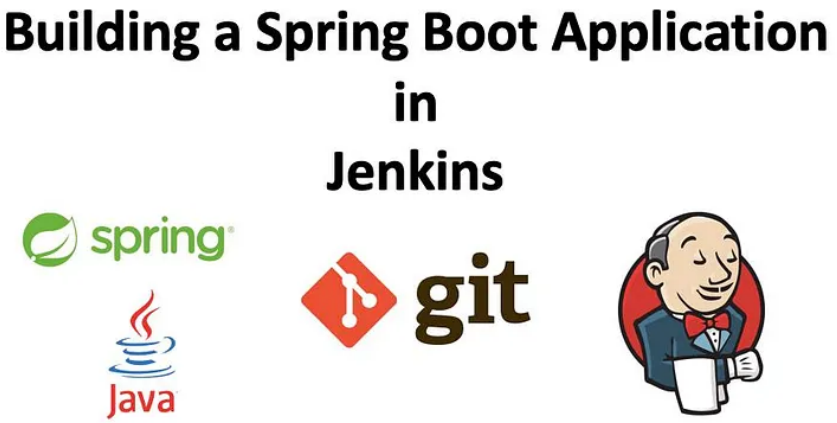
Streamline Your Spring Boot Deployments with Jenkins: A Step-by-Step Tutorial for 2024

fullstackjavadev.in · Follow
4 min read · May 26, 2024



Introduction

In the realm of modern software development, Continuous Integration and Continuous Deployment (CI/CD) have become essential practices. Jenkins, a widely-used open-source automation server, plays a pivotal role in enabling these practices. Integrating Jenkins with Spring Boot allows for automated building, testing, and deploying of applications, significantly improving development efficiency and reliability. This detailed tutorial will guide you through setting up Jenkins and using it to streamline your Spring Boot deployments in 2024.

Prerequisites

Before we begin, ensure you have the following:

- A Spring Boot application
- Jenkins installed on your machine or server
- Basic knowledge of Git
- Java Development Kit (JDK) installed

Step 1: Setting Up Jenkins

Installing Jenkins

On Windows:

- Download the Jenkins installer from the official website.
- Run the installer and follow the setup wizard instructions.

On Linux:

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add
```

Using Docker:

```
docker run -p 8080:8080 -p 50000:50000 jenkins/jenkins:lts
```

Starting Jenkins

- **Windows:** Jenkins starts automatically after installation.
- **Linux:** Use `sudo systemctl start jenkins`.

Accessing Jenkins

- Open your browser and navigate to `http://localhost:8080`.
- Follow the on-screen instructions to unlock Jenkins using the initial admin password found in `/var/lib/jenkins/secrets/initialAdminPassword`.

Step 2: Configuring Jenkins for Your Spring Boot Project

Creating a New Job

Create a New Item:

- Go to the Jenkins dashboard.
- Click on “New Item”.
- Enter a name for your job and select “Freestyle project”.
- Click “OK”.

Configuring Source Code Management

Git Integration:

- In the job configuration page, under “Source Code Management”, select “Git”.
- Enter the repository URL and credentials if necessary.

Configuring Build Triggers

Automate Builds:

- Under “Build Triggers”, select “Poll SCM” and set a schedule (e.g., `H/5 * * * *` for polling every 5 minutes) or choose "GitHub hook trigger for GITScm polling".

Adding Build Steps

Gradle Build:

- Under “Build”, click “Add build step” and select “Invoke Gradle script”.
- Specify `clean build` as the Tasks.

Maven Build (if using Maven):

- Under “Build”, click “Add build step” and select “Invoke top-level Maven targets”.
- Specify `clean install`.

Step 3: Creating a Jenkins Pipeline with Jenkinsfile

Instead of configuring everything through the UI, you can define your pipeline in a `Jenkinsfile`. This approach offers better version control and easier maintenance.

Creating a Jenkinsfile

Add a `Jenkinsfile` to the root of your Spring Boot project:

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo/spring-boot-app.git'
            }
        }
        stage('Build') {
            steps {
                sh './gradlew clean build' // Use 'mvn clean install' if using Maven
            }
        }
        stage('Test') {
            steps {
                sh './gradlew test' // Use 'mvn test' if using Maven
            }
        }
        stage('Package') {
            steps {
                sh './gradlew bootJar' // Use 'mvn package' if using Maven
            }
        }
        stage('Deploy') {
            steps {
                // Add your deployment steps here, e.g., using SCP, SSH, Docker, etc.
                sh 'scp build/libs/*.jar user@server:/path/to/deploy'
            }
        }
    }

    post {
        success {
```

```
            echo 'Build and Deploy succeeded!'
        }
        failure {
            echo 'Build or Deploy failed!'
        }
    }
}
```

Step 4: Running the Pipeline

Commit and Push:

- Commit your `Jenkinsfile` and push it to your repository.

Create a Pipeline Job:

- Go to Jenkins dashboard and create a new pipeline job.
- In the job configuration, point to your repository where the `Jenkinsfile` is located.
- Jenkins will automatically detect and execute the pipeline defined in the `Jenkinsfile`.

Step 5: Deploying Your Spring Boot Application

In the “Deploy” stage of your Jenkinsfile, add the necessary steps to deploy your Spring Boot application. This might involve:

- Copying the JAR file to a server using SCP or Rsync.
- Running Docker commands to build and deploy Docker containers.
- Using cloud provider-specific deployment commands.

Example deployment using SCP:

```
stage('Deploy') {
    steps {
        sh 'scp build/libs/*.jar user@server:/path/to/deploy'
        sh 'ssh user@server "nohup java -jar /path/to/deploy/your-app.jar >
    }
}
```

Conclusion

Integrating Jenkins with your Spring Boot application can greatly enhance your development workflow by automating builds, tests, and deployments. This step-by-step guide provides a comprehensive approach to setting up and using Jenkins to streamline your Spring Boot deployments in 2024. By following these steps, you can ensure a more efficient and reliable CI/CD pipeline, allowing you to focus more on coding and less on manual deployment processes.

Additional Resources

- Jenkins Documentation
- Spring Boot Documentation
- Gradle Documentation
- Maven Documentation

These resources can provide further reading and deeper insights into Jenkins and Spring Boot, helping you to extend and customize your CI/CD pipeline.

Spring | Spring Boot | DevOps | Jenkins | Kubernetes

Follow

Written by fullstackjavadev.in

56 Followers

Full stack java development