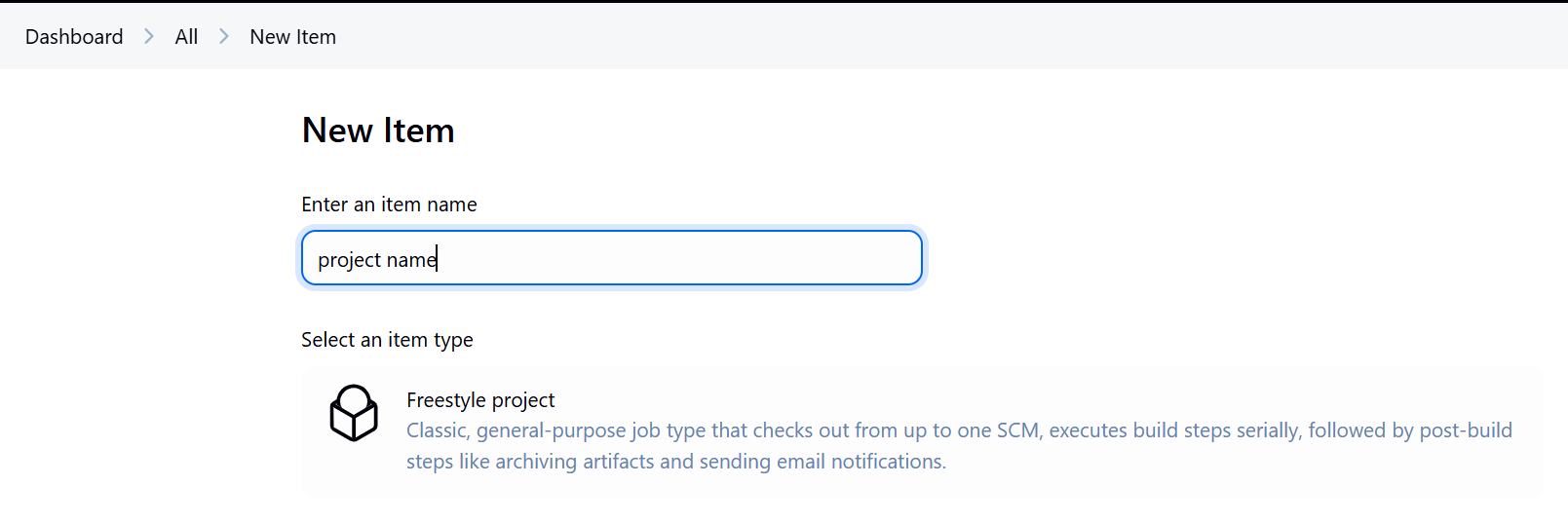
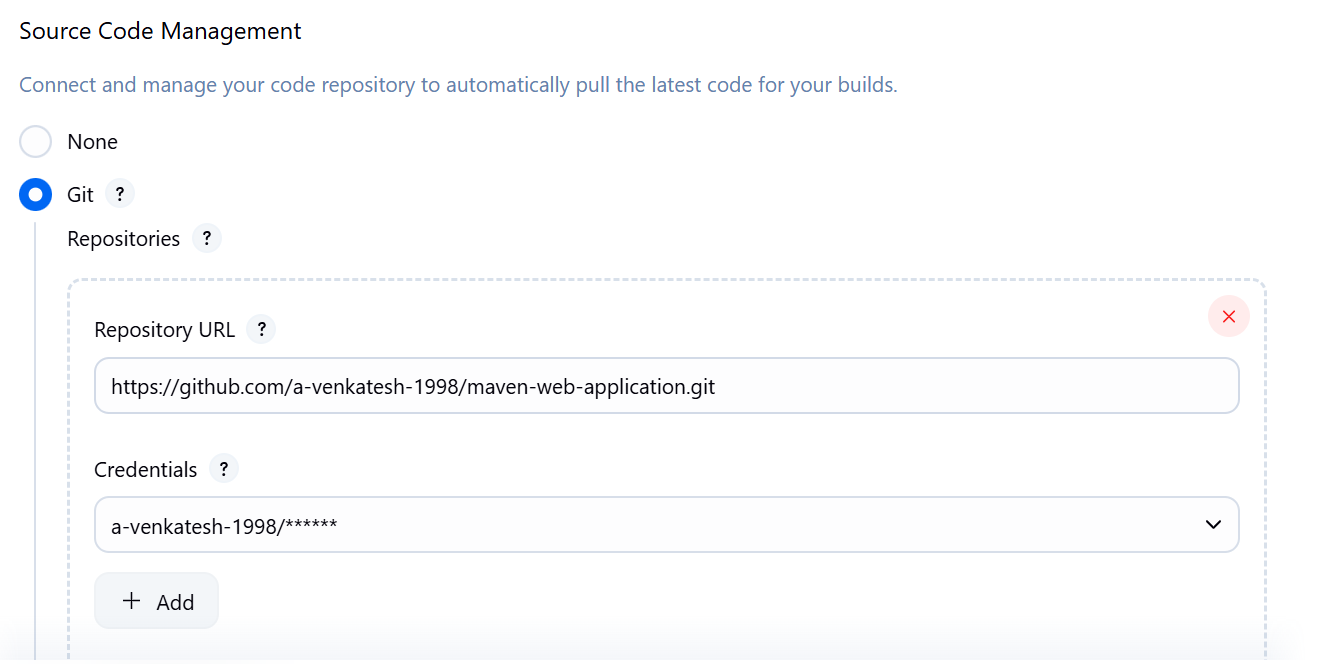
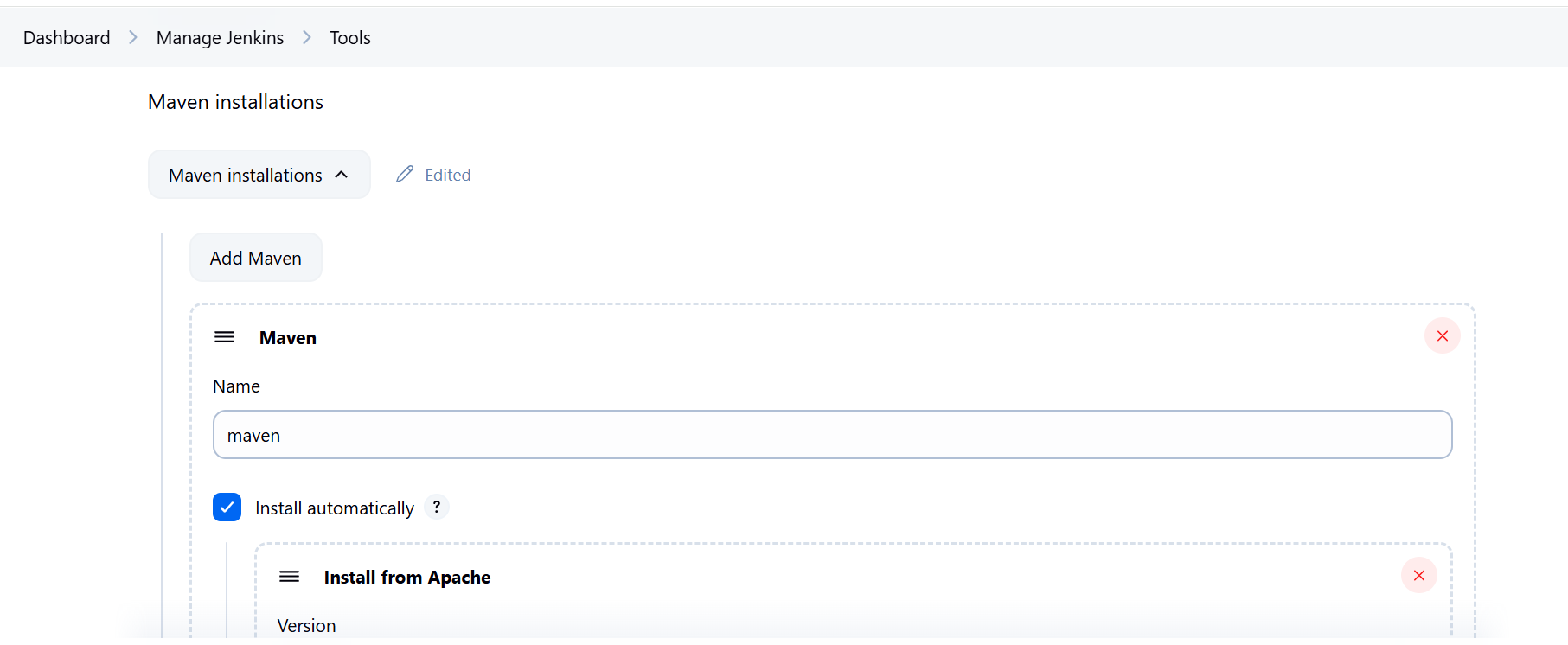
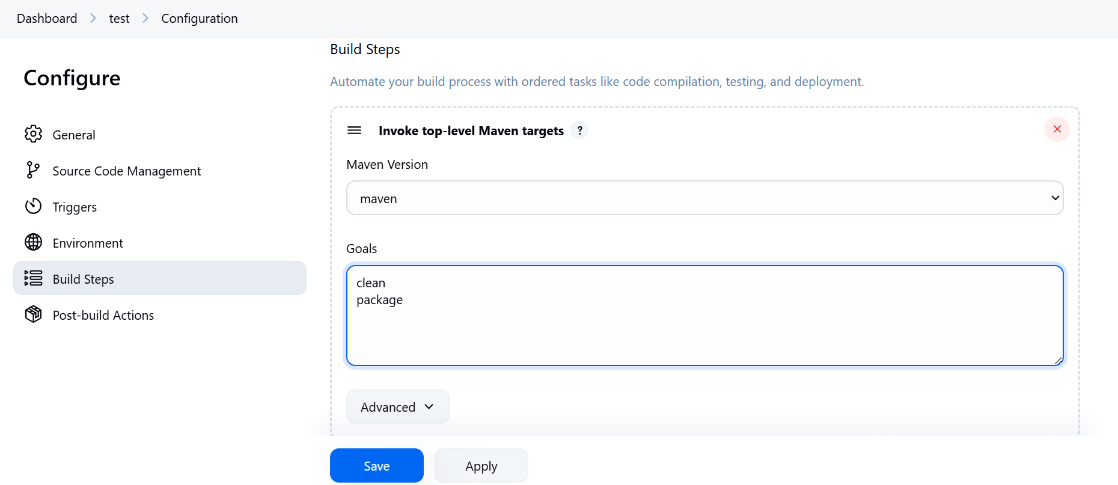
JENKINS

1. To integrate with GIT  
2. To integrate with MAVEN  
3. To integrate with SONARQUBE  
4.To integrate with NEXUS  
5.To integrate with TOMCAT  
6.Discard old builds  
7. Delete workspace before build starts and adding timestamp to console o/p  
8. Enable/Disable Project  
9. JACOCO – Plugin  
10. Email Integration  
11. Jenkins directory structure  
12. Plugin Management  
13. Jenkin File(port,defaultpath)  
14.Build with Parameter  
15.Create View  
16. Jenkins Security – users/level of access  
17. Pipeline Jobs  
18.Multibranch pipeline  
19. Thin backup  
20. Jenkins master-slave Arch & adding nodes  
21.Jenkins migration  
22.Jenkins CLI  
23. Upstream and downstream jobs  
24.Parallel Execution  
25.Shared libraries   
  
  
  
  
  
  
create a new item and select freestyle project  
  
  
1.To integrate with GIT  
under scm add git repository and provide credentials  
  
  
and select which branch you want to use  
  
2.To integrate with MAVEN  
In order to use maven use below steps ManagejenkinsàTools and install specific maven version  
  
  
  
add clean package and build it will sucess  
   
  
  
  
**mvn clean package --> jenkins will pull the code and build the packages**  
  
3.To integrate sonarqube  
  
change pom.xml in github  
  
<properties>

<jdk.version>1.8</jdk.version>

<spring.version>5.1.2.RELEASE</spring.version>

<junit.version>4.11</junit.version>

<log4j.version>1.2.17</log4j.version>

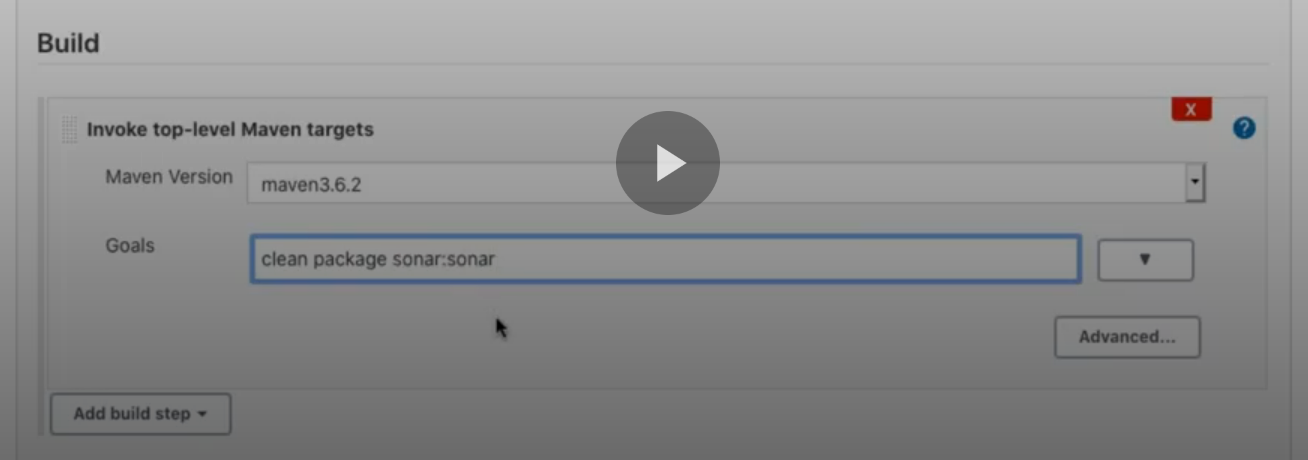
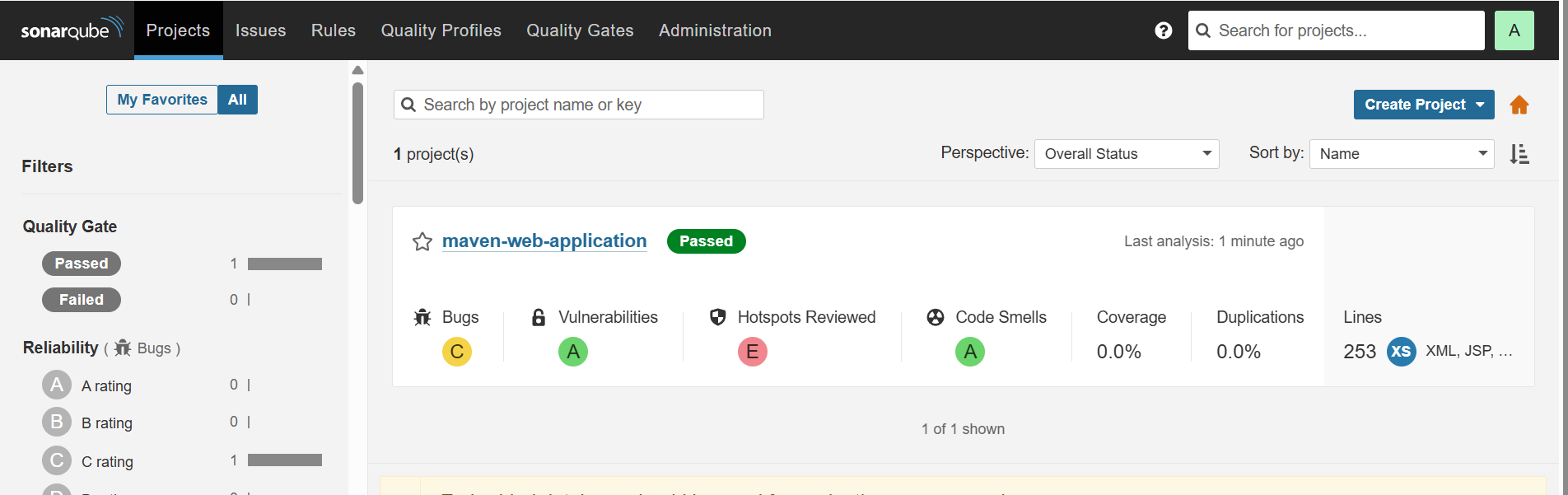
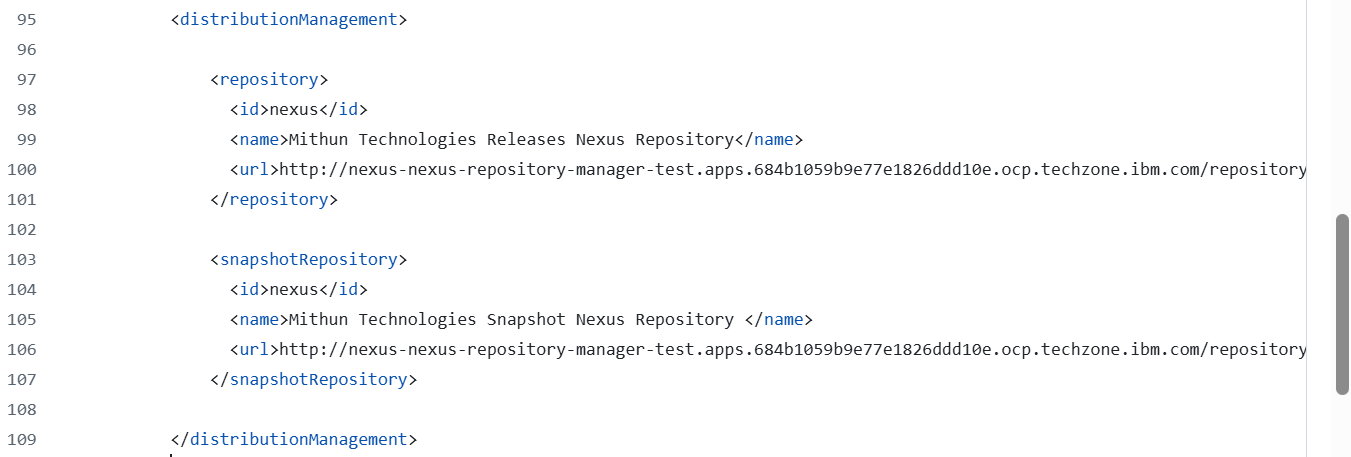
<sonar.host.url>http://172.31.33.143:9000/</sonar.host.url>

<sonar.login>admin</sonar.login>

<sonar.password>passw0rd</sonar.password>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

</properties>  
  
  
  
  
  
  
  
  
myaccountàsecurityà  
  
Add sonar:sonar in maven targets  
  
  
  
  
once after the build you will see the o/p  
  
  
  
**mvn clean package sonar:sonar --> it will package and create sonar type report**  
  
4.To integrate NEXUS  
  
login to nexus and create two repositories and both should be hosted  
  
  
  
  
  
  
  
In github pom.xml change the repo names  
  
  
or  
 <distributionManagement>

<repository>

<id>nexus</id>

<name>Mithun Technologies Releases Nexus Repository</name>

<url>http://172.31.42.154:9980/mithuntechnologies/repository/canarabank-release/</url>

</repository>

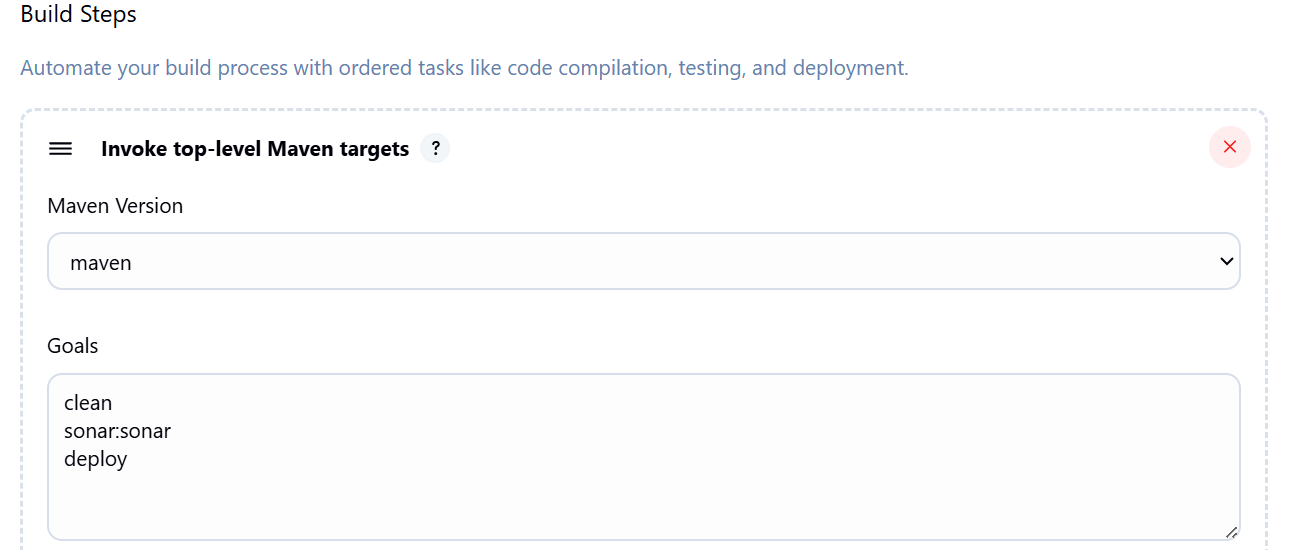
<snapshotRepository>

<id>nexus</id>

<name>Mithun Technologies Snapshot Nexus Repository </name>

<url>http://172.31.42.154:9980/mithuntechnologies/repository/canarabank-snapshot/</url>

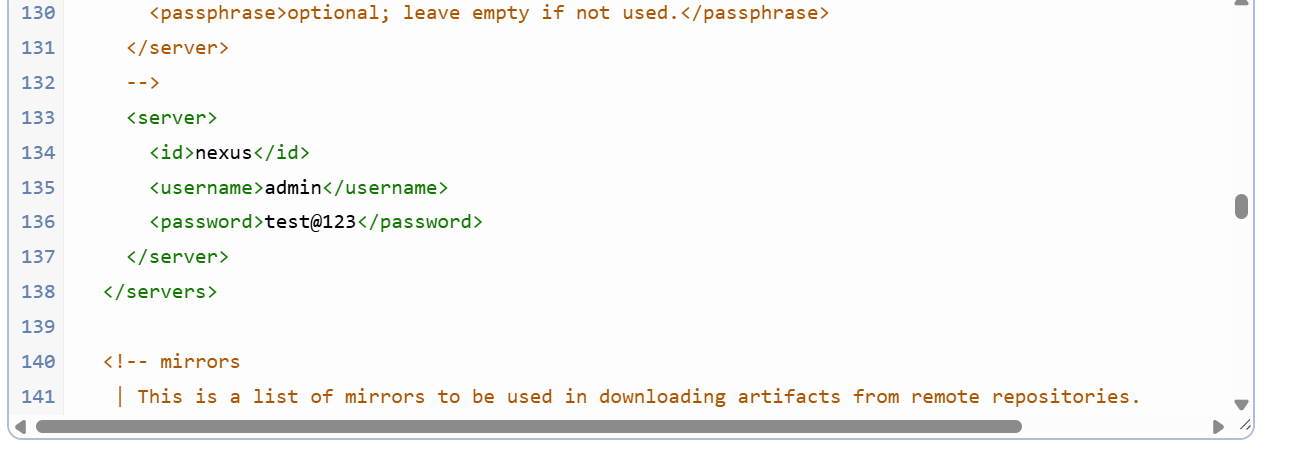
</snapshotRepository>

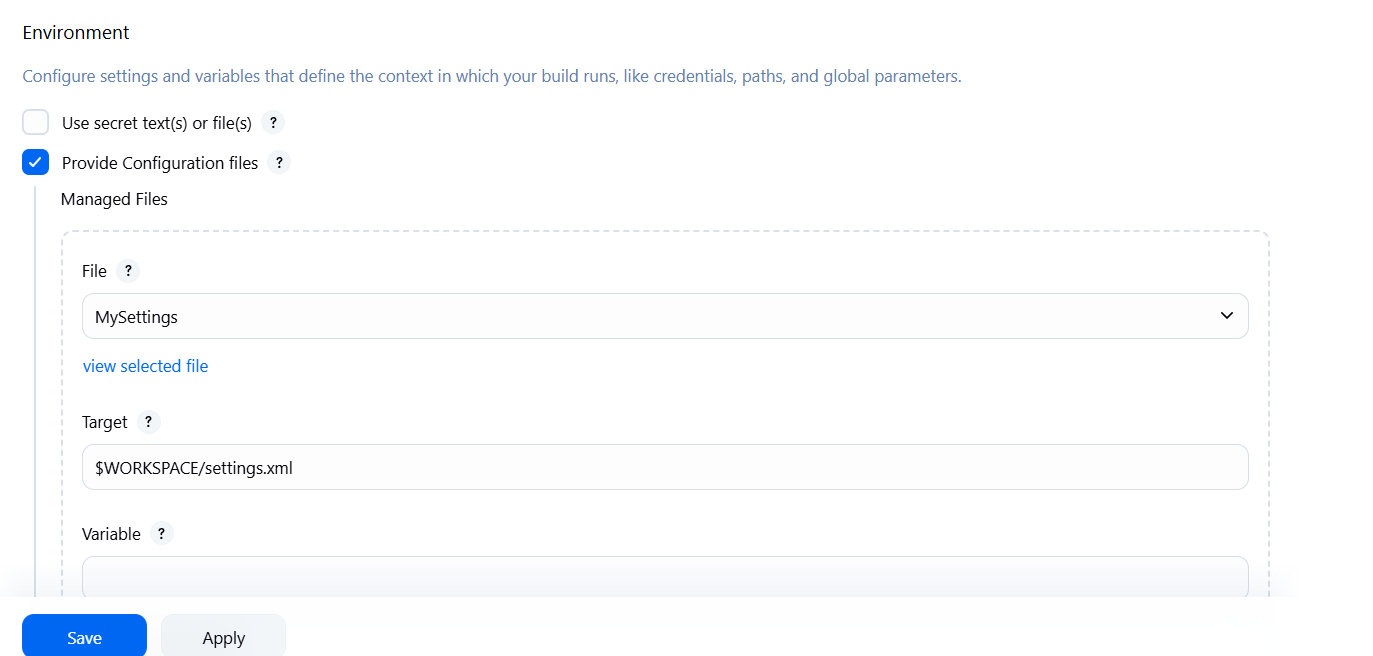
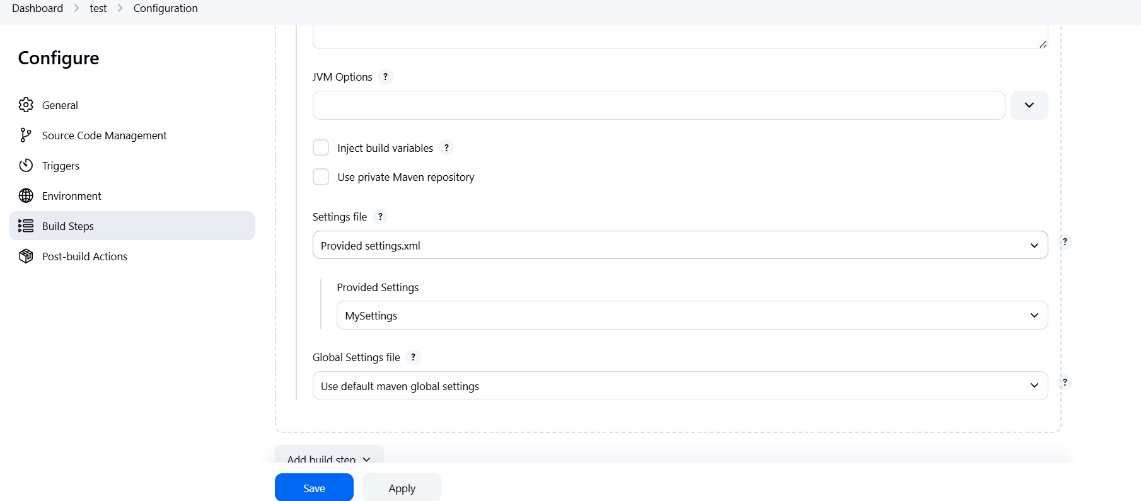
</distributionManagement>  
  
  
  
  
under goals add deploy  
  
  
  
  
  
need to add nexus credentials in setting.xml since i am using maven of Jenkins use below steps if it is installed in vm then go to the path and change  
  
Make sure the **Config File Provider Plugin** is installed:

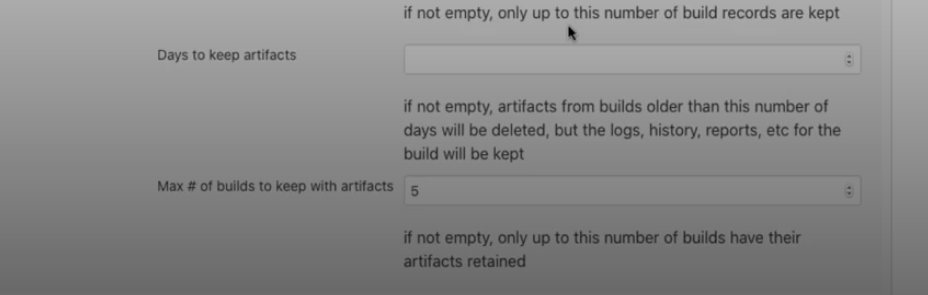
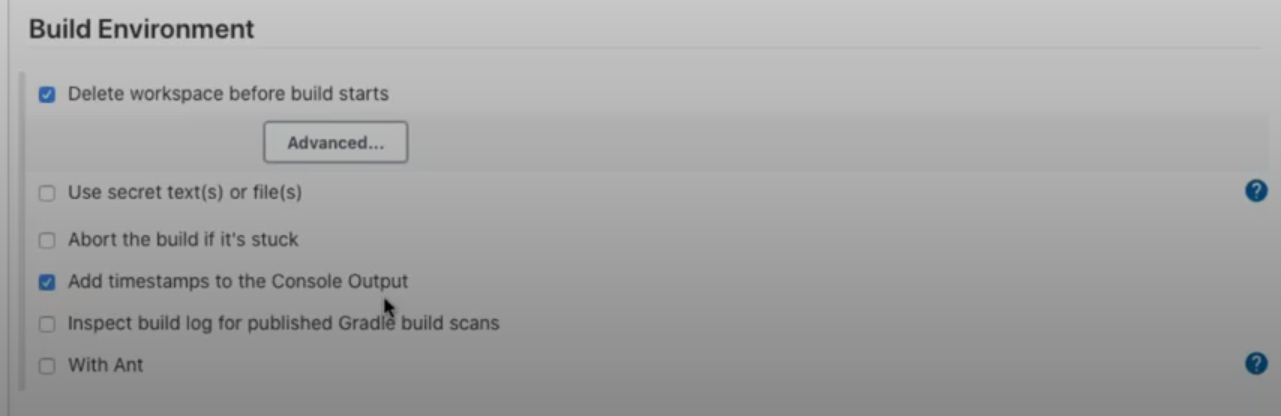
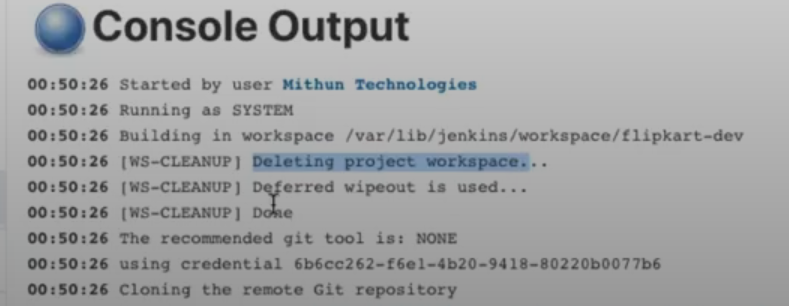
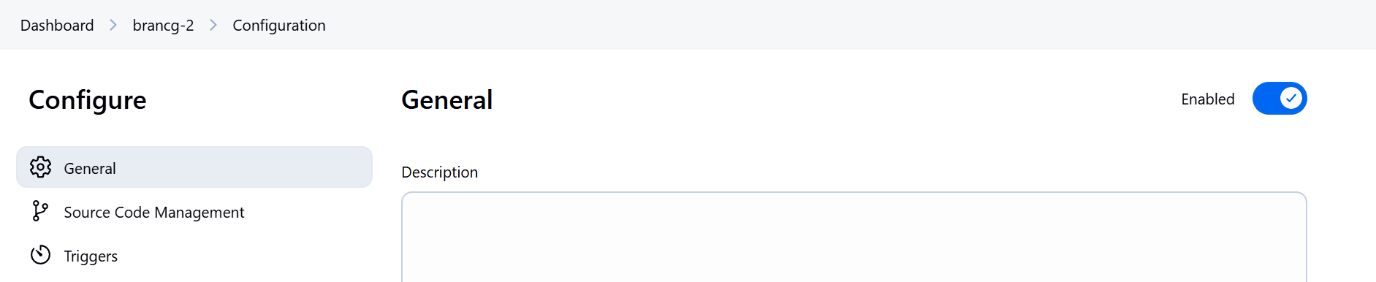
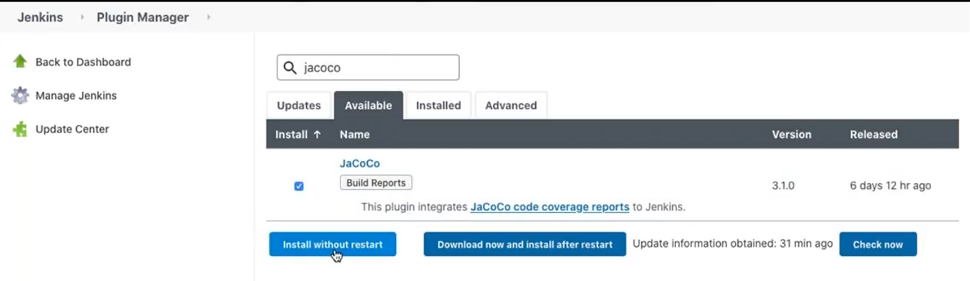
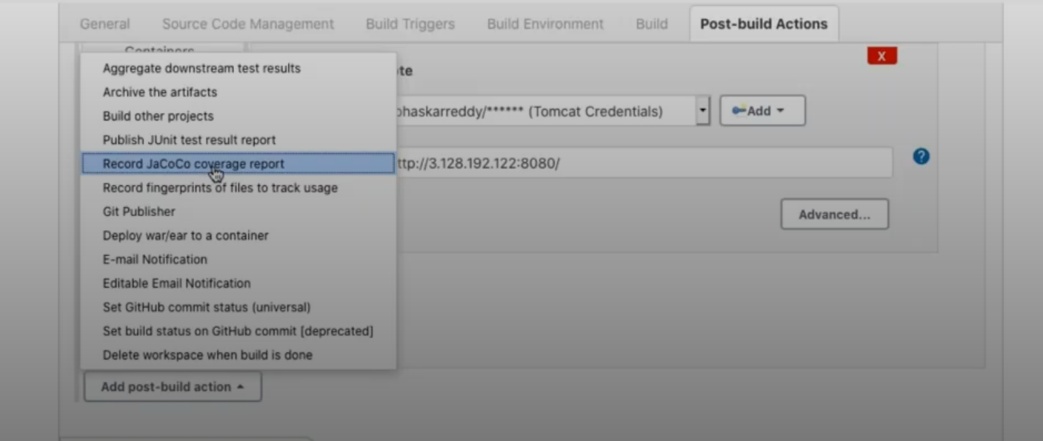
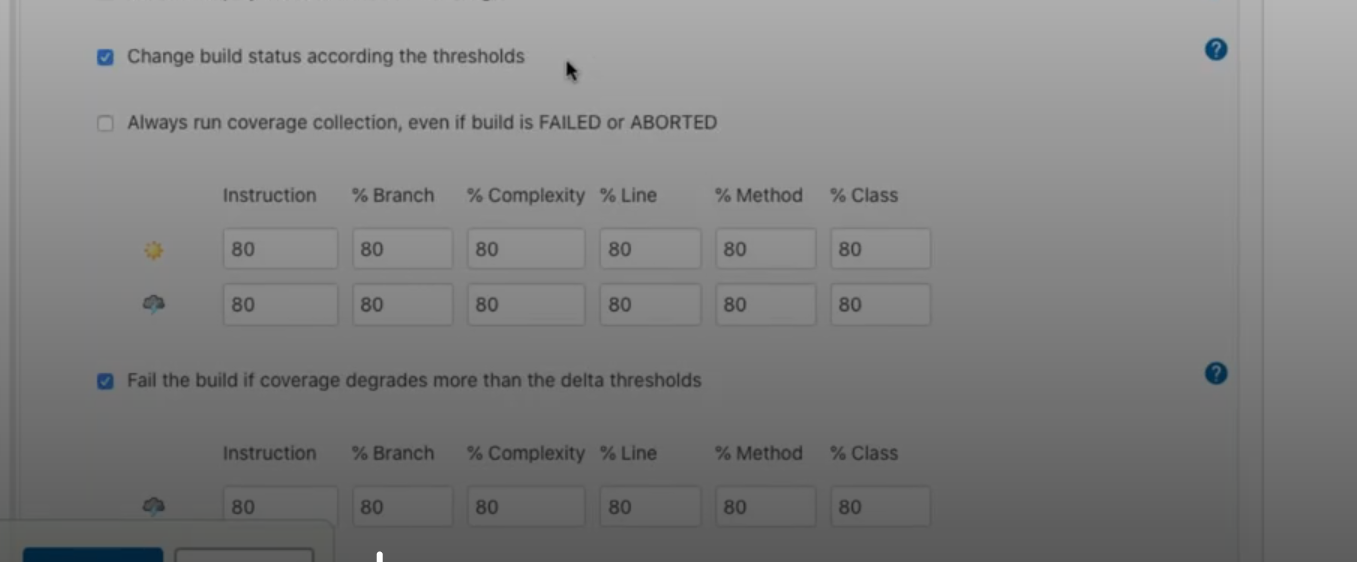
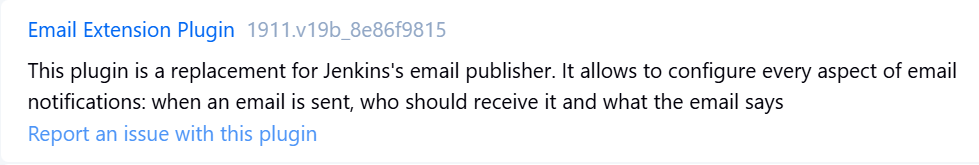
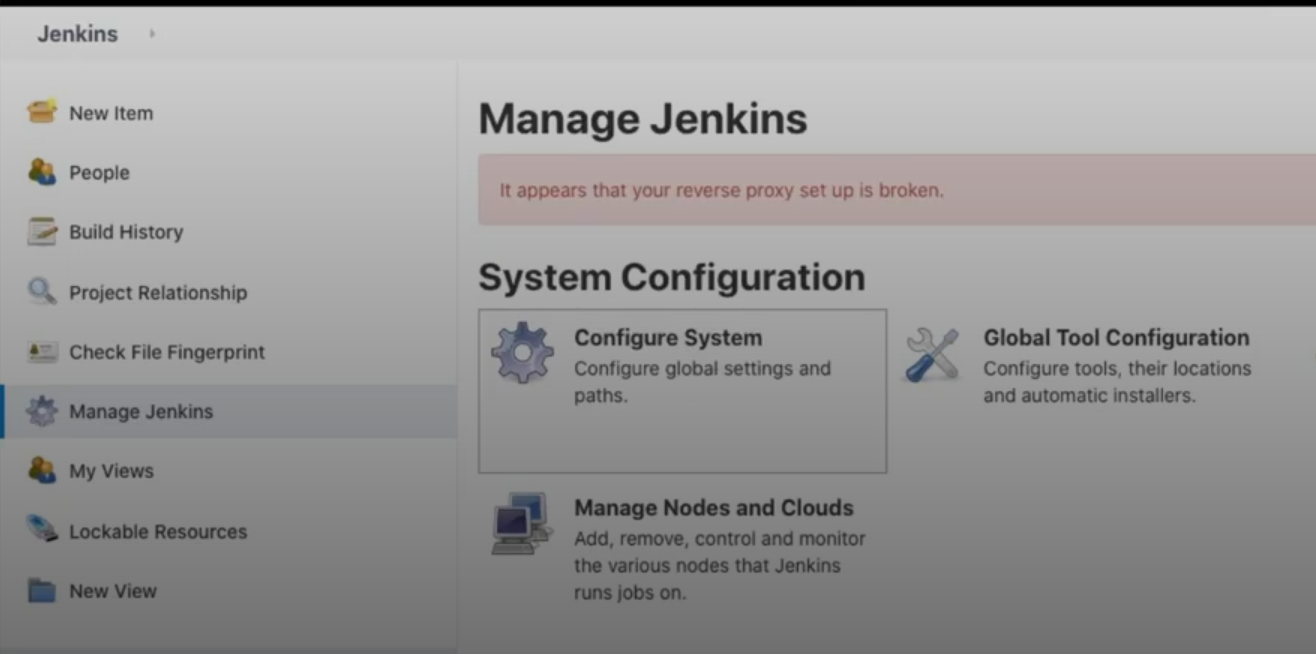
1. Go to **Manage Jenkins** → **Plugins** → **Installed** tab
2. Search for **"Config File Provider"**
3. If not found, go to **Available** tab, search and install it.

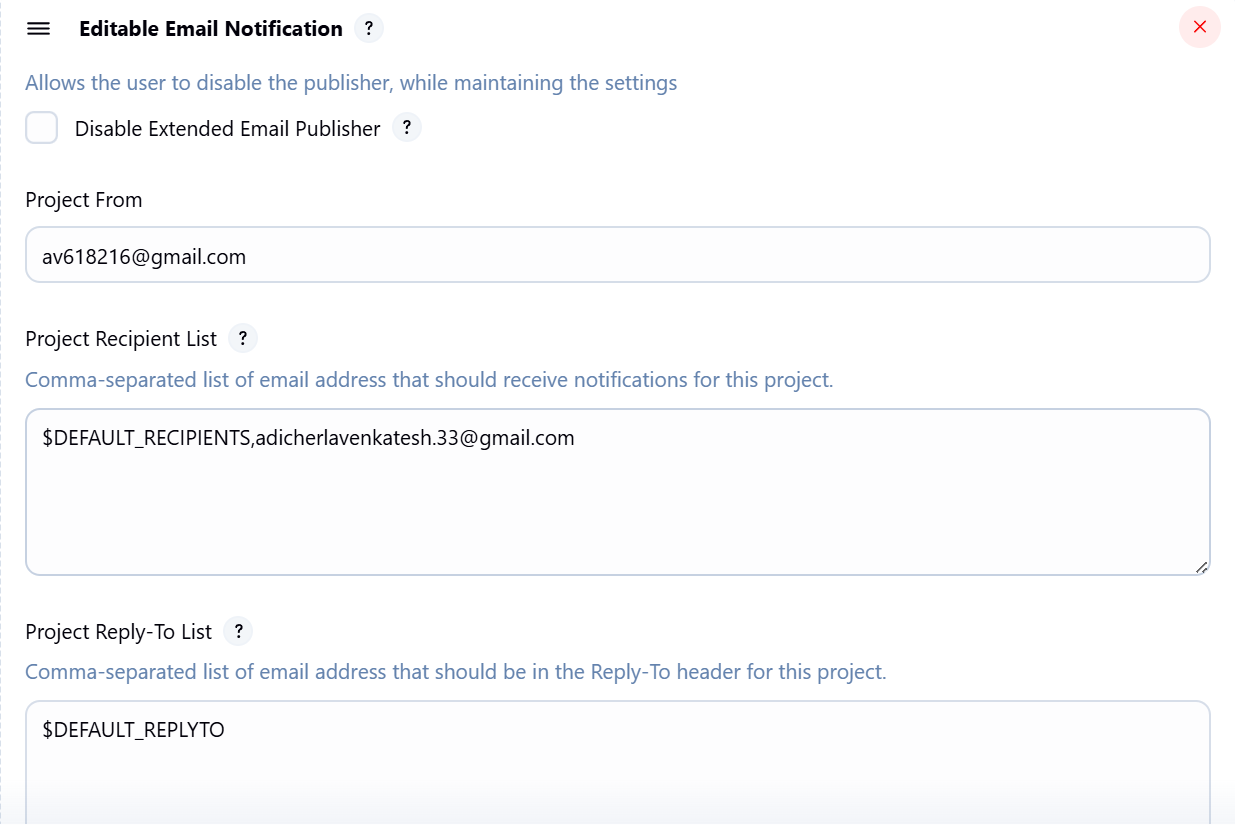
**Step 1: Create the settings.xml in Jenkins (Global Config)**

1. Go to **Manage Jenkins** → **Managed Files**  
   URL: http://<your-jenkins-host>/configfiles/
2. Click **"Add a new Config"** → Choose **"Maven settings.xml"**
3. Fill in:
   * **ID**: custom-maven-settings (or anything you prefer)
   * **Name**: Custom Maven Settings
   * **Comment**: (optional)
   * **Content**: Paste your full settings.xml contents here, e.g.:

Id nexus should match with id in the pom.xml -- paste total settings.xml  
  


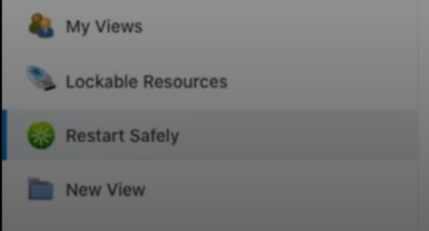
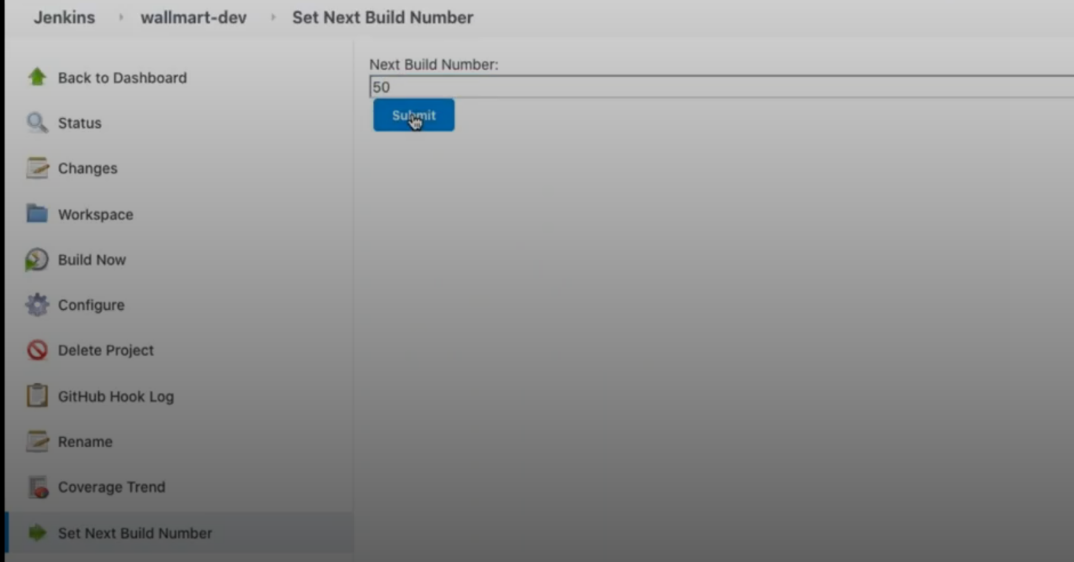
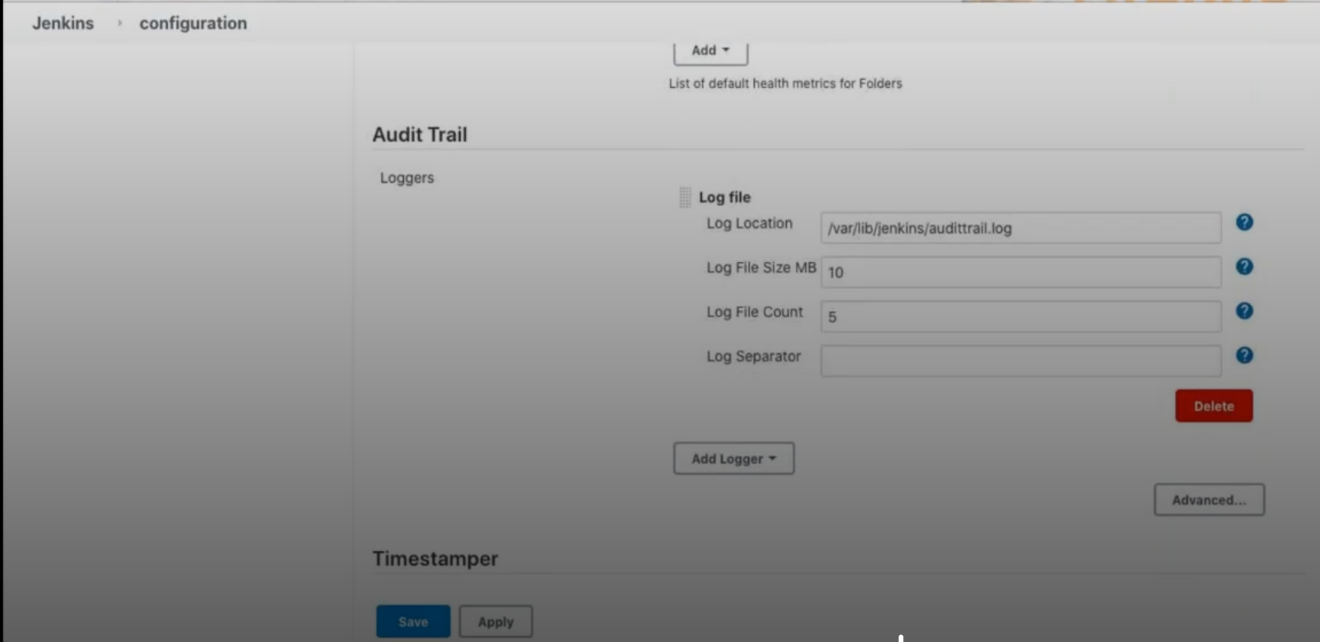
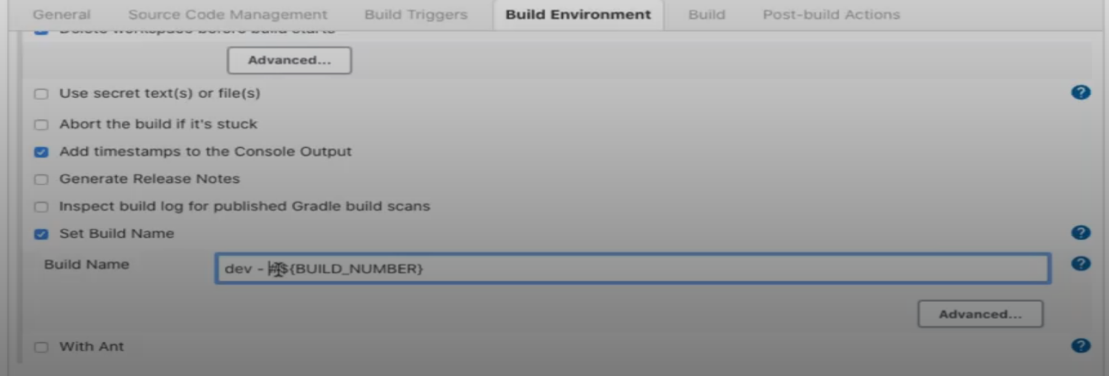
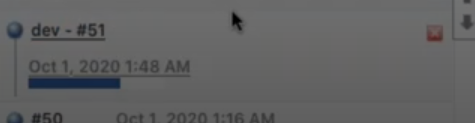
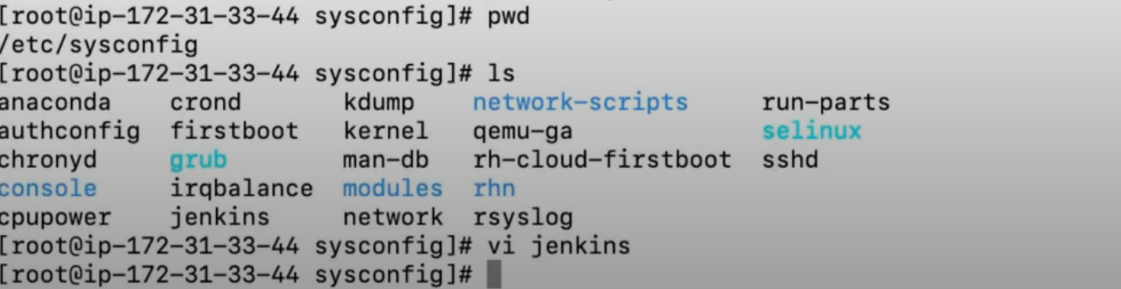
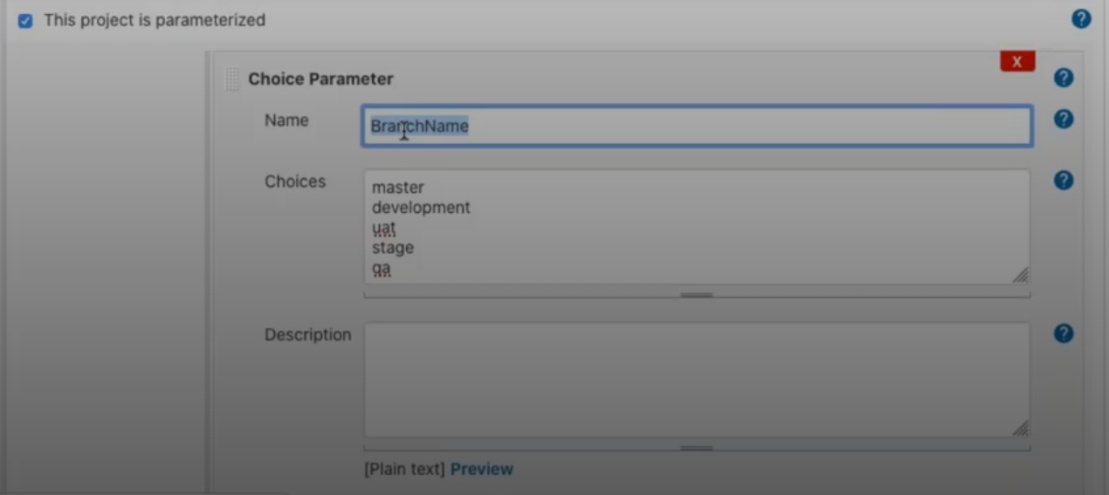
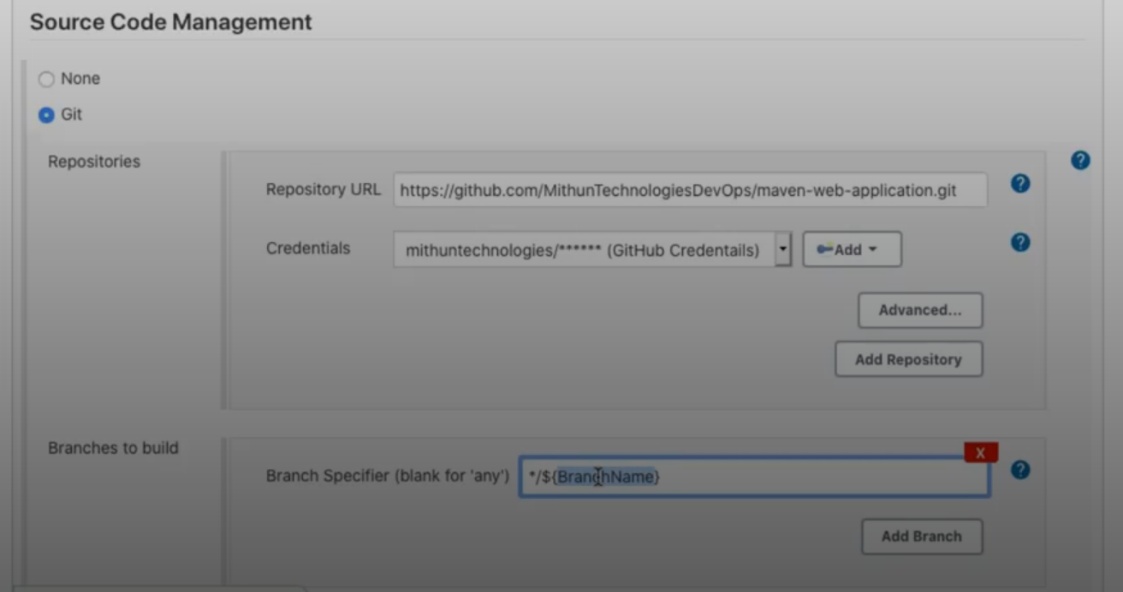
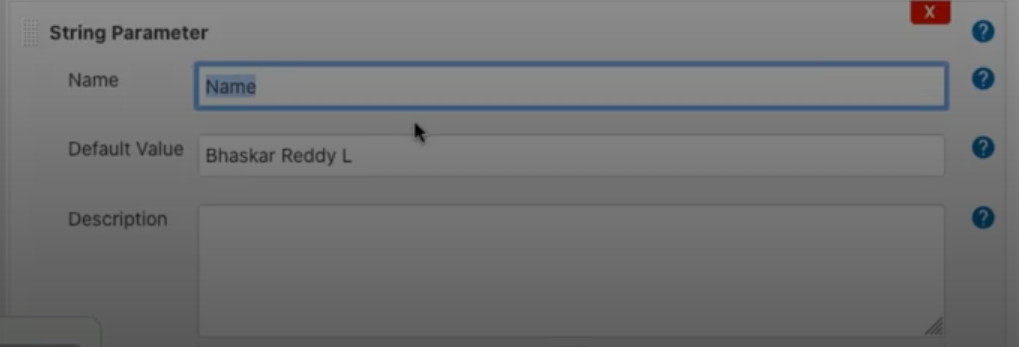
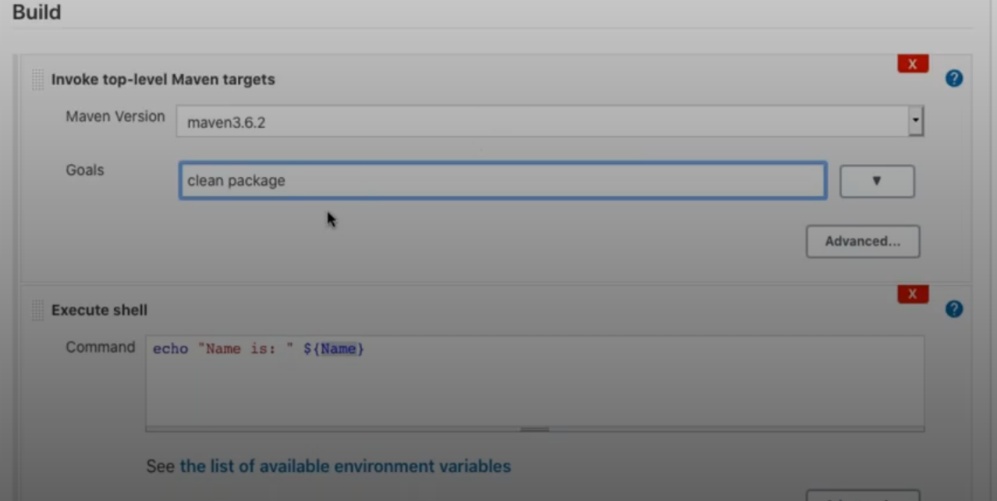
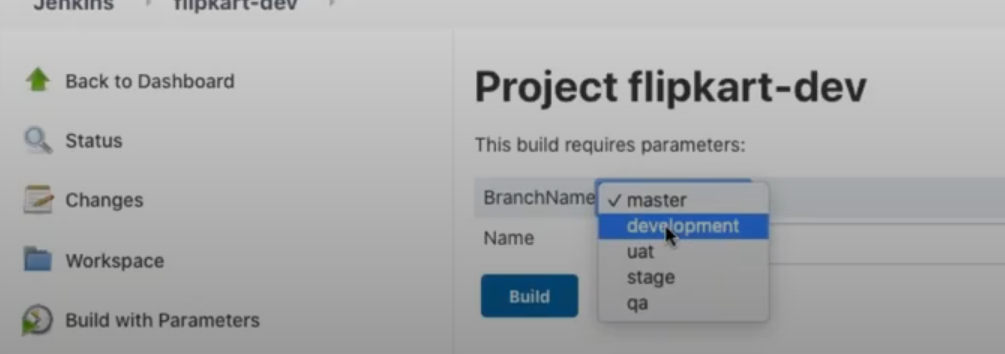
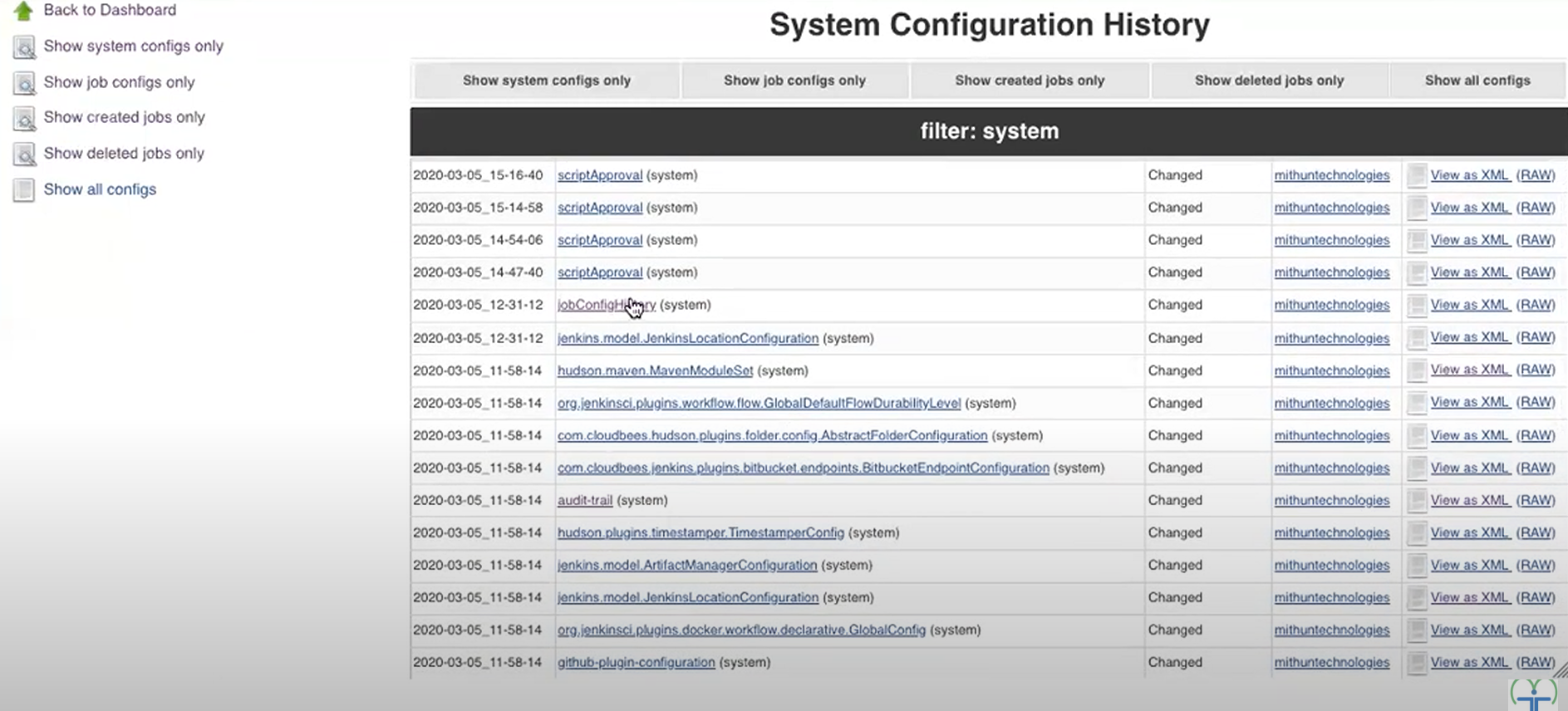
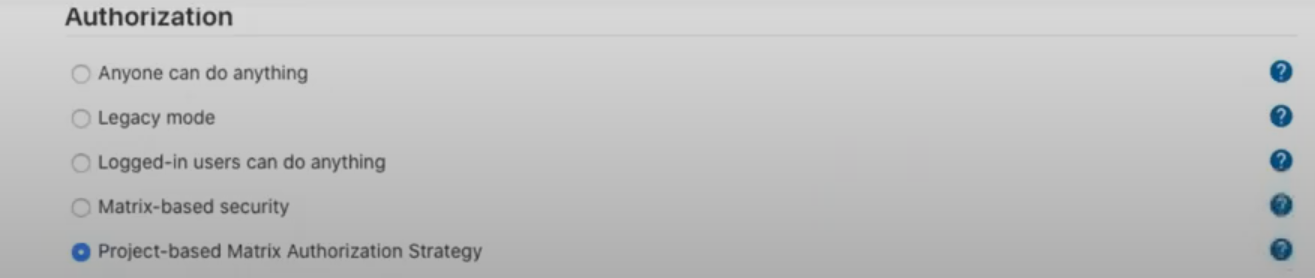
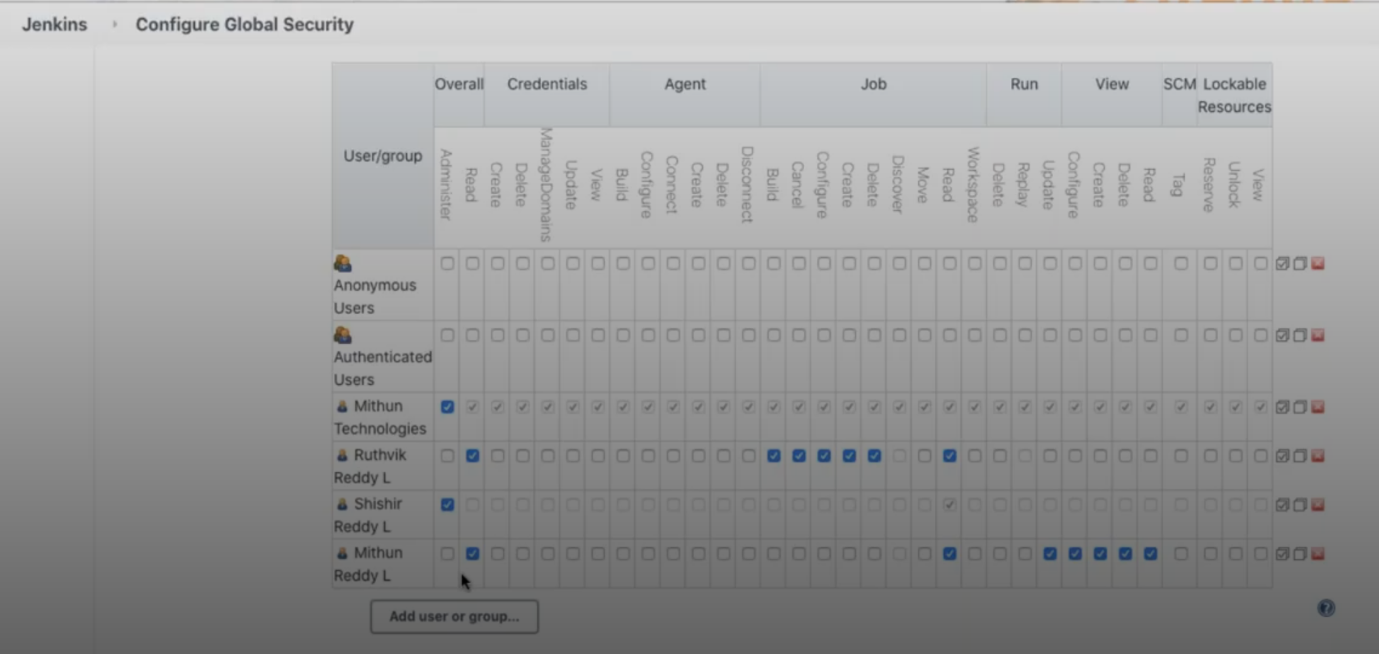
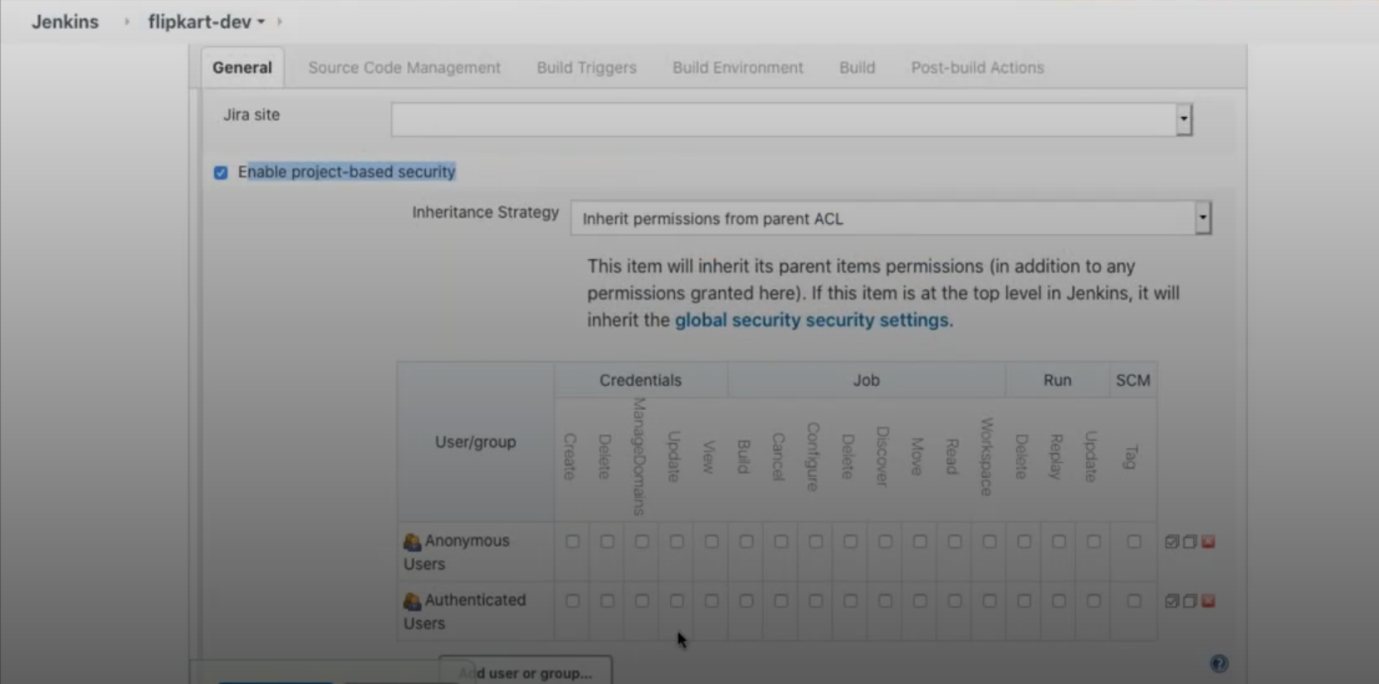
Now in the job configure use the setting file you have created  
this Environment step is not mandatory tested it is working without this step as-well  
  
  
  
  
  
  
now in maven advanced steps in settings file give the setting .xml which we have created  
  
  
  
  
5.To deploy in Tomcat

install the plugin deploytocontainer  
  
  
/opt/bitnami/tomcat/conf/tomcat-users.xml  
  
  
6.Discard old builds and artifacts up to 5days on below  
  
  
  
if you want to maintain the packages of old build mention on above  
  
output  
   
  
  
7.Delete workspace before build starts & Adding timestamps to console o/p   
if of enable the below option it will delete the workspace and pull the code freshly else it will pull the update code only  
output  
  
  
8.ENABLE/DISABLE  
Mainly used wehn nexus,tomcat,sonarqube etc..schecdule maintainace happened  
if we disable the project then we can run the build we need to enable and build it.  
  
  
  
  
  
  
  
  
  
  
**9.JACOCO – Plugin**  
  
install jacoco plugin for code coverage  
  
  
  
under postbuild Action cick  
  
  
  
  
and give the parameters  
if you enable the below checkboxes the builds will fail if the code coverage dosen’t meet  
1.change build status accordingly the threshold   
2. fail the build if the coverage degrades more that the delta thresholds  
  
  
  
  
10. SMTP Email integration for Alerts  
  
Details needed to integrate with smpt server  
  
smtp.gmail.com  
gmail  
password  
25 - non-secure  
465 – secure  
  
  
  
  
  


* Job-->configure-->post build action--> **Editable Email Notification** and all the details required  
    
    
    
  11. Jenkins Directory Structure  
    
  /var/lib/jenkins -- jenkins home directory  
  /var/log/jenkins -- jenkins log directory  
    
  **On Linux/Ubuntu:** /var/lib/jenkins (or ~/.jenkins in some cases)
* You can always find the exact JENKINS\_HOME location by going to **Manage Jenkins > System** in your Jenkins UI and looking under the "Home directory" heading.
* Here's a breakdown of the key directories and files you'll typically find within JENKINS\_HOME:
* **config.xml**: This is the main Jenkins configuration file, storing global settings for your Jenkins instance.
* **credentials.xml**: Stores credentials (encrypted).
* **fingerprints/**: This directory is used by Jenkins to keep track of artifact fingerprints, which helps identify if a specific artifact has been used in different builds or jobs.
* **jobs/**: This is one of the most important directories. It contains a subdirectory for each Jenkins build job you've configured.
  + **jobs/<JOB\_NAME>/**: Inside each job's directory, you'll find:
    - **config.xml**: The configuration file specific to that particular job.
    - **builds/**: This directory stores the build history for the job.
      * **builds/<BUILD\_ID>/**: Each build has its own subdirectory, named with a timestamp (e.g., 2025-06-28\_22-30-00). Within this directory:
        + **build.xml**: A summary of the build results.
        + **log**: The console output/log file for that specific build.
        + **changelog.xml**: Details about the changes that triggered the build.
        + Any artifacts archived during the build (if configured).
      * **permalinks/**: Contains symbolic links like lastSuccessfulBuild, lastStableBuild, etc., pointing to the respective build directories.
    - **workspace/**: This is the working directory where Jenkins performs the build operations for that job. It's where the source code is checked out, and any files generated by the build process are stored. This workspace is typically reused for successive builds of the same job.
* **plugins/**: Contains all the plugins installed on your Jenkins instance. Each plugin will have its own .jpi file (or a directory for exploded plugins).
* **secrets/**: Stores sensitive information, including the initial admin password and keys used for encrypting credentials.
* **updates/**: Internal directory used by Jenkins to store information about available plugin updates.
* **userContent/**: A directory where you can place custom content (e.g., HTML files, images) that can be accessed via <http://your_jenkins_url/userContent/>.
* **users/**: If you're using Jenkins' native user database, user accounts and their configurations are stored here.
* **war/**: This directory contains the expanded web application if Jenkins is run as a standalone application.

**Understanding this directory structure is crucial for:**

* **Troubleshooting:** Locating logs, configuration files, and build artifacts.
* **Backup and Restore:** Knowing which directories to back up to preserve your Jenkins instance.
* **Migration:** Moving a Jenkins instance to a new server.

**Disk Space Management:** Identifying areas that might consume a lot of disk space (e.g., builds/ and workspace/).  
  
  
12. Plugin Management  
  
1.**Deploy to container,Deploy Weblogic , websphere Deployer** – to deploy in tomcat , jboss,weblogic,websphere  
2. **Maven Integration** – to create job in Mave project type  
3. **Safe restar**t -  
  
   
  
without installing plugin use below  
<http://jenkins-test.apps.684b1059b9e77e1826ddd10e.ocp.techzone.ibm.com/restart> -- it will forcefully restart the jenkins even when the jobs are running  
[http://jenkins-test.apps.684b1059b9e77e1826ddd10e.ocp.techzone.ibm.com/safeRestart](http://jenkins-test.apps.684b1059b9e77e1826ddd10e.ocp.techzone.ibm.com/restart) - might wait till the jobs completed or hold the jobs  
  
4. **Next build number** - if you want to change the next build number  
  
  
  
5. **JACOCO** - to genereate the sonarqube report and want to stop the deployment if the build fail and meet the coverage report  
  
6 **SonarQube Scanner** - for sonarqube scanner within jenkins install this plugin and do setup  
  
7. **AuditTrail -** install this plugin for who performed which action to check  
under manage jenkins --> system and use below setup  
  
  
  
  
8. **Scheduled build** – to schedule build other than poll scm/Build-prediatically  
  
  
  
  
  
  
  
  
9. **Artifactory Plugin** - if we want to use nexus in jenkins install and do setup  
  
  
  
  
  
  
10. **Build Name and Description Setter -** if you want to provide name to the build is this plugin  
  
  
  
  
  
  
11. **Blue ocean** - for another UI of Jenkins  
  
12. **convert to pipeline** -- if you have build a job with normal freestyle and want to convert to pipeline you can use the above plugin  
  
13**. Stage view** -- to see each stage staus  
  
  
  
Publish Over SSH  
SSH Agent -- to do ssh from one server to another  
Thin backup  
jobconfig history plugin  
job import plugin  
  
**13.JENKINS FILE**  
if you want to change the default port number , default home directory of jenkins   
/etc/sysconfig - RedHat  
/etc/default - Ubuntu  
  
systemctl restart jenkins  
  
  
   
  
14. Build with Parameter  
if we chnage the branches frequently then we can use build with parameters  
  
  
  
  
  
  
  
Output  
  
  
  
  
 15.Create View  
If we want to segregate out jobs like prod jobs in prod section and Qa jobs in Qa section we can do that but end of the day all job will be available in ALL jobs  
  
  
  
  
  
  
  
  
16. Jenkins Security  
  
To create users  
Manage jenkins --> Mange users  
  
--> By default when you create a user. that user will have admin access, need to change and provide what level of access he needed.  
  
  
--> here we are using Jenkins own database but in realtime we will use like LDAP  
if we configure with the LDAP then user creation will be taken care in LDAP  
  
**17. Job config history plugin** – if we install this plugin we can see all the history who deleted,when created etc  
even we can restore the jobs which we have delete with this plugin  
  
  
 **18.Job import plugin**if there are two Jenkins servers A and B and if you want to import the jobs from server A to the server B then you can use job import plugin and achive it  
 **Plugin needs to install**  
Matrix Authorization Strategy  
Role-based Authorization  
LDAP - for integration with LDAP  
  
  
  
**Project-based Matrix Authorization Strategy (overall\_jenkins)**  
  
  
  
   
**Project-based Security(specific job level access)**  
job-->configure  
if there are two user Venky and kanki and there are amazon and flipkart jobs   
venky should have access to only amazon jobs  
kanki should have access to flipkart jobs only  
we can achieve that with the help of Project-based Security  
  
  
  
**17. Pipeline**  
  
  
  
**git pull - use snippet generator**  
  
stage('Git Pull') {

steps {

// This step pulls your code from the specified Git repository

// using the provided credentialsId.

git credentialsId: '68070b12-9864-43da-8415-35aa2e03dcc5', url: 'https://github.com/a-venkatesh-1998/maven-web-application.git'

}  
  
**Maven Build**  
  
**Option 1**  
  
**---> here in the pipeline tool name should match with the name in below pic how you defined**

stage('Prepare Maven') {

def mavenHome = tool 'maven'

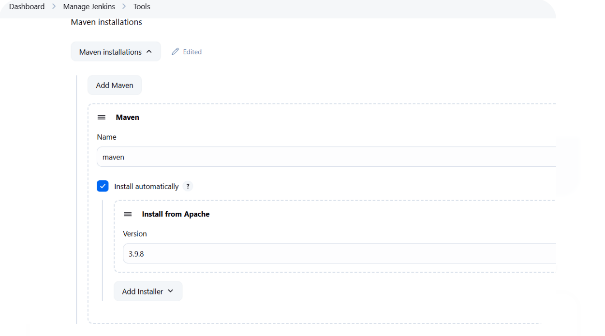
env.M2\_HOME = mavenHome

env.PATH = "${mavenHome}/bin:${env.PATH}"

}

stage('Maven Build') {

sh "mvn clean package"

}  
  
**Option 2**  
  
  
  
**---> here in the pipeline name should match with the name in above pic how you defined**  
  
pipeline {

agent any

**tools {**

**maven 'maven'**

**}**

stages {

stage('Git Pull') {

steps {

git credentialsId: '68070b12-9864-43da-8415-35aa2e03dcc5', url: 'https://github.com/a-venkatesh-1998/maven-web-application.git'

}

}

stage('Maven Build') {

steps {

sh 'mvn clean package'

}

}

}

post {

always {

echo 'Pipeline finished.'

}

success {

echo 'Build successful!'

}

failure {

echo 'Build failed!'

}

}

}  
  
**option 3**  
  
  
  
  
**SonarQube Report**  
  
 stage('SonarQube Report'){

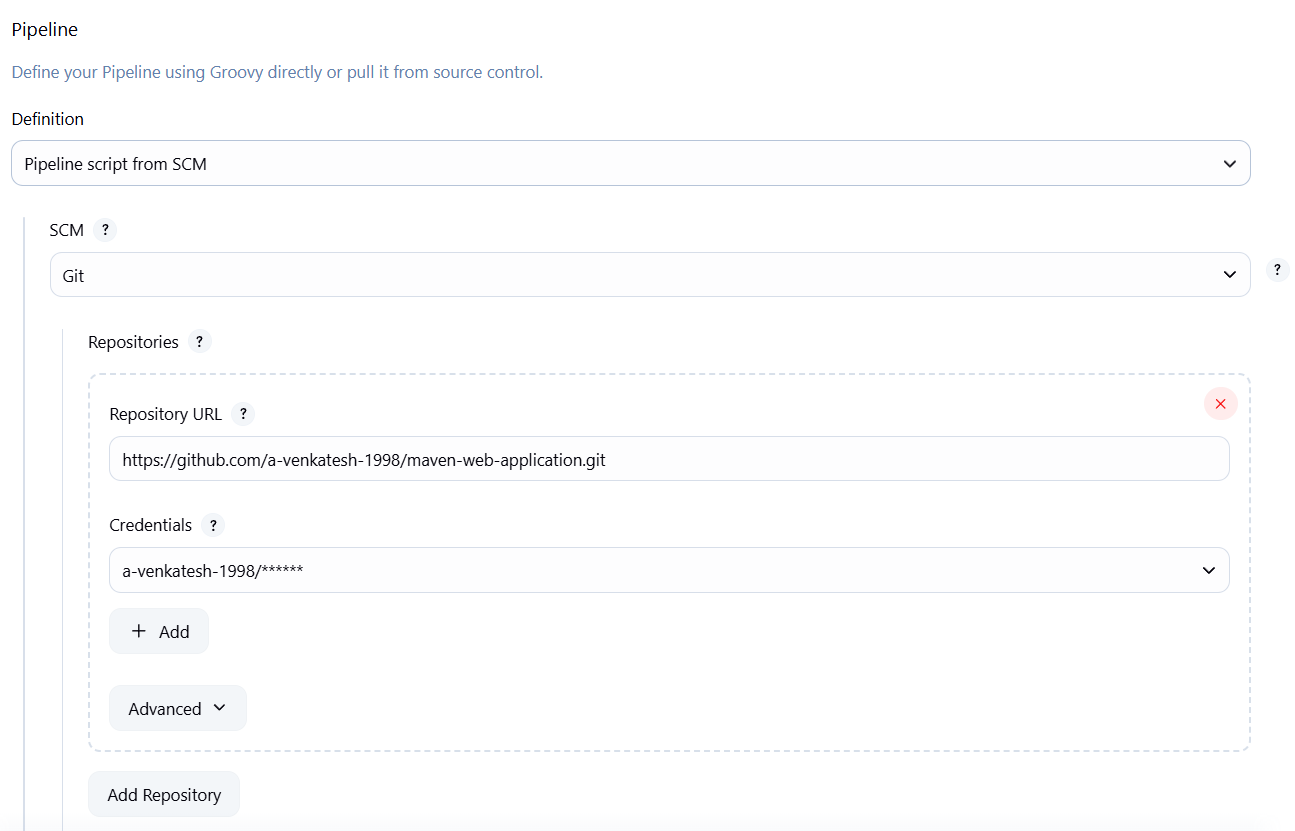
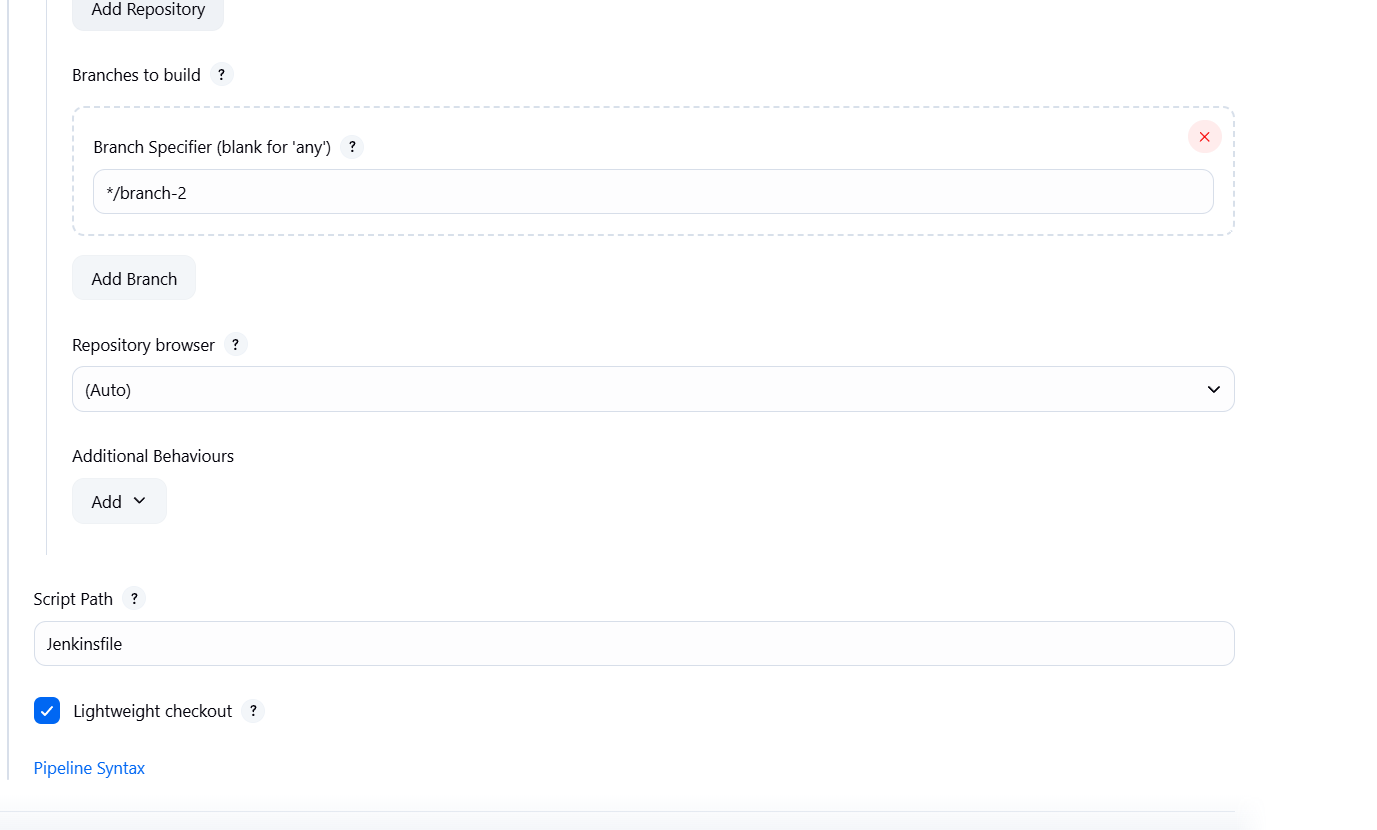
sh "mvn sonar:sonar"

}  
  
  
  
**Nexus Upload**  
  
 stage('Upload to Nexus') {

// Keep configFileProvider here as per request

configFileProvider([configFile(fileId: '6088a863-6ecc-4dc2-b790-d7b30fedfe4e', variable: 'MAVEN\_SETTINGS')]) {

sh "mvn deploy --settings $MAVEN\_SETTINGS"

}  
  
**To build pipeline from the git repo – jenkinsfile should present in that repo**  
  
  
  
  
**Pipeline code**  
node {

stage('Git Pull') {

git credentialsId: '68070b12-9864-43da-8415-35aa2e03dcc5', url: 'https://github.com/a-venkatesh-1998/maven-web-application.git'

}

stage('Prepare Maven') {

def mavenHome = tool 'maven'

env.M2\_HOME = mavenHome

env.PATH = "${mavenHome}/bin:${env.PATH}"

}

stage('Maven Build') {

// Removed configFileProvider as per request

sh "mvn clean package"

}

stage('SonarQube Report') {

// Removed configFileProvider as per request

sh "mvn sonar:sonar"

}

stage('Upload to Nexus') {

// Keep configFileProvider here as per request

configFileProvider([configFile(fileId: '6088a863-6ecc-4dc2-b790-d7b30fedfe4e', variable: 'MAVEN\_SETTINGS')]) {

sh "mvn deploy --settings $MAVEN\_SETTINGS"

}

}

}  
  
if you configure the sonarqube in Jenkins itself use below stage



**20.Multibranch pipeline**  
  
  
A Jenkins Multibranch Pipeline is a powerful feature in Jenkins that automates the creation and management of continuous integration/continuous delivery (CI/CD) pipelines for multiple branches within a single source code repository.

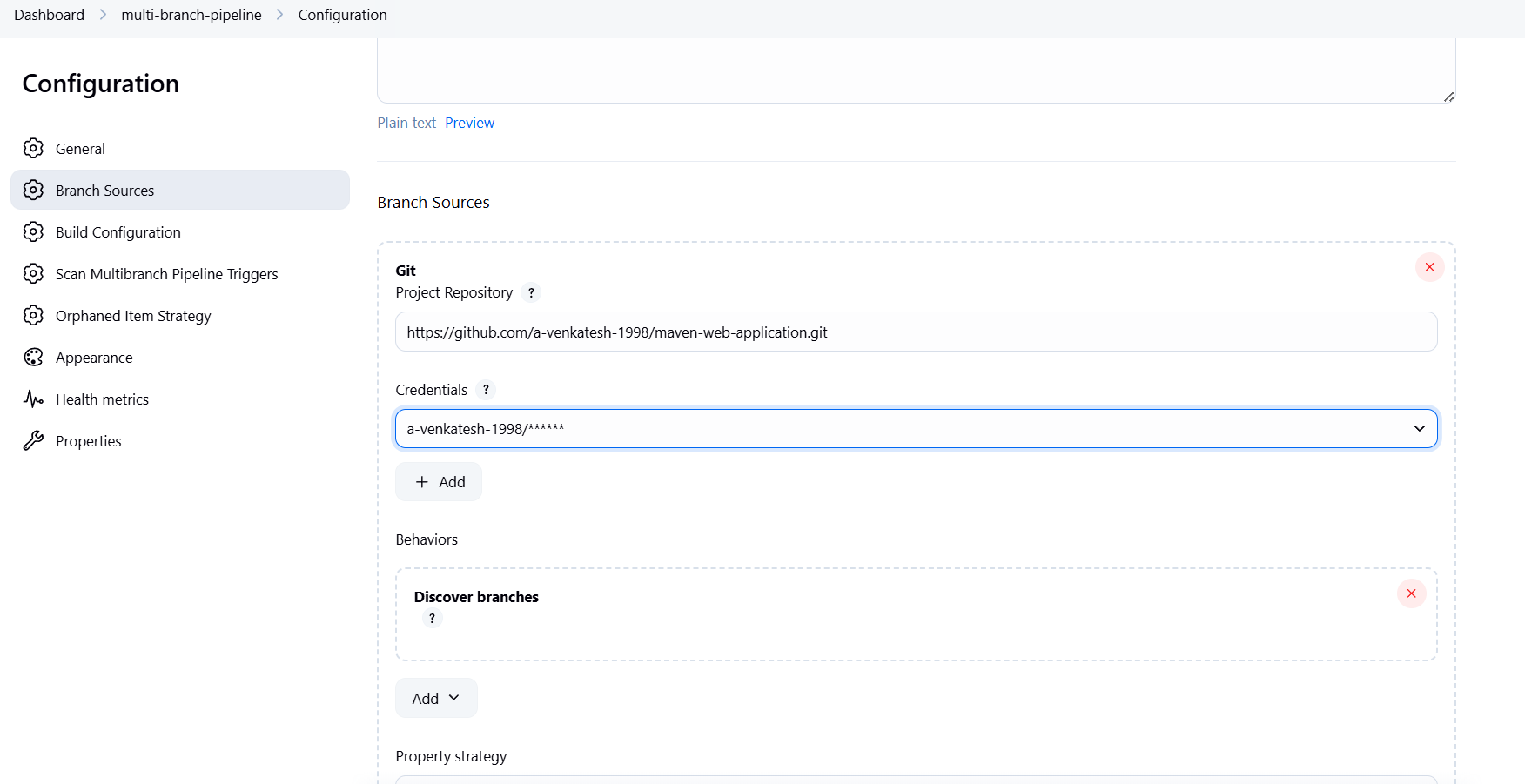
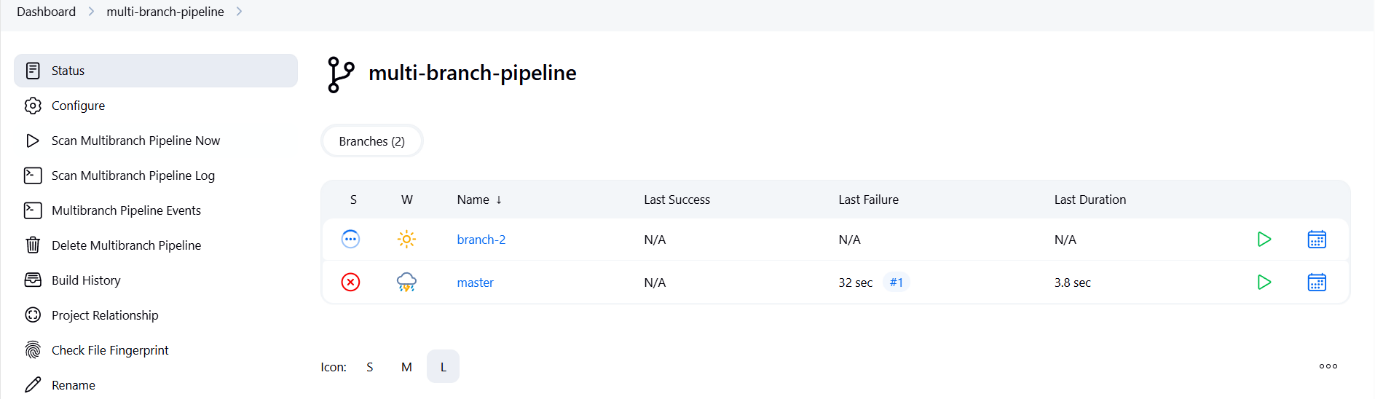
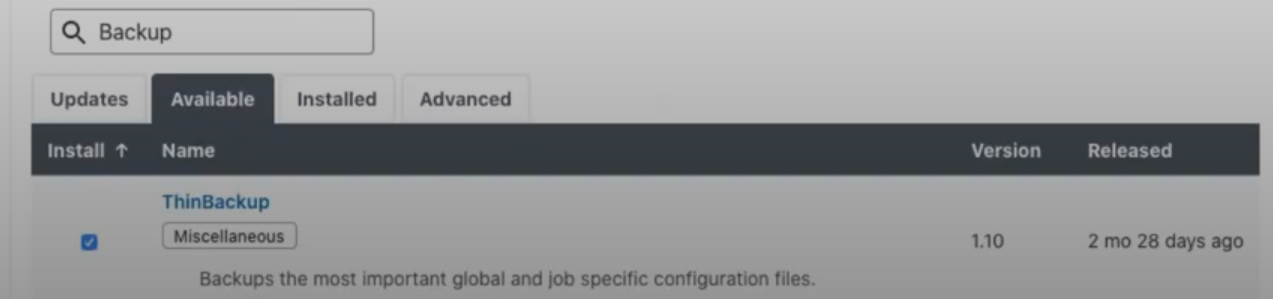
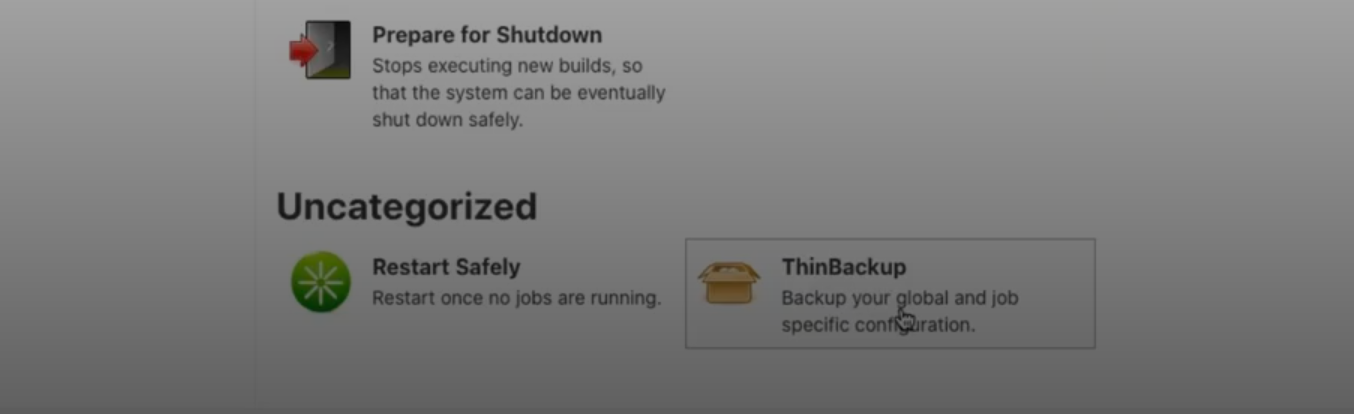
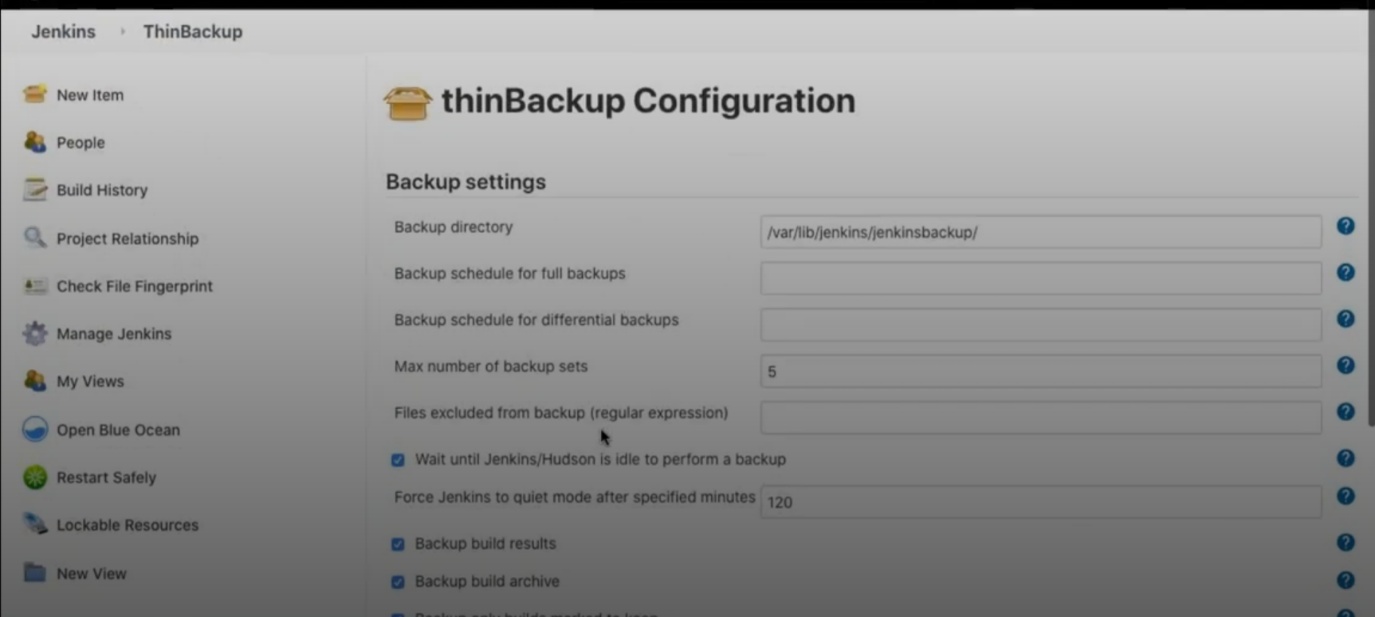
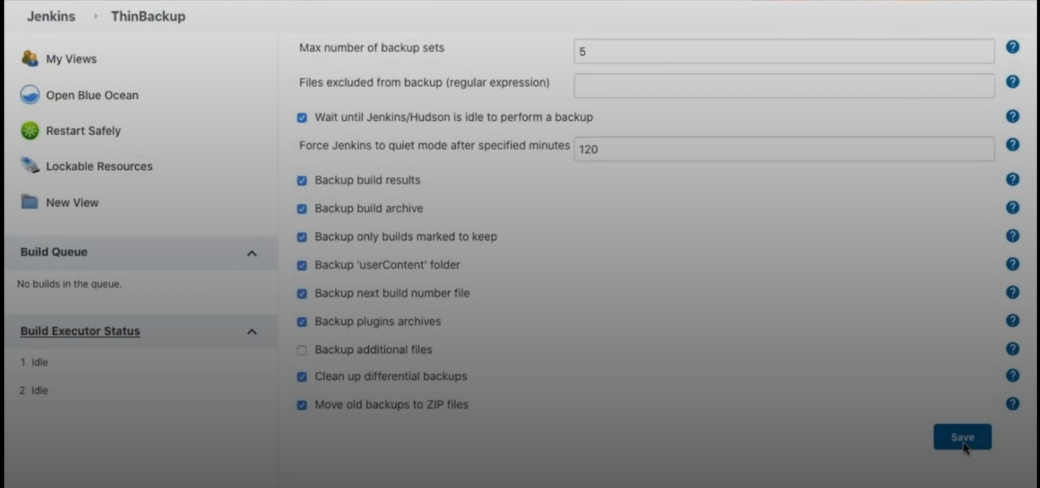
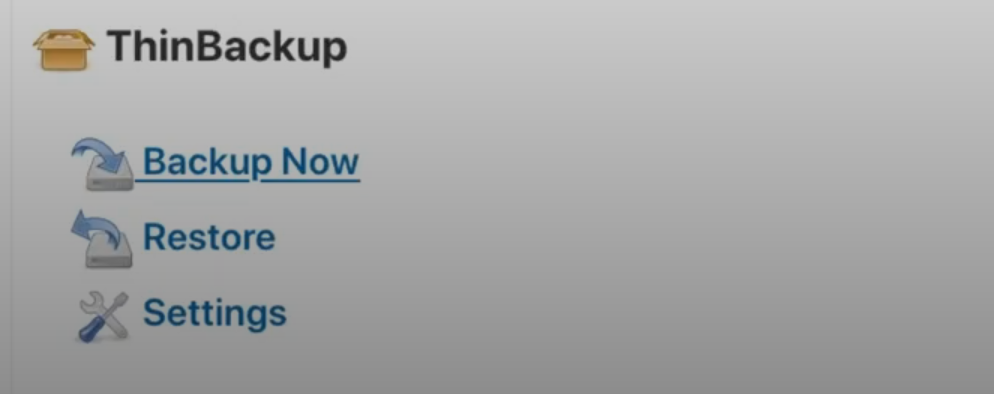
Here's a breakdown of what it is and why it's so useful:

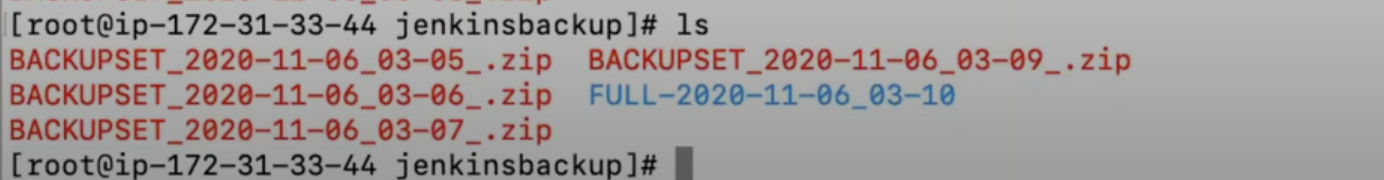
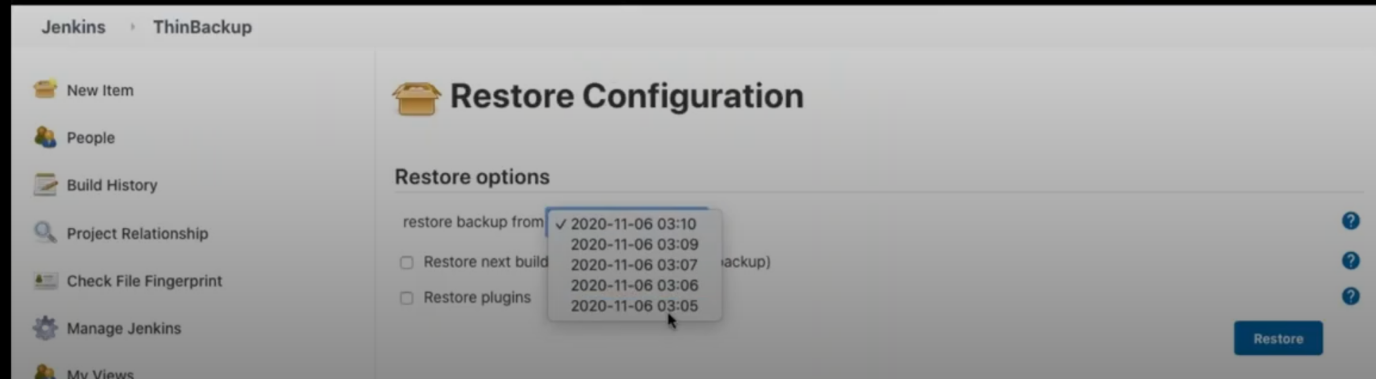
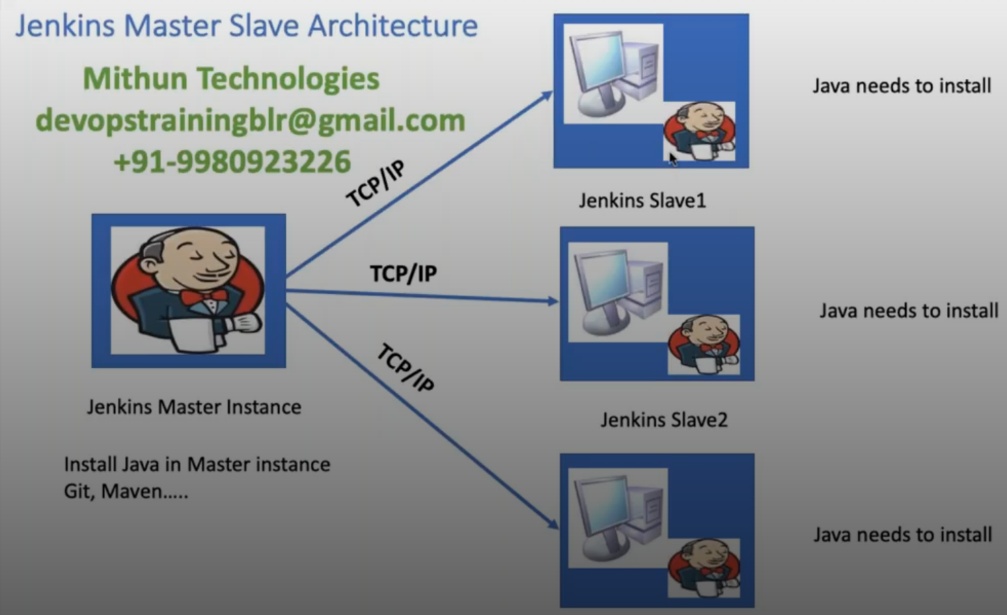
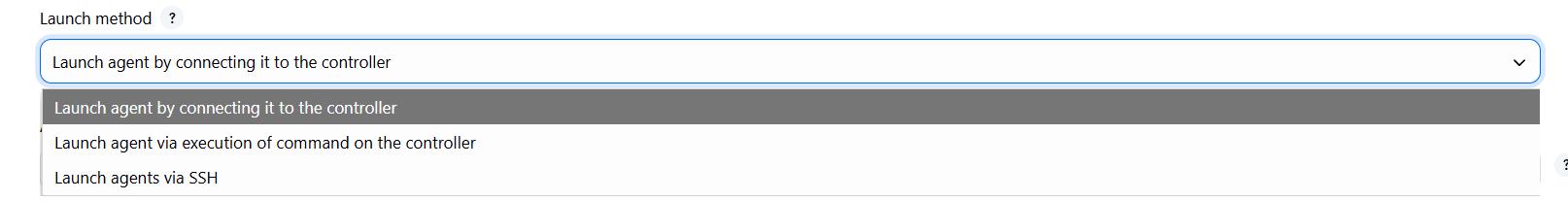
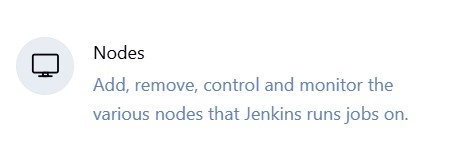
**What is a Multibranch Pipeline?**

Imagine you have a Git repository with several branches: main, develop, feature-X, bugfix-Y, etc. Traditionally, if you wanted a Jenkins pipeline for each of these branches, you'd have to create a separate Jenkins job for every single one. This quickly becomes cumbersome and hard to manage, especially as your project grows and branches are created and deleted frequently.

A Multibranch Pipeline solves this by:

* **Automatically discovering branches:** It scans your source code repository (e.g., Git, GitHub, Bitbucket, GitLab) and automatically detects all existing branches that contain a Jenkinsfile.
* **Creating a pipeline for each branch:** For every discovered branch with a Jenkinsfile, Jenkins automatically creates a dedicated pipeline job. This job uses the Jenkinsfile present in that specific branch to define its build, test, and deployment stages.
* **Managing pipelines lifecycle:** When a new branch is created, Jenkins automatically creates a new pipeline job for it. When a branch is deleted, Jenkins can be configured to automatically remove its corresponding pipeline job.
* **Supporting Pull Requests/Merge Requests:** With appropriate plugins (e.g., GitHub Branch Source, Bitbucket Branch Source), Multibranch Pipelines can also detect and build Pull Requests or Merge Requests, allowing you to run pre-merge checks and ensure code quality before it's integrated into a main branch.

**implementation**  
  
  
  
  
**19.ThinBackup**  
  
Install plugin ThinBackup  
  
  
  
  
  
go to manage jenkins  
  
  
  
create a directory jenkinsbackup and provide jenkins owner permissions and give the path in backup directory  
and provide max number of backup sets  
if you provide backup schedule for differential backups - \*\*\*\*\* --> every minute it will take one backup  
enable what are all needed for the backup  
c  
save the settings and click on backup now  


In the server you and see the backups  
  
  
  
  
  
  
  
click on the backup and restore what you need  
  
  
  
  
  
  
  
  
  
  
  
  
  
20. Jenkins master-slave Arch  
  
  
  
  
plugins need to install for below options  
Command Agent Launcher Plugin  
**SSH Build Agents Plugin**  
  
  
**Mange jenkins --> Nodes**  
  
  


**Number of executors:** 5

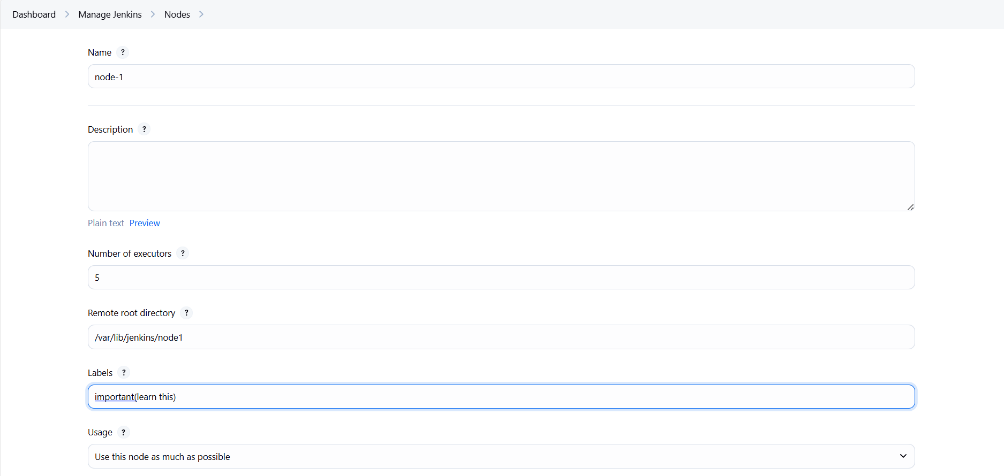
* This specifies how many concurrent builds this agent can handle. In this case, node-1 can run up to 5 Jenkins jobs simultaneously. If more than 5 jobs are queued for this agent, they will wait until an executor becomes available.

**Remote root directory:** /var/lib/jenkins/node1

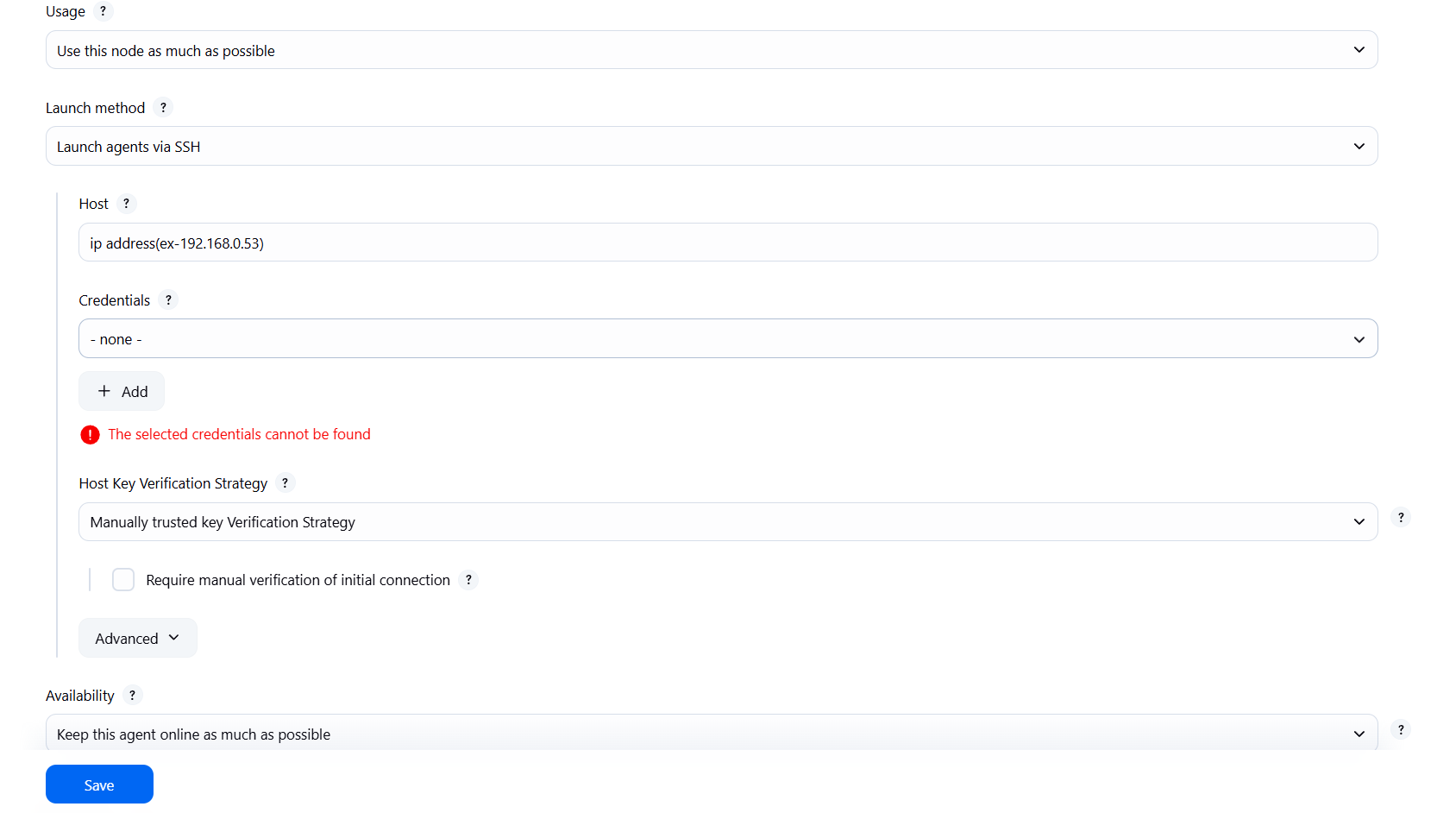
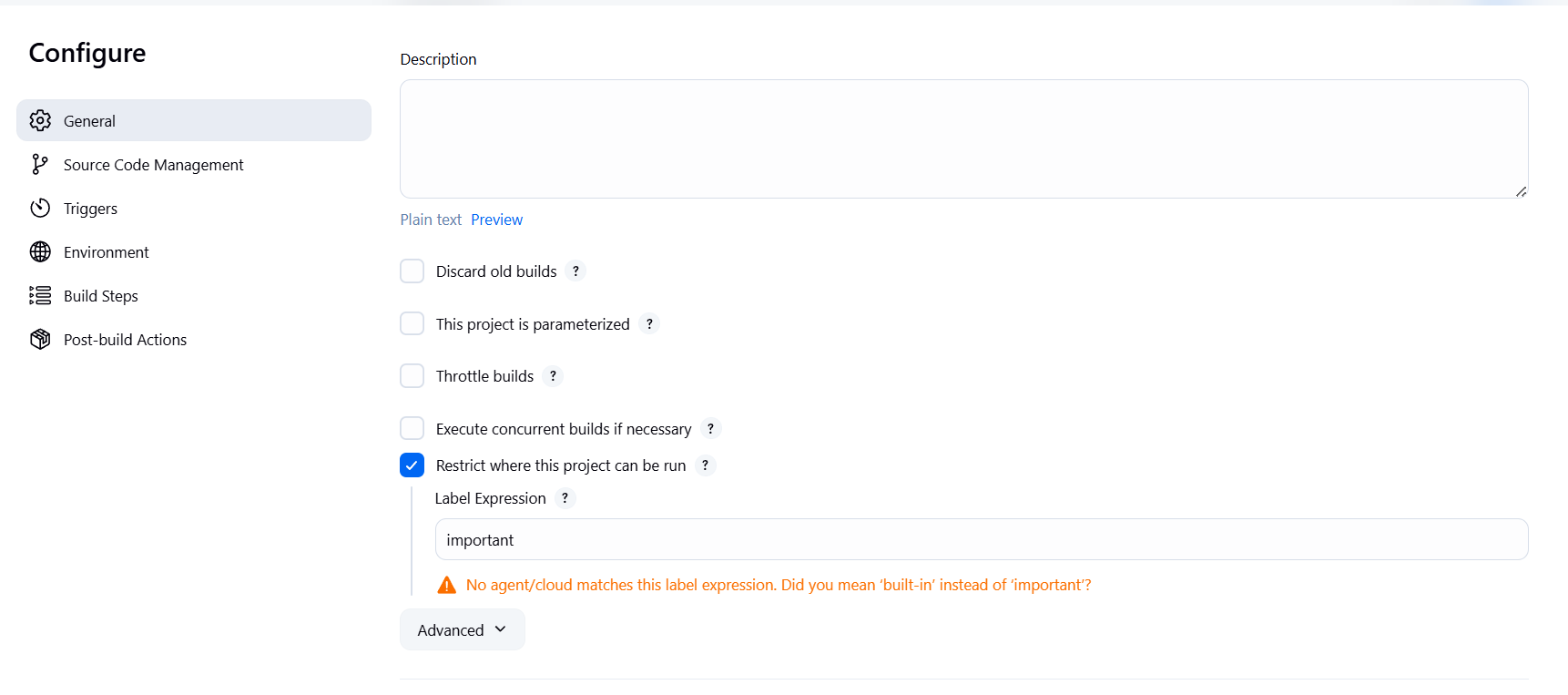
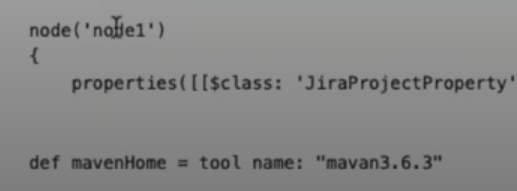
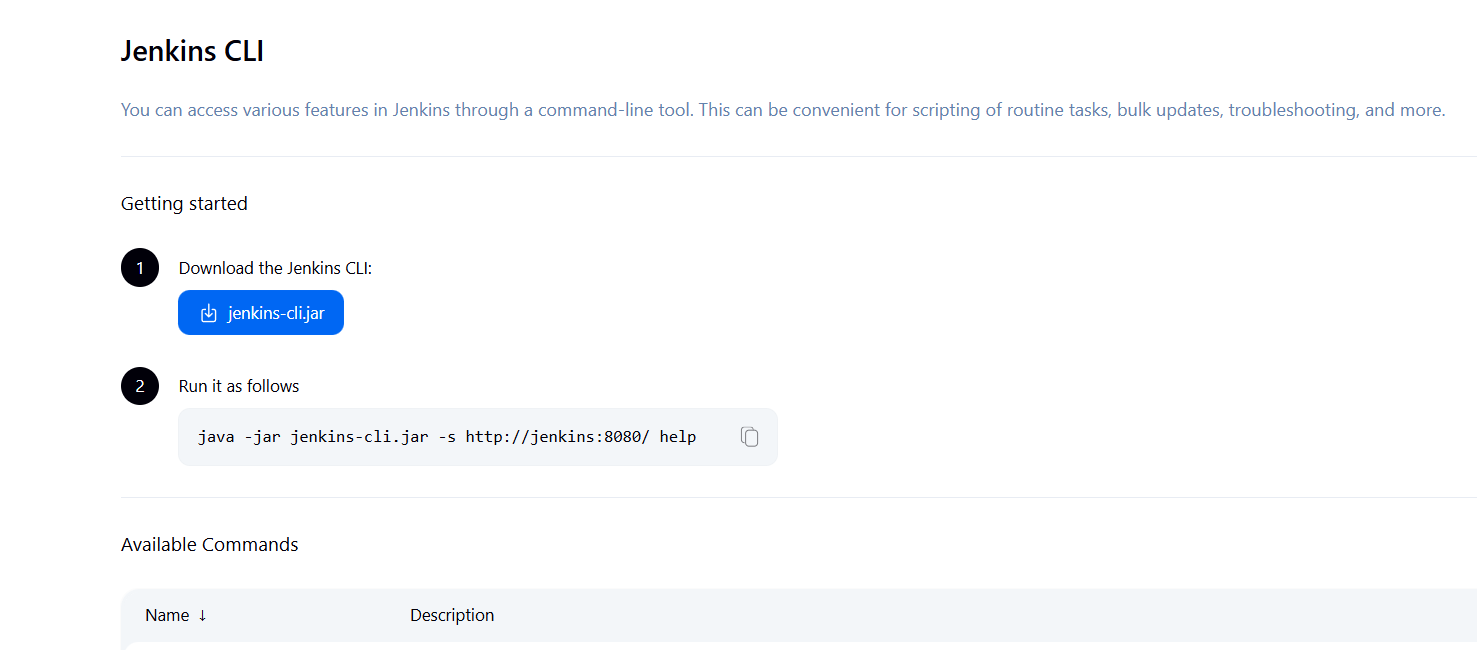
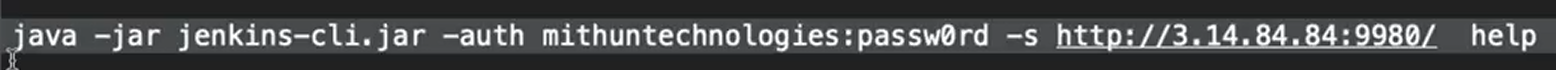
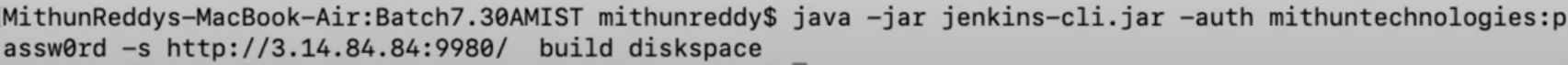
* This is the path on the *agent machine* where Jenkins will store its workspace for builds. When a job runs on node-1, its files will be checked out and processed within a subdirectory of /var/lib/jenkins/node1. This directory needs to exist and be writable by the user Jenkins connects as.

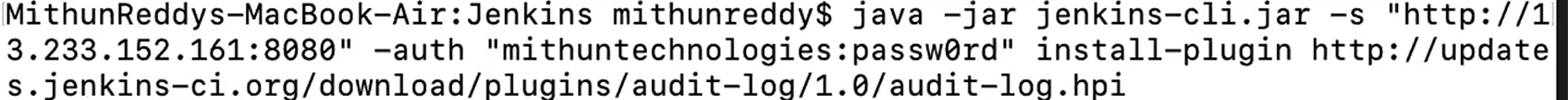
**Labels:** important(learn this)

* Labels are crucial for grouping and targeting agents. You can assign one or more labels to an agent.
* In a Jenkinsfile or job configuration, you can use the agent { label 'important' } syntax to specify that a particular pipeline stage or job should run on any agent that has the important label.

  
  
  
**Launch method:** Launch agents via SSH

* This is the most common and recommended way to connect to and launch Jenkins agents. It means Jenkins will use SSH to connect to the remote machine, execute commands, and manage the agent process. Other options might include "Launch agent by connecting it to the master" (for agents initiating the connection) or "Launch agent via Java Web Start"  
    
    
  install java in the linux server mandatory

  
provide the node label in the job level so that the job will run in the given node  
  
  
  
  
  
  
  
  
or in the pipeline mention the node name  
  
  
  
  
**22.Jenkins CLI**  
  
http://jenkins-test.apps.684b1059b9e77e1826ddd10e.ocp.techzone.ibm.com/cli  
  
  
  
download the jar and run the above command  
to display all Jenkins commands  
  
  
to run the job using cli  
  
  
to delete job  
  
  
to list all the jobs  
  
  
  
to install plugins

  
  
  
A screenshot of a computer

AI-generated content may be incorrect.  
  
  
**23.Upstream and downstream jobs**  
  
 **Upstream Job (The Initiator/Parent):** A job that, upon completion, triggers one or more other jobs. It's earlier in the process.

 **Downstream Job (The Triggered/Child):** A job that is initiated by one or more other jobs. It depends on the completion of an upstream job.

Imagine you have a simple web application. Your goal is to:

1. **Build** the application (compile code, create an executable/deployable artifact).
2. **Test** the built application (run automated tests).
3. **Deploy** the application to a development server.

We'll set this up using three Jenkins jobs:

* **Job 1: Build\_WebApp (Upstream)**
* **Job 2: Run\_Tests\_WebApp (Downstream of Build\_WebApp)**
* **Job 3: Deploy\_To\_Dev\_WebApp (Downstream of Run\_Tests\_WebApp)**

Once the Job1 is completed then job 2 will trigger  
once job2 is completed the job 3 will trigger  
  
  
  
  
  
  
  
**24.Parallel Execuation**  
ex  
if you want to run build and sonarqube report parallelly then we can use below parallel execution  
  
