Test Automation Academy

JENKINS INTERVIEW NOTES

Interview Success Strategies and Knowledge Nuggets

Written by Jatin Shharma

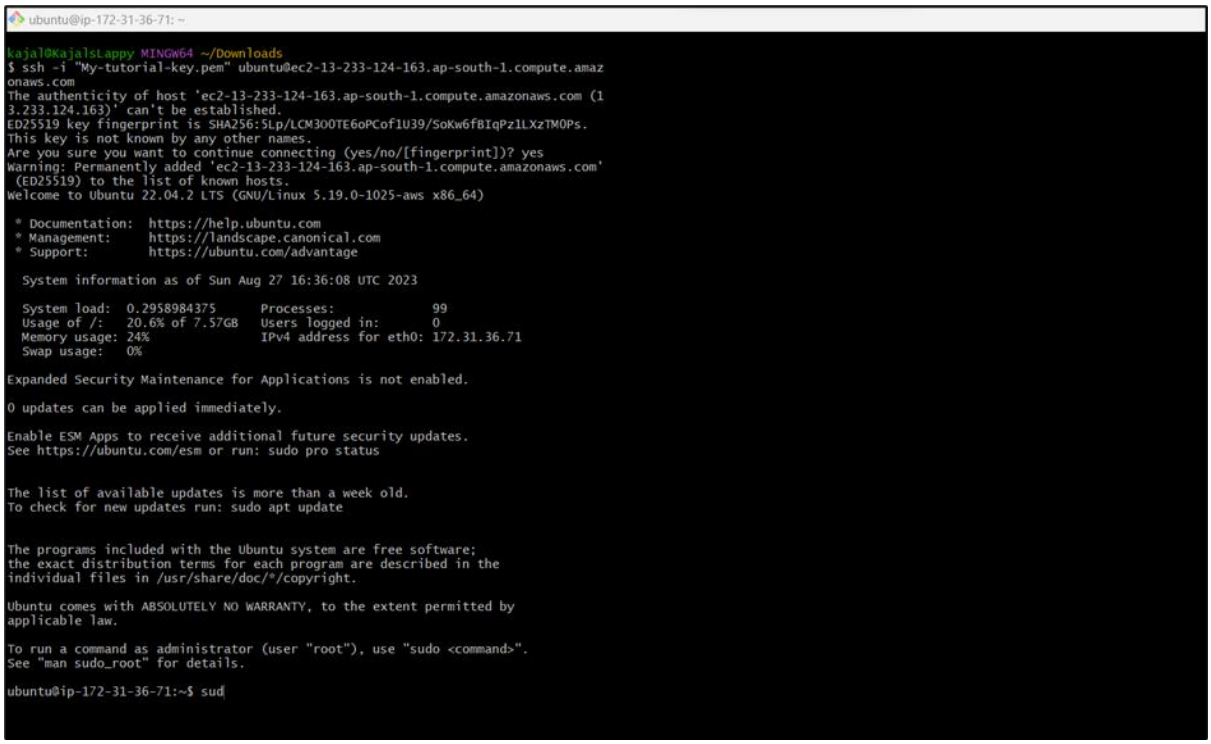
Jatin Shharma

51 Jenkins Interview Questions

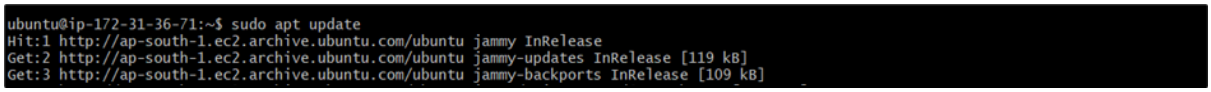
Jenkins Interview Questions

1. How to create a slave in Jenkins?

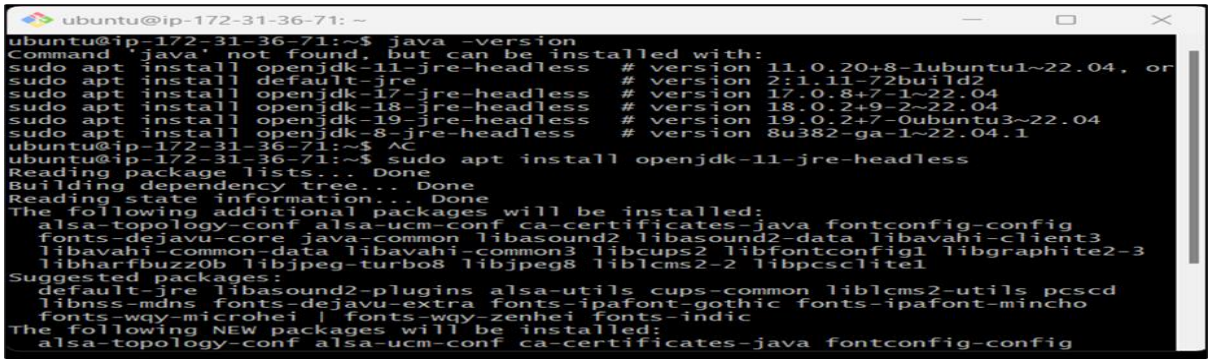
1. Connect to the server



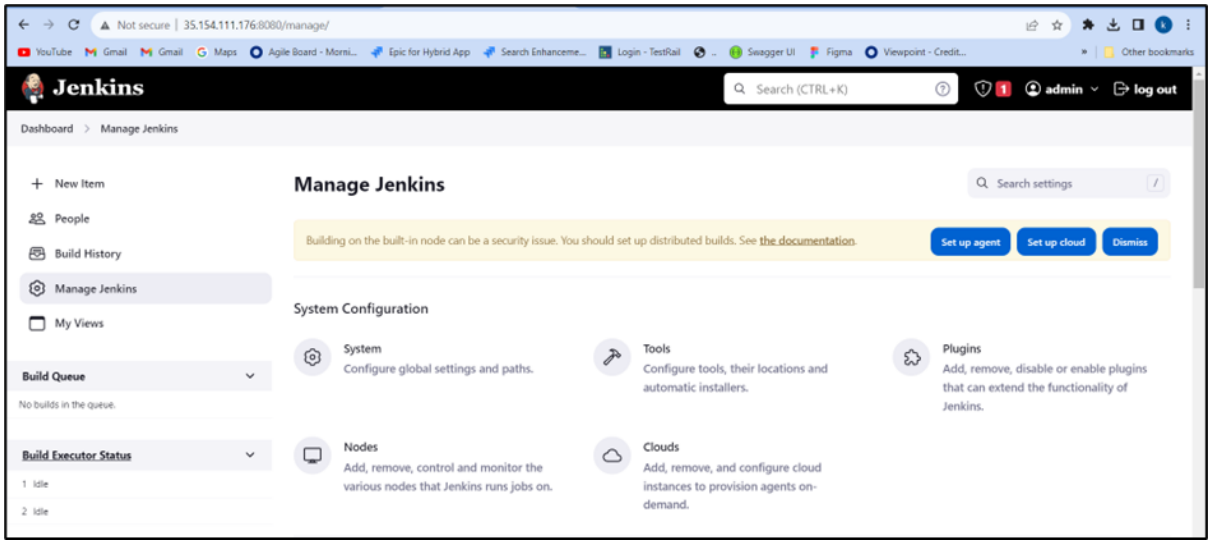
2. Sudo apt update



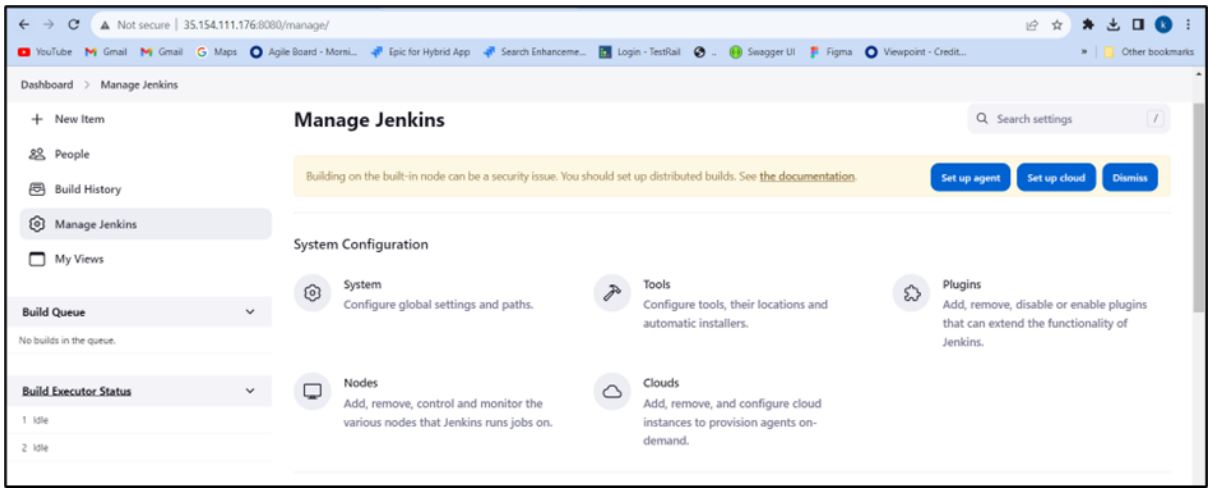
3. Install Java



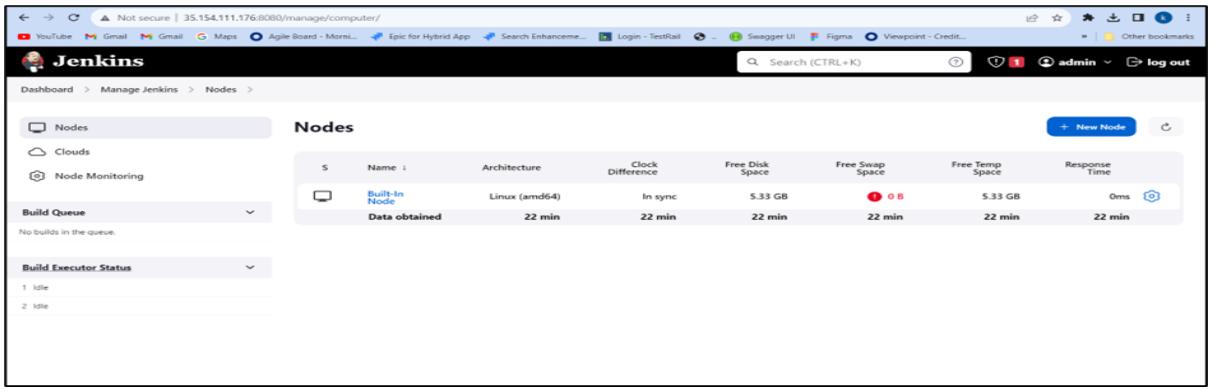
4. Click on manage Jenkins



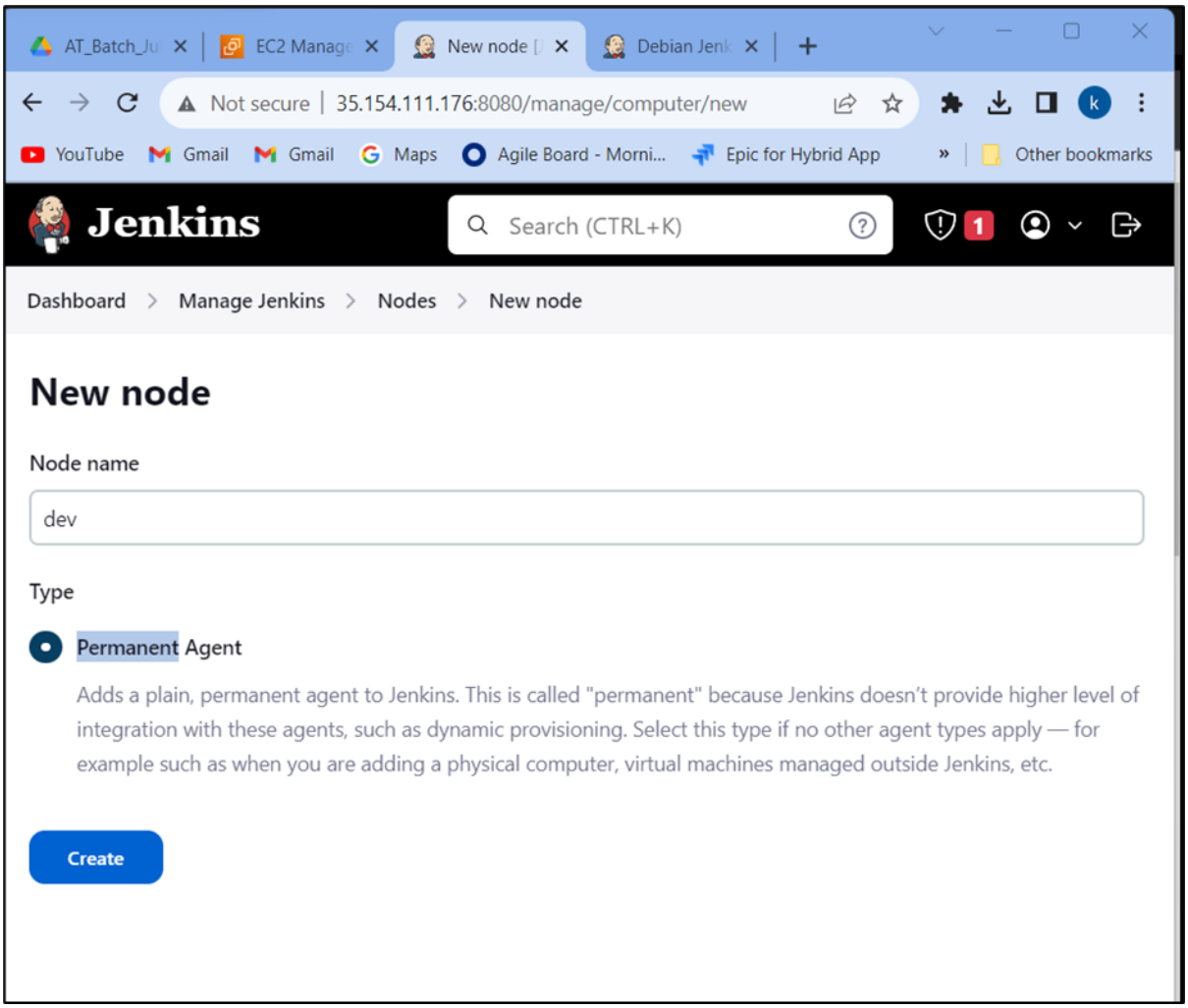
5. Click on ‘Nodes’



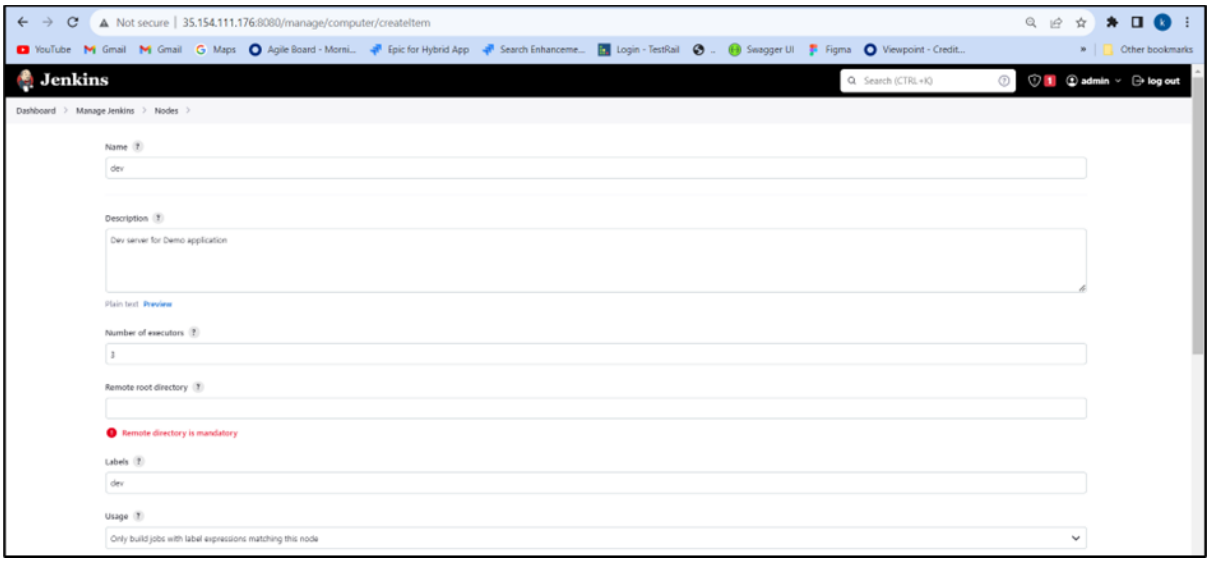
6. Click on ‘New Node’

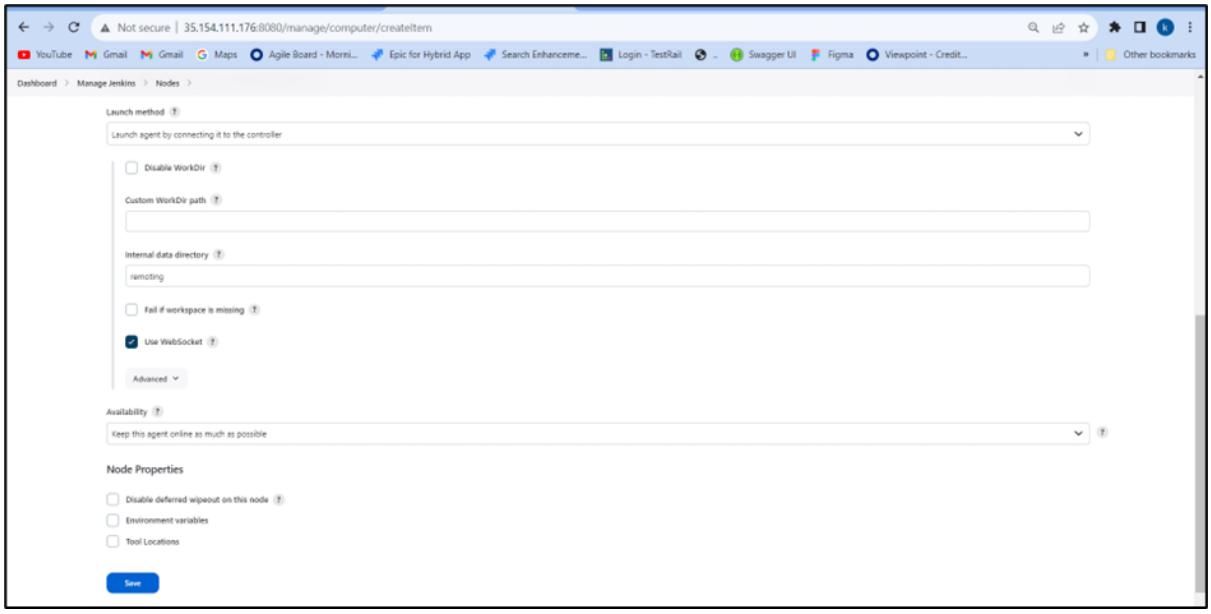


7. Give name and select ‘Permanent Agent’ and click on ‘create’ button

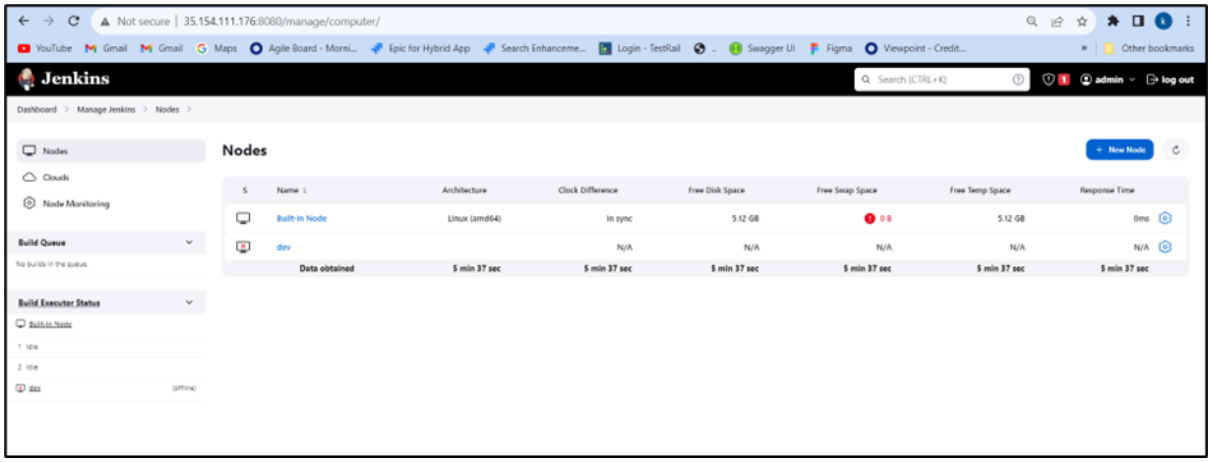


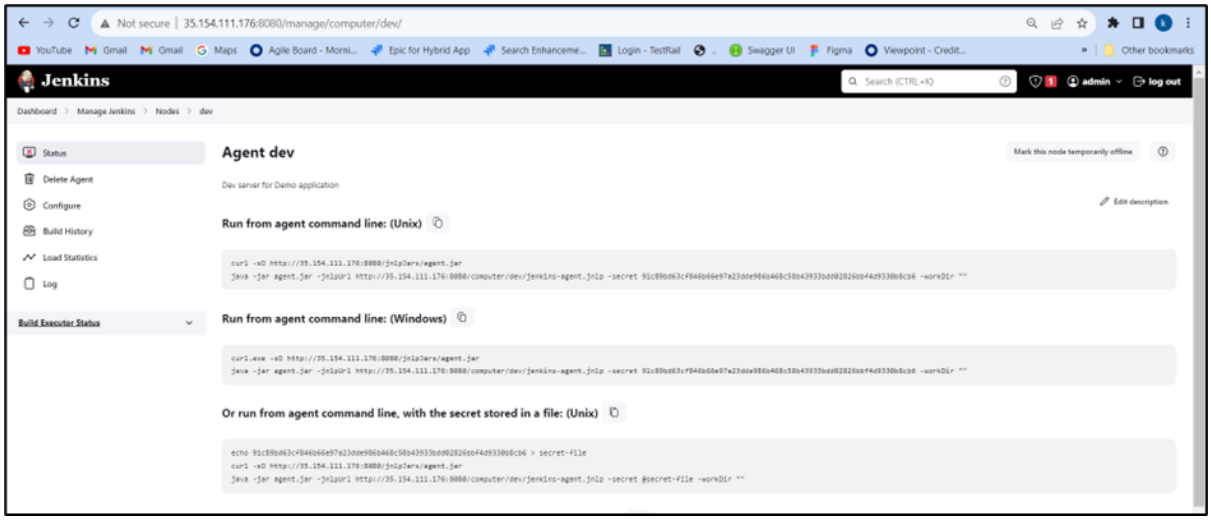
8. Fill the fields as given below





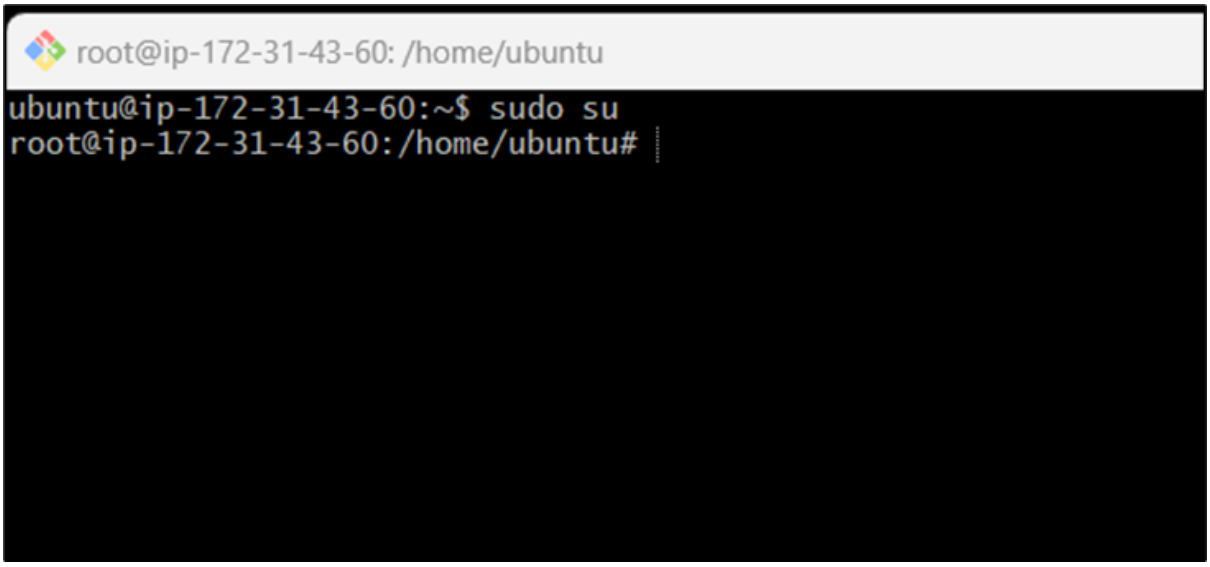
9. Click on the newly created node

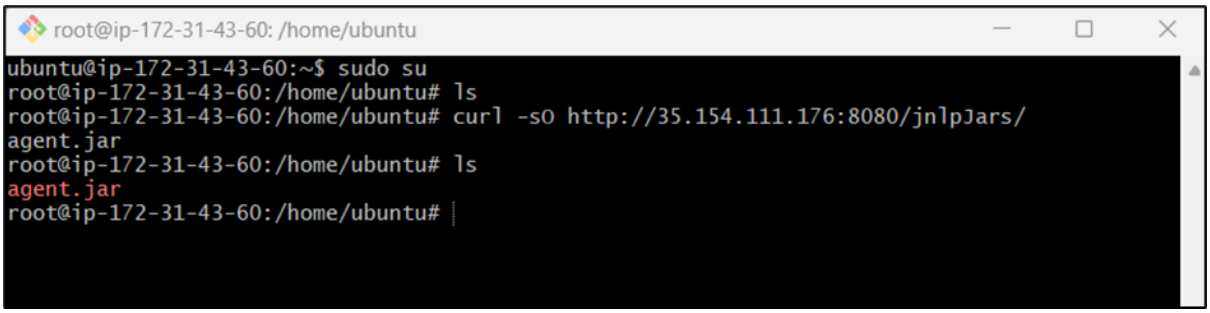




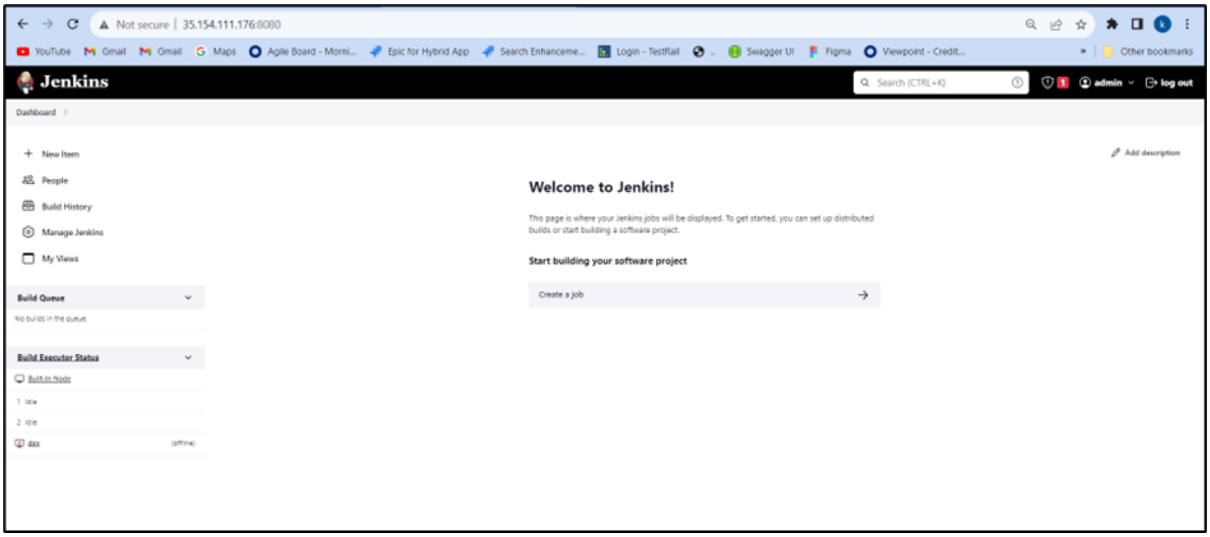
10. Execute these command one by one on the server and make sure to check the IP address is same for jenkins and commands

11. Before executing commands ***,Go to the sudo mode*** *by using* ***sudo su***

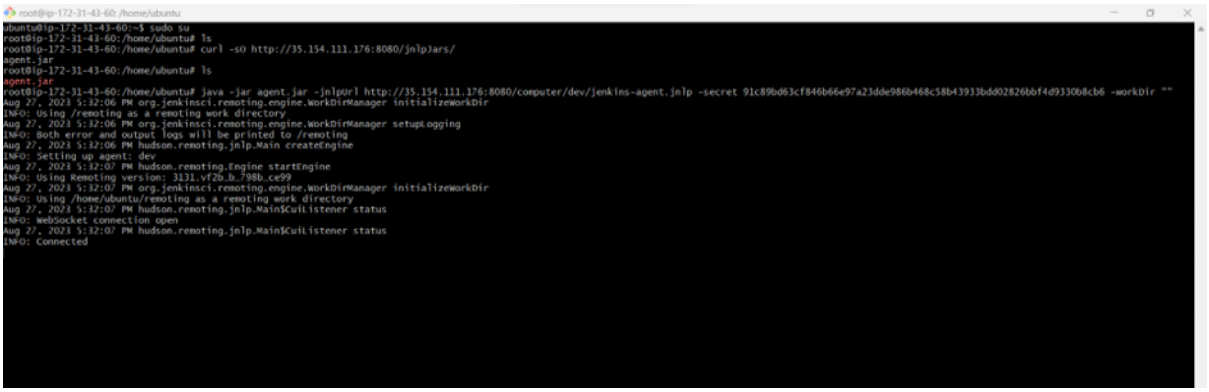




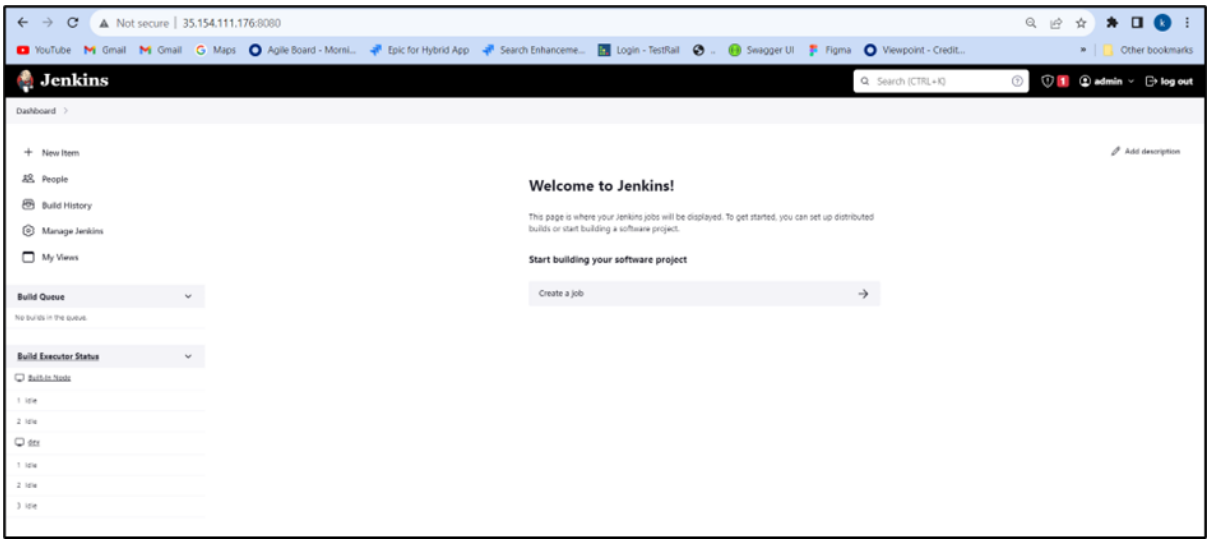
12. Check if the dev server is offline before executing the next command



13. Execute the next command

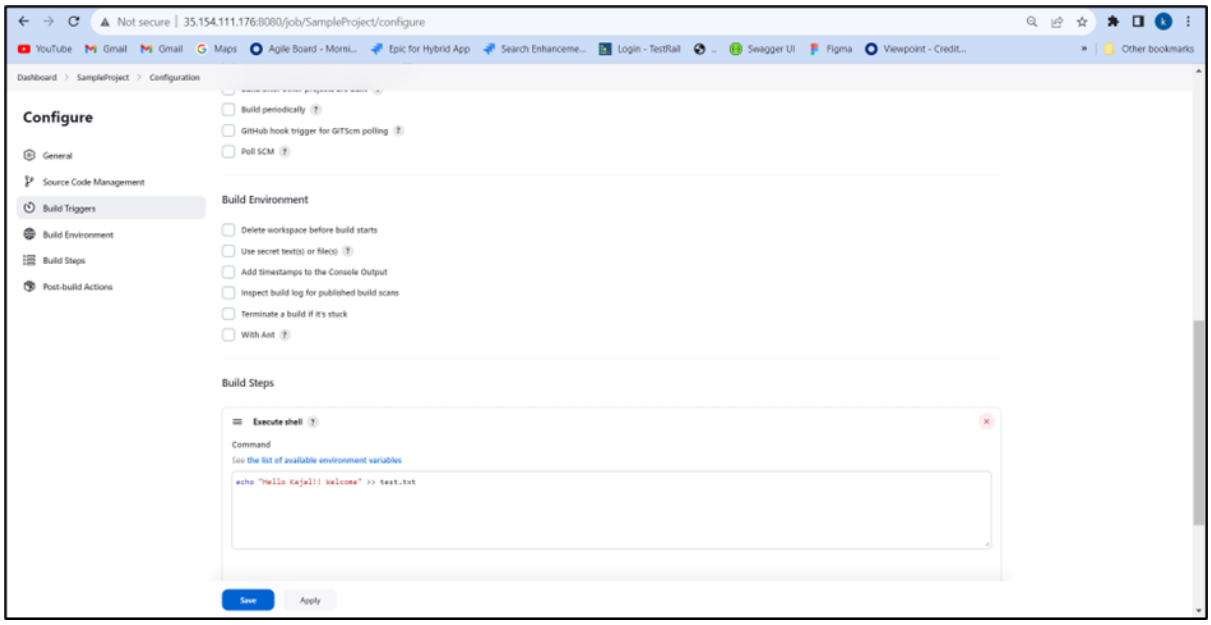


14. Check the status of ‘Dev’ server

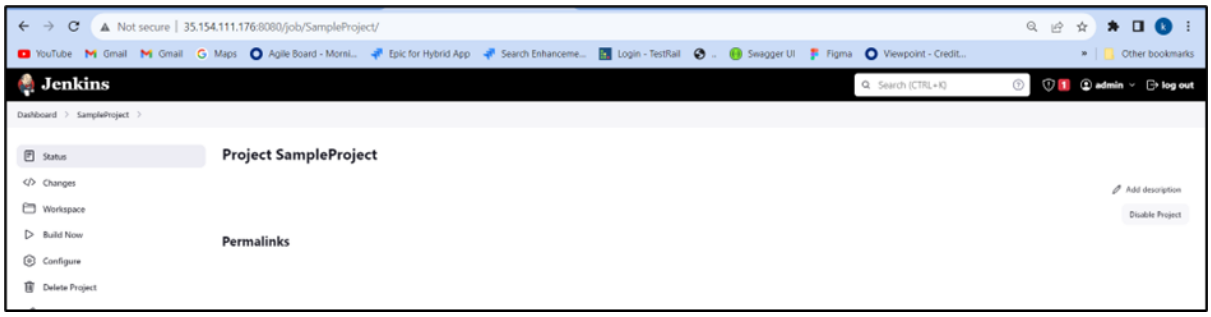


15. Create ‘Freestyle Project’ in Jenkins. make sure to select ‘Restrict where this Project can be run’ and select the label.

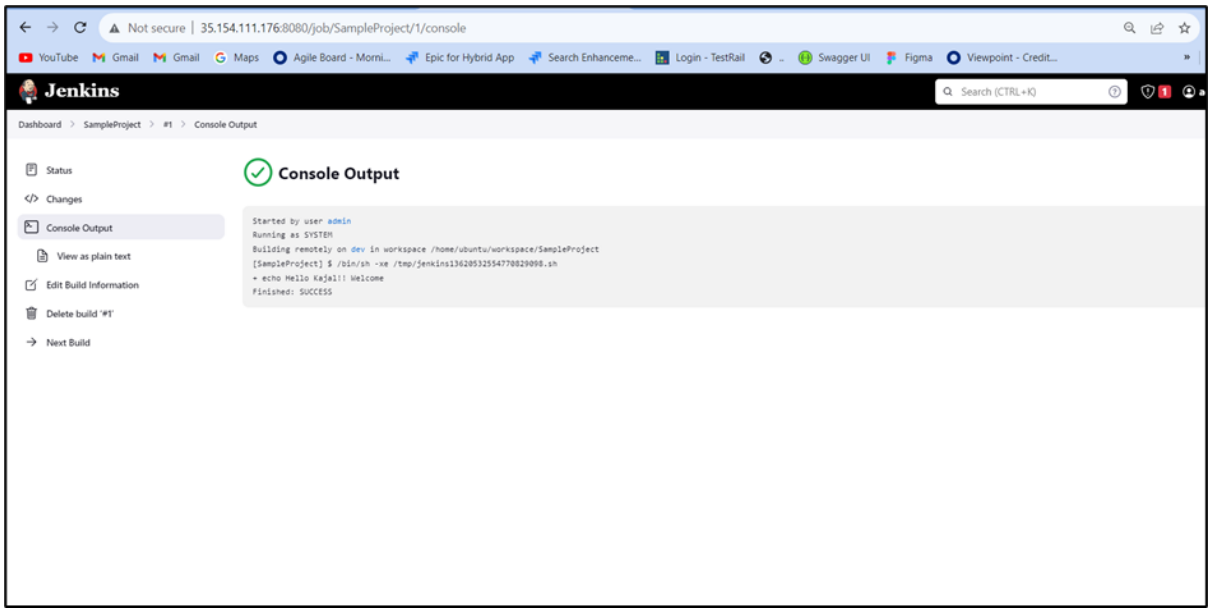
16. Select the ‘Build Steps’



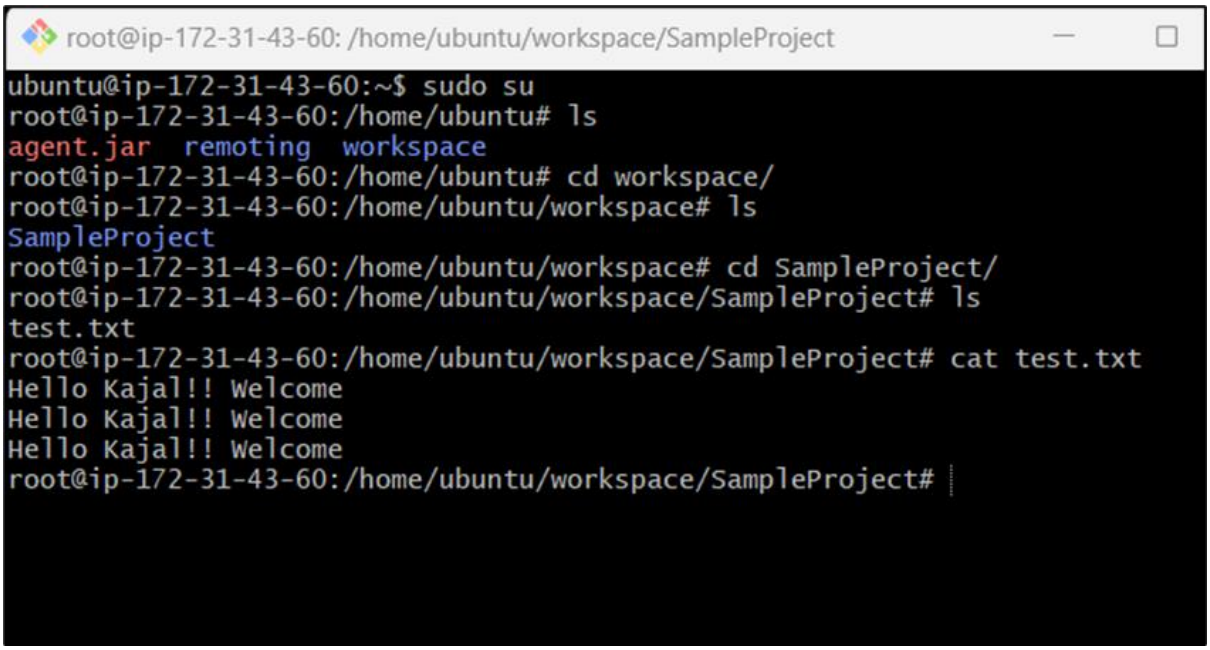
17. Click on ‘Build now’



18. Check the ‘console output’

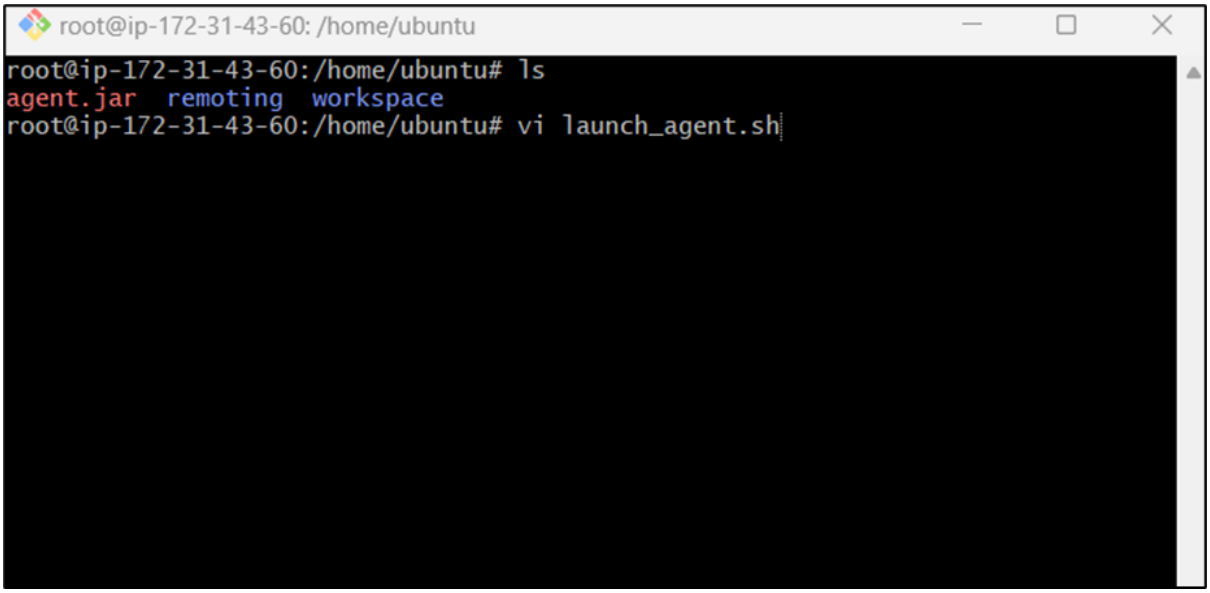


19. Check the server if the file that we added on the freestyle project is available or not



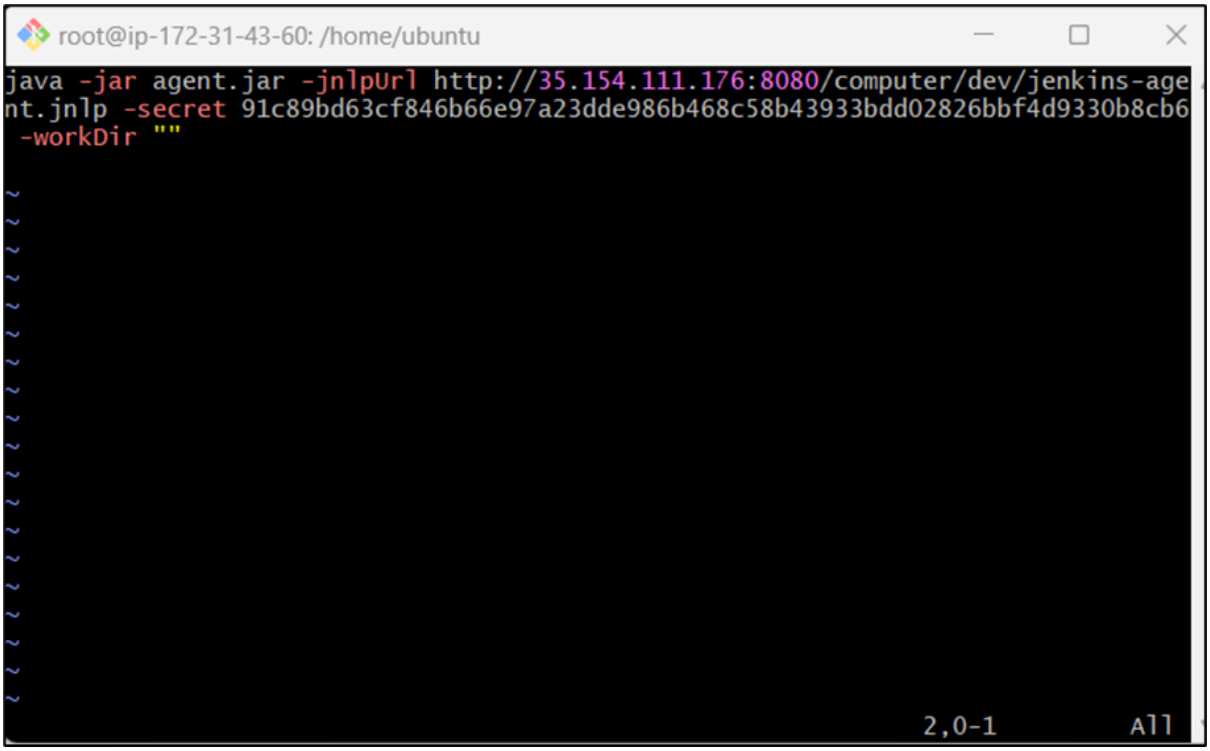
20. To make the dev server daemon, follow below steps

- Open launch_agent.sh

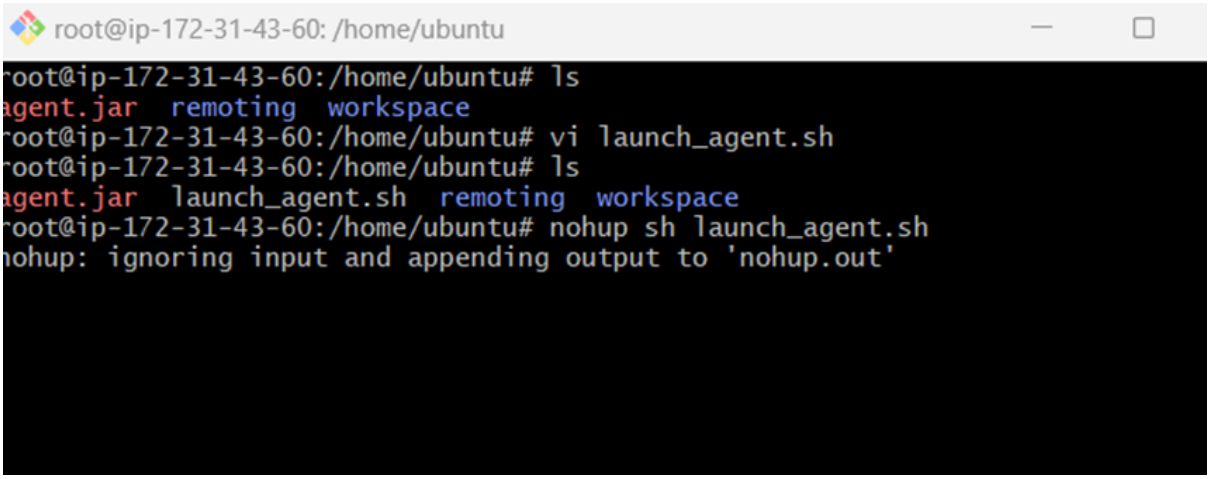


21. Paste that command

java -jar agent.jar -jnlpUrl http://35.154.111.176:8080/computer/dev/jenkins-agent.jnlp -secret 91c89bd63cf846b66e97a23dde986b468c58b43933bdd02826bbf4d9330b8cb6 -workDir ""



22. To run ‘sh’ file in background process use ‘nohup’ command.



Note:

Please note that if the server is disconnected and then reconnected, the IP address will change. Consequently, the IP address will also need to be updated in the following command:

java -jar agent.jar -jnlpUrl http://35.154.111.176:8080/computer/dev/jenkins-agent.jnlp -secret 91c89bd63cf846b66e97a23dde986b468c58b43933bdd02826bbf4d9330b8cb6 -workDir ""

Ensure to modify the IP address accordingly before executing the command after the server reconnection.

2. What is a pipeline in Jenkins?

- A Jenkins pipeline is a way to define, manage, and visualize the entire software delivery process in code. It encompasses building, testing, and deploying code. You can create pipelines using the "Pipeline DSL" or through visual editors like Blue Ocean. This approach allows for versioning, automation, and tracking of the CI/CD process, promoting efficiency and consistency.

3. How to schedule jobs in Jenkins?

- You can schedule jobs in Jenkins using the "Build Triggers" section in the job configuration. You can set up schedules using the "Build periodically" option, specifying a cron-like schedule expression. For example, to run a job every day at 3 AM, you can use 0 3 * * *.

    - ***Minute (0-59):*** *Specifies the minute for job triggering (0-59).*
    - ***Hour (0-23):*** *Specifies the hour for job triggering (0-23).*
    - ***Day of the Month (1-31):*** *Specifies the day of the month for job triggering (1-31).*
    - ***Month (1-12 or JAN-DEC):*** *Specifies the month(s) for job triggering (1-12 or JAN-DEC).*
    - ***Day of the Week (0-7 or SUN-SAT):*** *Specifies the day(s) of the week for job triggering (0-7 or SUN-SAT).*

    - ***Examples of cron expressions:***
    - *0 3 * * *: Daily at 3 AM*
    - **/15 * * * *: Every 15 minutes.*
    - *0 0 1 * *: First day of every month at midnight.*
    - *0 9 * * MON-FRI: Every weekday at 9 AM.*
    - *0 0 * 1,6 *: Midnight on the 1st and 6th of every month.*
- ***Note****: Always setup timezone prior to the cron.*

  TZ =Asia/Kolkata

  *0 3 * * *: Daily at 3 AM*

- *This will ensure that the job will run at 3AM IST.*

4. How to find the errors in Jenkins?

Jenkins error logs are typically stored **in the Jenkins master's file system**. The location of these logs can vary depending on your Jenkins installation and operating system. Here are some common locations:

Jenkins Master Log:

The primary log file for Jenkins is often referred to as the "Jenkins Master Log" or "Jenkins System Log."

On Linux systems, you can typically find this log file at: **/var/log/jenkins/jenkins.log**

Job-Specific Logs:

Each Jenkins job or build generates its own log***, which can be accessed from the build's details page.***

You can find the job-specific logs under the Jenkins workspace directory for that job. The path would be something like:
$JENKINS_HOME/workspace/YourJobName/builds/BuildNumber/log

5. What is the difference between schedule jobs and poll SCM in Jenkins?

- **Scheduled Jobs:** Scheduled jobs in Jenkins run at specific times based on a predefined schedule (e.g., every day at a certain time). They are time-triggered and not dependent on changes in source code repositories.

- **Poll SCM:** Polling SCM involves checking the source code repository for changes at regular intervals. When changes are detected, Jenkins triggers a build. Polling SCM is event-driven and depends on changes in the repository.

6. If you have 6 jobs, can we run some jobs on a Windows slave and some jobs on a Linux slave in Jenkins?

- Yes, you can run different jobs on different types of Jenkins slaves (nodes/agents).
- You can label your slaves with tags (e.g., "QA" or "Dev"), and then in your job configuration, specify which label a job should run on.
- Jenkins will schedule the job on an appropriate slave based on the label you've defined.
- **[Refer Jenkins Master slave setup]**

7. What is the use of Jenkins?

- Jenkins is a popular open-source automation server that's crucial for Continuous Integration (CI) and Continuous Delivery (CD) in software development.
- Its core purpose is to automate tasks like building, testing, and deploying software, making development faster and more reliable.
- Jenkins also offers a vast selection of plugins to customize and extend its functionality.

8. What is Continuous Integration (CI)?

Continuous Integration, often abbreviated as CI, is a software development practice and process. It revolves around the idea of frequently integrating code changes made by multiple developers into a shared codebase. The core principles of CI include:

- **Frequent Integration**: Developers regularly merge their code changes into a central repository, multiple times a day if possible. This ensures that everyone's work is continuously integrated.

- **Automated Build and Testing**: As code changes are integrated, an automated CI system kicks in. It automatically builds the software and runs a suite of tests, including unit tests and integration tests, to identify issues and ensure that the code functions correctly.

- **Immediate Feedback**: CI systems provide immediate feedback to developers. If a code change causes a problem or breaks existing functionality, developers are alerted right away. This quick feedback loop helps in addressing issues promptly.

- **Isolation of Changes**: CI encourages small, focused code changes. If a problem arises during integration, it's easier to pinpoint the specific code change that caused the issue, making debugging and fixing problems more efficient.

- **Version Control:** Version control systems, like Git, are essential in CI. They allow developers to track changes, collaborate effectively, and revert to previous code states if necessary.

- **Deployment Readiness**: CI pipelines often include tasks related to preparing code for deployment to various environments, ensuring that the code is always in a deployable state.

The primary benefits of CI include early issue detection, consistent and reproducible builds, faster development cycles, and improved collaboration among development teams. CI ultimately contributes to higher software quality and more reliable releases

9. How does Jenkins support Continuous Integration?

- Jenkins automates the building, testing, and deployment of code changes, helping to maintain a consistent development workflow.

10. What are the two types of Jenkins pipelines?

- Declarative pipeline and Scripted pipeline.

Aspect	Declarative Pipeline	Scripted Pipeline
Syntax	Uses a more structured and simplified syntax.	Uses a Groovy-based scripting language.
Ease of Use	Designed for simplicity and readability.	More flexible but requires greater expertise.
Abstraction Level	Provides a higher-level abstraction.	Allows low-level control over the pipeline.
Readability	Generally easier to read and understand.	Can become complex and harder to read.
Extensibility	Limited extensibility compared to Scripted.	Highly extensible and customizable.
Best Use Cases	Ideal for simple to moderately complex tasks.	Suitable for highly customized workflows.
Error Handling	Provides some built-in error handling.	Requires explicit error handling code.
Parallel Execution	Supports parallel stages but with limitations.	Supports more advanced parallelization.
Built-in Features	Fewer built-in features and functions.	Offers a wide range of built-in functions.

11. How do you create a simple Jenkins job?

- In Jenkins, create a new job, select the appropriate job type (e.g., Freestyle project), configure the job details, and define build steps.

12. What is a Jenkins agent (or slave)?

- A Jenkins agent is a **machine** that connects to the Jenkins ***server to execute tasks as part of a build or deployment process.***

13. How do you define environment variables in Jenkins?

- Environment variables can be defined globally in the Jenkins configuration or within a specific job's configuration.

Declarative Pipeline:

In a Declarative Pipeline, you can access environment variables using the environment block. Here's an example:

```
pipeline {
    agent any
    environment {
    MY_VARIABLE = 'some_value'
    }
    stages {
        stage('Example Stage') {
            steps {
            echo "The value of MY_VARIABLE is ${env.MY_VARIABLE}"
            }
        }
```

```
        }
    }
```

In this example, we define the ***MY_VARIABLE environment variable*** within the environment block, and then we access it using **${env.MY_VARIABLE}** within the echo step.

Scripted Pipeline:

In a Scripted Pipeline, you can access environment variables directly using the env object. Here's an example:

```
node {
    def myVariable = env.MY_VARIABLE
    echo "The value of MY_VARIABLE is ${myVariable}"
}
```

In this example, we assign the value of the MY_VARIABLE environment variable to a variable called myVariable, and then we echo its value.

Jenkins automatically provides several environment variables that you can access, such as **BUILD_NUMBER, JOB_NAME, WORKSPACE**, and many more. You can also set your custom environment variables as shown in the examples above.

14. What is a Jenkinsfile?

- A Jenkinsfile is a text file that defines the structure and steps of a Jenkins pipeline. It serves as the configuration file for a specific pipeline job in Jenkins. Jenkins pipelines are used for defining and automating the entire CI/CD process, including building, testing, and deploying applications.

- **Key points about Jenkinsfiles:**
    - **Pipeline Configuration**: A Jenkinsfile contains the configuration for a pipeline job. It defines the stages, steps, and other instructions that Jenkins should follow to build, test, and deploy your software.

    - **Declarative and Scripted Syntax:** Jenkins supports two types of syntax for defining pipelines within Jenkinsfiles: Declarative and Scripted. Declarative syntax is more structured and easy to read, while Scripted syntax provides more flexibility and control through Groovy scripting.

    - **Version Control**: Jenkinsfiles are typically stored alongside your application code in your version control system (e.g., Git). This ensures that your pipeline configuration is versioned and can be easily tracked over time.

    - **Reproducibility**: Jenkinsfiles help ensure that your CI/CD process is reproducible across different environments. By defining the pipeline steps in code, you minimize the "it works on my machine" problem.

    - **Pipeline as Code**: Jenkinsfiles are a key component of the "Pipeline as Code" approach, where the entire CI/CD pipeline is defined, versioned, and managed as code. This makes it easier to collaborate on and automate complex workflows.

    - **Extensibility**: Jenkinsfiles can be extended with various plugins and integrations to support specific build, test, and deployment requirements. You can include custom scripts, conditional logic, and more.

    - **Multibranch Pipelines:** Jenkinsfiles are often used in conjunction with multibranch pipelines, where Jenkins can automatically discover and create pipelines for different branches of your code repository.

- Here's a simple example of a Declarative Jenkinsfile:

```
pipeline {
        agent any
    stages {
            stage('Build') {
                    steps {
                        sh 'make build'
                    }
            }
            stage('Test') {
                    steps {
                        sh 'make test'
                    }
            }
            stage('Deploy') {
                    steps {
                        sh 'make deploy'
                    }
                }
        }
}
```

15. How do you trigger a Jenkins job based on a code commit?

- The most efficient way to trigger Jenkins on code commits is by setting up a webhook in your version control system. This webhook will send a notification to Jenkins whenever a commit is made to the repository.

- ***In GitHub or Gitlab, for example, you can go to your repository's settings, select "Webhooks," and add a new webhook that points to your Jenkins server's webhook URL***

16. How do you install plugins in Jenkins?

- In the Jenkins dashboard, navigate to "Manage Jenkins" > "Manage Plugins" and install the desired plugins from the available list.

17. What is a "Freestyle project" in Jenkins?

- A Freestyle project is a type of Jenkins job that allows you to configure a series of build steps and post- build actions using a graphical interface.

18. How do you configure a Jenkins pipeline using the Declarative syntax?

- Define the pipeline in a Jenkinsfile using the Declarative syntax, which provides a structured way to define stages, steps, and post-actions.

19. How can you ensure that Jenkins jobs are triggered in a specific order?

- Use the "Build after other projects are built" option in the job configuration to define the dependencies between jobs.

20. What is Jenkins' Blue Ocean plugin?

- Blue Ocean is a modern interface for Jenkins pipelines, providing a more intuitive visualization of pipeline stages and status.

21. How can you parameterize a Jenkins job?

- In the job configuration, enable the "This project is parameterized" option and define the parameters you want to use.

22. How do you archive build artifacts in Jenkins?

- Use the "Archive the artifacts" post-build action in the job configuration to save important files for future reference.

23. How can you schedule a Jenkins job to run at a specific time?

- Use the "Build periodically" option in the job configuration to define a cron-like schedule for job execution.
- **SETUP a CRON JOB**
- You can schedule jobs in Jenkins using the "Build Triggers" section in the job configuration. You can set up schedules using the "Build periodically" option, specifying a cron-like schedule expression. For example, to run a job every day at 3 AM, you can use 0 3 * * *.

    - ***Minute (0-59):*** *Specifies the minute for job triggering (0-59).*
    - ***Hour (0-23):*** *Specifies the hour for job triggering (0-23).*
    - ***Day of the Month (1-31):*** *Specifies the day of the month for job triggering (1-31).*
    - ***Month (1-12 or JAN-DEC):*** *Specifies the month(s) for job triggering (1-12 or JAN-DEC).*
    - ***Day of the Week (0-7 or SUN-SAT):*** *Specifies the day(s) of the week for job triggering (0-7 or SUN-SAT).*

    - ***Examples of cron expressions:***
    - *0 3 * * *: Daily at 3 AM*
    - **/15 * * * *: Every 15 minutes.*
    - *0 0 1 * *: First day of every month at midnight.*
    - *0 9 * * MON-FRI: Every weekday at 9 AM.*
    - *0 0 * 1,6 *: Midnight on the 1st and 6th of every month.*
- ***Note****: Always setup timezone prior to the cron.*

  TZ =Asia/Kolkata

  *0 3 * * *: Daily at 3 AM*
- *This will ensure that the job will run at 3AM IST.*

24. What is the Jenkins "Workspace"?

- The Workspace is a directory on the Jenkins agent where the job's code and build artifacts are stored during the build process.

- In Jenkins, the "workspace" refers to a directory on the Jenkins master or agent node where a specific job's files and build artifacts are stored during the execution of that job. Each Jenkins job has its own dedicated workspace, and this workspace is used to store the source code, build scripts, and any other files required for the job's build or automation process.

25. How can you trigger a Jenkins job remotely?

- There are several ways to trigger a Jenkins job remotely, depending on the specific use case and requirements of the job. Here are some common methods:

- **Jenkins Remote API (HTTP Requests):** You can use HTTP POST requests to trigger a Jenkins job remotely. For example, you can use tools like curl or scripts to send a POST request to the job's URL.

    - ***curl -X POST http://JENKINS_URL/job/JOB_NAME/build***

- **Jenkins CLI (Command Line Interface):** The Jenkins CLI allows you to trigger jobs from the command line using commands like build or build-job.

    - ***java -jar jenkins-cli.jar -s http://JENKINS_URL/ build JOB_NAME***

- **Webhooks**: If you want to trigger jobs automatically based on external events (e.g., code pushes), you can set up webhooks. When an event occurs, such as a code push to a repository, a webhook can trigger the associated Jenkins job.

- **Scheduled Builds (Polling):** Jenkins supports scheduled builds, where you can configure a job to poll a version control system for changes and trigger a build when changes are detected.

- **Third-Party Plugins**: Jenkins has a rich ecosystem of plugins. Some plugins, like the "GitHub Plugin" or "GitLab Plugin," allow you to trigger Jenkins jobs as a result of specific events in your version control system.

- **Authentication and Security**: Security is critical when triggering jobs remotely. You need to configure proper authentication and authorization mechanisms in Jenkins to ensure that only authorized users or systems can initiate job builds.

- **Token-based Authentication**: Jenkins supports token-based authentication, where users or external systems include a token in the request URL to authenticate.

26. What is a Jenkins "Freestyle Maven project"?

- A Freestyle Maven project in Jenkins is used for building and managing Maven-based Java projects using a graphical interface.

27. How do you integrate Jenkins with version control systems like Git?

- In the job configuration, select the appropriate Git integration option (e.g., "Source Code Management" > "Git") and provide the repository URL.

28. What is Jenkins "Pipeline as Code"?

- Pipeline as Code refers to defining Jenkins pipelines in a version-controlled Jenkinsfile, allowing for automation, versioning, and collaboration.

29. How can you set up email notifications in Jenkins?

- In the job configuration, use the "Editable Email Notification" post-build action to configure email notifications for build results.

30. How can you secure your Jenkins instance?

- Use security plugins, configure user authentication, and implement access control to secure your Jenkins environment.

31. How do you configure a Jenkins pipeline using the Scripted syntax?

- "Pipeline as Code" in Jenkins refers to the practice of defining and managing your Continuous Integration and Continuous Delivery (CI/CD) pipelines using code. Instead of configuring pipelines through the Jenkins web interface, Pipeline as Code allows you to define your pipelines in code files (typically in a version-controlled repository) using either Declarative or Scripted Pipeline syntax.

    - **Version Control:** Pipeline configurations are stored in code files (e.g., Jenkinsfiles), which can be versioned using a source code management system (e.g., Git). This enables you to track changes, collaborate with others, and roll back to previous configurations if needed.

    - **Reproducibility**: With pipeline configurations defined in code, the entire CI/CD process becomes reproducible. You can recreate the exact pipeline environment and steps at any time, reducing the "it works on my machine" problem.

    - **Collaboration**: Multiple team members can contribute to pipeline configurations using standard development practices, such as code reviews and pull requests.

    - **Automation**: Pipelines defined as code are automatically executed by Jenkins whenever code changes occur, ensuring consistent and repeatable builds and deployments.

    - **Flexibility**: Code-based pipelines can include custom logic, conditional statements, and integrations with third-party tools and services. This flexibility allows you to tailor your CI/CD process to your specific needs.

32. What is the purpose of a "Jenkins Shared Library"?

- A Jenkins Shared Library is a valuable component in Jenkins that serves multiple purposes to streamline and improve the efficiency of CI/CD processes. Its main purpose is to promote code reuse, maintainability, and consistency across various Jenkins pipelines. Here are the key reasons why organizations use Jenkins Shared Libraries:

    - **Code Reusability**: Shared Libraries allow teams to define and share common functions, steps, and resources, reducing duplication of code across projects. This promotes the 'Don't Repeat Yourself' (DRY) principle and makes it easier to manage and update shared functionality.

- **Consistency**: By centralizing commonly used logic in a Shared Library, you ensure that all pipelines adhering to the library follow the same standards and best practices. This consistency enhances the reliability and quality of CI/CD processes.

- **Encapsulation of Complexity**: Complex logic, integration with external tools, and custom business processes can be encapsulated within a Shared Library. This simplifies pipeline configurations in individual projects, making them more readable and maintainable.

- **Maintenance and Updates**: When changes or improvements are required in CI/CD processes, updates can be made to the Shared Library in a single location. All pipelines using the library automatically benefit from these updates, eliminating the need to modify each project individually.

- **Version Control**: Shared Libraries can be version-controlled using tools like Git, allowing teams to track changes, roll back to previous versions, and manage library releases effectively.

- **Collaboration**: Shared Libraries promote collaboration between DevOps or CI/CD specialists and application development teams. Specialists can develop and maintain shared functionality independently, facilitating cross-team cooperation.

- **Security and Compliance**: Shared Libraries can include security and compliance checks, ensuring that all pipelines using the library adhere to necessary security and compliance standards.

33. How can you integrate Jenkins with Docker for build and deployment?

- Use Docker plugins or Docker commands within Jenkins pipeline stages to build, test, and deploy Docker containers.

34. How do you trigger downstream jobs in Jenkins?

- Use the "Build other projects" post-build action in the job configuration to trigger downstream jobs upon successful completion.

35. How do you archive and publish JUnit test reports in Jenkins?

- Use the "Publish JUnit test result report" post-build action to publish JUnit test results for visualization and analysis.

36. What is the role of the "Jenkins Artifactory Plugin"?

- The Artifactory Plugin integrates Jenkins with JFrog Artifactory, enabling artifact management and distribution.

37. How can you prevent concurrent executions of a Jenkins job?

- Use the "Execute concurrent builds if necessary" option in the job configuration to control concurrent execution of the same job.

38. What is the purpose of the "Jenkins Pipeline Syntax" tool?

- The Pipeline Syntax tool helps you generate and experiment with various pipeline syntax options using a web interface.

39. How can you archive build logs in Jenkins?

- Use the "Archive the artifacts" post-build action and specify the path to the log files to archive build logs.

40. How do you use the "Promoted Builds" plugin in Jenkins?

- The Promoted Builds plugin allows you to promote specific build artifacts to higher-level environments using manual approval steps.

41. What is the role of the "Jenkins GitHub Integration" plugin?

- The GitHub Integration plugin enables seamless integration between Jenkins and GitHub repositories, including webhooks and status updates.

42. How can you set up a Jenkins job to trigger on multiple conditions?

- Use the "Build after other projects are built" option and define multiple project names to trigger a job on the completion of any of those projects.

43. What is a Jenkins "Parameterized Trigger" plugin used for?

- The Parameterized Trigger plugin allows you to trigger downstream jobs with parameters based on the outcome of a build.

44. How can you archive build artifacts to an external repository like Nexus?

- Use Jenkins plugins like the Nexus Artifact Uploader plugin to automatically publish build artifacts to external repositories.

45. What is the purpose of the "Jenkins Pipeline Unit" testing framework?

- Jenkins Pipeline Unit allows you to write and execute unit tests for your Jenkins pipeline code.

46. How can you trigger a Jenkins pipeline on a Git branch other than the default?

- Use the Git plugin along with branch specifier in the job configuration to trigger the pipeline on a specific branch.

47. What is the Jenkins "Matrix Project" job type used for?

- A Matrix Project job allows you to run a single job with multiple configurations, enabling comprehensive testing on various platforms.

48. How do you create a Jenkins job to deploy a Java application to Tomcat?

- Use plugins like the "Deploy to container" plugin or integrate the deployment script in a Jenkins pipeline.

49. How do you use the "Conditional BuildStep" plugin in Jenkins?

- The 'Conditional BuildStep' plugin is a useful Jenkins plugin that allows you to define conditions under which specific build steps should be executed within a Jenkins job. It provides a way to introduce conditional logic into your build process.
- Here's how you can use it:
    - **Install the 'Conditional BuildStep' Plugin (if not already installed):**
        - Before you can use the plugin, make sure it's installed on your Jenkins instance. You can install it from the Jenkins Plugin Manager.
    - **Create or Open a Jenkins Job:**
        - You can either create a new Jenkins job or open an existing one that you want to add conditional build steps to.
    - **Configure Your Build Steps:**
        - Within the job configuration, locate the section where you define the build steps (e.g., Execute shell, Windows batch command, etc.).
    - **Add a Conditional BuildStep:**
        - Click on the "Add build step" dropdown, and you should see an option for "Conditional step (single)". Select it.
    - **Configure the Conditional Step:**
        - Once you've added the conditional step, you'll see a configuration interface where you can define the conditions and the steps to execute.
        - In the "Conditions" section, specify the condition under which you want the build steps to run. You can choose from various conditions, such as:
        - Execute only if the build status is successful or failed.
        - Execute based on the result of a previous build step.
        - Execute based on the value of an environment variable or a build parameter.
        - Customize the condition using a Groovy script.
        - In the "Steps" section, define the build steps that should run when the condition is met. You can use any build step available in Jenkins, such as running shell commands, invoking scripts, or triggering other Jenkins jobs.

- **Save Your Job Configuration:**
  - Once you've configured the conditional build step, save your Jenkins job configuration.
- **Build Your Job:**
  - When you build the job, the Conditional BuildStep plugin will evaluate the specified condition(s). If the condition(s) are met, the associated build steps will be executed; otherwise, they will be skipped.
- **Monitor the Job Output:**
  - During the job execution, you can monitor the console output to see which build steps were executed based on the conditions.

50. What is a "Jenkins Multibranch Pipeline" job?

- A Multibranch Pipeline job automatically creates pipelines for each branch in a Git repository, facilitating CI/CD for multiple branches.

51. How can you trigger a Jenkins job remotely using a token?

- Use a URL in the format JENKINS_URL/job/JOB_NAME/build?token=TOKEN_NAME to trigger a job remotely with a specific token.