DevOps Shack

Jenkins Part-2: Jenkins Installation and Configuration Guide

[Click Here To Enrol To Batch-6 | DevOps & Cloud DevOps]

Prerequisites

Before installing Jenkins, ensure that Java is installed on your system. Jenkins requires Java to run. The following command installs OpenJDK 17:

sudo apt install openjdk-17-jre-headless -y

This command installs the necessary Java Runtime Environment to ensure Jenkins functions correctly.

Installing Jenkins

Step 1: Create a Script for Installation

Create a shell script to automate the installation process. This script will download the Jenkins key, add the Jenkins repository, and install Jenkins.

1. Create a script file named install-jenkins.sh:

vi install-jenkins.sh

2. Copy the following content into the script:

#!/bin/bash

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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

sudo apt-get update

sudo apt-get install jenkins

Step 2: Set Permissions and Run the Script

1. Change the script's permissions to make it executable:

sudo chmod +x install-jenkins.sh

2. Execute the script to install Jenkins:

./install-jenkins.sh

This script automates the installation, making it easier to deploy Jenkins on your server.

Accessing Jenkins

Once Jenkins is installed, it can be accessed through a web browser.

1. Use your server's public IP address and port 8080 to access Jenkins:

http://your_public_ip:8080

2. Retrieve the initial admin password needed to log in:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

This password allows you to log in and complete the Jenkins setup.

Installing Plugins & Configuring JDK

Step 1: Installing Plugins

1. Navigate to Manage Jenkins -> Manage Plugins.
2. Under the Available tab, search for "Eclipse Temurin Installer."
3. Install the plugin.

Step 2: Configuring JDK

1. Go to Manage Jenkins -> Global Tool Configuration -> JDK installations.
2. Add a JDK installation:
    - Provide a name for the JDK.
    - Select the "Install automatically" option.
    - Choose "Install from adoptium.net" and configure JDK 17.

This configuration ensures that your Jenkins instance can use JDK 17 for various tasks.

Jenkins Job Types

In Jenkins, different types of jobs allow you to define and automate various stages of the software development lifecycle. Here are some common types:

1. Freestyle Project

- A general-purpose job type where you can define a series of build steps, such as running shell commands, executing scripts, and performing other tasks.

- Suitable for simple tasks and projects that don't require complex workflows.

2. Pipeline Project

- Utilizes the Jenkins Pipeline plugin to define and manage build, test, and deployment workflows as code using a Jenkinsfile.
- Supports both declarative and scripted pipeline syntax, allowing for highly customizable and complex pipelines.

3. Multibranch Pipeline

- A pipeline job that automatically creates sub-jobs for each branch in your version control repository.
- Suitable for managing and automating CI/CD pipelines for multiple branches.

Creating and Configuring a Freestyle Jenkins Job

Creating and configuring a Jenkins Freestyle job is straightforward. Follow these steps:

Step 1: Access Jenkins

1. Log in to the Jenkins web interface.

Step 2: Create a New Freestyle Job

1. Click on "New Item" in the Jenkins dashboard.
2. Enter a name for your job (e.g., "MyFreestyleJob").
3. Select "Freestyle project" and click "OK."

Step 3: Configure General Settings

1. In the job configuration page, start with the "General" section:
    - Enter a brief description for the job (optional).
    - Choose the "Discard old builds" option to manage build retention.

Step 4: Configure Source Code Management (Optional)

1. If your project requires source code management, select the appropriate SCM system (Git, Subversion, etc.) and provide the necessary repository information.

Step 5: Configure Build Steps (Continued)

1. **Execute Shell**: Run shell commands or scripts.
2. **Execute Windows Batch Command**: Run batch commands on Windows.
3. **Invoke Ant**: Run Apache Ant tasks.
4. **Invoke Gradle**: Run Gradle build tasks.

5. **Invoke Maven**: Run Maven build tasks.

These steps allow you to define the build process for your project. For instance, if you're working on a Java project, you might use "Invoke Maven" to compile and build your code.

Step 6: Configure Post-Build Actions

After defining the build steps, you can specify post-build actions that will be executed after the build completes. Common post-build actions include:

1. **Archive the Artifacts**: This option allows you to archive build artifacts (e.g., JAR files, reports) for future reference. It's useful for keeping a record of what was built.
2. **Publish JUnit Test Result Report**: If your project includes unit tests, this option allows you to publish test results and view them within Jenkins.
3. **Send Build Artifacts Over SSH**: This action lets you transfer build artifacts to a remote server via SSH. It's useful for deployment or further processing.

Step 7: Configure Build Triggers (Optional)

Build triggers allow you to define conditions under which the build should start automatically. Some common triggers include:

1. **Build Periodically**: Schedule the build to run at specific intervals (e.g., nightly builds).
2. **Poll SCM**: Poll the source code repository for changes and trigger builds when changes are detected.

These triggers are helpful for automating the build process and ensuring that your project is always up-to-date.

Step 8: Save Configuration

Once you've configured all the settings for your Freestyle job, click "Save" to save your changes. Your job is now ready to be run.

Step 9: Run the Job

After saving the job configuration, you can manually trigger the job by clicking "Build Now" on the Jenkins dashboard. This will start the build process according to the steps you've defined.

Step 10: View Build Results

Once the build is complete, you can view the build results, including the console output, any artifacts produced, and test results if applicable. This information is accessible from the Jenkins job page under the "Build History" section.

Jenkins Pipeline Job

Here's a more detailed explanation of a Jenkins pipeline job using the Declarative Pipeline syntax:

Example: Simple Maven Pipeline

This pipeline example includes stages for checking out code, compiling, testing, and packaging a Maven project:

```
pipeline {
    agent any
    tools {
        jdk 'jdk_name'
        maven 'maven_name'
    }
    stages {
        stage('Checkout') {
            steps {
                // Check out the source code from the repository
                git branch: 'main', url: 'https://github.com/your/repo.git'
            }
        }
        stage('Compile') {
            steps {
                // Compile the code using Maven
                sh 'mvn compile'
            }
        }
```

```
        stage('Test') {
            steps {
                // Run tests using Maven
                sh 'mvn test'
            }
        }
        stage('Package') {
            steps {
                // Package the application using Maven
                sh 'mvn package'
            }
        }
    }
    post {
        always {
            // Archive the build artifacts
            archiveArtifacts artifacts: '**/target/*.jar', allowEmptyArchive: true
        }
    }
}
```

Explanation:

1. **Agent**: The agent any directive tells Jenkins to run the pipeline on any available agent.
2. **Tools**: The tools block specifies the tools required by the pipeline, such as JDK and Maven. Replace 'jdk_name' and 'maven_name' with the actual tool names configured in Jenkins.
3. **Stages**: The stages block defines the various stages of the pipeline. Each stage represents a step in the CI/CD process:
    - **Checkout**: Checks out the source code from the Git repository.
    - **Compile**: Compiles the source code using Maven.
    - **Test**: Runs tests using Maven.
    - **Package**: Packages the application (e.g., creating a JAR file).
4. **Post**: The post block contains actions that should always be executed after the pipeline runs. In this case, it archives the build artifacts, making them available for download from Jenkins.

Customization:

- Replace https://github.com/your/repo.git with the actual URL of your Git repository.
- Ensure that Maven (mvn) is installed on the Jenkins agent where the pipeline will run.

This pipeline provides a clear and structured way to automate the build, test, and package processes for your Maven project.

Setting Up a Trigger for a Pipeline Using a Generic WebHook Trigger

Step 1: Install Generic Webhook Trigger Plugin

1. Go to Jenkins dashboard.
2. Navigate to Manage Jenkins -> Manage Plugins.
3. In the Available tab, search for "Generic Webhook Trigger."
4. Install the plugin and restart Jenkins if necessary.

Step 2: Configure Jenkins Job

1. Create or open the Jenkins job you want to trigger.
2. In the job configuration, scroll down to the Build Triggers section.
3. Check the checkbox for Generic Webhook Trigger.

Step 3: Configure Post Parameters

1. In the Post Parameters section, add the following:
    - Variable: ref
    - Expression (JSON): $.ref

Step 4: Write a string as Token

1. Write any keyword string as token that will be used in the webhook URL.

Step 5: Configure Optional Filter

1. In the Optional Filter section, configure the following:
    - Expression: refs/heads/branch_name (Replace branch_name with the name of your branch)
    - Text: $ref

Step 6: Configure GitHub Webhook

1. Go to your GitHub repository settings.
2. Navigate to Webhooks.
3. Click Add webhook.

4. In the Payload URL field, enter the following URL:

Jenkins_URL/generic-webhook-trigger/invoke?token=github_token

Replace Jenkins_URL with the URL of your Jenkins instance and github_token with the token generated earlier. Example:

http://65.0.31.109:8080/generic-webhook-trigger/invoke?token=github_token

5. Set the Content type to application/json.
6. Select the events you want to trigger the webhook for (e.g., Pushes).

Step 7: Save Changes

1. Save your Jenkins job configuration.

Now, your Jenkins job is configured to trigger via a generic webhook whenever a push event occurs on the specified branch in your GitHub repository.

Setting Up a Trigger for a Multibranch Pipeline Using Multibranch Scan Webhook Trigger Plugin

Step 1: Install Multibranch Scan Webhook Trigger Plugin

1. Navigate to Jenkins dashboard.
2. Go to Manage Jenkins -> Manage Plugins.
3. In the Available tab, search for "Multibranch Scan Webhook Trigger."
4. Install the plugin and restart Jenkins if required.

Step 2: Configure Multibranch Pipeline Job

1. Create or open the multibranch pipeline job you want to trigger.
2. In the job configuration, navigate to Scan Multibranch Pipeline Triggers.
3. Select Scan by webhook.
4. Enter any trigger token in the Trigger token field.

Step 3: Configure GitHub Webhook

1. Go to your GitHub repository settings.
2. Navigate to Webhooks.
3. Click Add webhook.
4. In the Payload URL field, enter the following URL:

Jenkins_URL/multibranch-webhook-trigger/invoke?token=token_value

Replace Jenkins_URL with the URL of your Jenkins instance and token_value with the trigger token you configured in the Jenkins job. Example:

http://65.0.31.109:8080/multibranch-webhook-trigger/invoke?token=Devopsshack

5. Set the Content type to application/json.
6. Select the events you want to trigger the webhook for (e.g., Pushes).

Step 4: Save Changes

1. Save your Jenkins job configuration.

Now, your multibranch pipeline job is configured to trigger via webhook whenever a push event occurs on your GitHub repository.