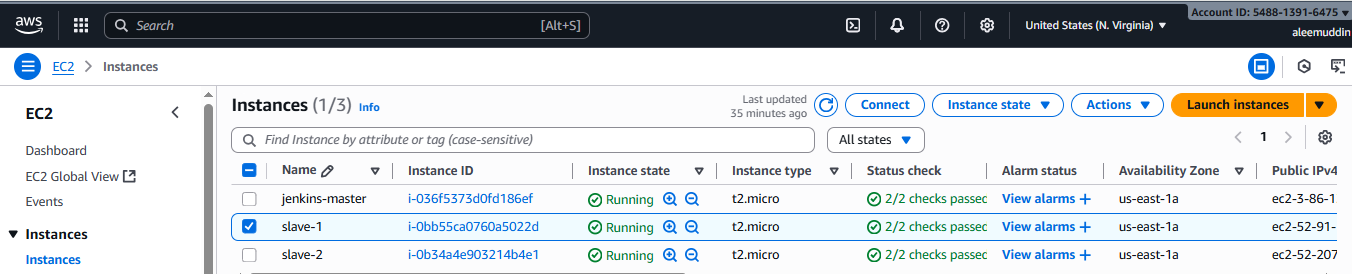
1. **Configure 2 slave machines in Jenkins master.**

**Steps on salve machine**

**Create or launch two instances slave01-ec2 and slave02-ec2**

****

**Install java and git on both ec2**

**sudo yum install -y git**

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

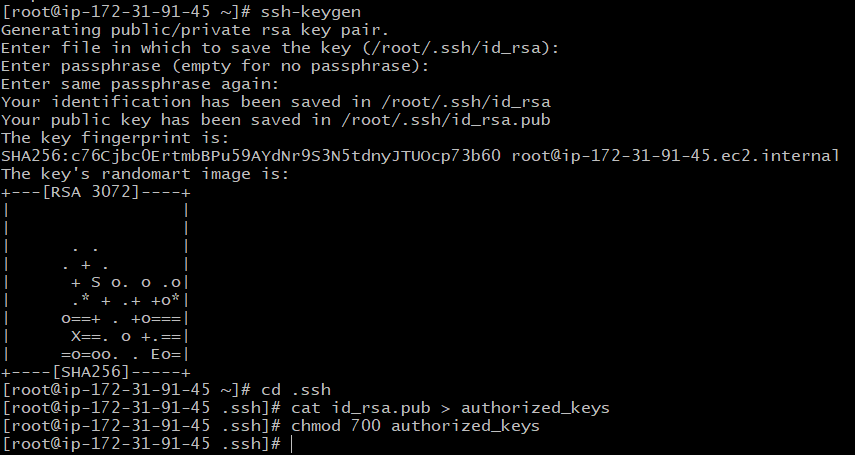
**Copy the key gen → ssh-keygen**

**cd .ssh**

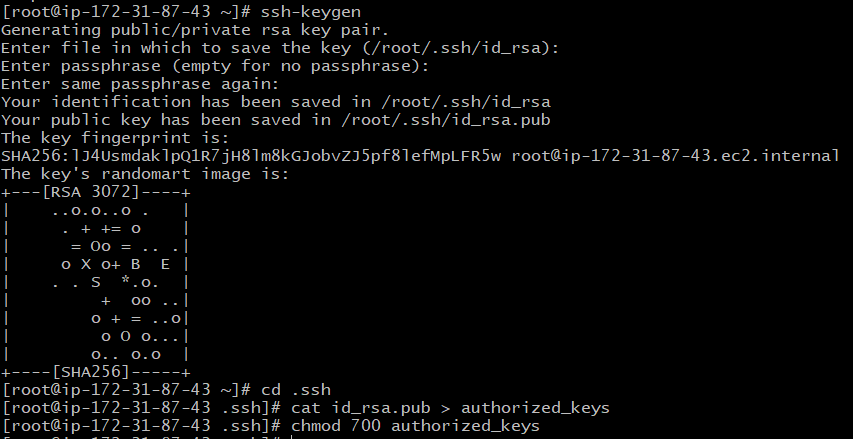
**cat id\_rsa.pub > authorized\_keys**

**chmod 700 authorized\_keys**

**Slave machine-1**

****

**Slave machine-2**

****

**Steps on Master Machine**

**Login to master machine.**

**Switch to root user.**

**Create Jenkins SSH directory:**

**mkdir -p /var/lib/jenkins/.ssh**

**cd /var/lib/jenkins/.ssh**

**ssh-keyscan -H SLAVE-NODE-PUBLIC-IP >> /var/lib/jenkins/.ssh/known\_hosts**

**chown jenkins:jenkins /var/lib/jenkins/.ssh/known\_hosts**

**chmod 644 /var/lib/jenkins/.ssh/known\_hosts**

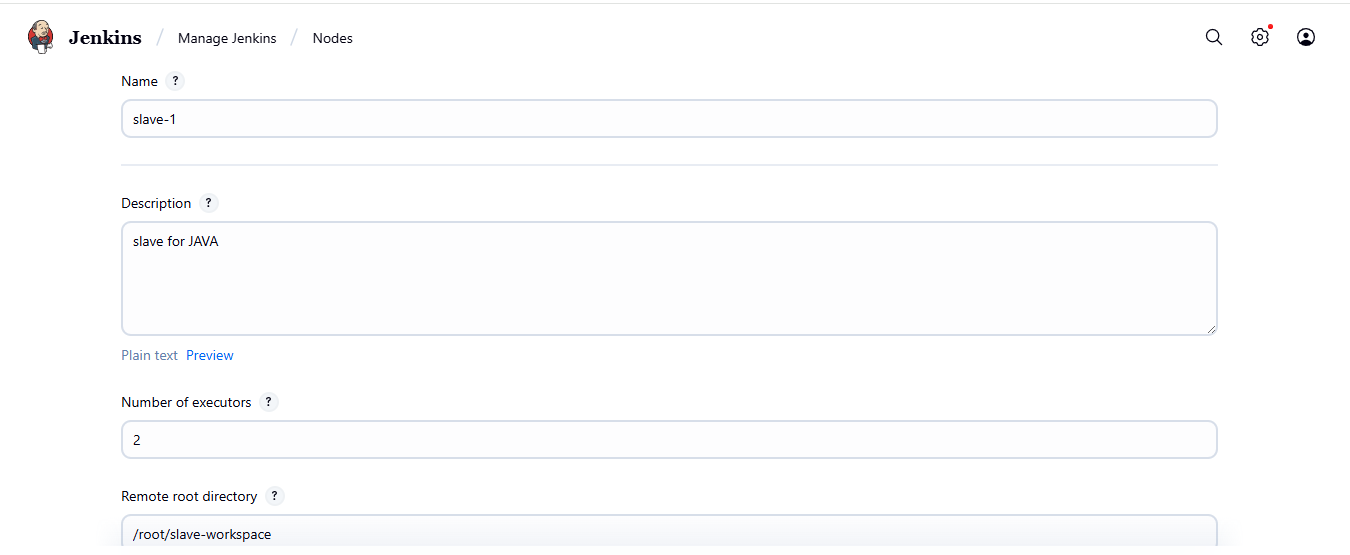
****

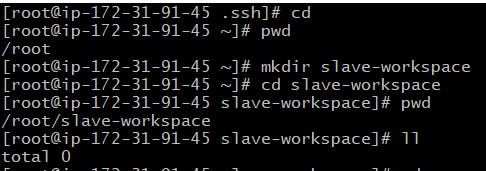
**Steps on Node creation**

### **Create the node in Jenkins GUI**

**Manage Jenkins → Nodes → New Node**

**Name: slave-1 → Type: Permanent Agent**

****

****

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

**1. This path is taken from slave machine**

**2. Slave ec2 → cat /etc/passwd**

**3. chown ec2-user:ec2-user slave-workspace**

**4. chmod 777 slave-workspace**

**Labels: slave2 java (as you like)**

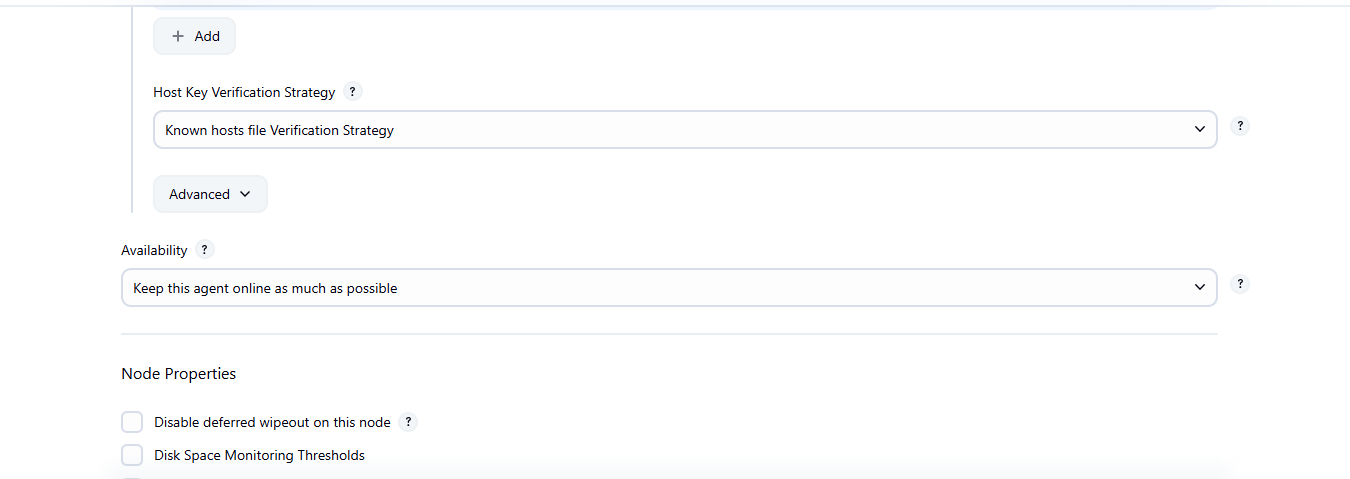
**Usage: “Only build jobs with label expressions” (optional)**

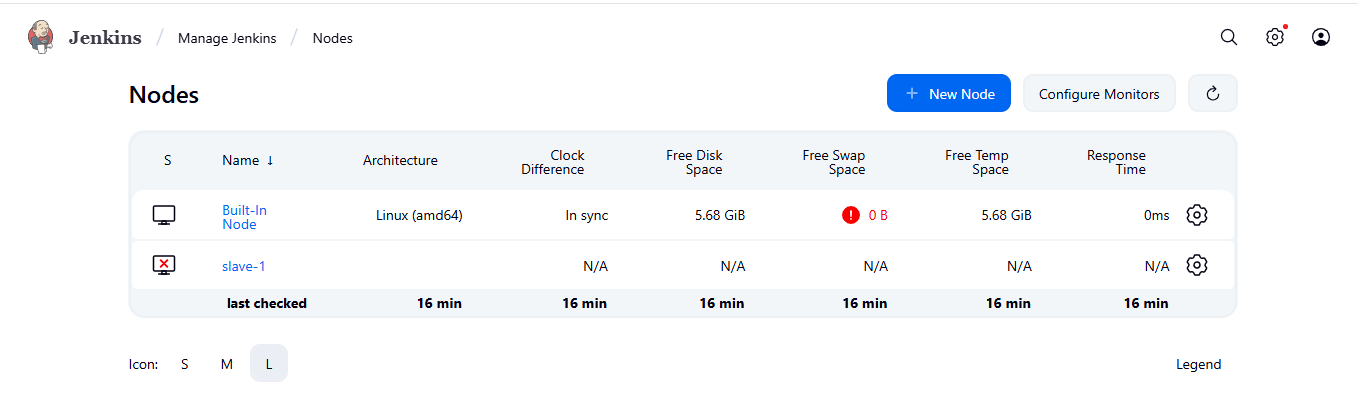
**Launch method: Launch agents via SSH  
Host: <NEW\_AGENT\_IP\_OR\_DNS>**

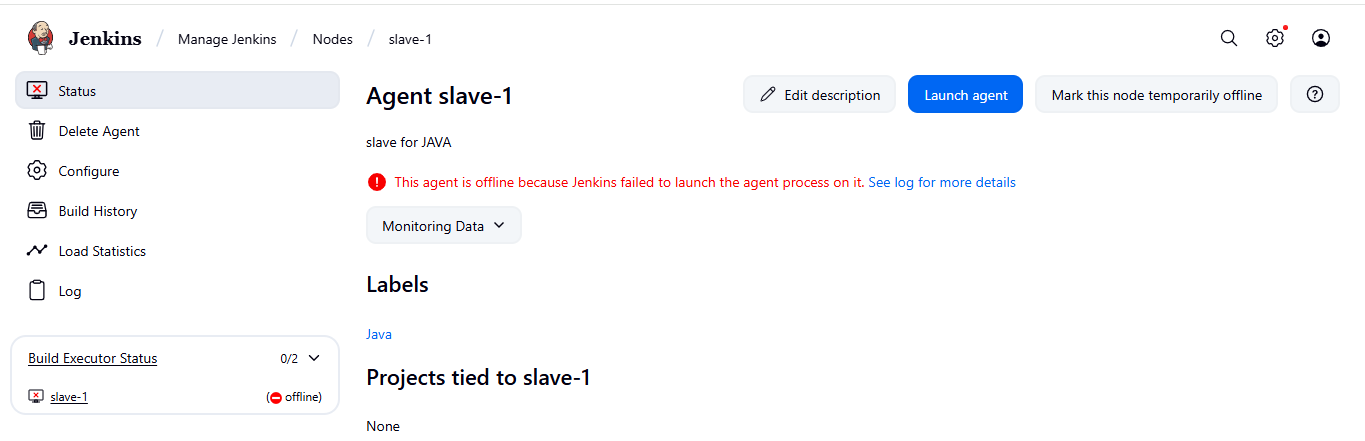
**Credentials: create/select “SSH Username with private key”  
1. Username: ec2-user  
2. Private key : paste content of /var/lib/jenkins/.ssh/id\_rsa from the master**

**Host Key Verification Strategy: Known hosts file (recommended)**

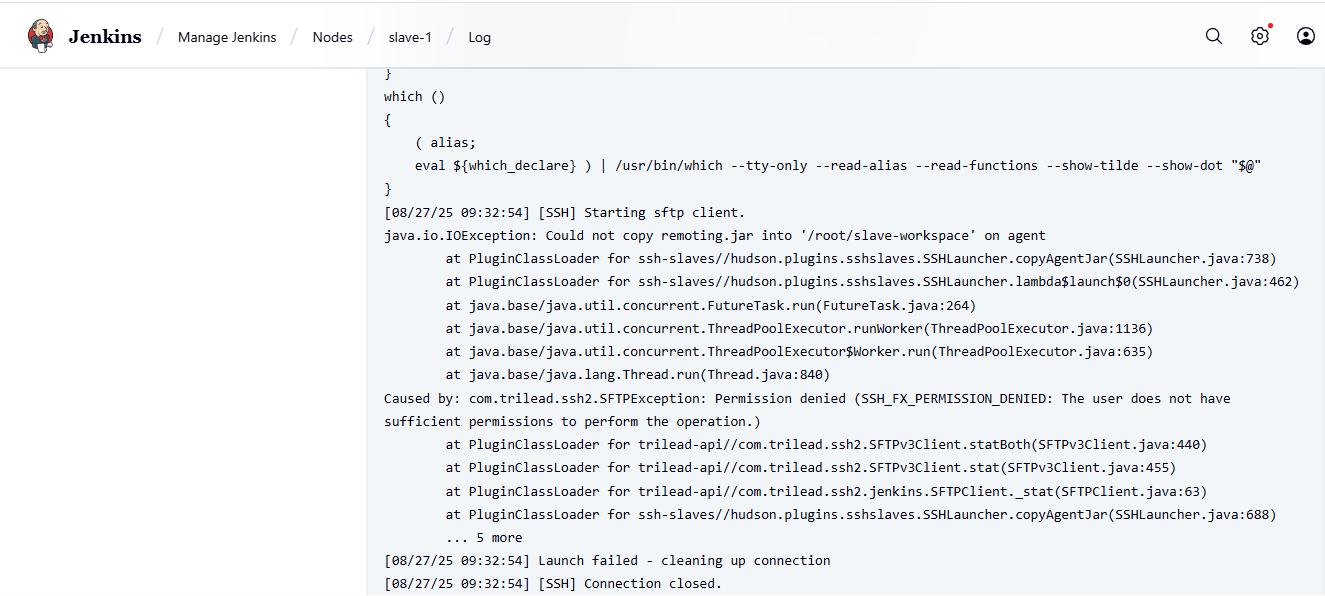
****

****

****

****

**If you see any error while launching the agent must check the logs to find the error and rectify it.**

****

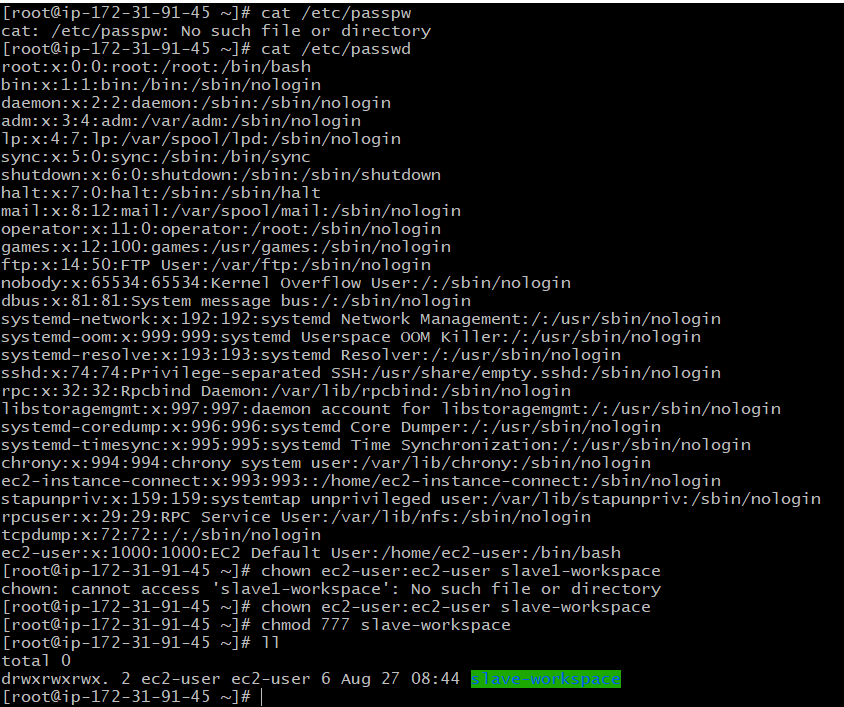
**Remote root directory: /home/ec2-user**

**1. This path is taken from slave machine**

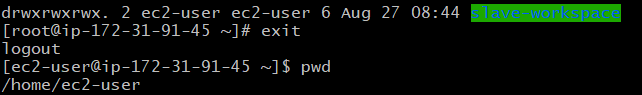
**2. Slave ec2 → cat /etc/passwd**

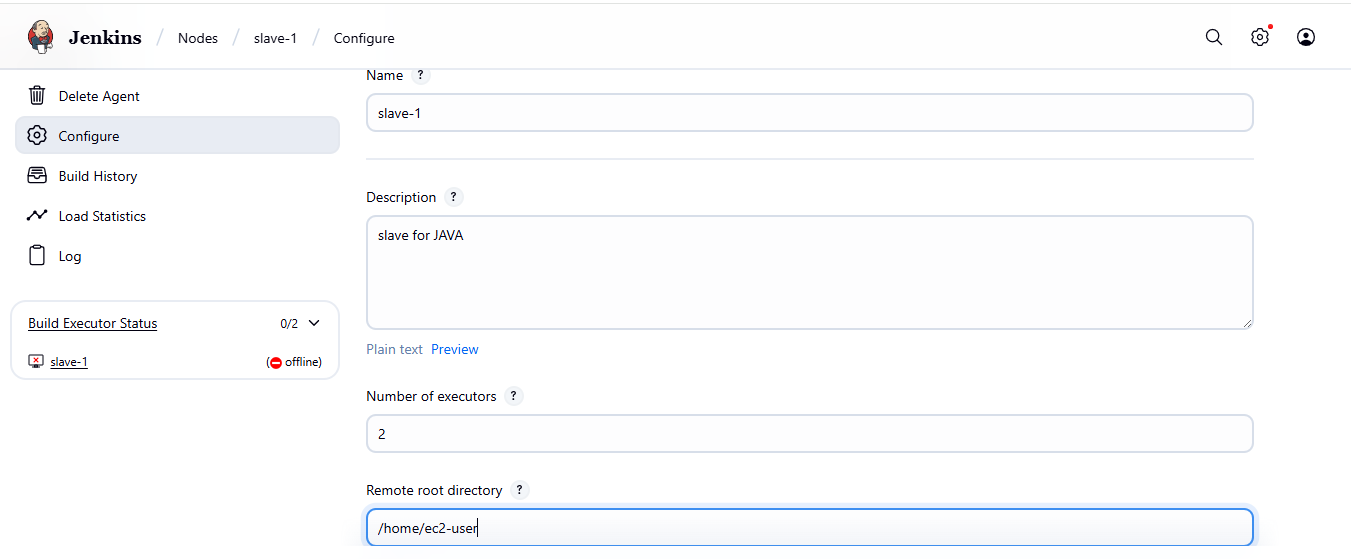
**3. chown ec2-user:ec2-user slave-workspace**

**4. chmod 777 slave-workspace**

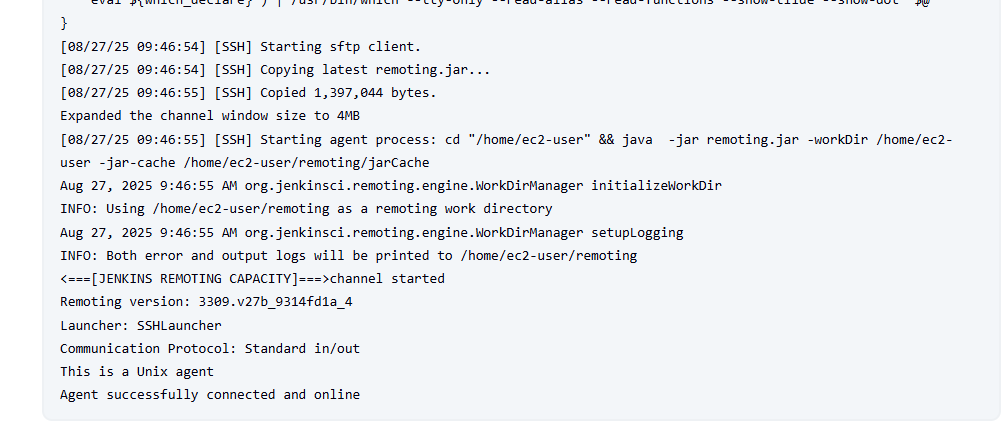
****

**Make some small changes again configuration the Remote root directory path as: /home/ec2-user**

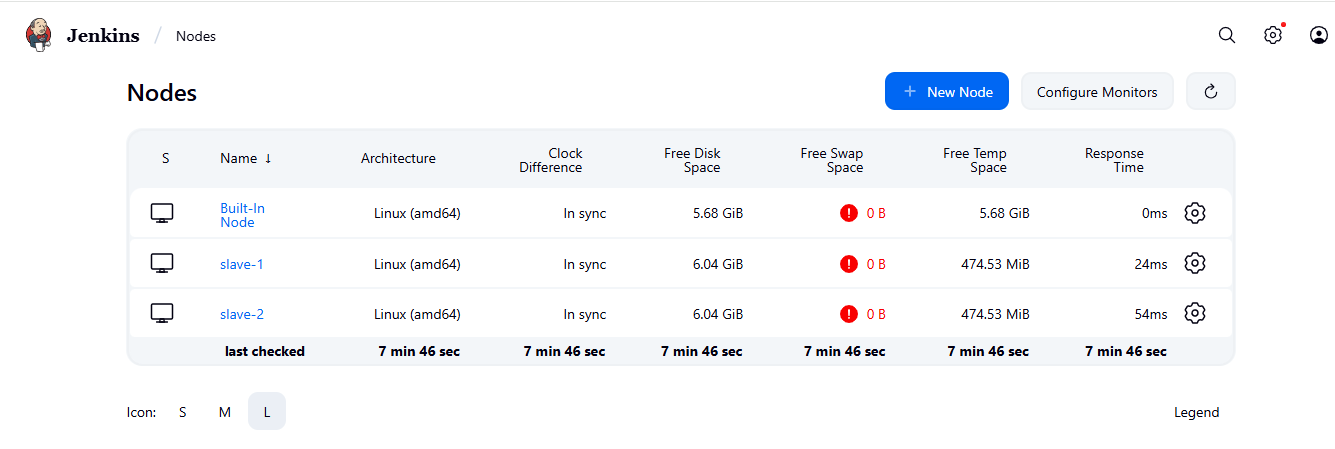
****

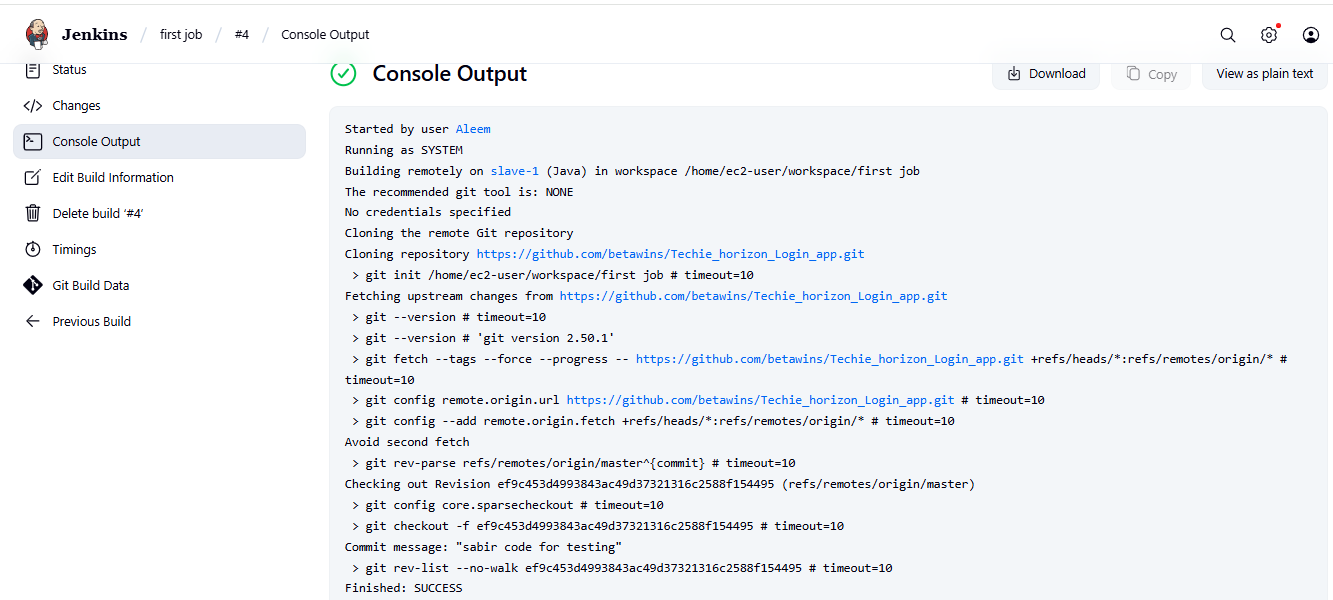
****

**Save → Jenkins should connect and show Online.**

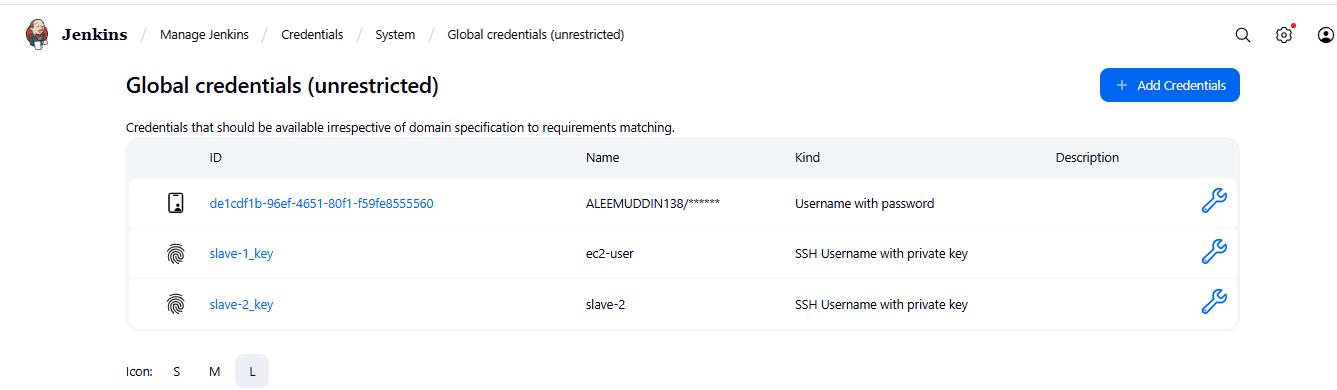
****

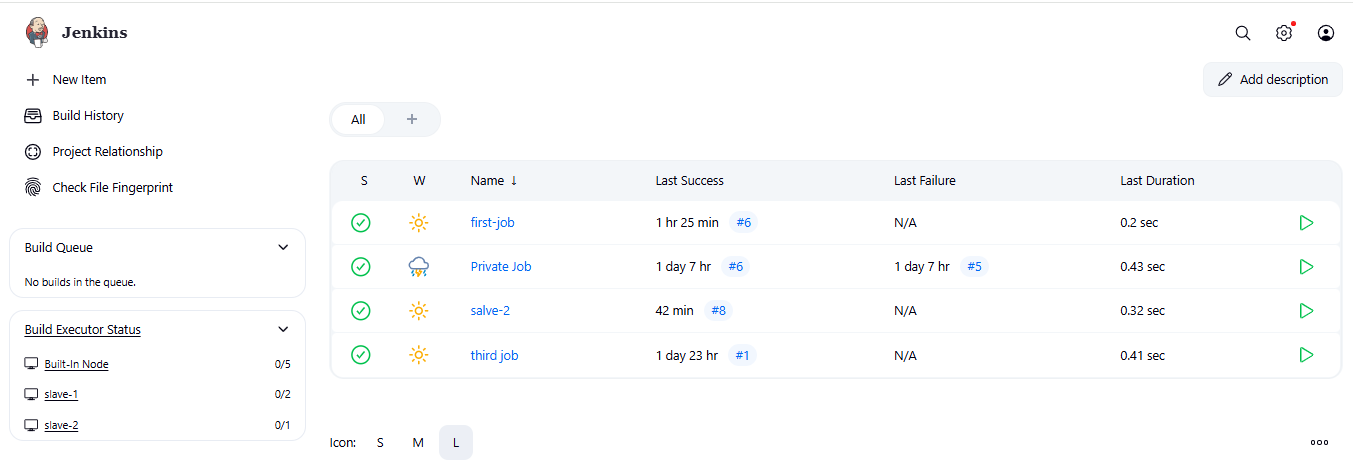
**After configuration successfully done, then you can see one master node and two slave nodes available.**

****

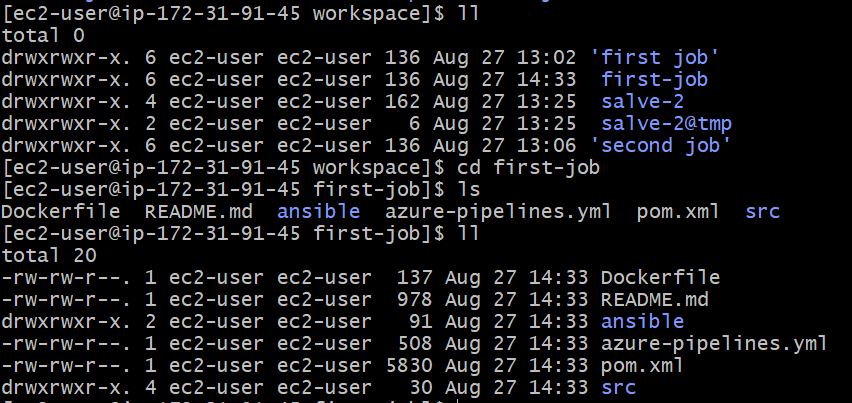
****

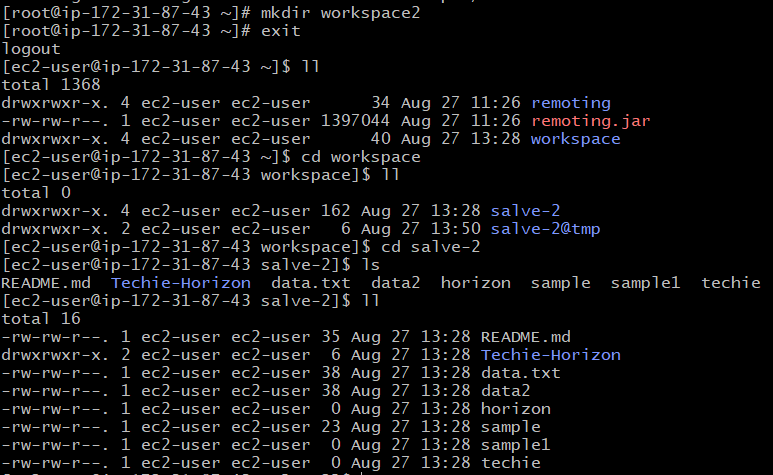
**Credentials that should be available irrespective of domain specification to requirements matching.**

****

**Related jobs has been done.**

**Slave-1 EC2 console**

****

**Slave-2 EC2 console**

**2) Configure webhooks to Jenkins job.**

**"GitHub hook trigger for GITScm polling" is a feature, primarily within Jenkins, that enables builds to be automatically triggered in response to events occurring in a GitHub repository, specifically those related to code changes.**

**Mechanism:**

* **GitHub Webhooks:**

**GitHub webhooks are configured within a repository's settings to send HTTP POST requests (payloads) to a specified URL whenever certain events occur (e.g., pushes to a branch, pull request activity).**

* **Jenkins Integration:**

**In Jenkins, a job is configured with the "GitHub hook trigger for GITScm polling" option enabled. This tells Jenkins to listen for incoming webhook events from GitHub.**

* **Triggering Builds:**

**When GitHub sends a webhook payload to Jenkins indicating a relevant event (like a push event), Jenkins receives this notification. Instead of constantly polling the SCM for changes (which is what "Poll SCM" does), this trigger kicks Jenkins' internal polling mechanism for the specific repository, prompting it to check for changes and initiate a build if new commits are detected.**

**Key Points:**

* **Event-Driven:**

**It shifts from a periodic polling model to an event-driven model, where builds are triggered immediately upon a relevant GitHub event, rather than waiting for a scheduled poll.**

* **Efficiency:**

**It reduces the load on Jenkins and GitHub by eliminating constant polling, as builds are only triggered when necessary.**

* **Real-time Feedback:**

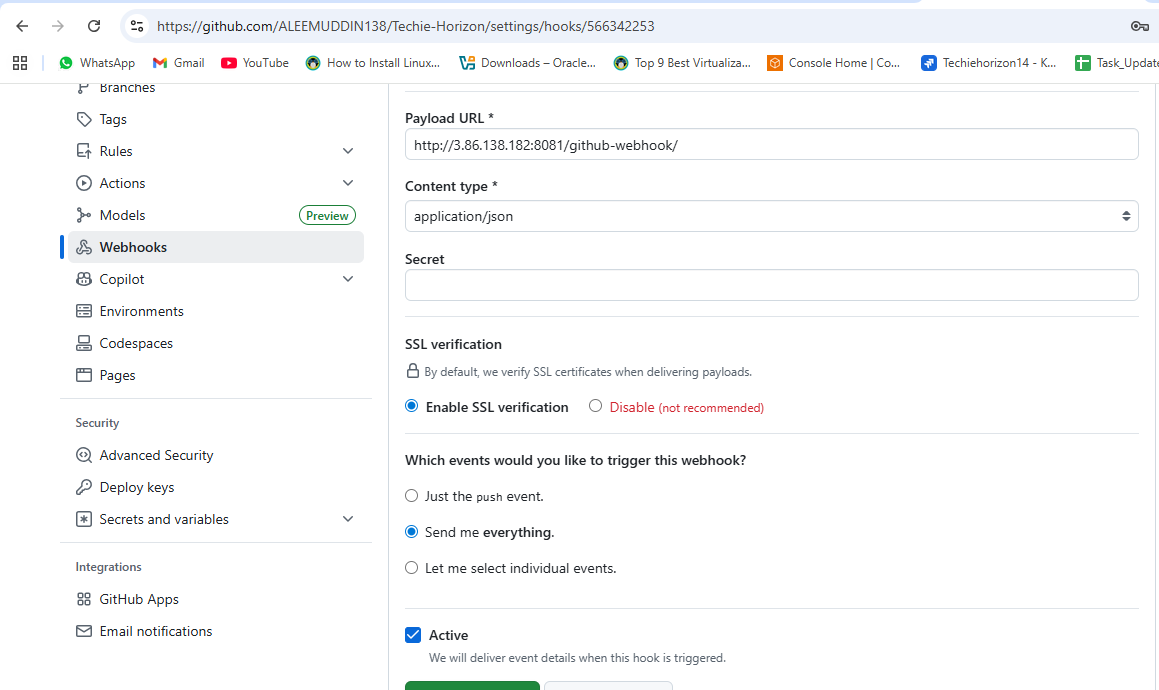
**It provides more immediate feedback on code changes by triggering builds faster.**

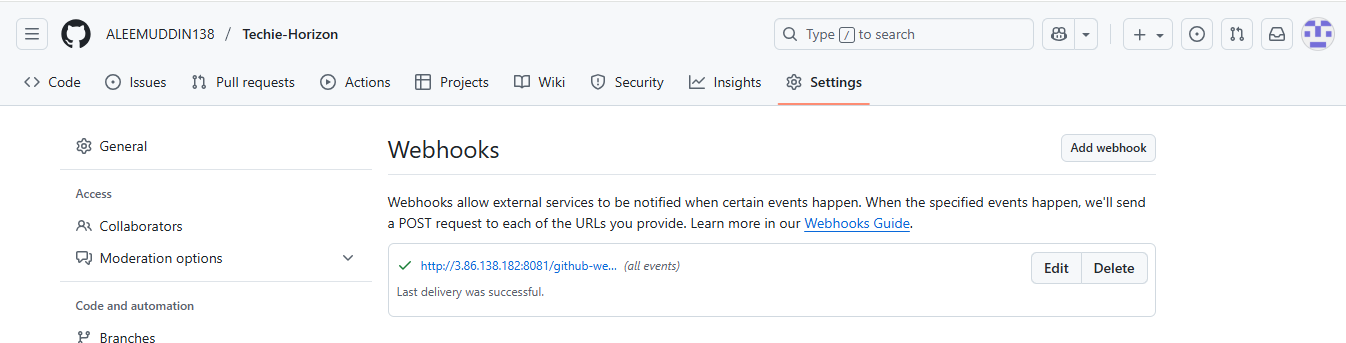
* **Configuration:**

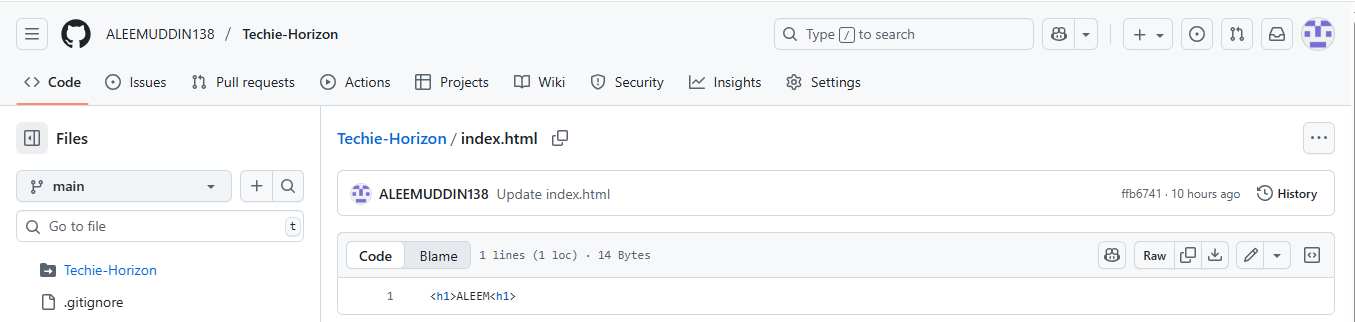
**It requires setting up webhooks in GitHub to point to the Jenkins instance and enabling the "GitHub hook trigger for GITScm polling" option within the Jenkins job's configuration.**

**Task started**

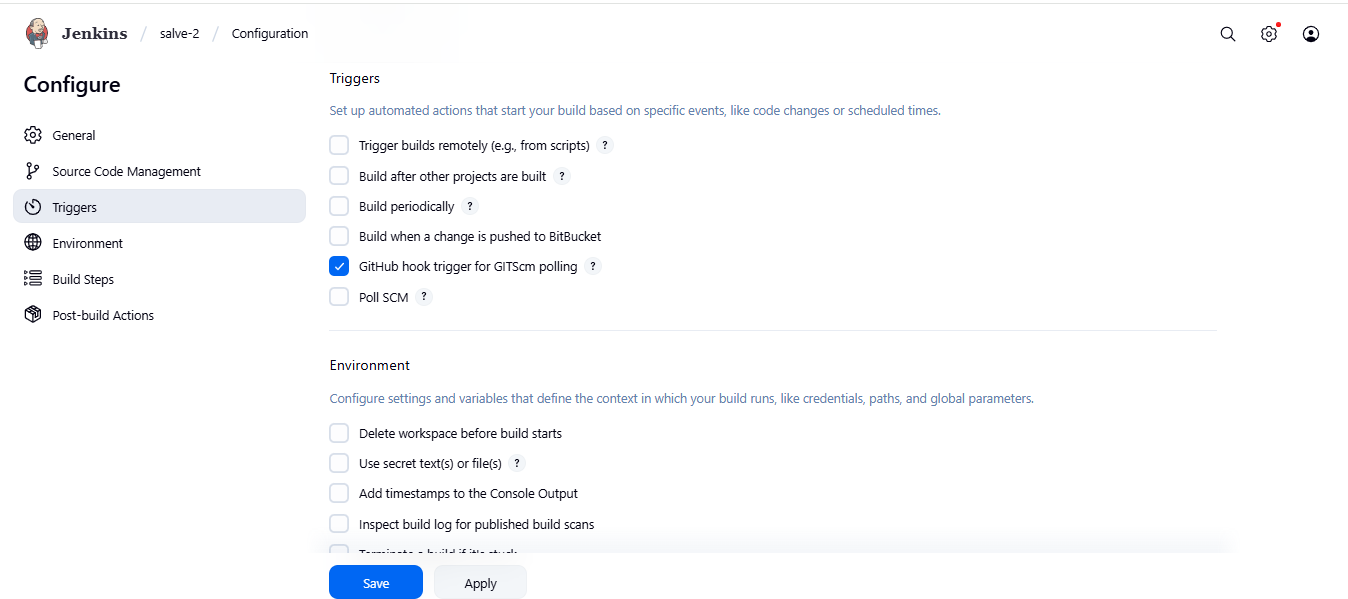
**open github repo → settings → webhooks → give payload (**[**http://3.86.138.182:8081/github-webhook/**](http://3.86.138.182:8081/github-webhook/)**) → content type (application/json) → select send me everything → add webhook**

****

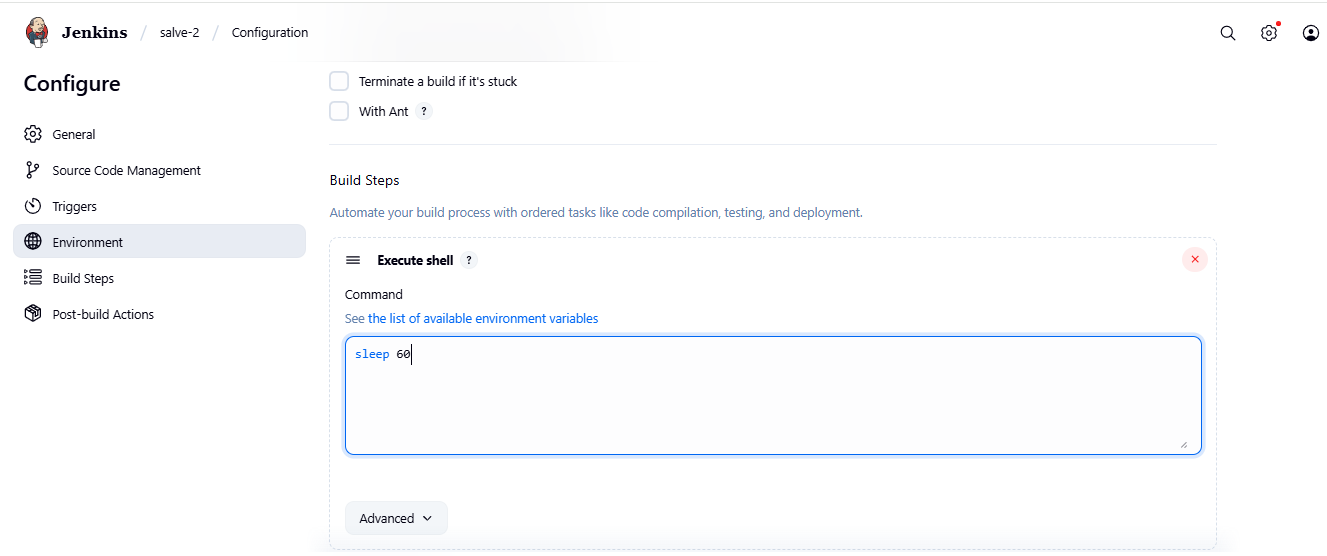
**Now go to your github repo and change or edit or update any file of the source code and commit**

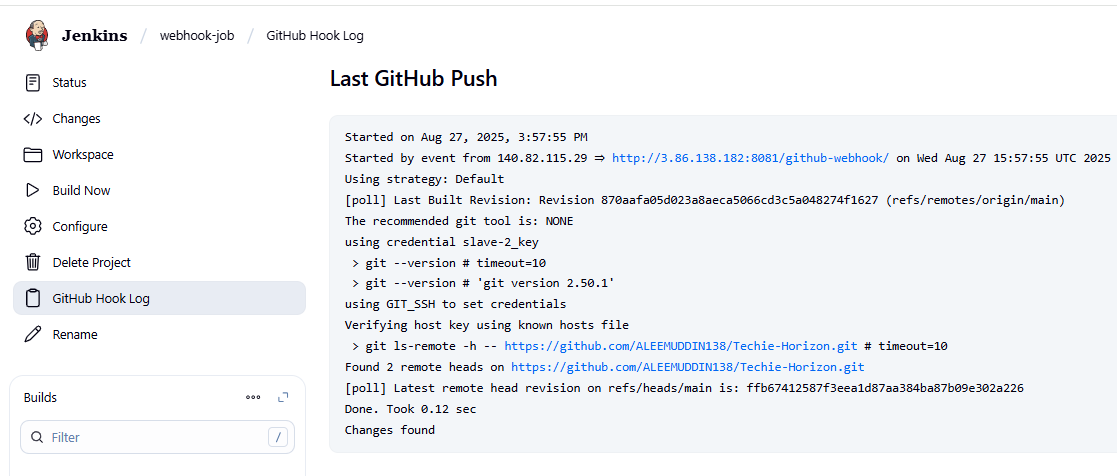
****

**Here select the option github hook trigger under configuration**

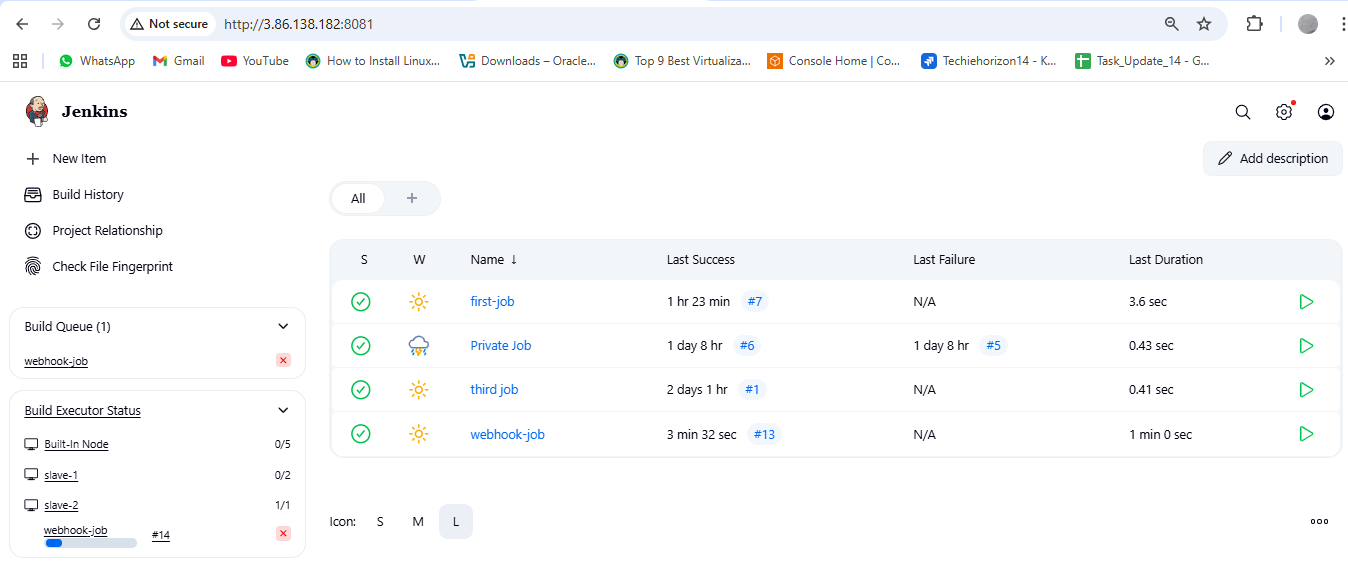
****

**This option is optional if you want to see the activity during testing/build the job.**

****

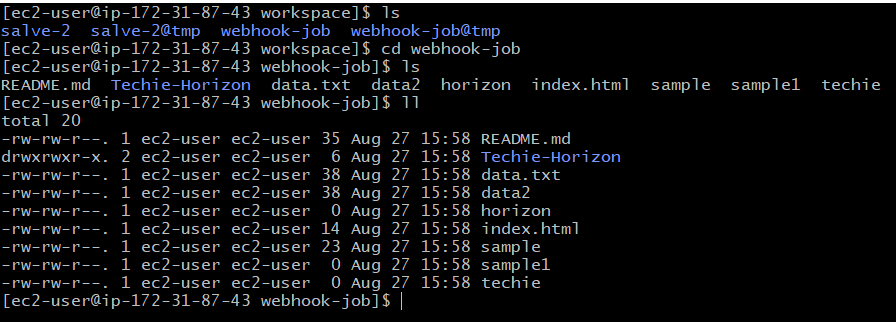
****

**You can see automatically webhook triggered and jenkins build the job automatically.**

****

**You can check files in our EC2 jenkins file.**

**cd /var/lib/jenkins/workspace**

****

**3) Configure poll scm and build periodical options in Jenkins job.**

**Poll SCM (Source Code Management) in Jenkins is a build trigger mechanism that periodically checks a configured source code repository for changes. If changes, such as new commits or updates, are detected since the last check, Jenkins automatically triggers a new build for the associated project.**

**This mechanism is crucial for achieving continuous integration, as it automates the process of monitoring your codebase and initiating builds whenever modifications are made.**

**How Poll SCM works:**

**Configuration:**

**You configure a Jenkins job to use Poll SCM and specify the schedule for polling using a cron-like syntax. This schedule dictates how frequently Jenkins will check the repository.**

**Polling:**

**At the scheduled intervals, Jenkins connects to the specified SCM repository (e.g., Git, SVN) and compares its current state with the state at the time of the last successful build.**

**Triggering:**

**If Jenkins identifies any changes in the repository, it then triggers a new build for the Jenkins job, allowing for automated compilation, testing, and deployment of the updated code.**

**Key aspects of Poll SCM:**

**Schedule:**

**The polling frequency is defined using a cron expression, allowing for flexible scheduling (e.g., every minute, every hour, daily).**

**Change Detection:**

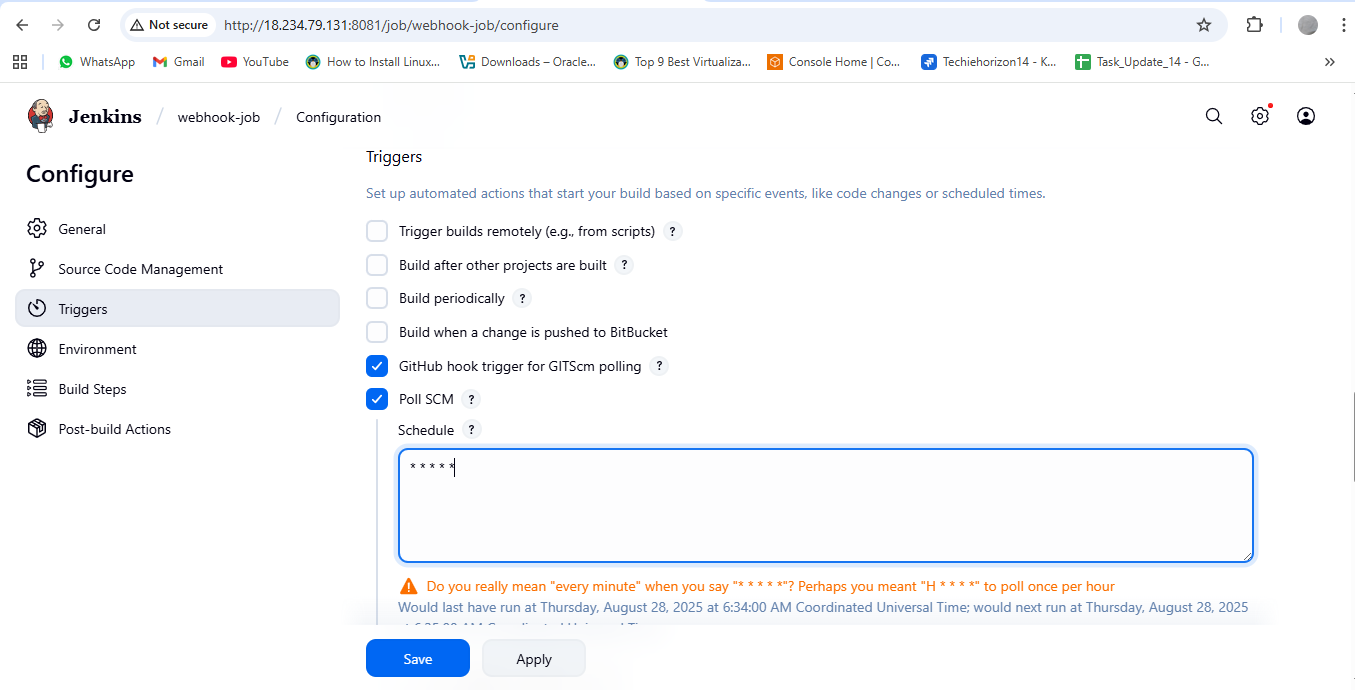
**Jenkins intelligently detects changes by comparing repository contents or commit history.**

**Automated Builds:**

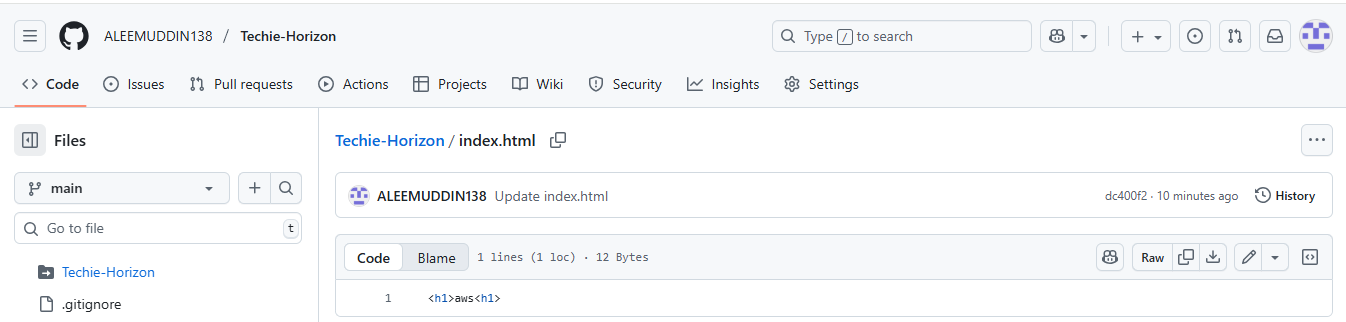
**It enables automated builds in response to code changes, facilitating continuous integration and early identification of integration issues.**

**Task started**

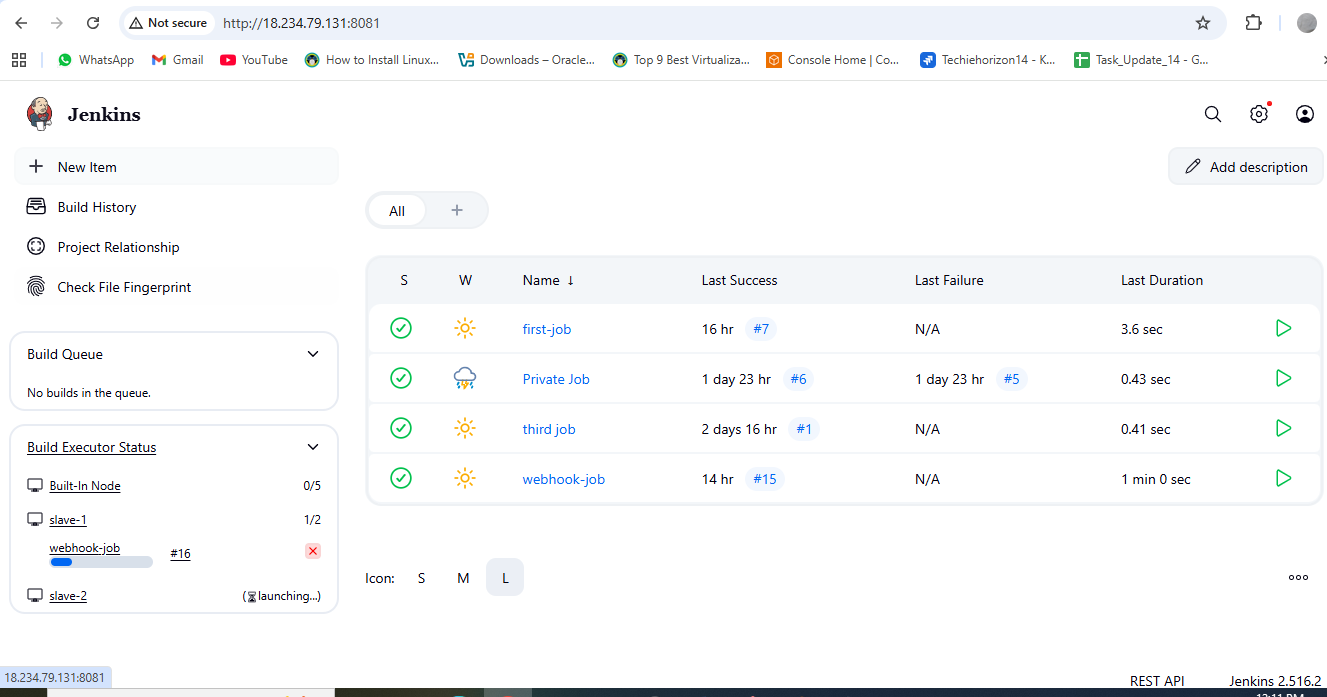
**While configure the job with poll scm and schedule with \* \* \* \* \* to the job its build every minute only if there is any changes happen in our github repo code.**

****

**Now go to your github repo and make some changes in code and check in jenkins.**

****

**Here you can see the testing under slave-1 node after make changes in the github file.**

****

**"Build periodically" is a feature, commonly in Jenkins, that schedules a software project's build process to run at pre-defined, recurring intervals, irrespective of any code changes in the Source Code Management (SCM) system. It uses a [Cron-like syntax](https://www.google.com/search?cs=0&sca_esv=ffd48def480ec45e&sxsrf=AE3TifOmWptLqtQ0uvbU3J92vl1Aonx3Zg%3A1756364361772&q=Cron-like+syntax&sa=X&ved=2ahUKEwj1j4CP96yPAxX91TgGHZSqLX8QxccNegQIBRAB&mstk=AUtExfC6F9mgRAFOCbmGXk_dRxAXdpdw2beFg52uSXSRl5YN7GVSIUyCnrr_m0wwwxNVgPHYtBl6B95RFEtyHP1Cwjz_yueKTisfs0121GddWz7w-PsNfwd3rck_v7rLJIfIbzcmHcgYBqFtcOR8lOEffPhgOF1RuR3x1Hs4CVB7gFJUJJY&csui=3" \t "_blank), consisting of five fields for minute, hour, day of the month, month, and day of the week, to define the specific timing for these automated builds, allowing for tasks to be executed at specific times or intervals like "every 15 minutes" or "daily at 8 AM".**

**How it works:**

**Scheduling with Cron Syntax:**

**You configure the "Build periodically" option with a Cron expression.**

**Example Syntax:**

**A Cron expression like H/15 \* \* \* \* would trigger a build every 15 minutes, while 30 8 \* \* 1-5 could schedule a build daily at 8:30 AM, Monday through Friday.**

**Automatic Execution:**

**The system automatically triggers the build at the scheduled times, automating tasks and reducing manual intervention.**

**"Hash" Symbol (H):**

**The letter "H" can be used instead of a specific number to evenly distribute load across multiple projects, ensuring builds don't all start at the exact same minute, according to the [GeeksforGeeks](https://www.geeksforgeeks.org/devops/what-is-build-periodically-in-jenkins/) article.**

**When to use it:**

**Regular Tasks:**

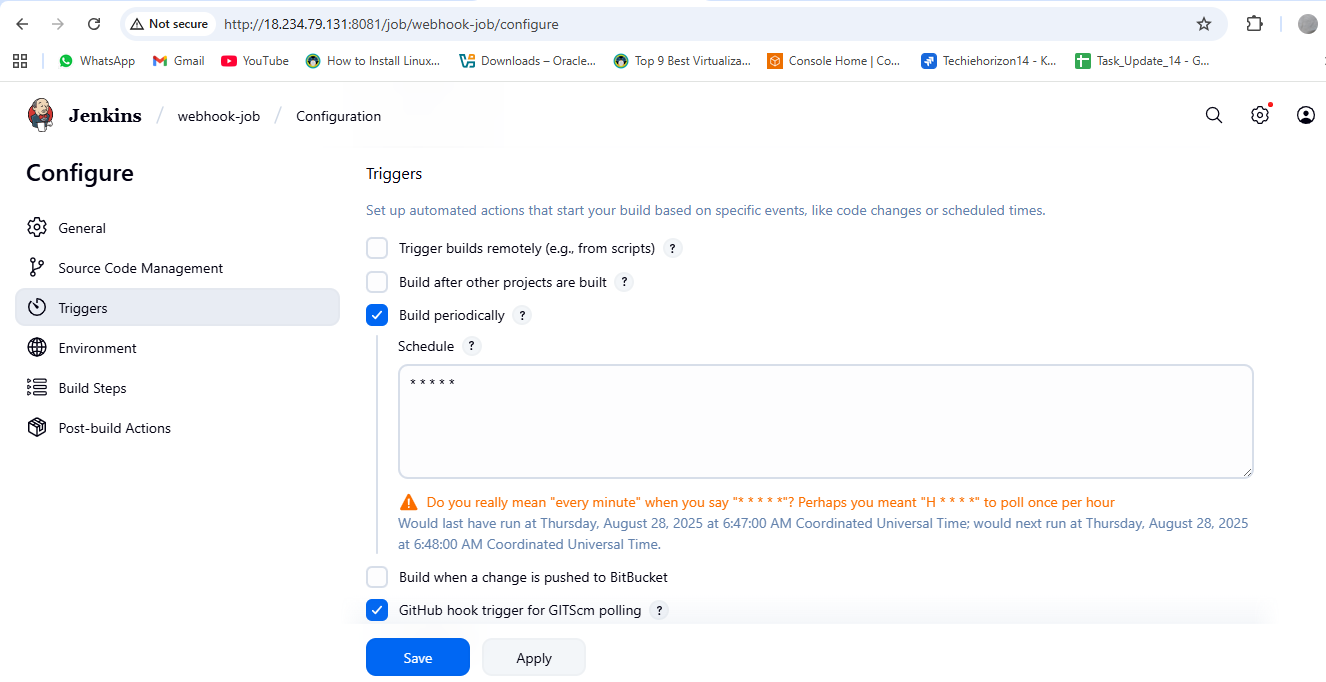
**Ideal for tasks that need to run on a fixed schedule, such as generating reports, performing regular maintenance, or running scheduled tests.**

**Pre-CI/CD Era:**

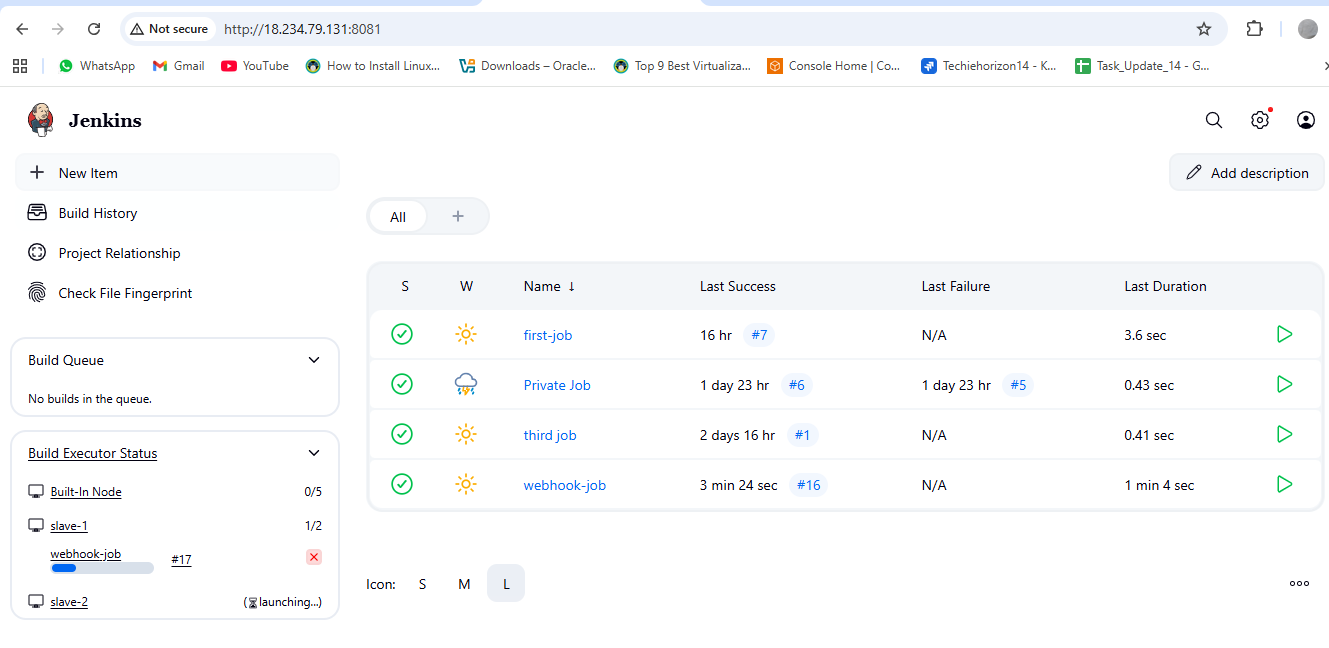
**While it still has uses, it's generally seen as a legacy approach compared to modern CI/CD practices, where builds are ideally triggered by code changes (using**[**Poll SCM**](https://www.google.com/search?cs=0&sca_esv=ffd48def480ec45e&sxsrf=AE3TifOmWptLqtQ0uvbU3J92vl1Aonx3Zg%3A1756364361772&q=Poll+SCM&sa=X&ved=2ahUKEwj1j4CP96yPAxX91TgGHZSqLX8QxccNegQILxAB&mstk=AUtExfC6F9mgRAFOCbmGXk_dRxAXdpdw2beFg52uSXSRl5YN7GVSIUyCnrr_m0wwwxNVgPHYtBl6B95RFEtyHP1Cwjz_yueKTisfs0121GddWz7w-PsNfwd3rck_v7rLJIfIbzcmHcgYBqFtcOR8lOEffPhgOF1RuR3x1Hs4CVB7gFJUJJY&csui=3)**or [webhooks](https://www.google.com/search?cs=0&sca_esv=ffd48def480ec45e&sxsrf=AE3TifOmWptLqtQ0uvbU3J92vl1Aonx3Zg%3A1756364361772&q=webhooks&sa=X&ved=2ahUKEwj1j4CP96yPAxX91TgGHZSqLX8QxccNegQILxAC&mstk=AUtExfC6F9mgRAFOCbmGXk_dRxAXdpdw2beFg52uSXSRl5YN7GVSIUyCnrr_m0wwwxNVgPHYtBl6B95RFEtyHP1Cwjz_yueKTisfs0121GddWz7w-PsNfwd3rck_v7rLJIfIbzcmHcgYBqFtcOR8lOEffPhgOF1RuR3x1Hs4CVB7gFJUJJY&csui=3" \t "_blank)) to provide faster feedback.**

**Task continue**

**While configure the job builds periodically and give \* \* \* \* \* then irrespective of changes in our github source code it builds as per given time every minute.**

****

**Here you can see the every minute testing will execute under slave-1 node after configuration build periodically.**

****

**4) Take backup of Jenkins server by using bash script.**

**Jenkins configuration is stored in /var/lib/jenkins/.**

**Create a script and give chmod permissions.**

**Vi test.bash**

**#!/bin/bash**

**set -euo pipefail # Fail on undefined vars, plus safer error handling**

**BACKUP\_DIR=${BACKUP\_DIR:-/backup/jenkins}**

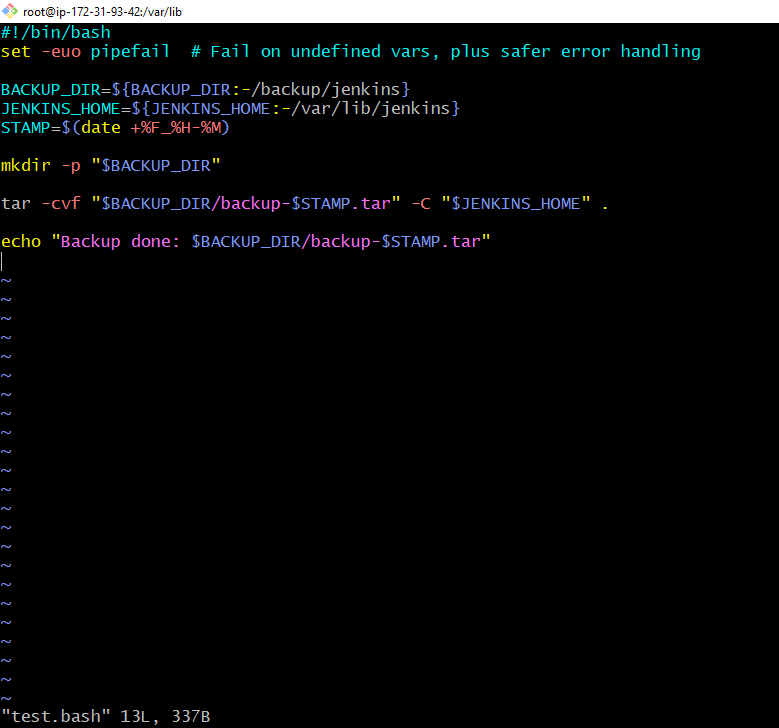
**JENKINS\_HOME=${JENKINS\_HOME:-/var/lib/jenkins}**

**STAMP=$(date +%F\_%H-%M)**

**mkdir -p "$BACKUP\_DIR"**

**tar -cvf "$BACKUP\_DIR/backup-$STAMP.tar" -C "$JENKINS\_HOME" .**

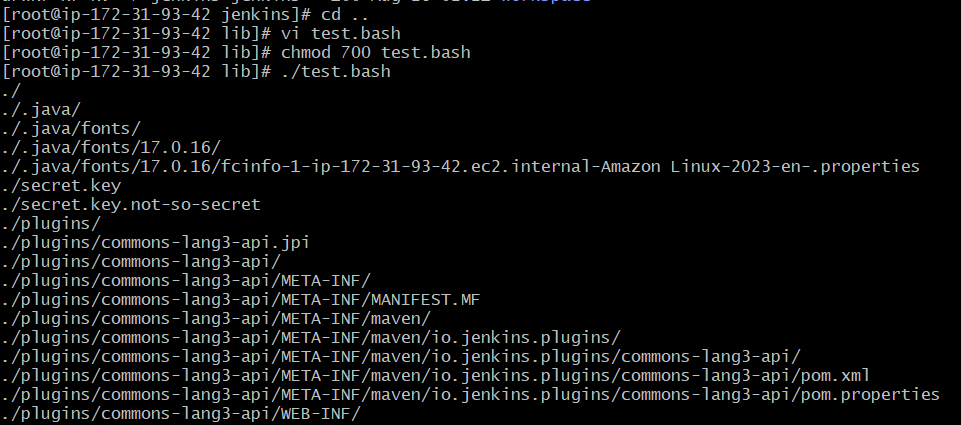
**echo "Backup done: $BACKUP\_DIR/backup-$STAMP.tar"**

****

**Save it and apply permissions to execute.**

**chmod 700 test.bash**

**./test.bash**

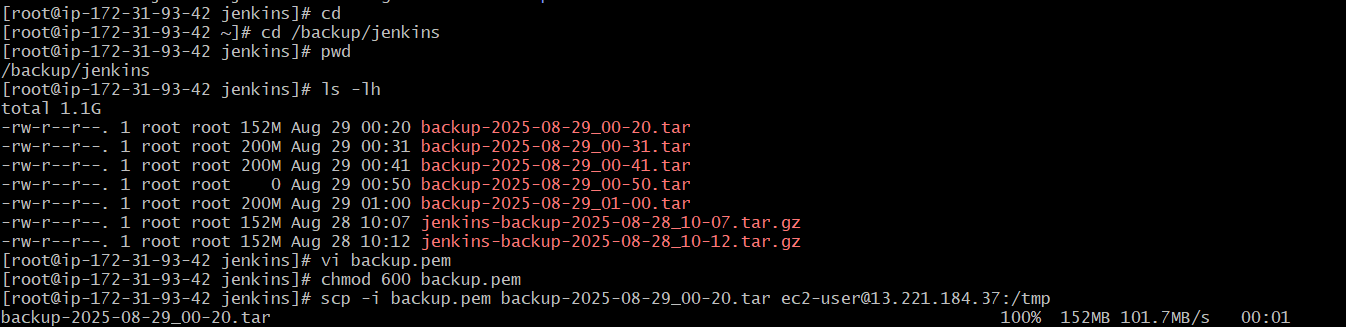
****

**To check where is backup are stored.**

**cd /backup/jenkins**

**pwd**

**ls -lh**

**  
5) Take backup of Jenkins using thin backup plugin.**

**Thin Backup plugin for Jenkins provides a way to back up and restore Jenkins configurations and job settings. It focuses on backing up the essential configuration files and not the entire workspace or build archives, leading to smaller, "thin" backups.**

**Key features and usage of the ThinBackup plugin:**

**Installation:**

**Navigate to "Manage Jenkins" > "Manage Plugins."**

**Go to the "Available" tab and search for "ThinBackup."**

**Install the plugin and restart Jenkins if required.**

**Configuration:**

**After installation, go to "Manage Jenkins" > "ThinBackup."**

**Click on "Settings" to configure the backup options.**

**Backup Directory: Specify the path where backup files will be stored. Ensure the Jenkins user has write permissions to this directory.**

**Backup Schedules: Define schedules for full and differential backups using cron notation.**

**Number of Backup Sets: Configure how many backup sets to retain.**

**Files Excluded/Included: Define regular expressions to exclude or include specific files during the backup process.**

**Backup Scope: Choose what components to include in the backup (e.g., global configurations, job settings, plugin archives).**

**Save the settings.**

**Backup and Restore:**

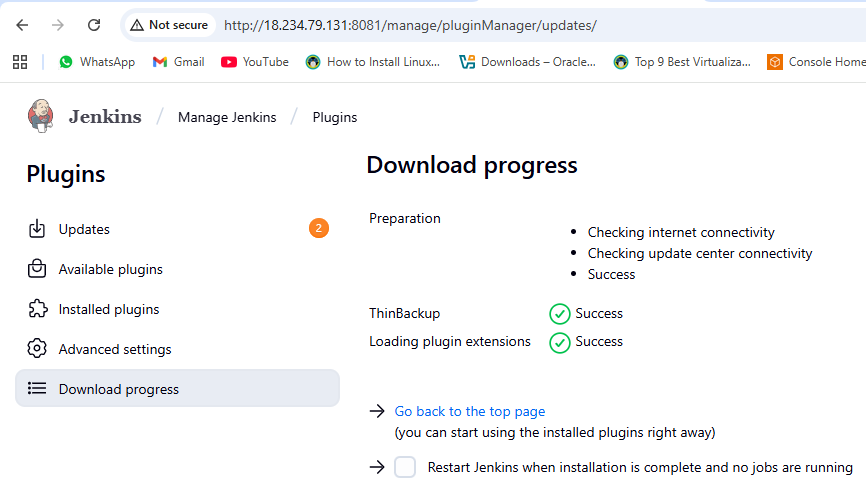
**Backup Now: Manually trigger a backup by clicking the "Backup Now" option.**

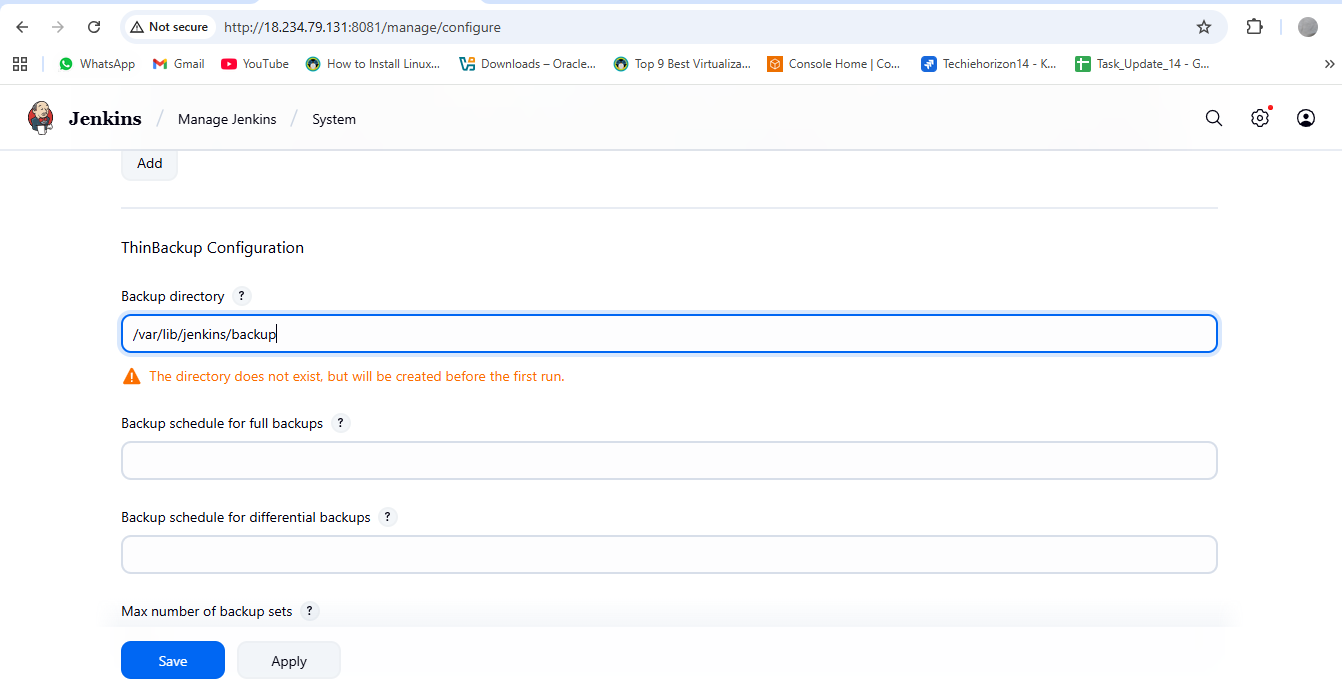
**Restore: Select a specific backup set from the available options and click "Restore" to revert Jenkins to a previous state. A Jenkins restart is typically required after a restore to activate the restored settings.**

**Task stared**

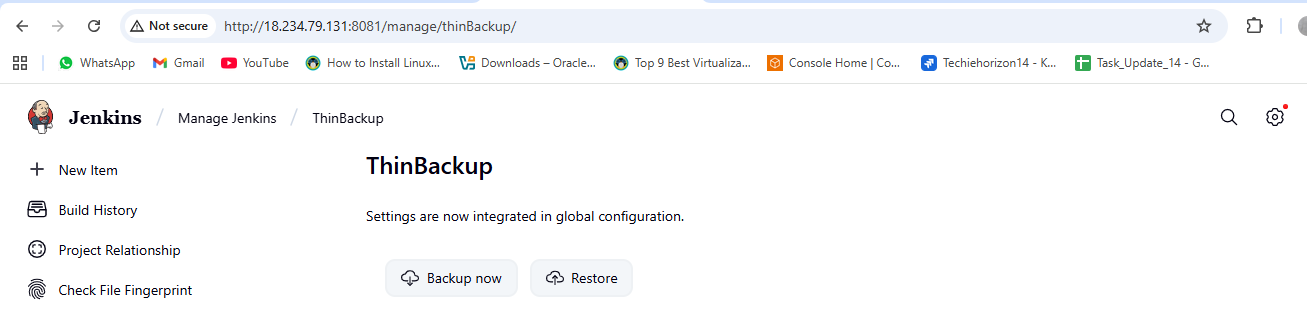
**Install plugins:**

**Available plugins > Search bar (Thin plugins) > Select and Install**

****

**Here need to do the thinback configuration and save configuration.**

**Click the option Backup now then the back stored in the local machine “ “ /var/lib/Jenkins/backup**

****

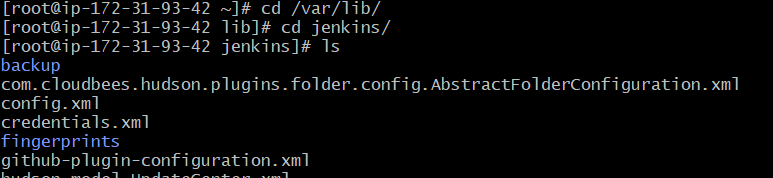
**After backup has been done it will reflect in the local machine at /var/lib/Jenkins/backup.**

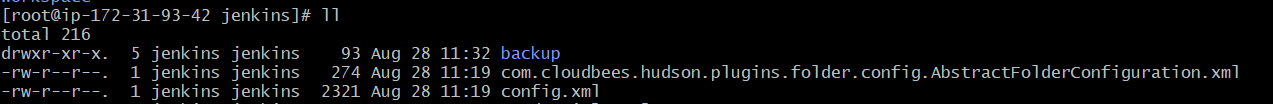
**To check backup use below commands**

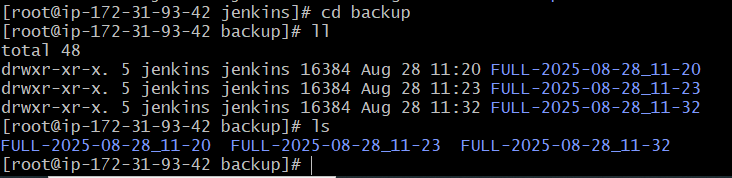
**cd /var/lib/ to enter in the liberary**

**cd jenkins/ to enter in the Jenkins**

**cd backup to enter in the backup and list details**

****

****

**  
6) Setup a new Jenkins server and dump the backup taken in task4.**

**Steps on backup machine**

**Create or launch one instances backup-master ec2**

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

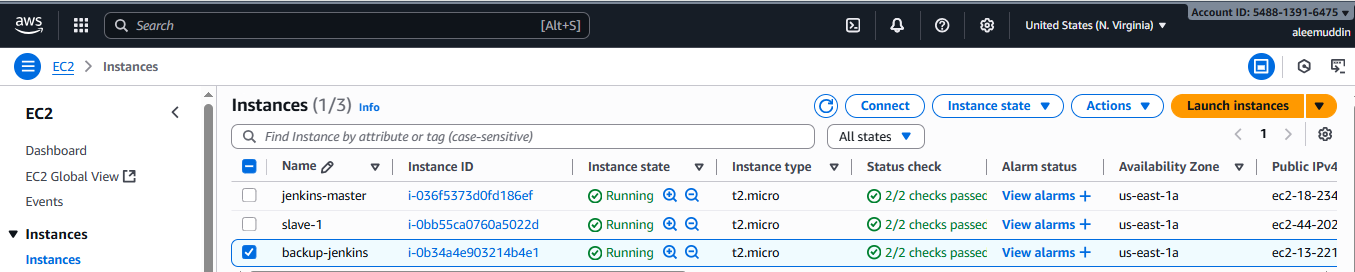
sudo wget -O /etc/yum.repos.d/jenkins.repo \ <https://pkg.jenkins.io/redhat-stable/jenkins.repo>

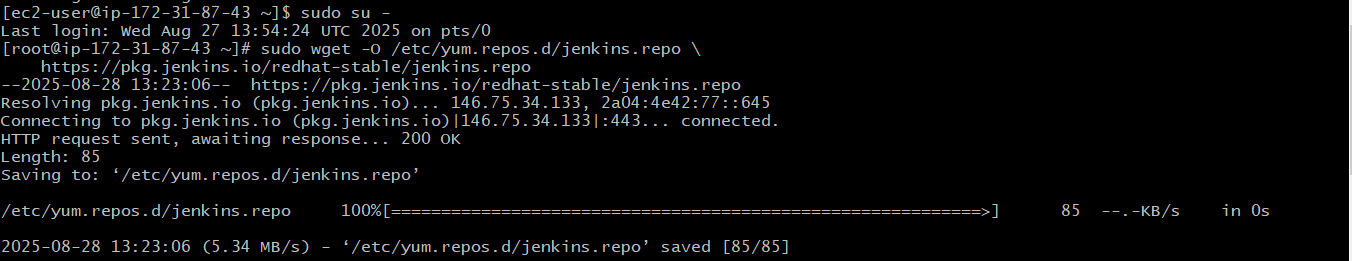
sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key>

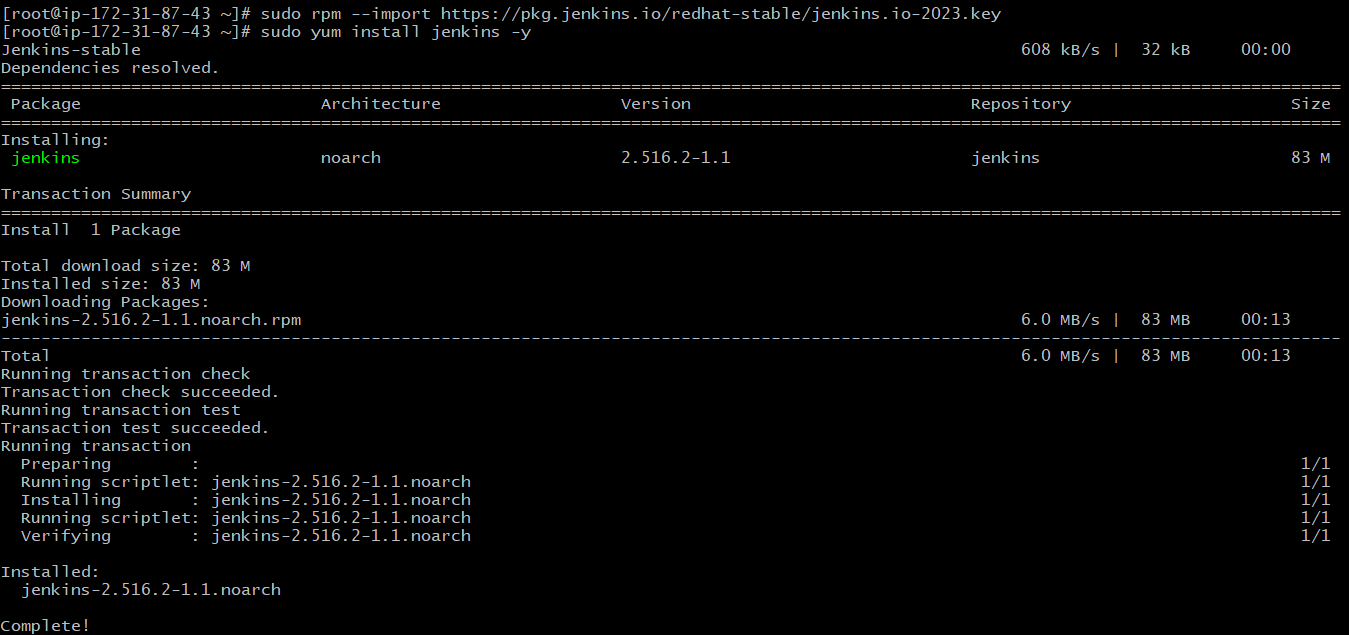
sudo yum install jenkins -y

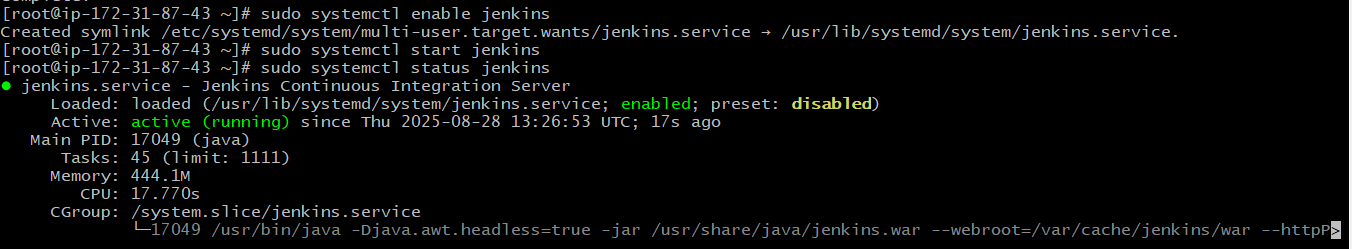
sudo systemctl enable jenkins  
sudo systemctl start jenkins

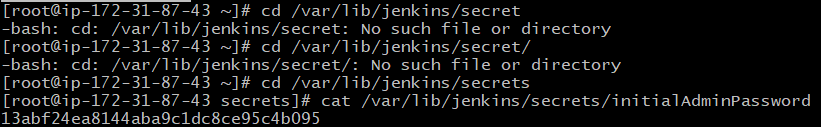
sudo systemctl status jenkins

****

****

****

****

****

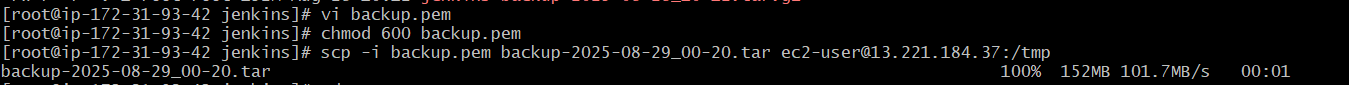
**Follow the steps like Jenkins task-1 to configure Jenkins by using ec2 instance public ip:8080 through browser.**

**Go to master Jenkins terminal**

**[root@ip-172-31-93-42 jenkins]# vi backup.pem**

**[root@ip-172-31-93-42 jenkins]# chmod 600 backup.pem**

**[root@ip-172-31-93-42 jenkins]# scp -i backup.pem backup-2025-08-29\_00-20.tar ec2-user@13.221.184.37:/tmp**

****

**Go to backup Jenkins terminal**

**[root@ip-172-31-87-43 ~]# cd /tmp/**

**[root@ip-172-31-87-43 tmp]# cd /var/lib/**

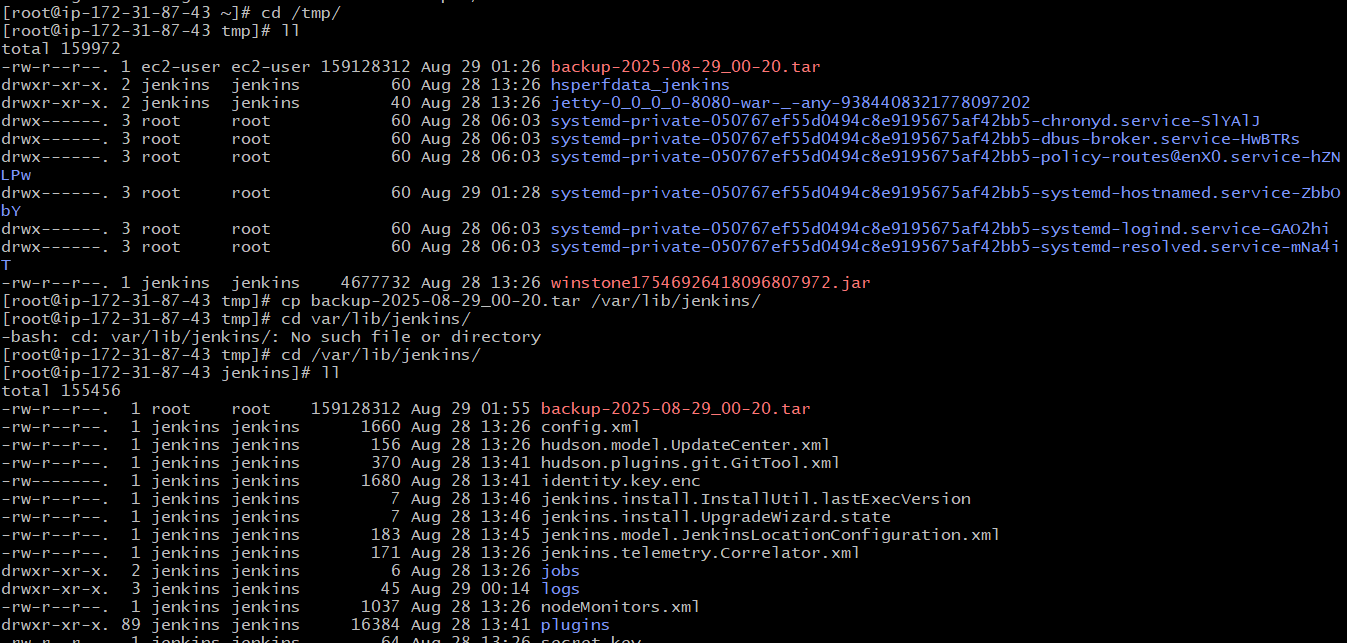
**[root@ip-172-31-87-43 tmp]# cp backup-2025-08-29\_00-20.tar /var/lib/**

**[root@ip-172-31-87-43 lib]# tar xvf backup-2025-08-29\_00-20.tar**

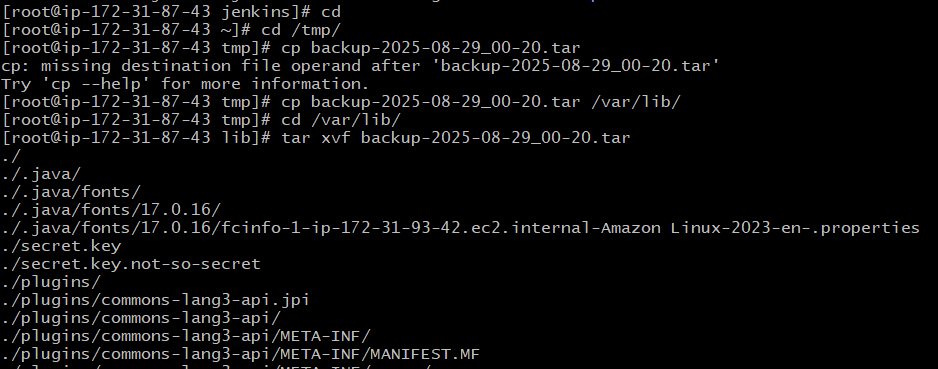
**[root@ip-172-31-87-43 lib]# cd jenkins**

**[root@ip-172-31-87-43 jenkins]# ll**

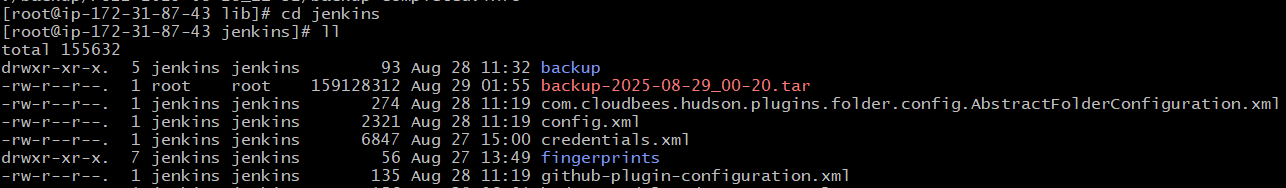
**Check .tar file reflecting or not and do the dumping process.**

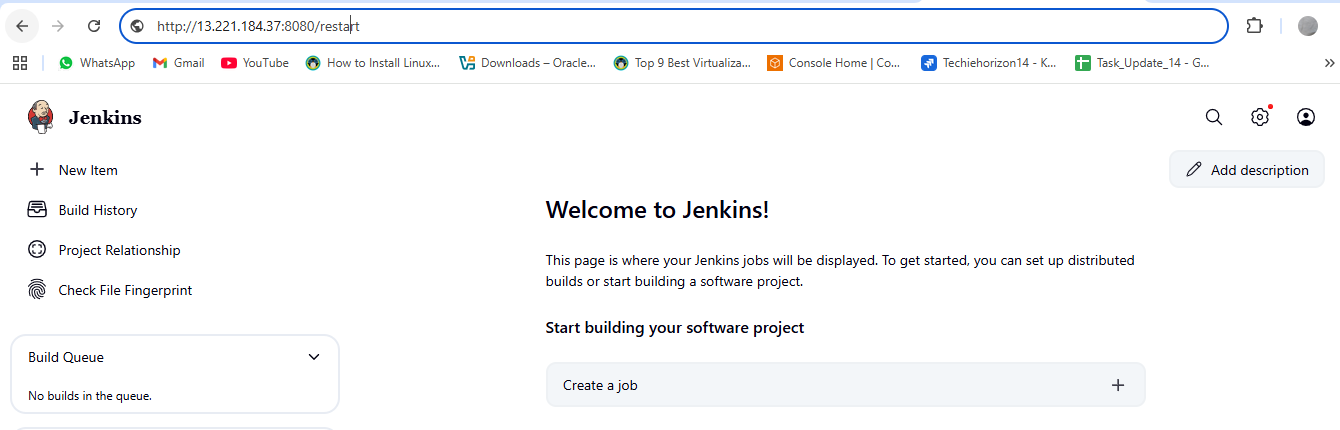
****

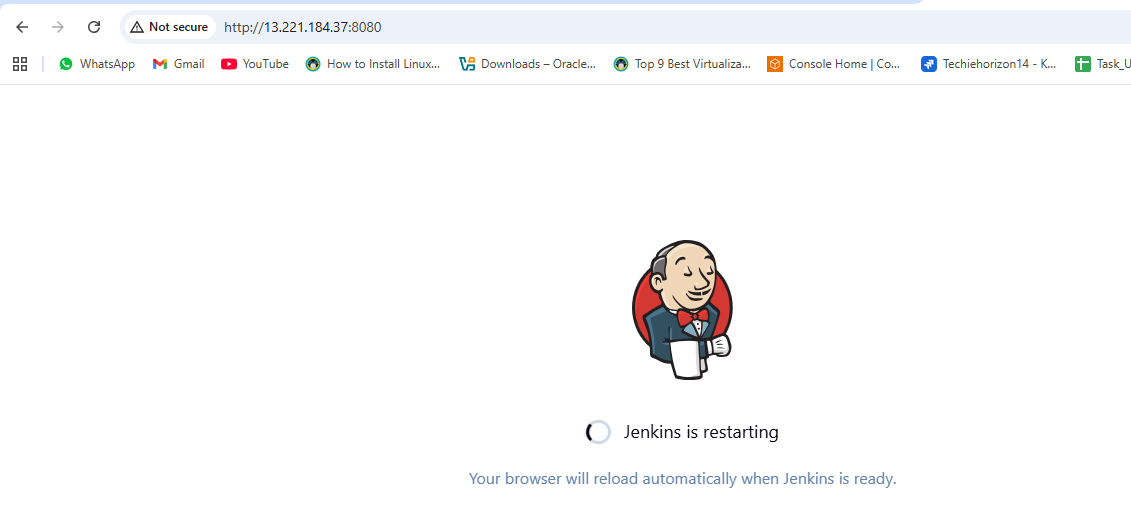
**Here copying and extracting**

****

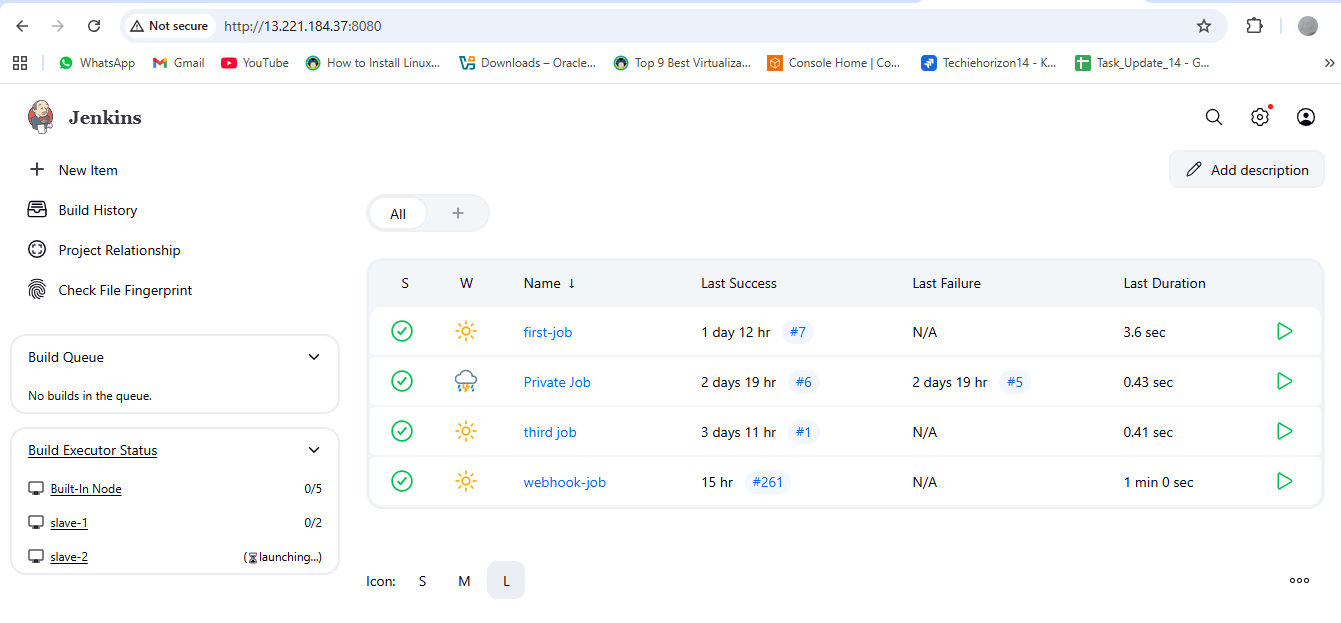
**Extract files reflecting in the Jenkins**

****

**Now need to restart jenkins**

****

**Login and check the all backup same as Master jenkins**

****

**Task has been completed**