Check on the web with the public ip and the deployed file name

<http://3.89.121.33:8080/simplecustomerapp-19-snapshot/>



**3) Setup a jenkins CICD pipeline using Scripted pipeline using feature-1.1 branch.**

**https://github.com/betawins/sabear\_simplecutomerapp/tree/feature-1.1**

**stages:**

**1) Git Clone**

**2) Sonarqube Integration**

**3) Maven Compilation**

**4) Nexus Artifactory**

**5) Slack Notification**

**6) Deploy On tomcat**

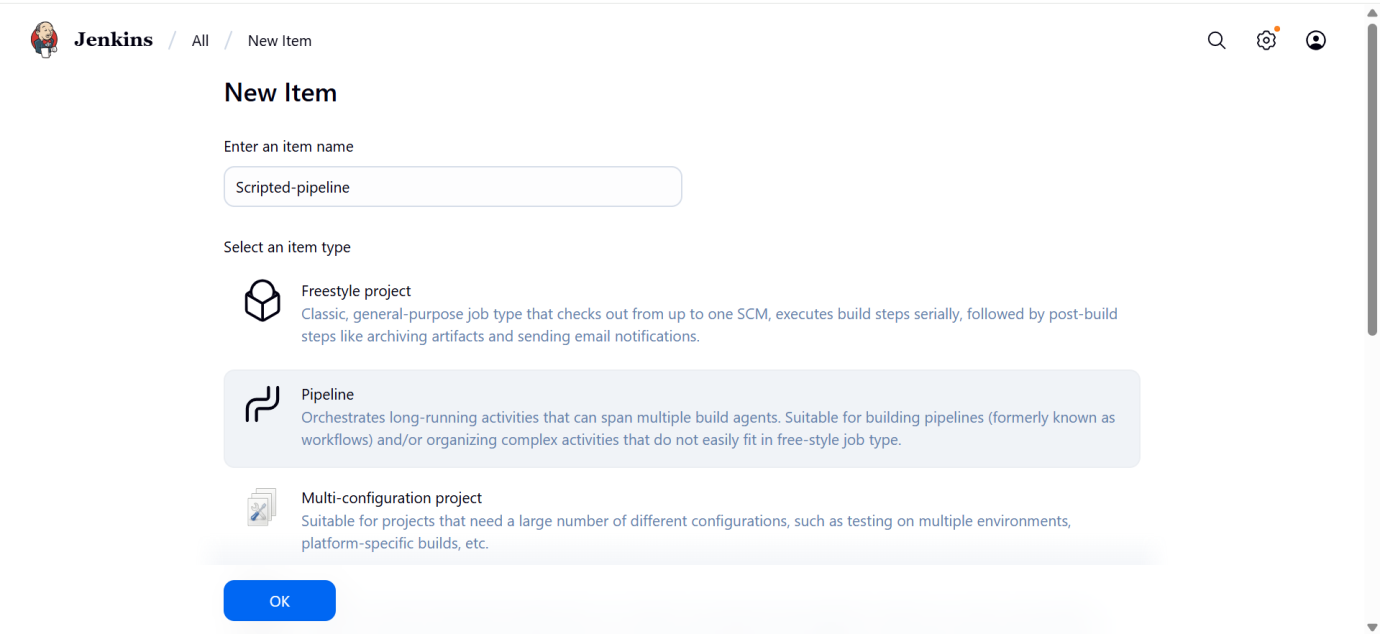
### Steps to Create a Scripted Pipeline in Jenkins

#### ****1. Log in to Jenkins****

* Open Jenkins in your browser (http://<jenkins-server>:8080/).
* Log in with your credentials.

#### ****2. Create a New Pipeline Job****

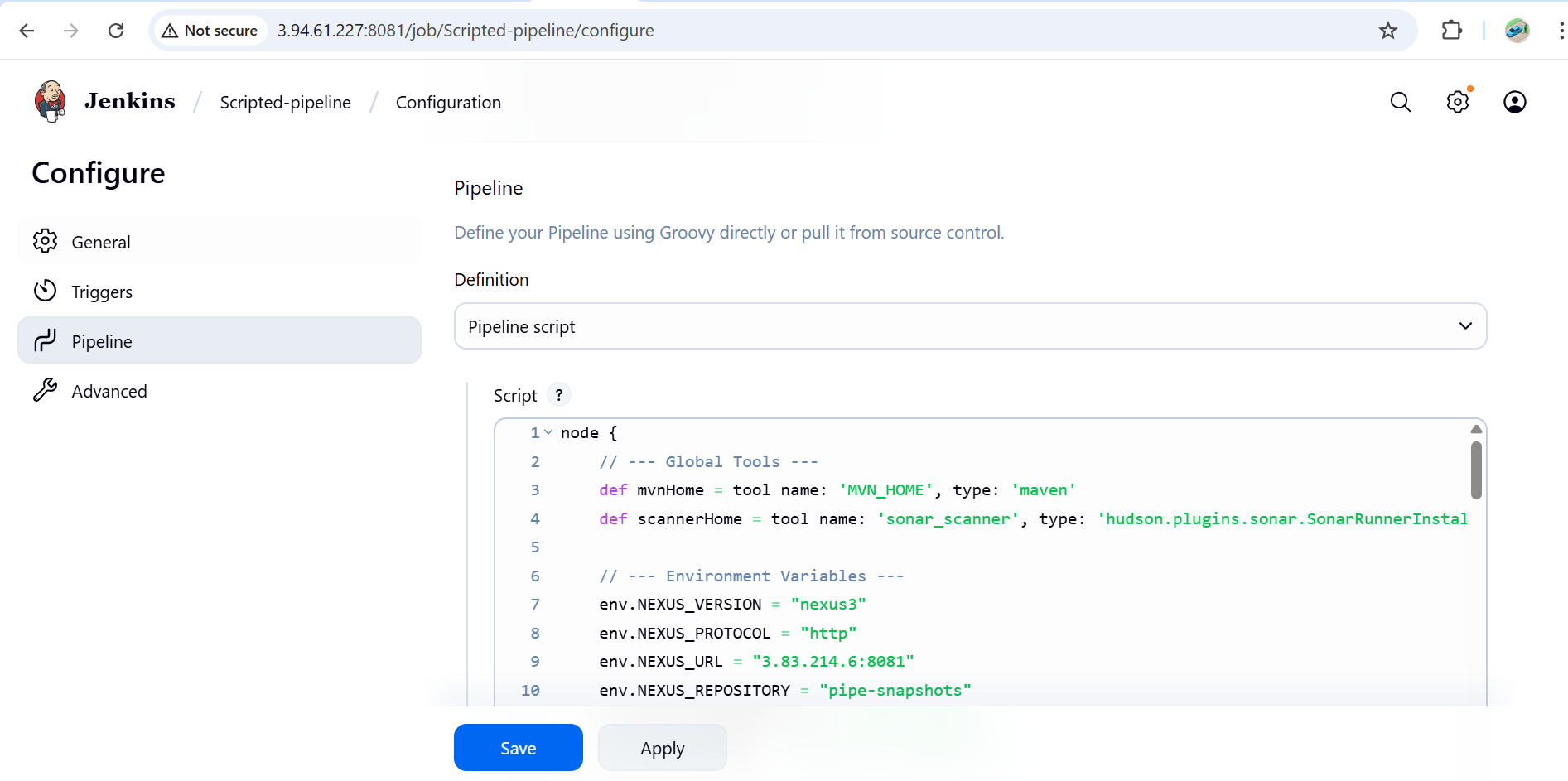
* From Jenkins Dashboard → Click **“New Item”**.
* Enter a **name** for your job (e.g., My-Scripted-Pipeline).
* Select **Pipeline** and click **OK**.



#### ****3. Configure Source Code Management (SCM)****

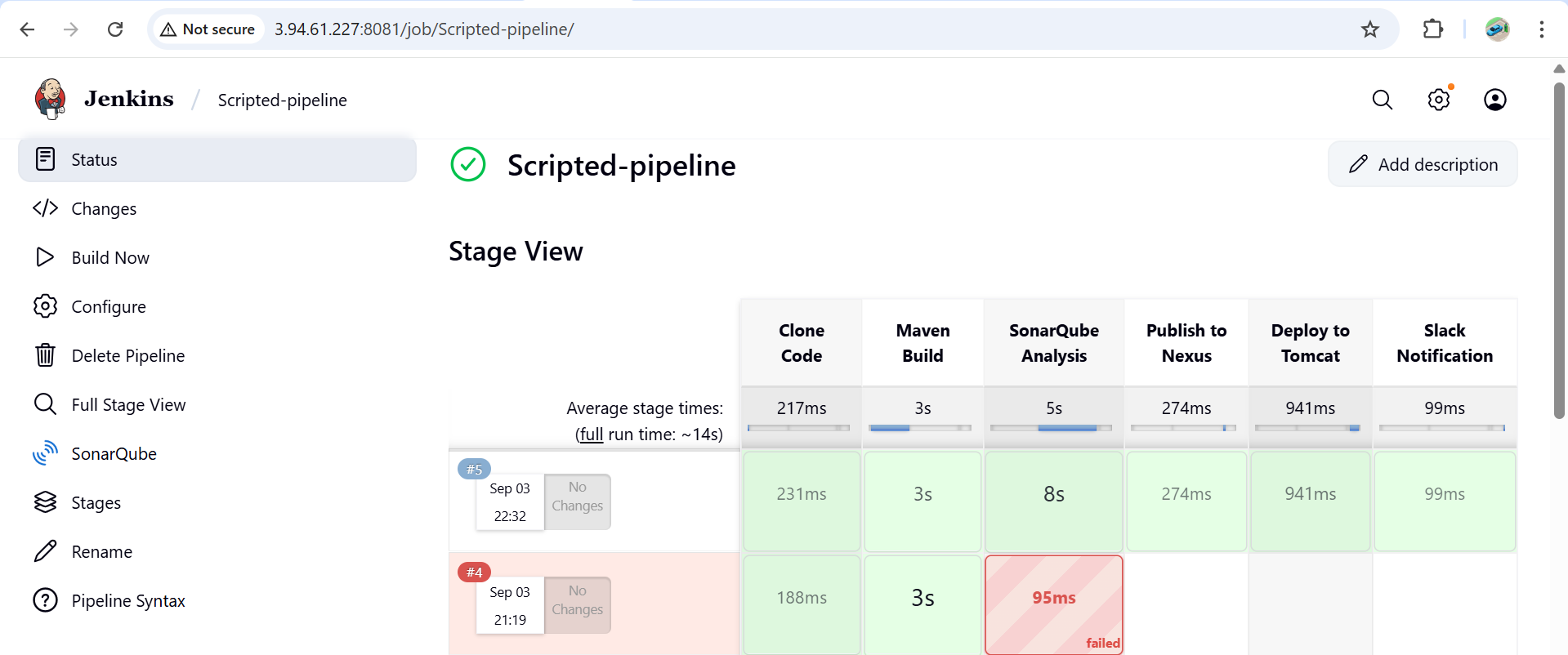
* In the job configuration:
  + Go to **Pipeline → Definition**.
  + Choose one of the two approaches:
    - **Pipeline script from SCM** (if your Jenkinsfile is already in your repo).
    - **Pipeline script** (if you want to write or paste the Scripted Pipeline directly in Jenkins).

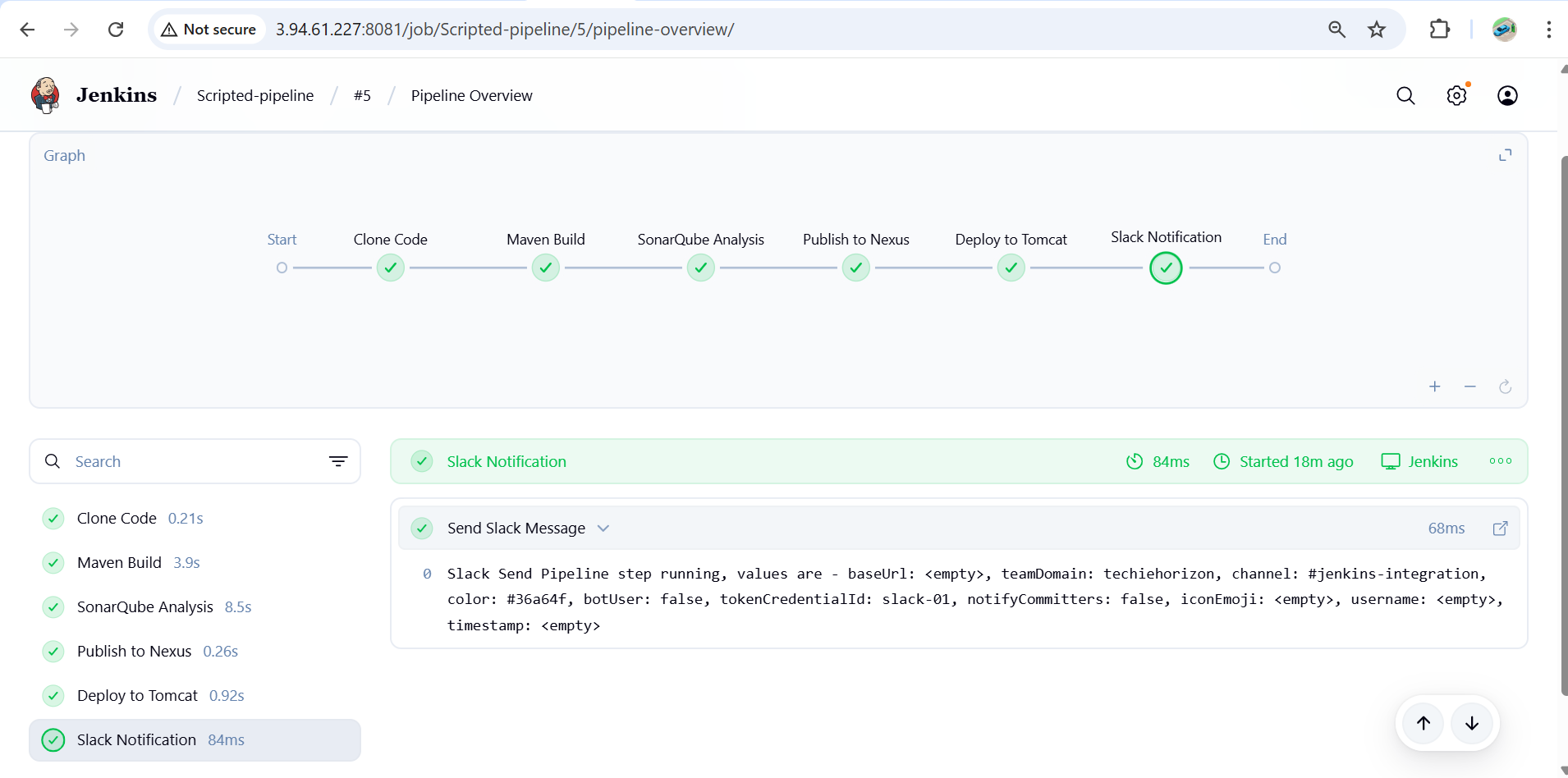
#### ****4. Scripted Pipeline Structure****

A **Scripted Pipeline** uses Groovy syntax.  


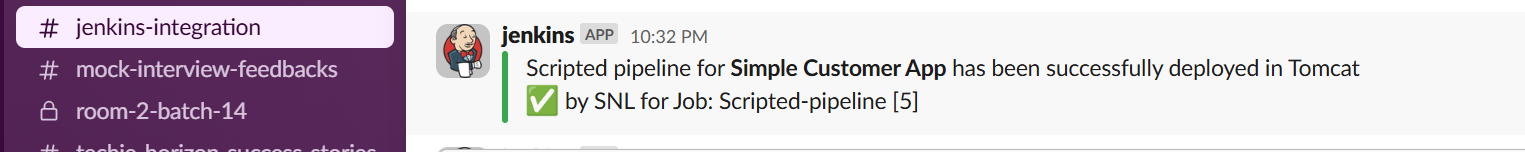
#### ****5. Save and Build****

* Click **Save**.
* Run the job → Click **Build Now**.
* Check console output to verify pipeline execution.





Slack Notification



# Validation checklist & common troubleshooting

Before running the pipeline, check:

* Jenkins agent can reach 3.83.214.6:8081 (Nexus), 3.89.121.33:8080 (Tomcat), and SonarQube server port.
* MVN\_HOME and sonar\_scanner are defined and tool binaries present on agent.
* nexusArtifactUploader plugin installed (or fallback to curl upload).
* Tomcat user has manager-script role.
* curl available on agent.
* Slack token works (test slackSend from a tiny job).

Common errors & fixes:

* findFiles returns empty → build failed or packaging not war → run mvn package locally and inspect target/.
* nexusArtifactUploader unknown step → plugin not installed.
* sonar-scanner not found → wrong sonar\_scanner tool name or not installed.
* waitForQualityGate times out → Sonar server unreachable or analysis not submitted; check Sonar project key and scanner logs.
* Tomcat deploy returns 401 → wrong tomcat creds or missing manager-script role.

**4) Write sample skeleton of pipelines.**

**Declarative Pipeline Skeleton**

**pipeline {**

**agent any // or agent { label 'my-node' }**

**tools {**

**maven 'MVN\_HOME' // optional**

**jdk 'JDK17' // optional**

**}**

**environment {**

**// Define env vars or credentials**

**APP\_ENV = 'dev'**

**// MY\_SECRET = credentials('secret-id')**

**}**

**options {**

**timestamps() // show timestamps in logs**

**}**

**stages {**

**stage('Checkout') {**

**steps {**

**git branch: 'main', url: 'https://github.com/your-org/your-repo.git'**

**}**

**}**

**stage('Build') {**

**steps {**

**sh 'mvn clean package'**

**}**

**}**

**stage('Test') {**

**steps {**

**sh 'mvn test'**

**}**

**}**

**stage('Deploy') {**

**steps {**

**echo "Deploying application..."**

**}**

**}**

**}**

**post {**

**always {**

**echo "Cleaning up..."**

**}**

**success {**

**echo "Pipeline succeeded ✅"**

**}**

**failure {**

**echo "Pipeline failed ❌"**

**}**

**}**

**}**

**Scripted Pipeline Skeleton**

node {

// Define tool paths

def mvnHome = tool 'MVN\_HOME'

stage('Checkout') {

git branch: 'main', url: 'https://github.com/your-org/your-repo.git'

}

stage('Build') {

sh "${mvnHome}/bin/mvn clean package"

}

stage('Test') {

sh "${mvnHome}/bin/mvn test"

}

stage('Deploy') {

echo "Deploying application..."

}

// Notifications / Cleanup

stage('Notify') {

echo "Send Slack or Email notification here"

}

}

# Scripted vs Declarative — stage-by-stage & config differences

(Your earlier Declarative pipeline → now Scripted)

* **Overall structure**
  + Declarative: pipeline { agent any; tools { ... } environment {...} stages {...} post {...} }
  + Scripted: imperative node { ... } + stage("...") { ... }. You call tool to resolve installations and assign to variables (e.g., mvnHome = tool 'MVN\_HOME').
* **Tools**
  + Declarative: tools { maven 'MVN\_HOME' } — Jenkins injects tool into PATH automatically for that node.
  + Scripted: def mvnHome = tool name: 'MVN\_HOME', type: 'maven' — you must reference the full path ${mvnHome}/bin/mvn.
* **Environment**
  + Declarative: environment { VAR = 'value' } — automatically available to all steps.
  + Scripted: env.VAR = 'value' or assign local def variables. Use withCredentials to temporarily set credentials.
* **Post / Notifications**
  + Declarative: built-in post { success { ... } failure { ... } } blocks for cleanup/notifications.
  + Scripted: you must implement your own try/catch/finally or place slackSend stages explicitly.
* **Quality Gate**
  + Declarative: you usually use waitForQualityGate abortPipeline: true inside a stage.
  + Scripted: use def qg = waitForQualityGate() and check qg.status to decide to error.
* **Readability & Safety**
  + Declarative: enforces structure (agent, stages) and is easier for linters and shared libraries.
  + Scripted: more flexible/imperative — useful for complex logic, but easier to introduce hard-to-debug flows.
* **Credentials handling**
  + Same in both: use withCredentials([...]) in scripted; declarative supports credentials directive in environment as well.

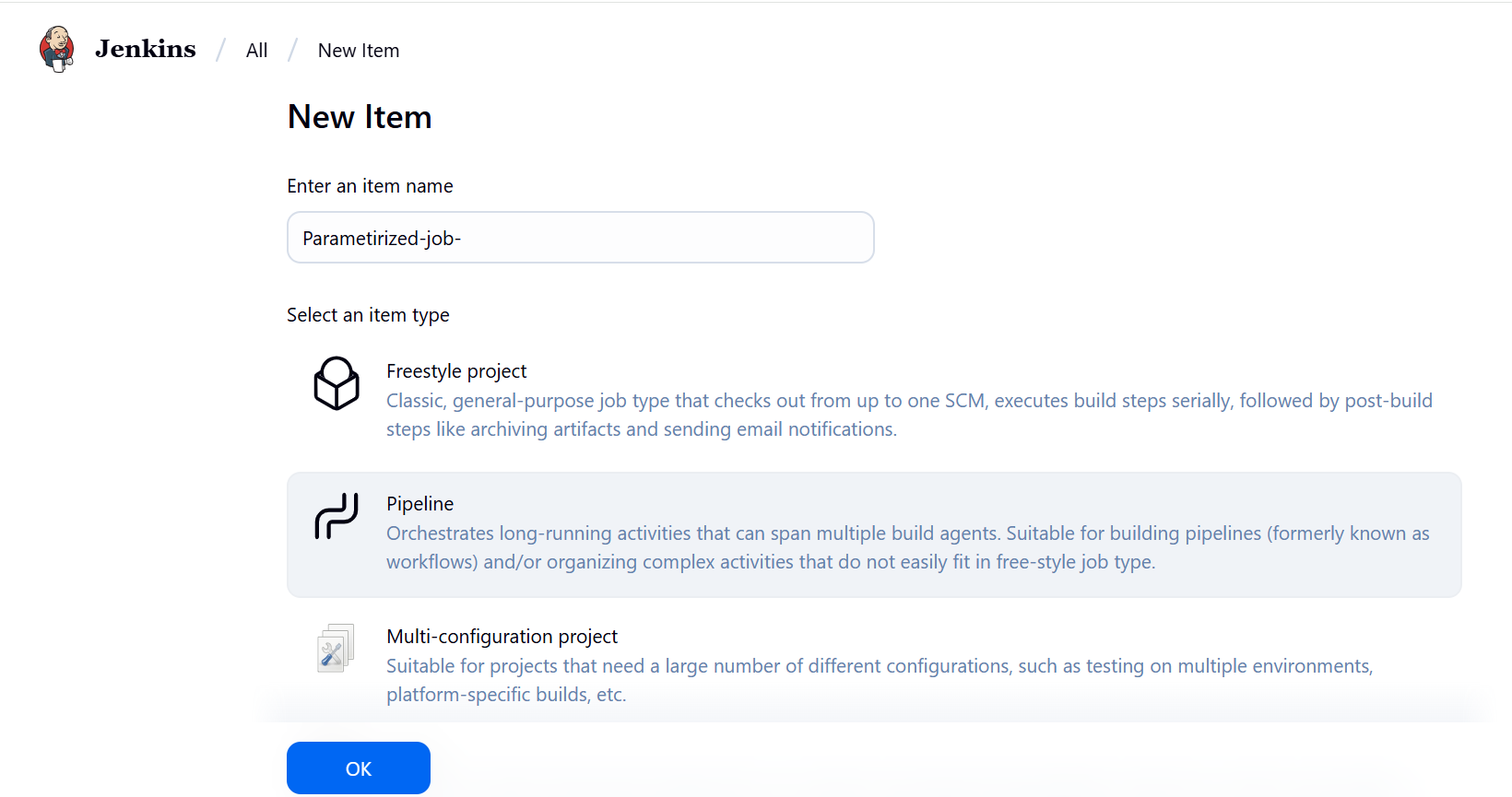
**5) Create a parametirized job in jenkins.**

<https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git>

### Creating a Parameterized Jenkins Pipeline

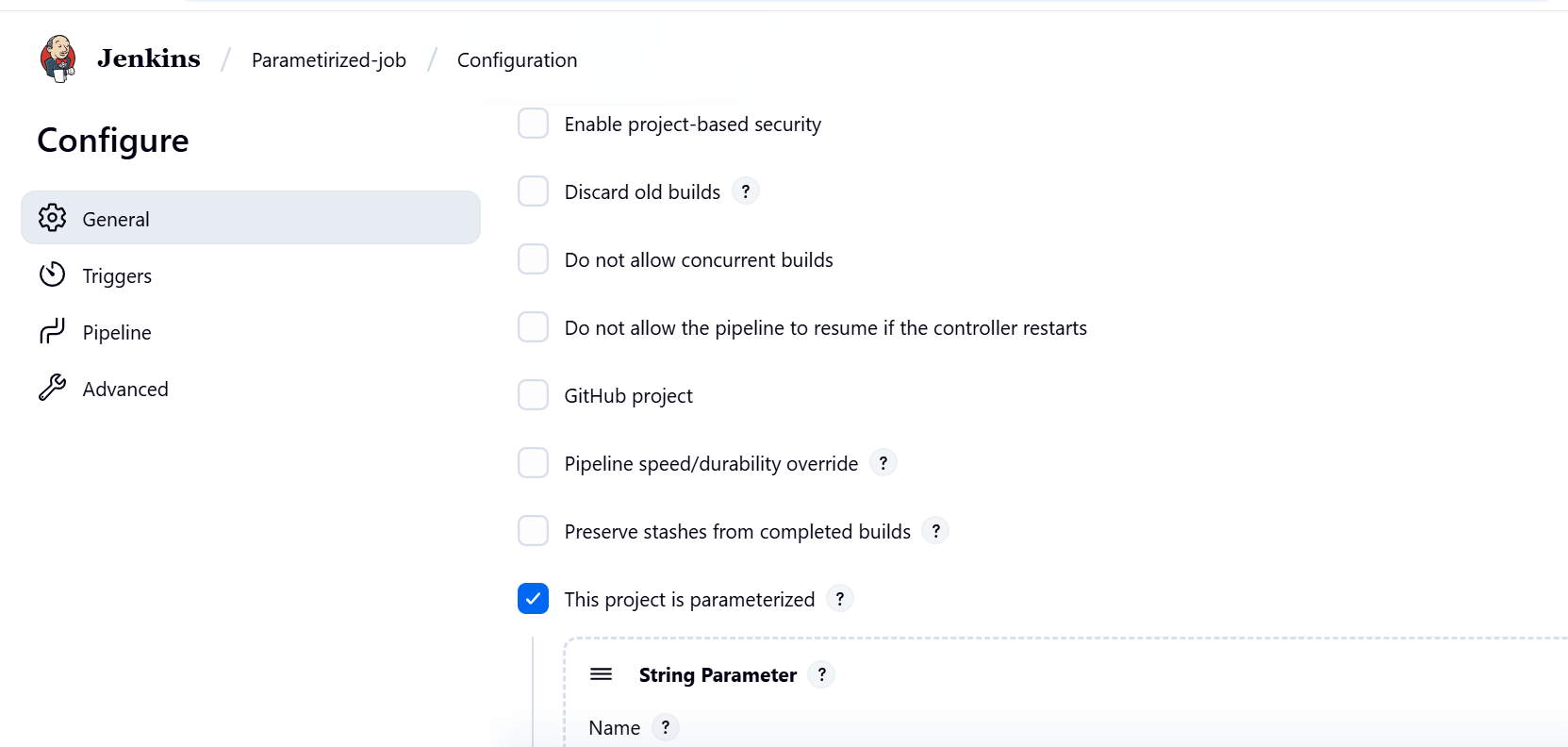
#### Step 1. ****Start a New Pipeline Job****

1. In Jenkins, click **New Item**.
2. Name your job and select **Pipeline**, then click **OK**.

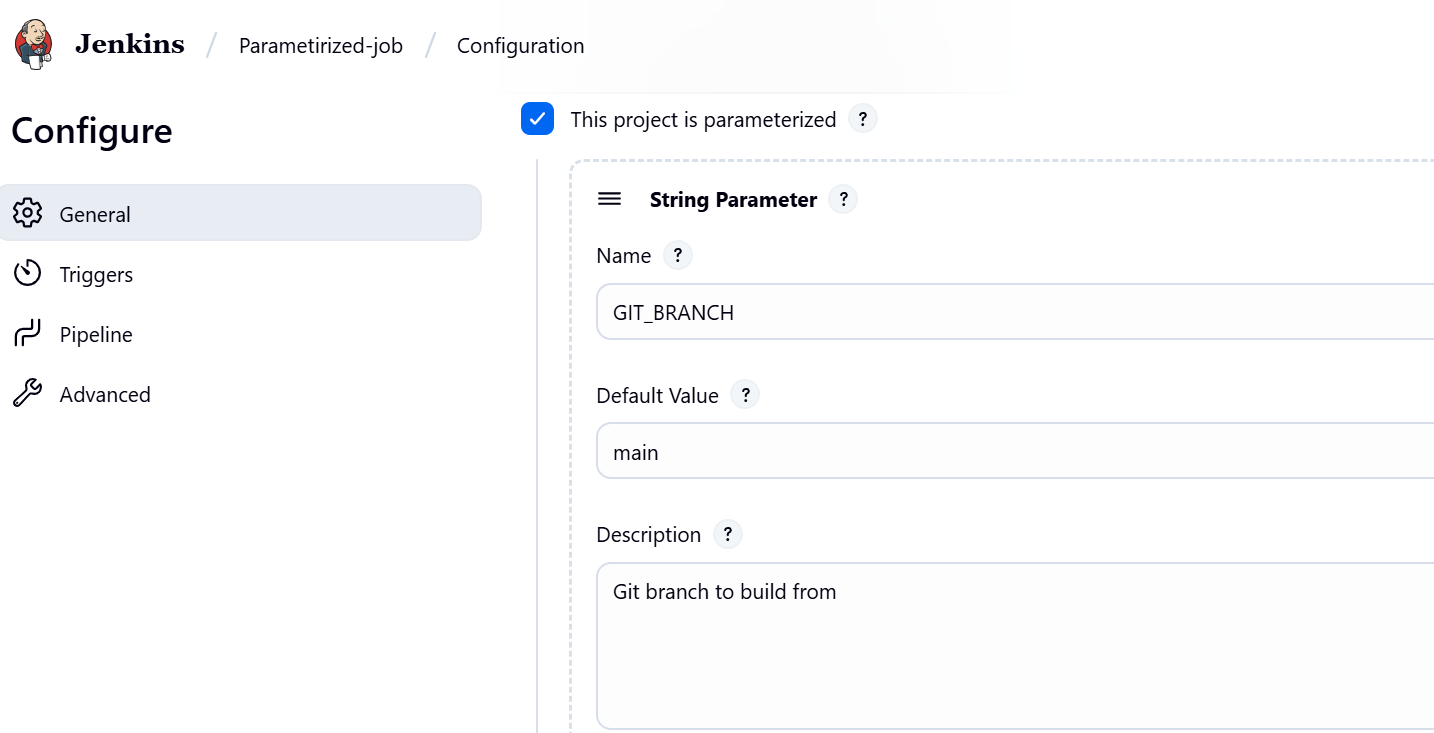


**Step 2: Add Parameters in Jenkins**

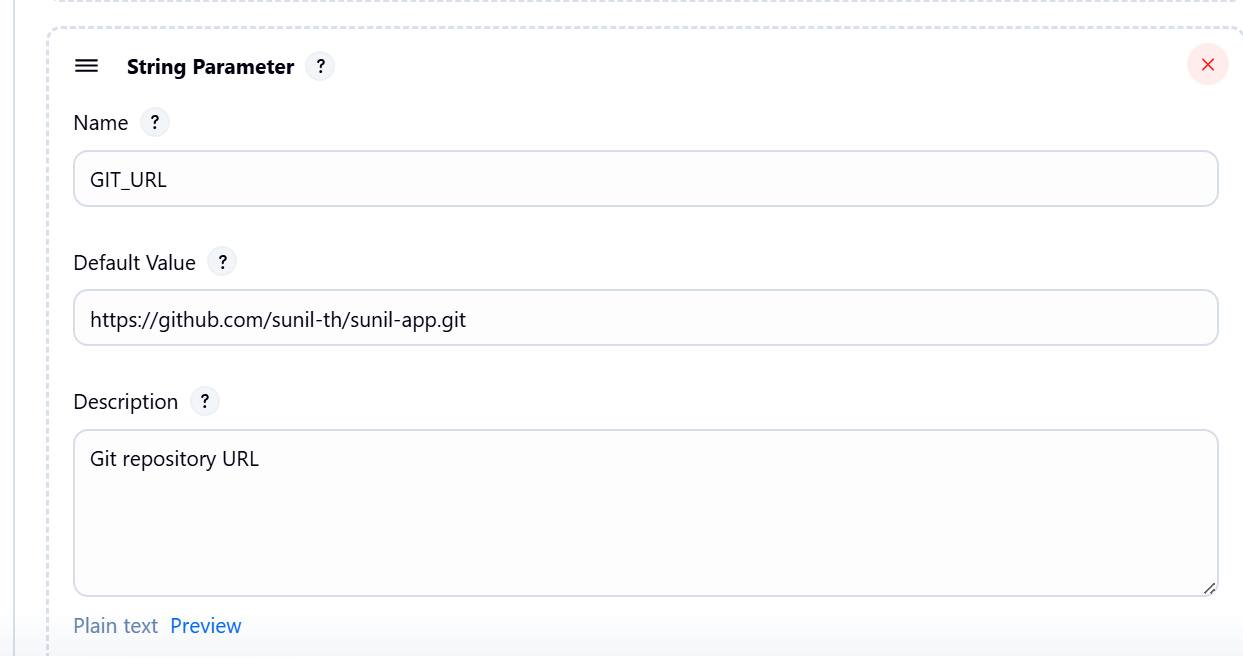
1. Go to **Jenkins Dashboard → Your Job → Configure**.
2. Check ✅ **This project is parameterized**.
3. Add the following parameters:



1. **String Parameter**
   * **Name**: GIT\_BRANCH
   * **Default Value**: main
   * **Description**: Git branch to build from



1. **String Parameter**
   * **Name**: GIT\_URL
   * **Default Value**: https://github.com/sunil-th/sunil-app.git
   * **Description**: Git repository URL



1. **String Parameter**
   * **Name**: APP\_VERSION
   * **Default Value**: 1.0
   * **Description**: Application version to deploy



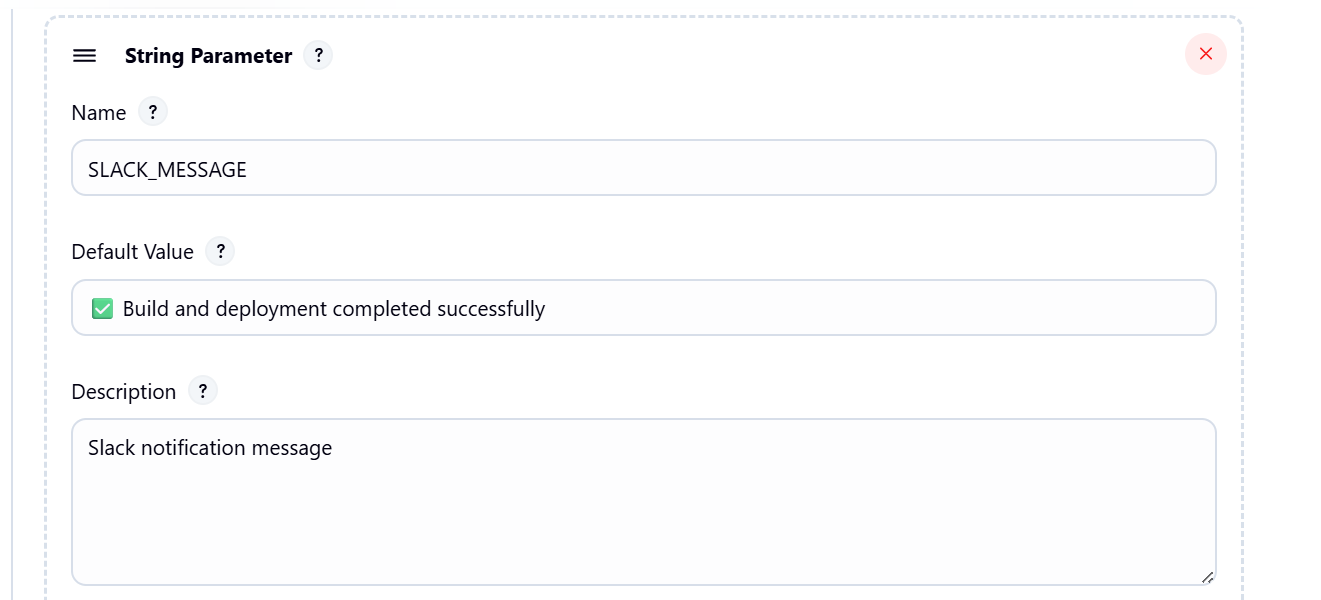
1. **String Parameter**
   * **Name**: NEXUS\_REPO
   * **Default Value**: sunil-app
   * **Description**: Nexus repository name



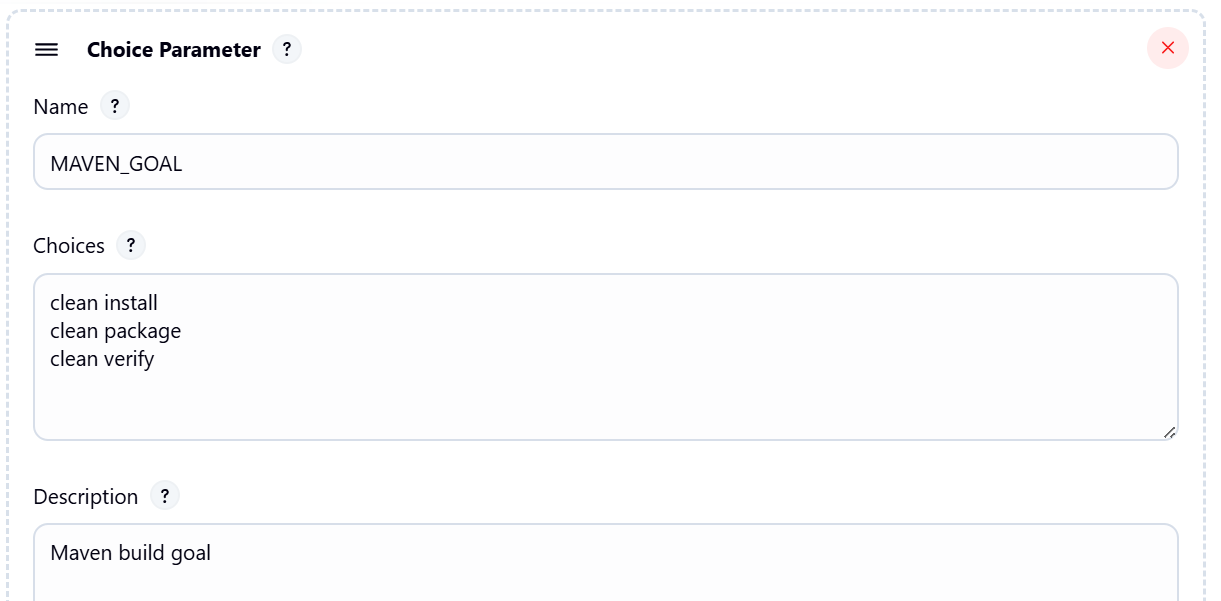
1. **String Parameter**
   * **Name**: SOURCE\_PATH
   * **Default Value**: src/main/java
   * **Description**: Path to source files (for SonarQube)



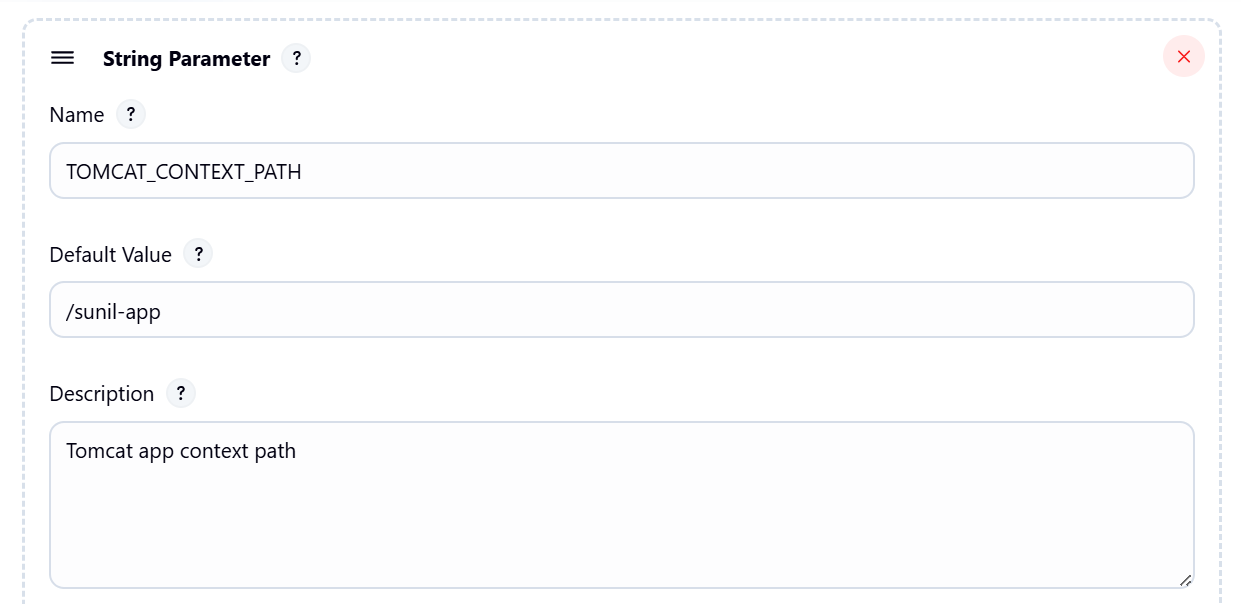
1. **String Parameter**
   * **Name**: SLACK\_MESSAGE
   * **Default Value**: ✅ Build and deployment completed successfully
   * **Description**: Slack notification message



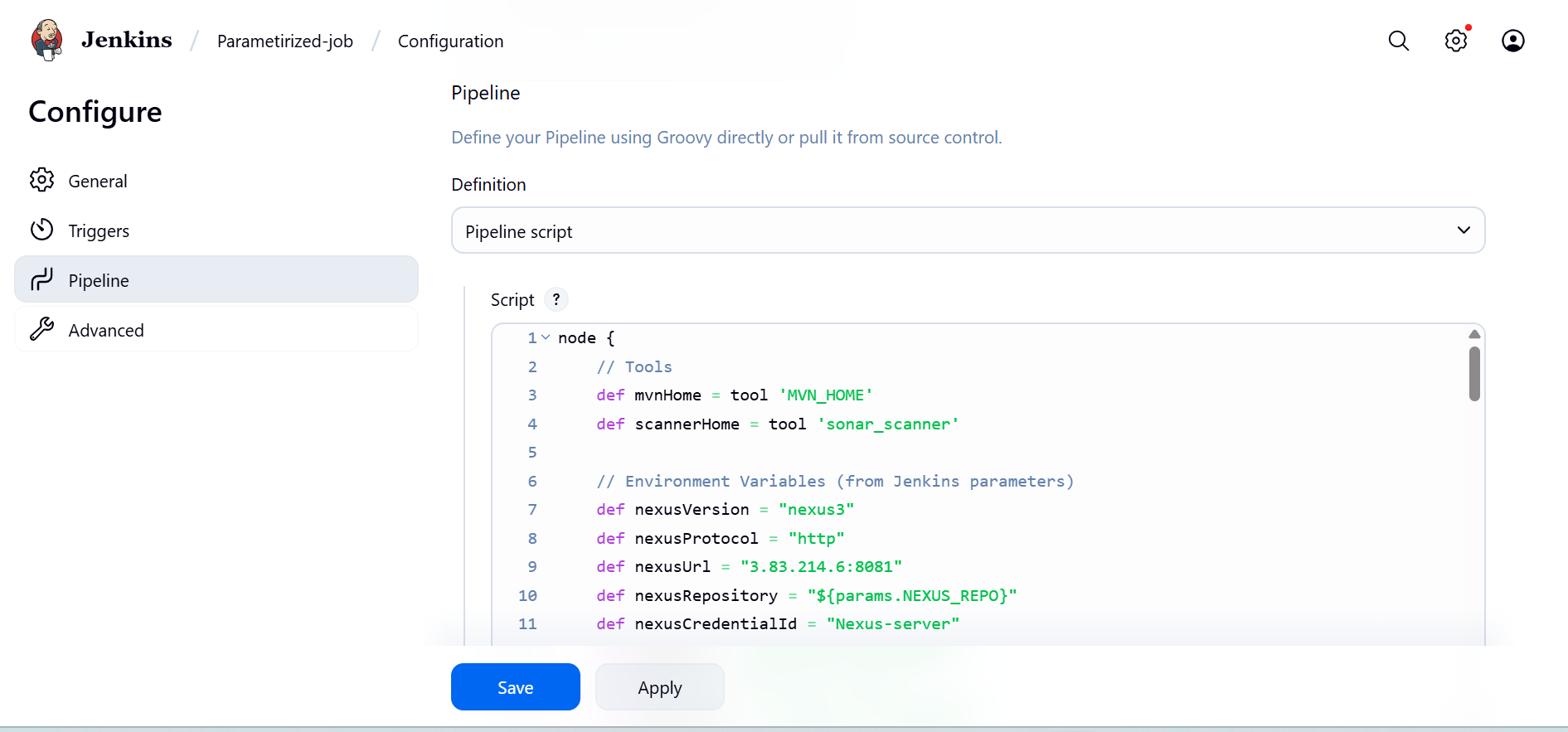
1. **Choice Parameter**
   * **Name**: MAVEN\_GOAL
   * **Choices**:
   * clean install
   * clean package
   * clean verify
   * **Description**: Maven build goal



1. **String Parameter**
   * **Name**: TOMCAT\_CONTEXT\_PATH
   * **Default Value**: /sunil-app
   * **Description**: Tomcat app context path

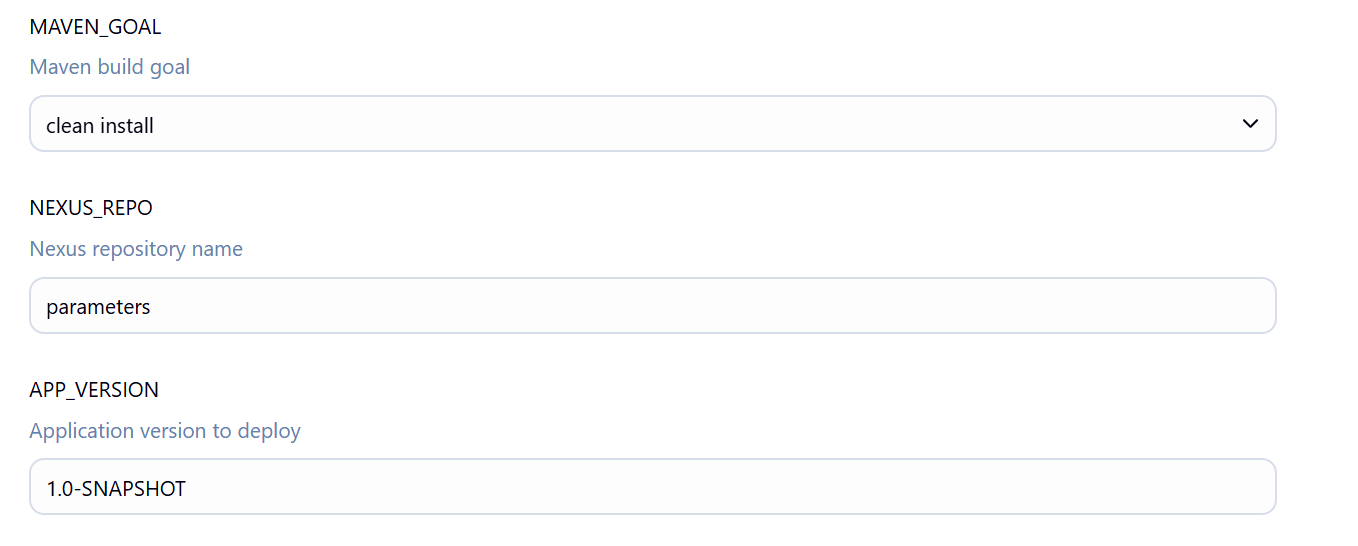


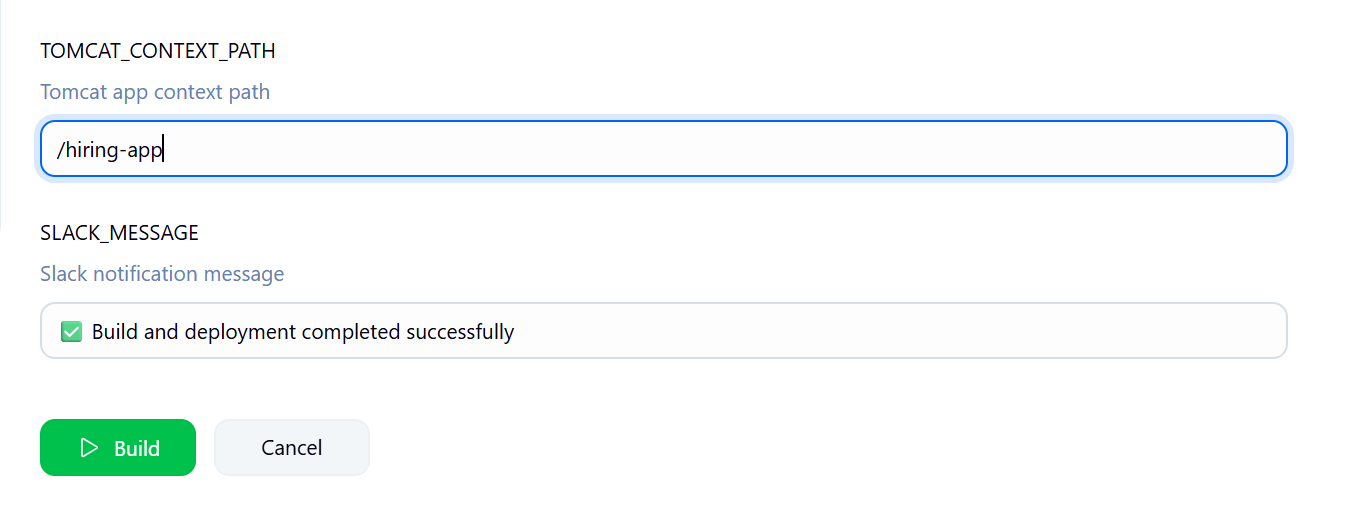
**Step 2: Updated Scripted Pipeline Code**

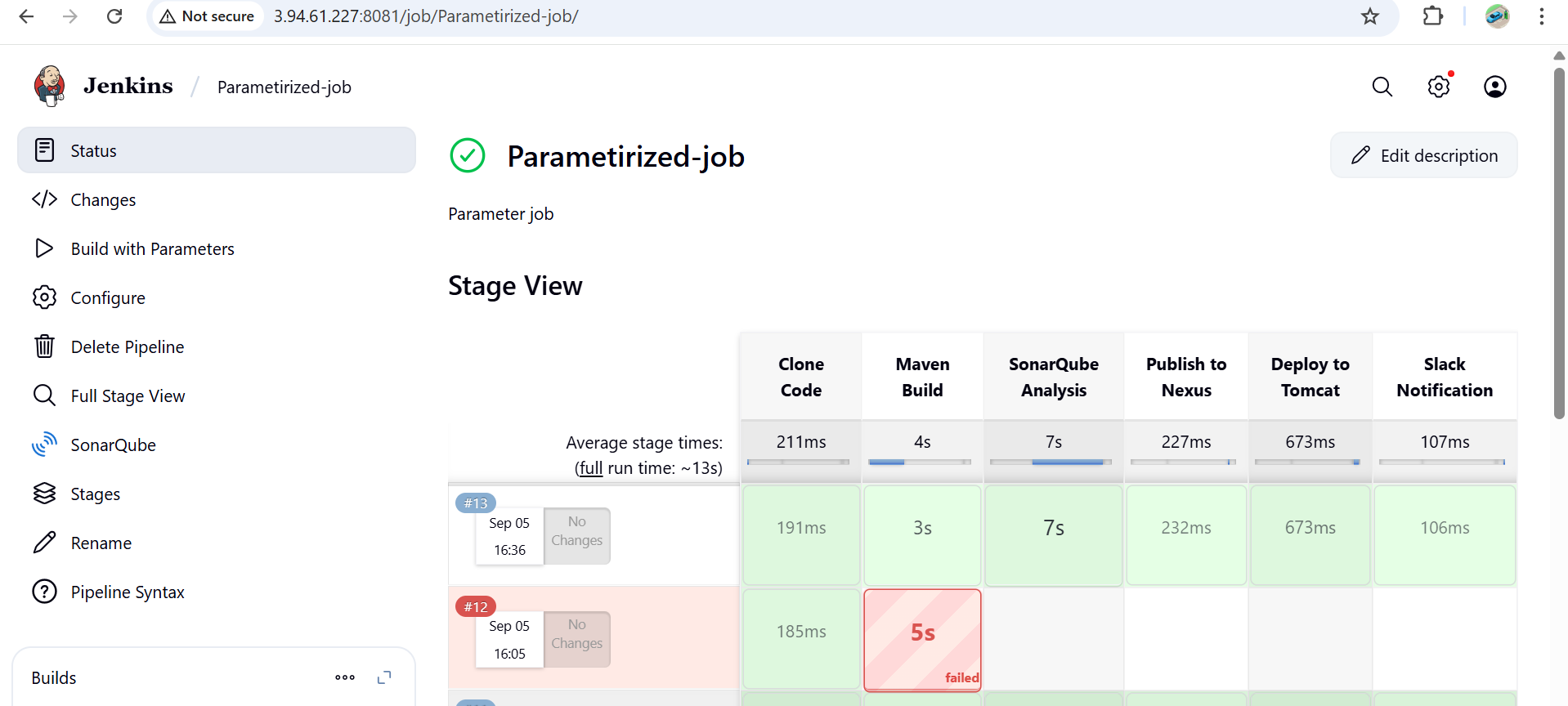


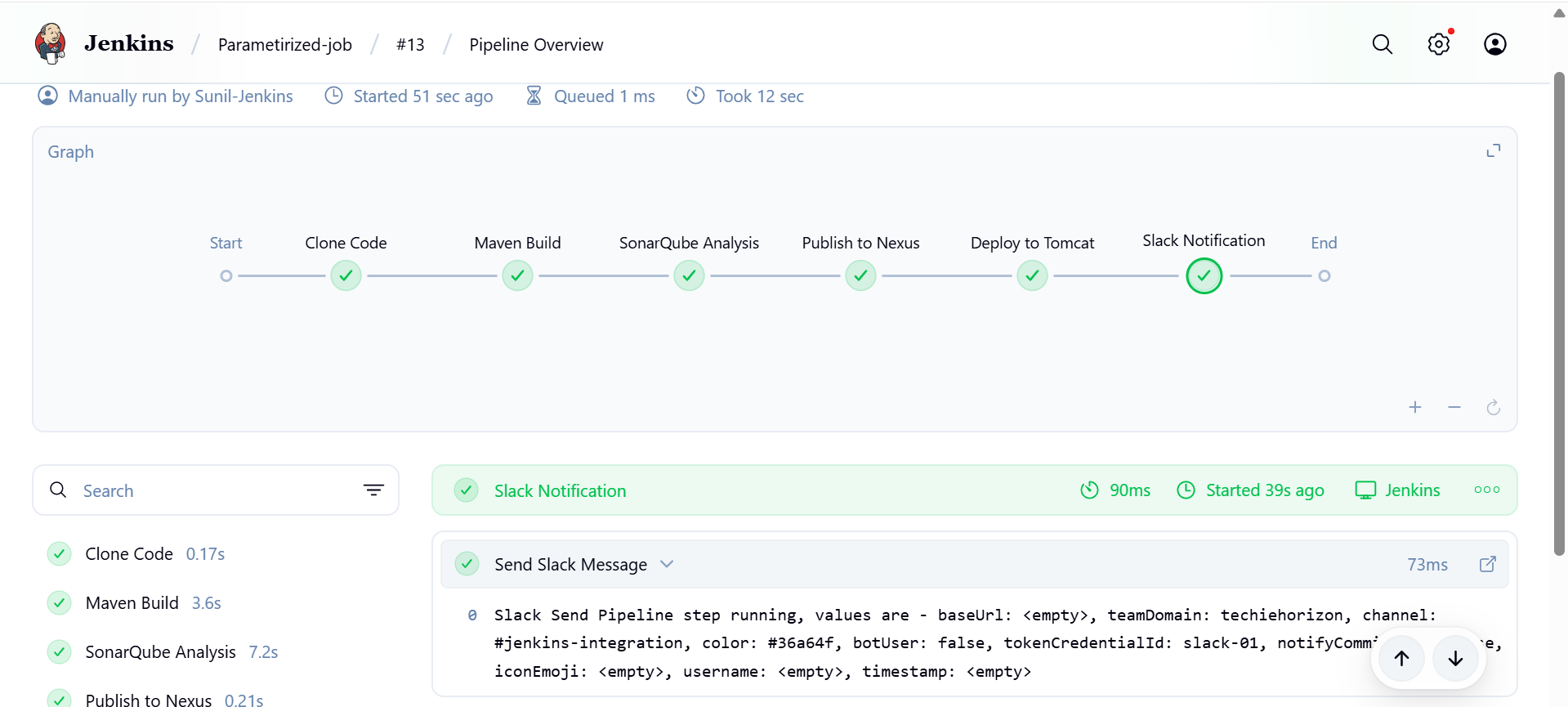
## Step 3: Save and Build with Parameter

## 





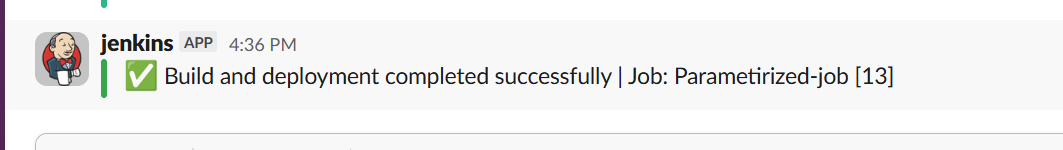




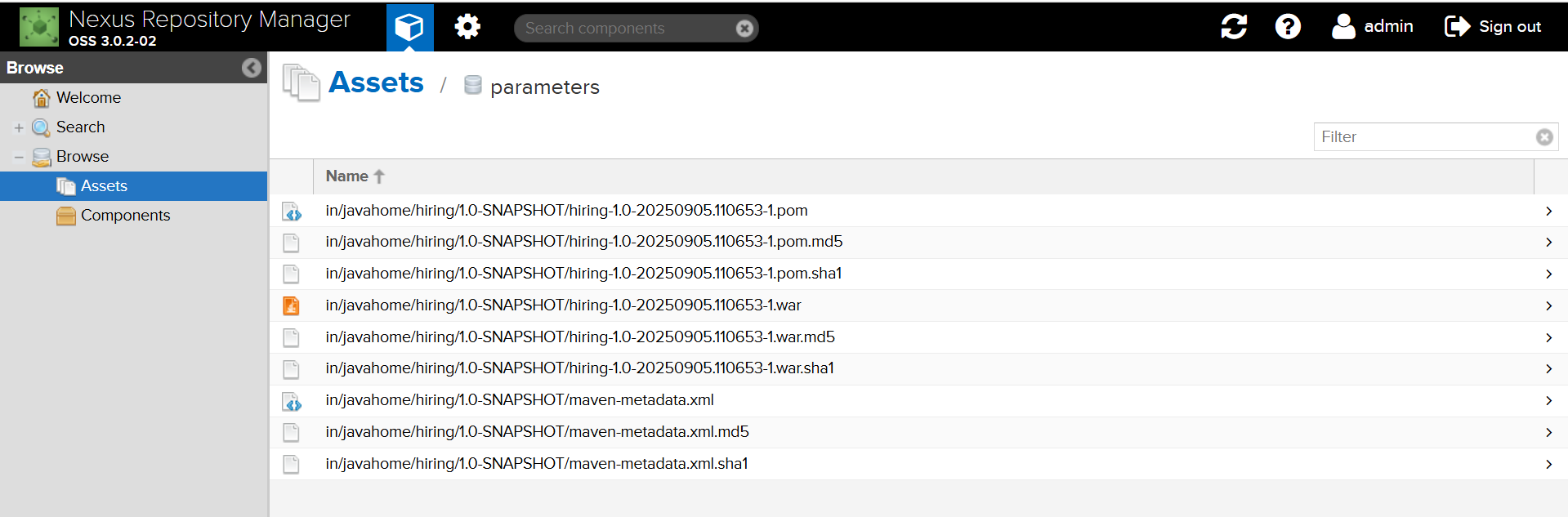
## How It Works

* When you click **Build with Parameters**, Jenkins will ask you to enter:
  + Branch, Repo URL, App Version, Nexus Repo name, Maven goal, Tomcat context, Slack details.
* Pipeline runs with those values.
* Example: you can build branch with feature-1.1 or master without touching the pipeline code.
* You can change Slack message on each run if needed.

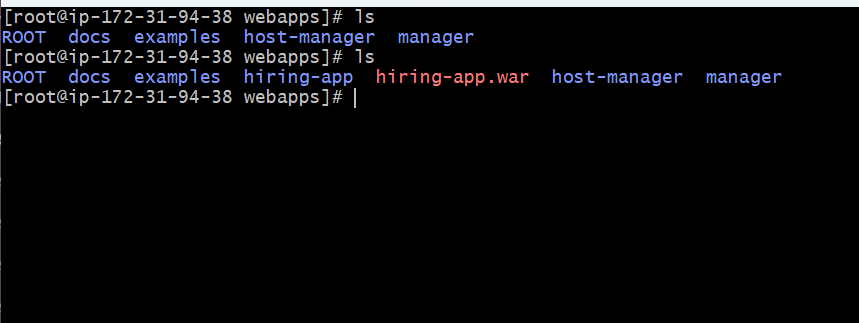
**Slack**



**Nexus**

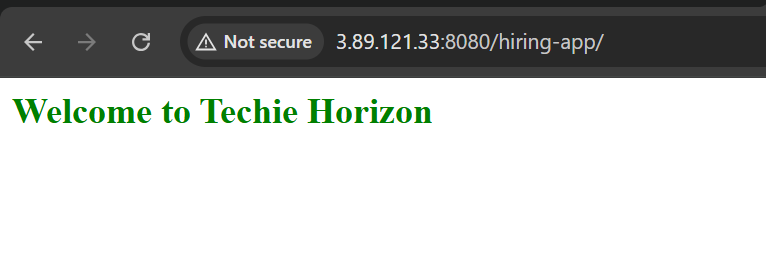
****

**Tomcat**

****

**Access in the web through the tomcat server public ip and the contextpath you given in the**

**http://3.89.121.33:8080/hiring-app/**

****

**6) Setup one slave machine for jenkins.**

Jenkins Master–Slave Configuration

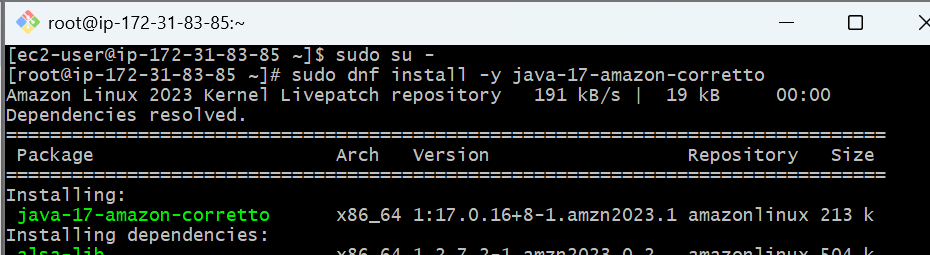
Steps to be done on slave

**Step-1** : Create an ec2 –slave

Login to slave machine..

Install Java on slave

sudo dnf install -y java-17-amazon-corretto

****

Check the java version

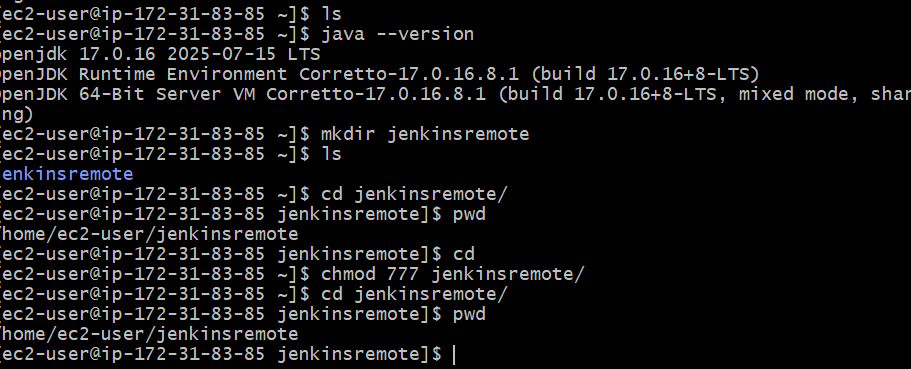
Java –version

Create a folder **jenkinsremote** folder

And give permissions to that folder

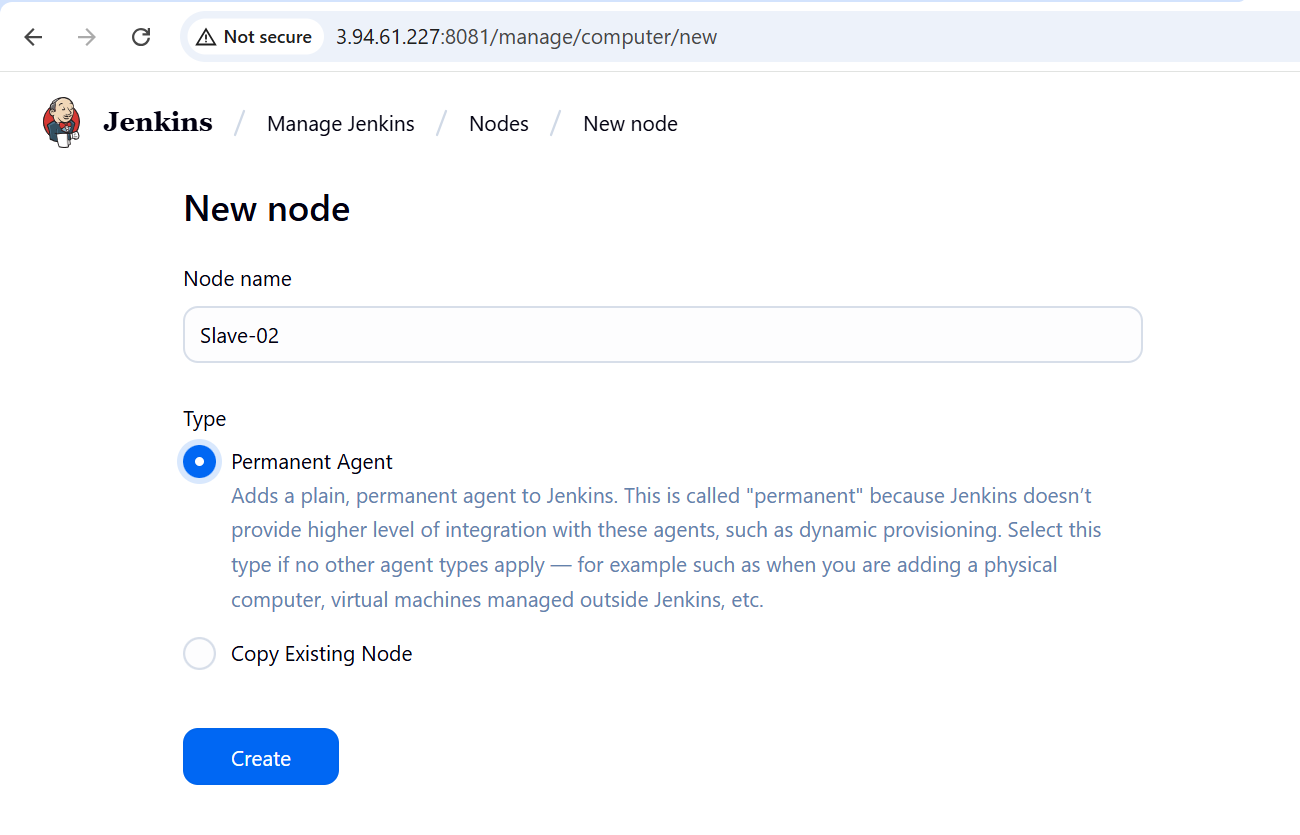
**Chmod 777 jenkinsremote**

and Copy the pwd **/home/ec2-user/jenkinsremote**

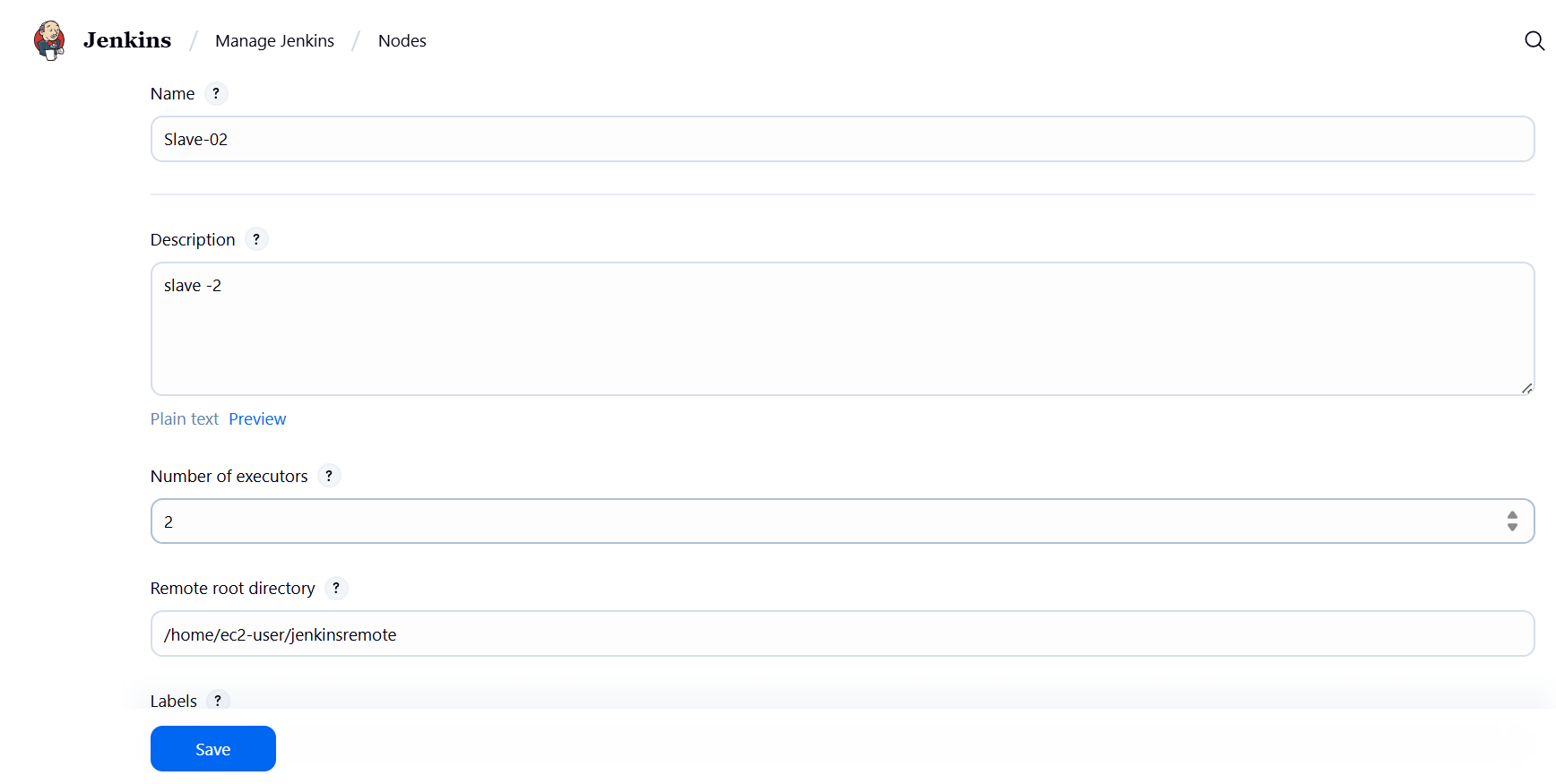


### ****Configure Node in Jenkins Master****

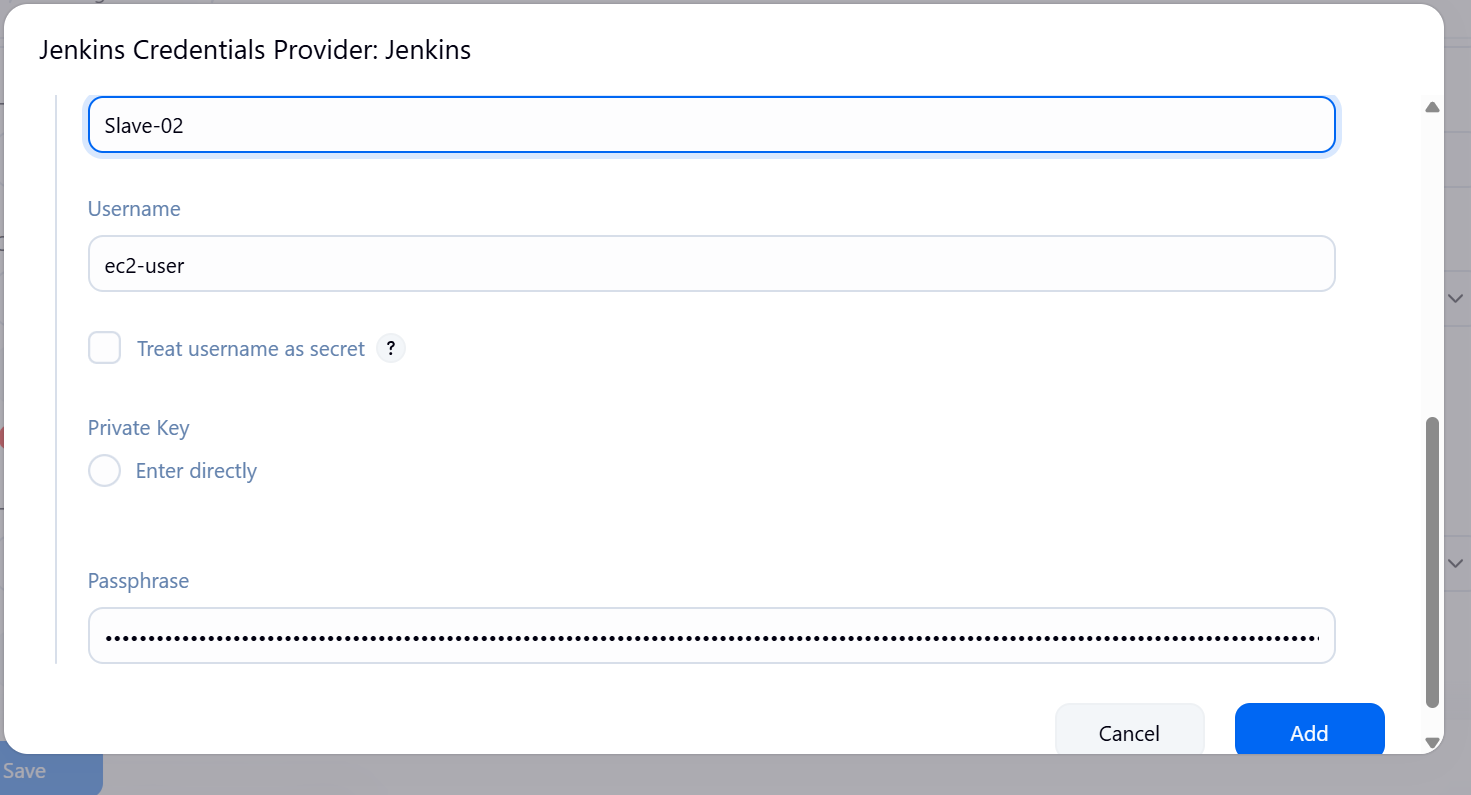
1. **Go to Jenkins Dashboard → Manage Jenkins → Nodes and Clouds → New Node**
2. **Enter node name (e.g., slave-1) → Select Permanent Agent → OK**

****

1. **Configure:**
   * **Remote root directory: (/home/ec2-user/jenkinsremote)**
   * **Labels: give a label (e.g., linux, build) to target jobs**
   * **Number of executors** → usually 1 (how many jobs can run in parallel on this agent).

****

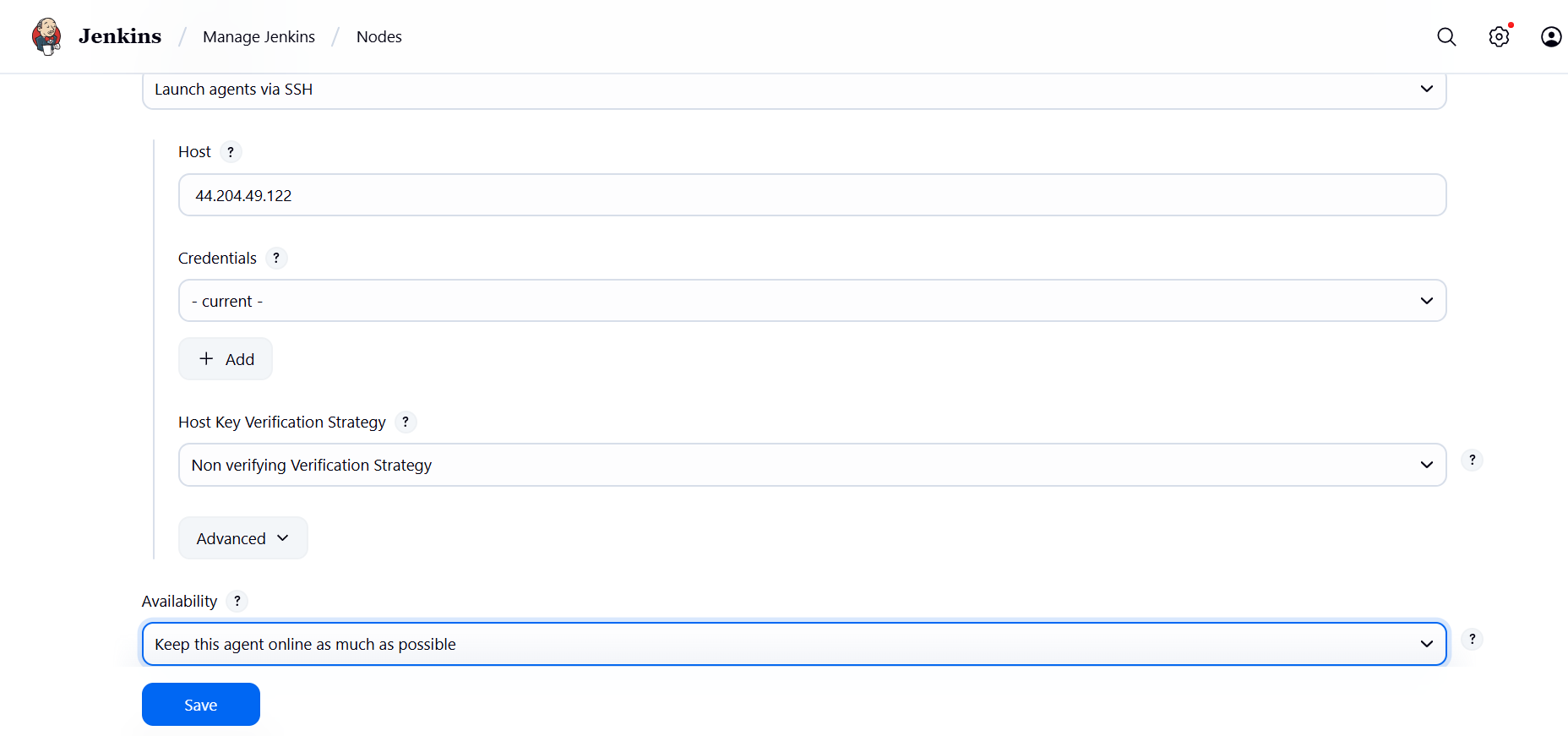
**Add credentials (Jenkins user SSH key)**

****

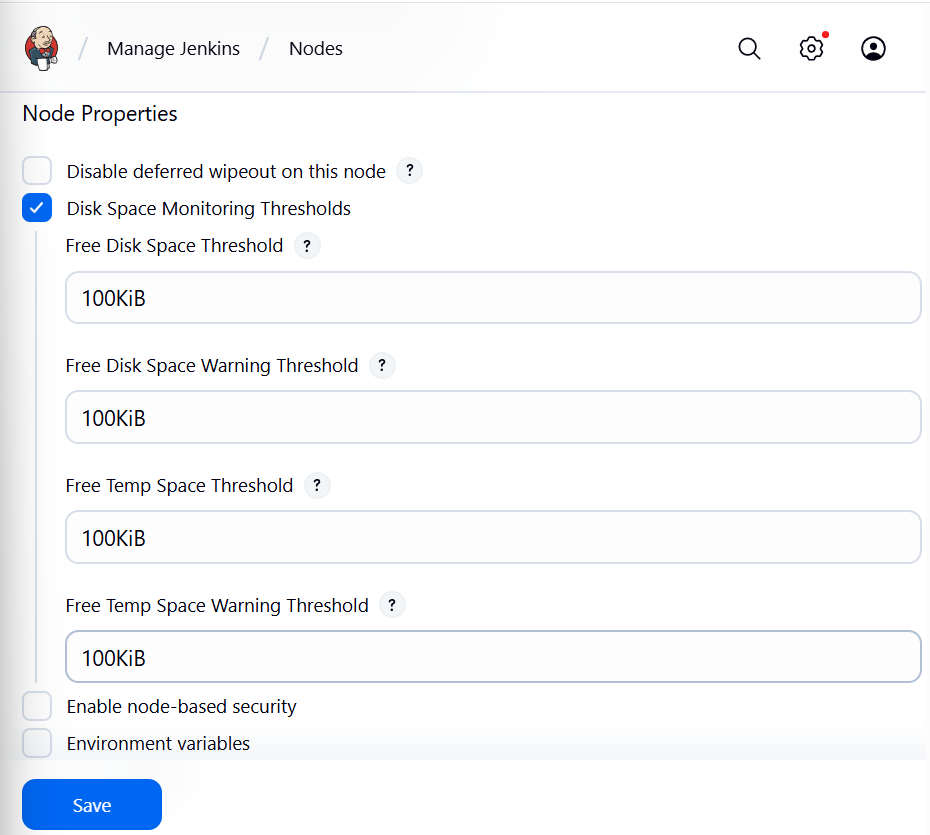
**Enter public /IP of ec2 slave**

**OR Launch agent via Java Web Start (JNLP) if SSH not possible**

**Usage** → Choose Use this node as much as possible.

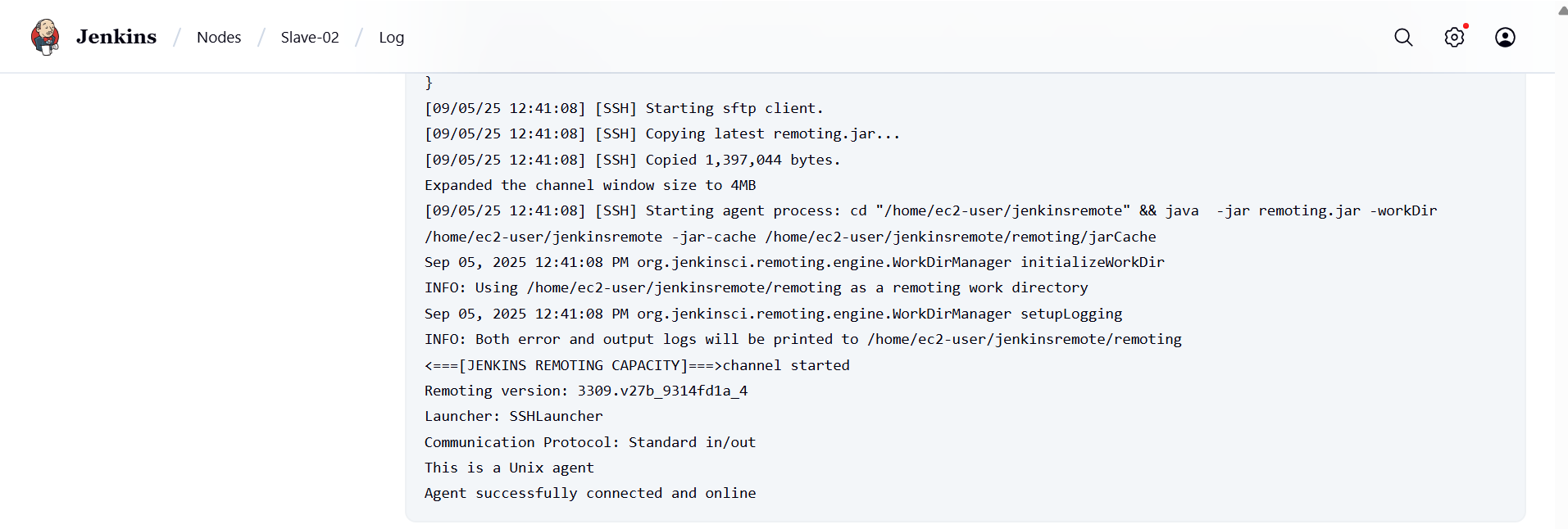
****

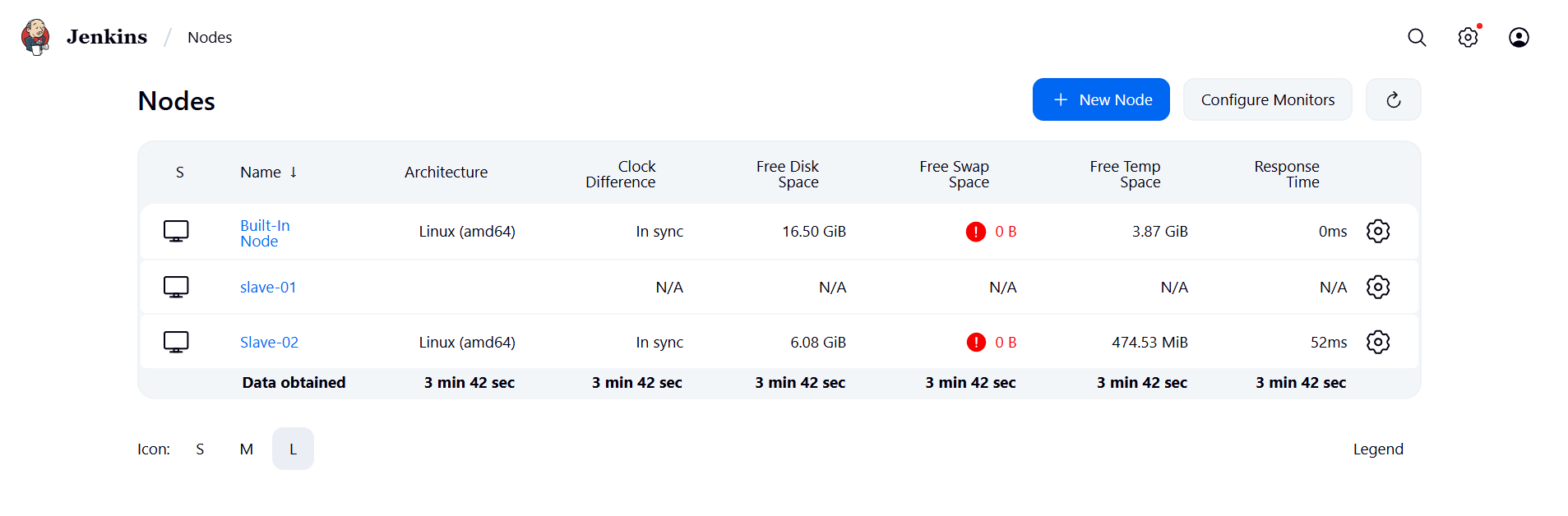
**Modify the disk space threhold**

****

### ****. Test Connection****

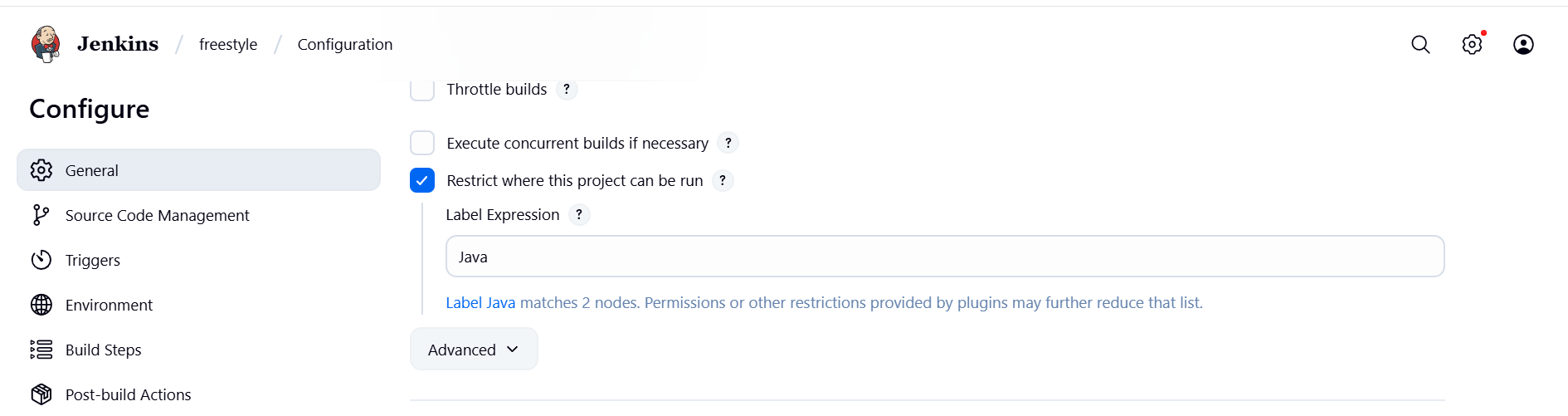
* Click **Save** → Jenkins will try to connect.
* Check node status in **Build Executor Status** (bottom left).
* If successful, node shows as **online**.

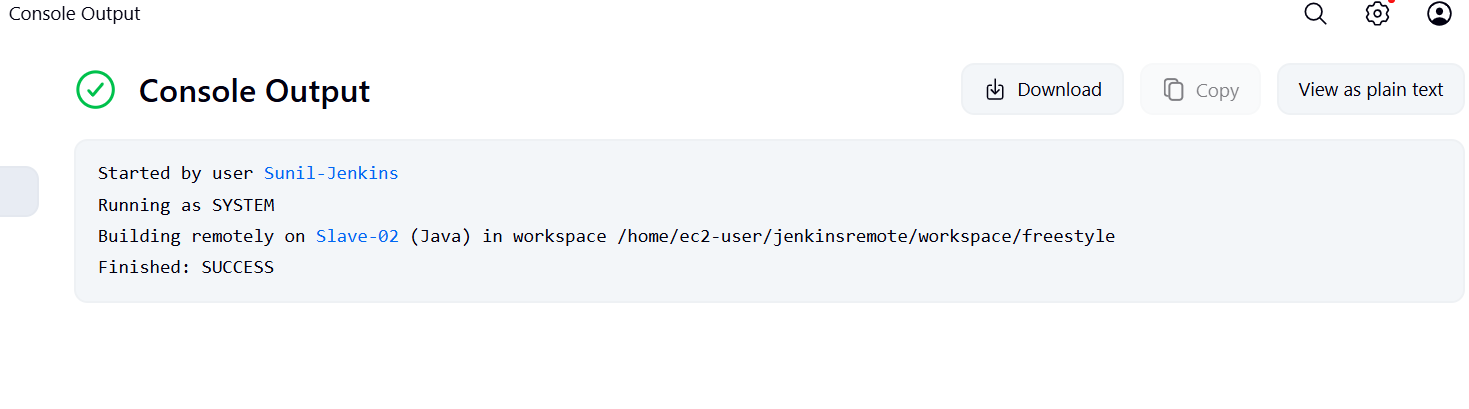
****

****

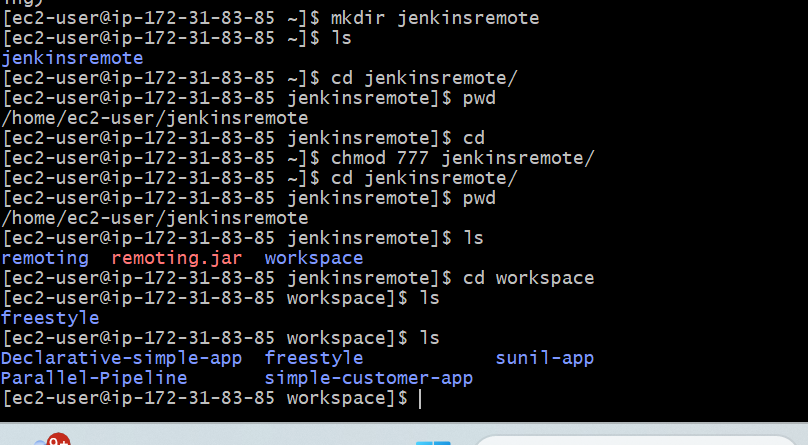
### ****Run a Test Job****

* Create a simple job and restrict it to the new agent by **label**.



****

Now your Jenkins slave is ready and connected.

****