Jenkins Tutorial for beginners

Introduction

Most students find the complicated setup, jargon-baited language, and intimidation that Continuous Integration/Continuous Delivery (CI/CD) is only suitable for experts. Jenkins is the de facto automation tool for automating your build, test, and deployment pipelines. We're going to begin simple, taking the automation out of complicated and into easy and accessible through this Jenkins tutorial for beginners. Ready to take on CI/CD? Take a look at the complete **Jenkins course syllabus** today!

Why Student or Freshers Learn Jenkins

Students and fresher should master Jenkins as it is the most widely used, open-source automation server in the world, and hence a key skill for contemporary tech jobs:

- **Key DevOps Skill:** Jenkins is responsible for automating CI/CD (Continuous Integration/Continuous Delivery), which is at the core of contemporary software development.

- **High Employability:** It's an absolute must-have tool for DevOps, Software Development, and Quality Assurance jobs, greatly enhancing the resume.
- **Practical Experience:** It enables freshers to create and execute actual build, test, and deployment pipelines, translating theoretical experience into practical skills.
- **Industry Standard:** Jenkins expertise reflects awareness of automated processes and faster, quality code deployment.

Preparing for your job search? Download our comprehensive **Jenkins Interview Questions and Answers** now!

Step-by-Step Jenkins Tutorial for Beginners

This in-depth, step-by-step guide walks new users through installing and configuring Jenkins, and then building your first automated job, an instance of a pipeline.

Step 1: Jenkins Installation and Setup

Since Jenkins is coded in Java, you need to have the Java Development Kit (JDK) installed beforehand.

Step 1.1: Install Java (Prerequisite)

Java is needed by Jenkins. We'll install OpenJDK 17, a stable release.

Code (Linux/Ubuntu):

sudo apt update

sudo apt install openjdk-17-jdk -y

java -version # Verify installation

Step 1.2: Install Jenkins

The simplest method of installing Jenkins on a Debian-based system (such as Ubuntu) is from the official package repository.

Add the Jenkins repository key:

*curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee *

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

Add the repository to the system's sources list:

*echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] *

*https://pkg.jenkins.io/debian-stable binary/ | sudo tee *

/etc/apt/sources.list.d/jenkins.list > /dev/null

Update and install Jenkins:

sudo apt update

sudo apt install jenkins -y

Start and enable the Jenkins service:

sudo systemctl start jenkins

sudo systemctl enable jenkins

sudo systemctl status jenkins # Check that it's active and running

Step 1.3: Unlock Jenkins (Web Interface Setup)

1. **Access Jenkins:** Open your browser and go to http://<Your-Server-IP-or-Hostname>:8080. The default Jenkins port number is 8080.
2. **Get the Initial Admin Password:** The setup page will prompt for an initial administrative password. Get it from the server:
3. **Finish the Setup:**
    1. Paste the output from the above command into the Administrator password field on your browser.
    2. Click Continue.
4. **Install Plugins:** On the following screen, select "Install suggested plugins." Jenkins will install the necessary plugins for general use, such as Git, Pipeline, and a variety of build tools, automatically.
5. **Create Admin User:** After the installation of plugins, create your initial admin user (username, password, name, and email). This will be your login of record.
6. **Instance Configuration:** The last step validates the Jenkins URL. Use the default or modify it if you need to, and then Save and Finish. You will be redirected to the Jenkins Dashboard.

Step 2. Setting Up the Development Environment (Git Integration)

In order to construct code, Jenkins must communicate with a Version Control System, most commonly Git (for example, GitHub, GitLab).

Step 2.1: Install Git on the Jenkins Server

Make sure that Git is installed on your server so that Jenkins can check out (clone) source code.

Code (Linux/Ubuntu):

sudo apt install git -y

Step 2.2: Set Global Tool Settings

Jenkins must be aware of where it can locate its build tools

1. From the Jenkins Dashboard, go to **Manage Jenkins → Global Tool Configuration**.
2. Scroll down to the **Git** section.
3. Click **Add Git**.
4. Insert a name (e.g., Default Git). The **Path to Git** executable is typically /usr/bin/git.
5. Click **Save**.

Step 3. Your First Pipeline

A Jenkins Pipeline is an automated, customizable statement of your entire CI/CD process. We'll utilize a Declarative Pipeline, declared in a text file referred to as a Jenkinsfile.

Step 3.1: The Jenkinsfile (Pipeline as Code)

In the root directory of your source code repository, you require a file named Jenkinsfile. This example mimics a simple Node.js application build process.

Jenkinsfile Code:

```
pipeline {

    agent any // This tells Jenkins to run the pipeline on any available agent

    stages {

        stage('Checkout Code') { // Stage 1: Get the code
```

```
    steps {
        // Assuming you are using Git
        echo 'Checking out code from repository…'
        // You would typically use 'checkout scm' here for real projects
    }
}
stage('Install Dependencies') { // Stage 2: Install required packages
    steps {
        echo 'Installing Node.js dependencies…'
        sh 'npm install' // Runs the npm install command
    }
}
stage('Run Tests') { // Stage 3: Test the code
    steps {
        echo 'Running tests…'
        sh 'npm test' // Runs unit tests defined in your package.json
    }
```

```
    }

    stage('Build Artifact') { // Stage 4: Create the final package

        steps {

            echo 'Building the application…'

            sh 'npm run build' // Creates the deployable code package

        }

    }

  }

}
```

Step 3.2: Create the Jenkins Job

1. From the Jenkins Dashboard, click **"New Item" (or "Create a job")**.
2. Enter an **Item Name** (e.g., my-first-ci-pipeline).
3. Choose **"Pipeline"** as the type.
4. Click **OK**.

Step 3.3: Set up the Pipeline Job

1. In the setup screen, scroll down to the **Pipeline** area.
2. Change Definition from **"Pipeline script"** to **"Pipeline script from SCM"** (Source Code Management).
3. Configure the SCM section:
    1. **SCM:** Choose **Git**.

2. **Repository URL:** Enter the URL of your Git repository (e.g., https://github.com/your-user/your-repo.git).
3. **Credentials:** If the repository is public, choose (none). If it is private, click **Add** → **Jenkins** and provide your **Git username/password** or **SSH key**.
4. **Branch Specifier (optional):** Enter **/main or */master* to specify which branch Jenkins should watch.
5. **Script Path:** Make sure this is left at ***Jenkinsfile*** (the default filename).

4. Click **Save**.

Step 4. Running and Monitoring the Pipeline

Step 4.1: Trigger the First Build

- Go to your **new pipeline job** (my-first-ci-pipeline).
- In the left panel, click on **"Build Now"**.

Jenkins will proceed to clone the repository and begin running the Jenkinsfile steps.

Step 4.2: Monitor the Build

- The build will be visible in the **Build History** panel on the left. Click on the build number (i.e., #1) to see the details.
- In the left sidebar, click on **"Console Output"**.

The Console Output is the live log of the pipeline. You will observe the output for every stage listed in your Jenkinsfile:

[Pipeline] { (Checkout Code) [Pipeline] echo Checking out code from repository… … [Pipeline] { (Install Dependencies) [Pipeline] echo Installing Node.js dependencies… [Pipeline] sh + npm install …

Step 4.3: Automate the Trigger (Polling)

To turn this Continuous Integration, you need to set up Jenkins to automatically build whenever a developer checks in code.

- Return to the job configuration page (Configure).
- Under the Build Triggers field, select “Poll SCM”.
- In the Schedule field, enter a syntax similar to cron to define how frequently Jenkins should poll the repository for changes (e.g., every 5 minutes):

Code (Poll SCM Schedule):

*H/5 * * * **

(This means: Poll the SCM repository every 5 minutes for new commits.)

- Click Save. Now, each time a fresh commit is pushed into the target branch, Jenkins will automatically initiate a fresh build, and your very first basic CI cycle is complete!

Essential Jenkins Concepts)

To fully utilize Jenkins, you need to learn several key concepts:

A. The Jenkins Pipeline (Jenkinsfile)

Rather than copying the script into the Jenkins UI, best practice is to keep the pipeline definition in a file called Jenkinsfile in the root of your Git repository.

B. Agents (Master-Agent Architecture)

- **Jenkins Controller (Master):** The primary server for scheduling jobs, handling configurations, and keeping results.
- **Jenkins Agent (Slave/Node):** External computers (physical or virtual) that run the actual build jobs.

Why Agents? To relieve build workloads, execute builds on alternate operating systems (e.g., Linux and Windows), and keep the Controller stable. Our Docker environment utilized the built-in agent (agent any).

C. Plugins (Extending Functionality)

Plugins are the fuel of Jenkins. They enable integration with nearly any tool.

Our popular guide that contains **Jenkins challenges and solutions** will help you know more about CI/CD with Jenkins.

Real Time Examples for Jenkins Tutorial for Learners

The following are some real-time examples for Jenkins tutorial for learners illustrating the key Jenkins concepts:

1. Automated Unit Testing (Continuous Integration)

Suppose a big team is developing an e-commerce website. Developers commit code changes to GitHub day in, day out.

Jenkins Concept: Build Trigger and Pipeline Stage.

- **Real-Time Action:** Jenkins has a Poll SCM trigger configured to poll the Git repository every minute. If a developer commits code, Jenkins picks up the change immediately, triggering a new build.
- **Pipeline Flow:** The pipeline performs the “Test” phase, executing several hundred unit tests (e.g., with JUnit or Jest). Should any one test fail, Jenkins flags the build as Failed and sends immediate notification (e.g., to Slack) to the team. This catches bugs and fixes them in minutes, ensuring Continuous Integration.

2. Deploying to Various Environments (Pipeline Parameters)

A software firm wishes to roll out a new feature, but to the Staging environment initially for inspection.

Jenkins Concept: Parameterized Build and Declarative Stage Selection.

- **Real-Time Action:** The Jenkins job is set up to take a parameter named TARGET_ENV (with values such as “Staging” and “Production”).
- **Pipeline Flow:** The team member clicks “Build with Parameters” and chooses Staging. The pipeline employs an environment variable to run only the Deploy_Staging stage, bypassing the Deploy_Production stage. Safe, manual control ensures review of new code prior to impacting live users.

3. Managing Heavy Loads (Master-Agent Architecture)

There is a popular game company that has Java, Python, and C++ projects, each needing certain operating systems and hardware for build.

Jenkins Concept: Master-Agent (Distributed Build) Architecture.

- **Real-Time Action:** The primary Jenkins Controller (Master) is dedicated to scheduling and configuration. The job is outsourced to dedicated Agents (Slave nodes).
- **Pipeline Flow:** As the Java build initiates, the Master allocates the task to the Linux Agent, which is installed with Java and Maven. At the same time, another project written in C++ can be allocated to a Windows Agent. This loads the work differently, does not overload the Master, and enables the builds to get executed in the targeted environments where they are needed, thus enabling efficient and concurrent processing.

Click here for more **Jenkins Project Ideas for Learners**.

FAQs About Jenkins Tutorial for Beginners

1. What is Jenkins Used For?

Jenkins is utilized for automating the software delivery pipeline, simply building, testing, and deploying applications in order to obtain Continuous Integration/Continuous Delivery (CI/CD).

2. Is Jenkins CI/CD?

Yes, Jenkins is the most widely used, open-source automation tool applied to apply the process of Continuous Integration (CI) and Continuous Delivery/Deployment (CD).

3. How can I learn Jenkins?

Begin with installation on a local system (or Docker), learn how to create a simple Pipeline based on a Jenkinsfile, and exercise putting it together with a Git repository and a build tool.

4. What are the 7 C's of DevOps?

The 7 C's are the stages of the DevOps life cycle: Continuous Code/Development, Continuous Integration, Continuous Testing, Continuous Feedback, Continuous Monitoring, Continuous Deployment, and Continuous Operations.

5. What are the two types of DevOps?

DevOps is not divided into two "types," but two core practices are typically emphasized: Continuous Integration (CI) and Continuous Delivery/Deployment (CD).

6. Is Jenkins ETL tool?

No. Jenkins is a CI/CD tool utilized for automating. Although it can be utilized to schedule and invoke Extract, Transform, Load (ETL) jobs, it is not an ETL processing tool.

7. Which CI tools are used in Jenkins?

Jenkins supports a number of tools through plugins such as Git (VCS), Maven/Gradle (Build), JUnit/Selenium (Testing), and Docker/Kubernetes (Deployment).

8. Is Jenkins a tool or language?

Jenkins is a tool (an open-source automation server). Its Pipeline configuration is expressed through a Domain Specific Language (DSL) based on the Groovy programming language.

9. Can I use Jenkins for Python?

Yes. Jenkins is language-agnostic and can be utilized for Python projects to automate tasks such as running pytest, code quality checking using Pylint, and packaging apps.

10. What are the 4 stages of CI CD?

The major steps of a typical CI/CD pipeline are Source (Commit), Build, Test, and Deploy (which encompasses Continuous Delivery or Deployment).

11. What code does Jenkins use?

The Jenkins server is mostly in Java. Its automation scripts are usually specified by using Groovy DSL in a Jenkinsfile for building pipelines.

12. Is Jenkins free or paid?

Jenkins is open-source and free. Although the basic tool is free, organizations can pay for the underlying infrastructure (cloud, servers) and voluntary enterprise support.

Conclusion

You've completed the Jenkins tutorial for beginners with ease, acing installation, foundational concepts, and creating your first automated pipeline. You now see the power of Jenkins that expedites manual drudgery and fosters efficiency with CI/CD. This ground knowledge is your passport to a successful DevOps career. Remember, practice every day is what makes these steps second nature to you. Ready to take this skill further? Sign up for our complete **Jenkins course in Chennai** to become expert in advanced pipelines and integrations.