

# SPRING BOOT CI/CD WITH JENKINS



- Rohan Thapa  
[thaparohan2019@gmail.com](mailto:thaparohan2019@gmail.com)

# What is Jenkins?

Jenkins is an **open-source** automation server written in **Java** that helps **automate** parts of software development related to **building**, **testing**, and **deploying**, facilitating continuous integration and continuous delivery (**CI/CD**).

It's one of the most popular **CI/CD** tools used by developers to create a seamless pipeline for code delivery.

# History of Jenkins

- Originally developed as **Hudson** at **Sun Microsystems** in 2004.
- Jenkins emerged after a dispute between Oracle and the Hudson open-source community in 2011.
- Now, Jenkins is maintained by a **large community** and has become the **go-to tool for CI/CD**.

# Key Concepts of Jenkins

## Jenkins Pipeline

A **pipeline** in Jenkins is a **set of tasks** that are automated to build, test, and deploy your software. It's a workflow that defines the steps to take your code from **version control** to **production**.

## Jobs

A job in Jenkins represents a **task or a step** in a **pipeline**. Jenkins can have multiple jobs for different stages in the pipeline (e.g., build, test, deploy).

# Key Concepts of Jenkins

## Builds

A **build** is the **result of executing a job**. Builds can represent compiled code or simply the outcome of running automated tests.

## Nodes and Executors

- **Nodes:** Jenkins uses master-slave architecture to manage different machines that execute jobs.
- **Master Node:** Handles scheduling and orchestration of jobs.
- **Slave/Agent Nodes:** Execute jobs, and they can be on different machines/servers.
- **Executors:** Slots on a node that execute jobs. You can configure how many jobs an executor can run simultaneously.

# Key Concepts of Jenkins

## Plugins

Jenkins has a **plugin-based architecture** that allows it to integrate with almost any development tool (SCM systems, build tools, deployment tools, etc.).

Popular plugins include:

- **Git Plugin:** For source code integration.
- **Maven Plugin:** To build Maven projects.
- **Docker Plugin:** To interact with Docker.
- **Slack Plugin:** For sending notifications.

# Setting Up Jenkins

## Prerequisites

- **Java:** Jenkins runs on Java, so you need to have Java installed on your machine (**JDK 17 is recommended**).
- **Web Browser:** Jenkins is accessed via a **web interface**.
- **Server:** You can run Jenkins on your **local machine**, a **server**, or **in the cloud**.

# Setting Up Jenkins

## Installation

You can install Jenkins on **different platforms**. Let's go through a few common options.

On Windows:

- Download Jenkins from the official website and run the installer.

On Linux(Ubuntu/Debian):

```
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
sudo systemctl start jenkins
```

# Initial Setup and Unlocking Jenkins

- After installation, Jenkins runs on <http://localhost:8080>.
- You will be prompted to enter an initial admin password, which can be found in the log file at:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

- Complete the setup wizard and install recommended plugins.

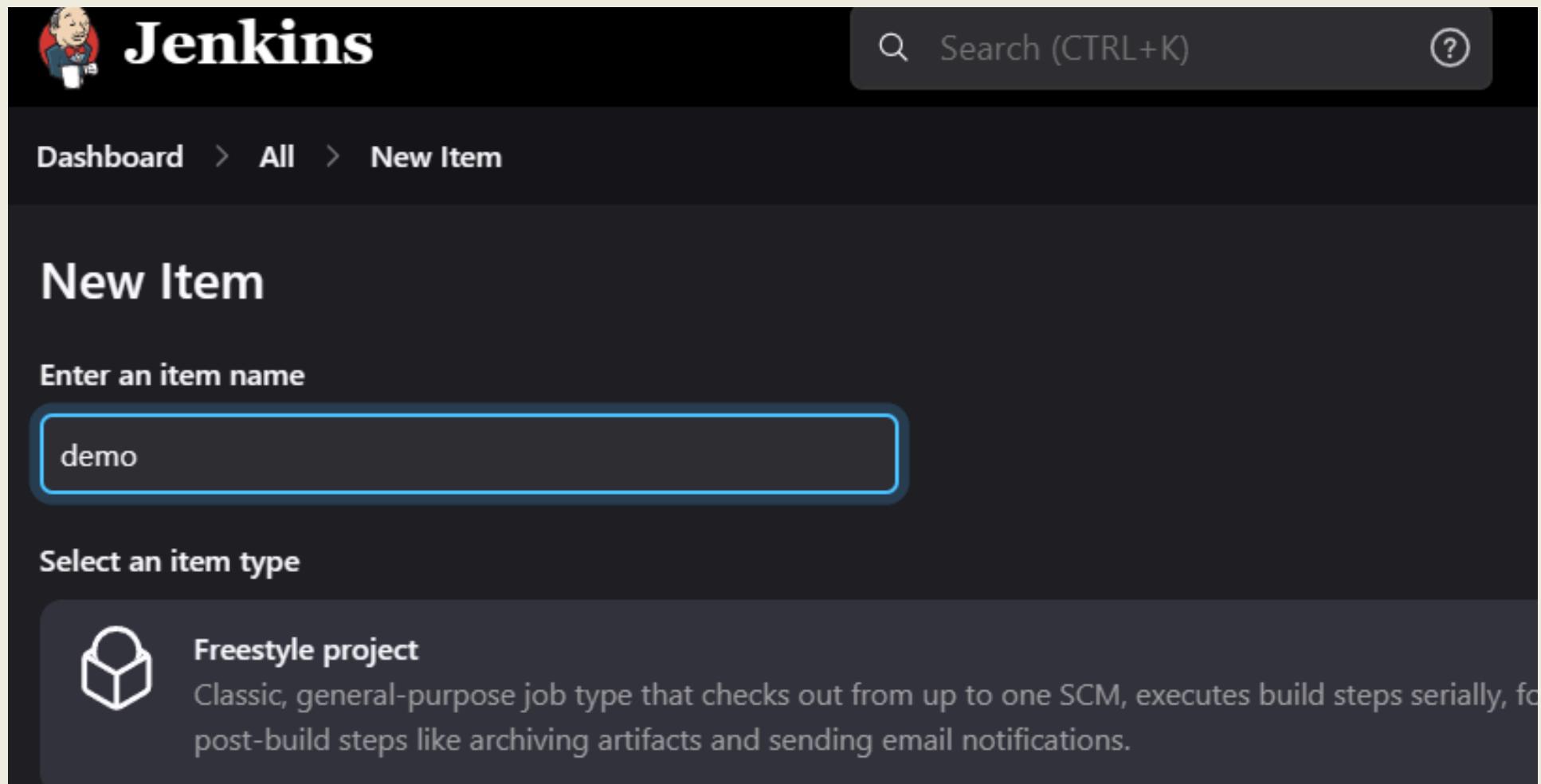
# Jenkins Basics

## Freestyle Projects

A **Freestyle Project** is the simplest type of Jenkins job. It allows you to define a series of build steps that Jenkins will execute.

## Steps to create a Freestyle Project:

1. Click on "New Item" and select "Freestyle Project."



The screenshot shows the Jenkins interface for creating a new item. At the top, there's a navigation bar with a logo, the word 'Jenkins', a search bar labeled 'Search (CTRL+K)', and a help icon. Below the navigation, the path 'Dashboard > All > New Item' is visible. The main section is titled 'New Item' and has a sub-instruction 'Enter an item name'. A text input field contains the value 'demo', which is highlighted with a blue border. Below this, another instruction 'Select an item type' is shown, followed by a card for 'Freestyle project'. The card includes a small icon of a cube, a brief description, and a longer explanatory text at the bottom.

Dashboard > All > New Item

New Item

Enter an item name

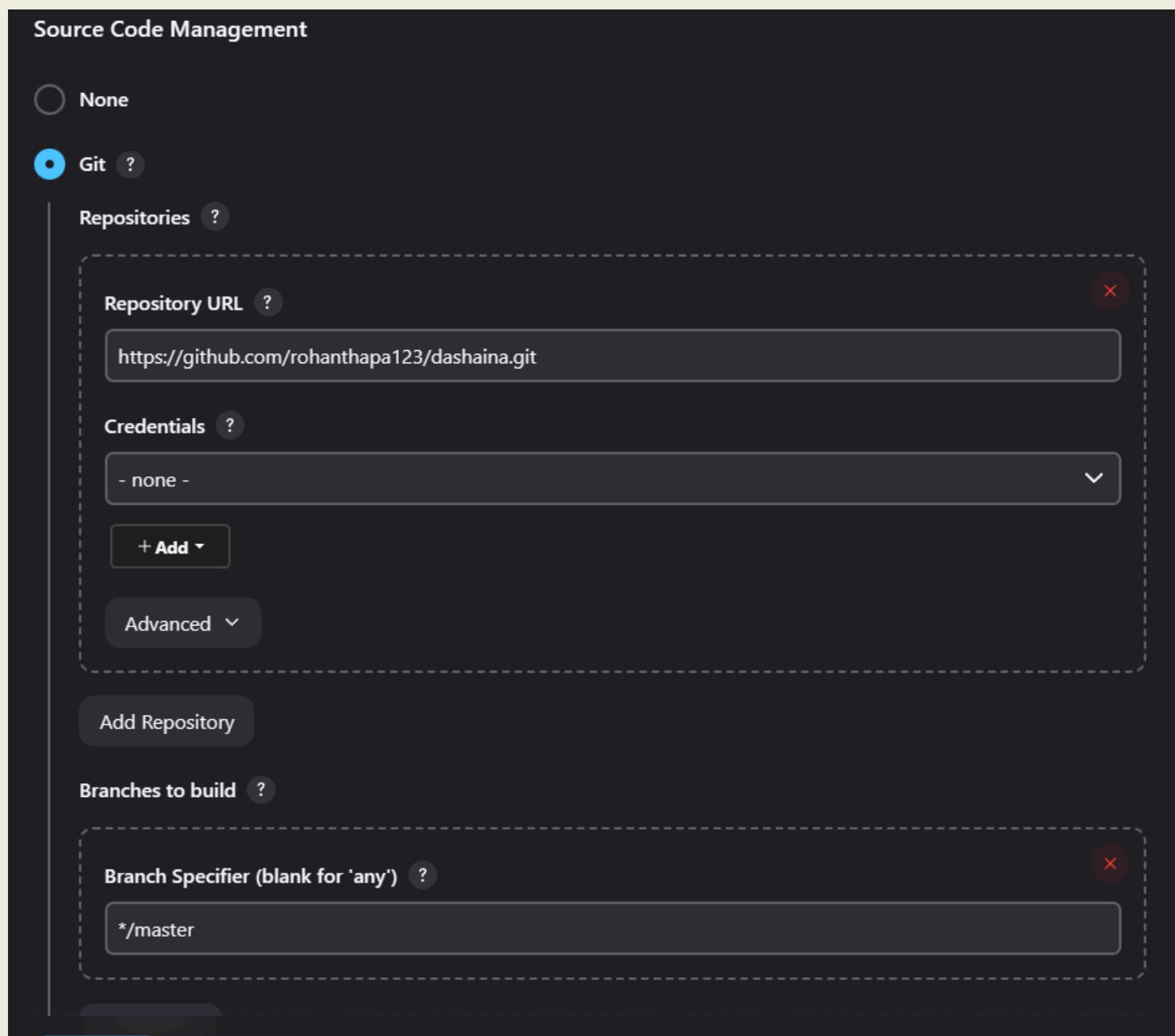
demo

Select an item type

 Freestyle project  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, for post-build steps like archiving artifacts and sending email notifications.

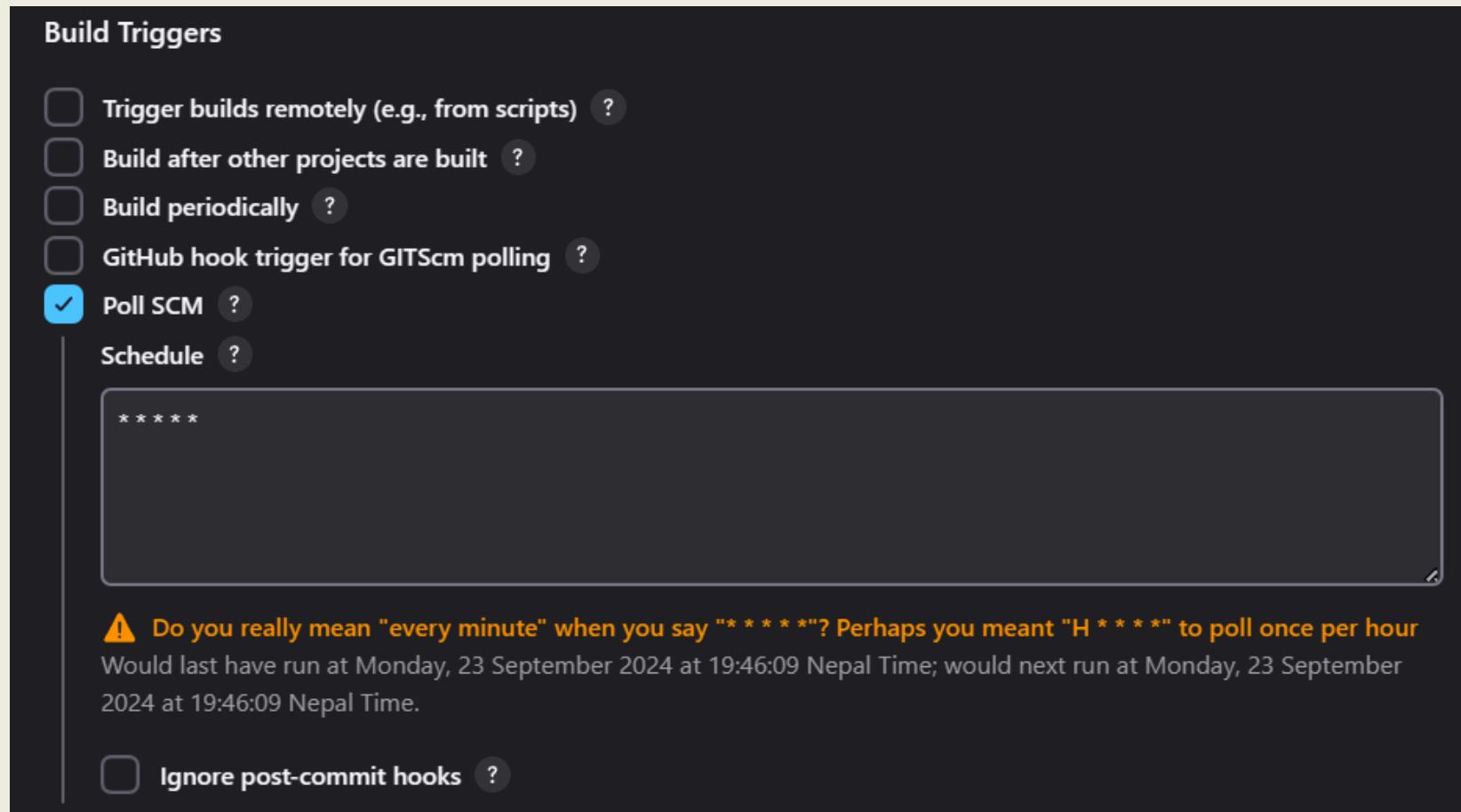
# Jenkins Basics

## 2. Define your **Source Code Management** (SCM) system (e.g., **Git**).



# Jenkins Basics

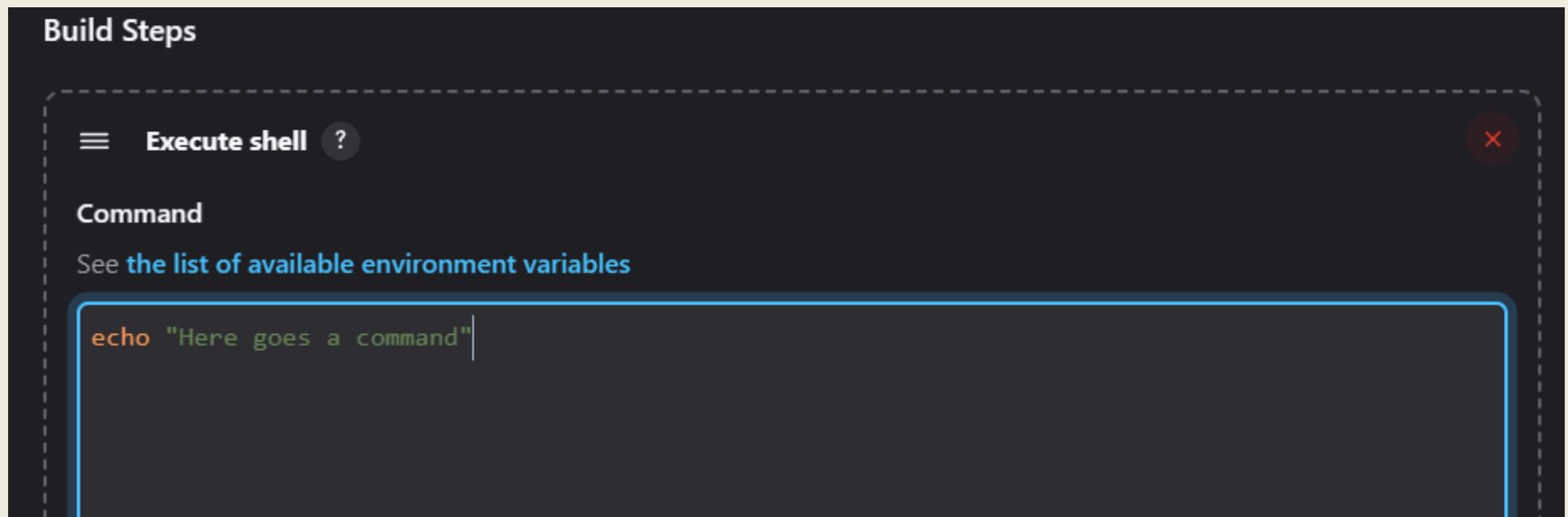
3. Define Build Triggers (when you want the job to run—manually, periodically, on a SCM change, etc.).



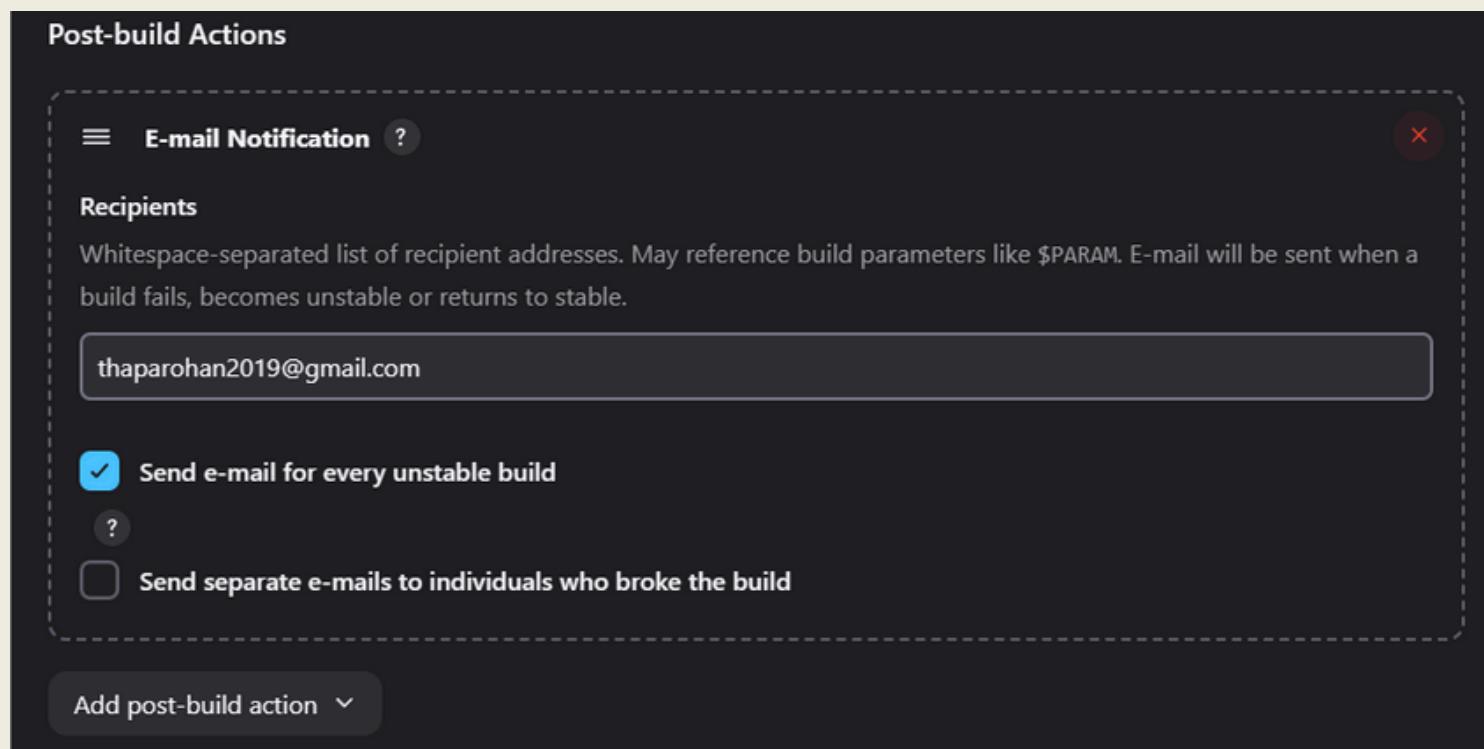
- Add Build Steps (e.g., **invoking a shell command, running a Maven build**).
- Define Post-Build Actions (e.g., **sending notifications, archiving artifacts**).

# Jenkins Basics

## 4. Add Build Steps (e.g., invoking a shell command, running a Maven build).



## 5. Define Post-Build Actions (e.g., sending notifications, archiving artifacts).



# Jenkins Basics

## Pipelines

**Jenkins Pipelines** are scripted workflows that give more control and flexibility than Freestyle Projects.

You define pipelines using **Jenkinsfile**, a text file that contains the definition of the pipeline.

## Declarative vs Scripted Pipelines

- **Declarative Pipelines:** A simpler, structured way to define a pipeline using pipeline syntax.
- **Scripted Pipelines:** Uses more advanced Groovy syntax, offering more flexibility and control.

# Jenkins Basics

## Pipelines

**Jenkins Pipelines** are scripted workflows that give more control and flexibility than Freestyle Projects.

You define pipelines using **Jenkinsfile**, a text file that contains the definition of the pipeline.

## Declarative vs Scripted Pipelines

- **Declarative Pipelines:** A simpler, structured way to define a pipeline using pipeline syntax.
- **Scripted Pipelines:** Uses more advanced Groovy syntax, offering more flexibility and control.

# Jenkins Basics

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
                sh 'mvn clean package'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
                sh 'mvn test'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying...'
                sh 'scp target/*.jar user@server:/path/to/deploy'
            }
        }
    }
}
```

# **CI/CD with Jenkins**

**Jenkins** enables **Continuous Integration** (CI) by running automated **builds** and **tests** when code is pushed to **version control**. **Continuous Deployment** (CD) is achieved by **automatically deploying** the application to **staging/production** if the build and tests are successful.

# Basic CI/CD Pipeline

Here's a simple **CI/CD** workflow using **Jenkins**:

## 1. Build:

- Jenkins **pulls code from GitHub**.
- Runs the build process (**using Maven/Gradle**).
- Stores build artifacts (e.g., **JAR/WAR** files).

## 2. Test:

- Jenkins runs **unit tests, integration tests**, or other tests (e.g., using JUnit).

## 3. Deploy:

- If tests pass, Jenkins can deploy the application to a **production** or **staging** environment.

# Triggering Builds

- **Poll SCM:** Jenkins periodically checks the repository for changes.
- **Webhooks:** Jenkins triggers a build when GitHub/Bitbucket sends a webhook event.
- **Manual:** You can manually trigger builds.

# Advanced Jenkins Concepts

## Jenkins Agents (Nodes)

Jenkins uses a **master-slave** architecture. **Master** manages the **build process**, while **Agents** (formerly called "slaves") **execute the build tasks**.

## Agent Types:

- **Static agents:** Pre-configured machines or VMs.
- **Dynamic agents:** Can be spun up on-demand (e.g., Docker containers, Kubernetes pods).

# Advanced Jenkins Concepts

## Distributed Builds

Jenkins can distribute build jobs across different machines to balance the load and speed up the process.

## Jenkins Pipelines as Code

You can define Jenkins pipelines as code using the **Jenkinsfile**. This allows version control for your pipeline and makes it easier to maintain.

# Jenkins Plugins for CI/CD

- **Pipeline Plugin:** Allows Jenkins to support **pipeline-as-code**.
- **Blue Ocean:** Provides a modern UI for Jenkins.
- **JUnit Plugin:** Displays test results within Jenkins.
- **Email Extension Plugin:** Sends notifications based on the status of builds.
- **GitHub Plugin:** Integrates GitHub as an SCM provider.
- **Docker Pipeline Plugin:** Builds and deploys Docker images.

# **Jenkins and Docker**

**Jenkins and Docker** are often used together to:

- **Build Docker images** within Jenkins.
- **Deploy applications** using Docker containers.
- **Run Jenkins** inside a Docker container (Jenkins-in-Docker).

# Build and Push Docker Images in Jenkins

- Install Docker Pipeline plugin.
- Define a pipeline to build and push Docker images:

```
pipeline {  
    agent {  
        docker { image 'maven:3.6.3-jdk-11' }  
    }  
    stages {  
        stage('Build') {  
            steps {  
                sh 'mvn clean package'  
            }  
        }  
        stage('Build Docker Image') {  
            steps {  
                script {  
                    docker.build('my-app-image')  
                }  
            }  
        }  
        stage('Push Docker Image') {  
            steps {  
                script {  
                    docker.withRegistry('https://registry.hub.docker.com', 'docker-hub-credentials') {  
                        docker.image('my-app-image').push('latest')  
                    }  
                }  
            }  
        }  
    }  
}
```

# Complete CI/CD Example

Let's walk through a **full CI/CD pipeline** setup with **Jenkins**, using **GitHub** as the source code repository. In this example, we'll have a workflow that:

- Runs **tests** on every **code push** to GitHub.
- Builds a **Docker image** if the **tests pass**.
- Deploys the application (e.g., to a server or cloud).
- **Sends notifications** (e.g., via **Slack** or **email**) after the build completes.

# Complete CI/CD Example

This workflow will cover:

- **Continuous Integration:** Running tests on every push.
- **Continuous Delivery:** Automatically building and deploying if tests pass.
- **Notifications:** Sending notifications based on the result.

# Complete CI/CD Example

## Set Up Jenkins

If Jenkins isn't already installed, follow the installation steps for your system. Refer to the **previous explanation** for detailed installation steps.

- Ensure Jenkins is running and accessible at **<http://localhost:8080>**.
- Install the necessary plugins for this pipeline:
  - a. **Git Plugin** (for GitHub integration)
  - b. **Pipeline Plugin** (for using Jenkins pipelines as code)
  - c. **Docker Pipeline Plugin** (for Docker integration)
  - d. **Slack Notification Plugin** or **Email Extension Plugin** (for notifications)

# Complete CI/CD Example

[Git plugin 5.5.1](#)

This plugin integrates [Git](#) with Jenkins.

[Report an issue with this plugin](#)

[Docker Pipeline 580.vc0c340686b\\_54](#)

Build and use Docker containers from pipelines.

[Report an issue with this plugin](#)

[Slack Notification Plugin 741.v00f9591c586d](#)

Integrates Jenkins with Slack, allows publishing build statuses, messages and files to Slack channels.



[Report an issue with this plugin](#)

[Email Extension Plugin 1814.v404722f34263](#)

This plugin is a replacement for Jenkins's email publisher. It allows to configure every aspect of email notifications: when an email is sent, who should receive it and what the email says



[Report an issue with this plugin](#)

[Pipeline 600.vb\\_57cdd26fdd7](#)

A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code with Jenkins](#) for more details.



[Report an issue with this plugin](#)

# Complete CI/CD Example

## GitHub Repository Setup

Ensure to have code repository on GitHub. We'll assume the following structure for the project:

 rohanthapa123	Jenkins file and controller	9a65b6d · now	
 .mvn/wrapper	fresh start	1 hour ago	
 src	Jenkins file and control...	now	
 .gitignore	fresh start	1 hour ago	
 Dockerfile	jenkins	1 hour ago	
 Jenkinsfile	Jenkins file and control...	now	
 mvnw	fresh start	1 hour ago	
 mvnw.cmd	fresh start	1 hour ago	
 pom.xml	jenkins	1 hour ago	

# Complete CI/CD Example

## Create a Jenkins Pipeline (Jenkinsfile)

In the root of your GitHub repository, create a file named Jenkinsfile to define your pipeline.

```
pipeline {
    agent any

    environment {
        // Define Docker image name and Slack webhook URL as environment variables
        DOCKERHUB_CREDENTIALS = credentials('docker-hub-access-token')
        DOCKER_IMAGE = 'rohanthapa02/jenkinsdemo'
        SLACK_WEBHOOK_URL = credentials('slackjenkinsci')
        GITHUB_CREDENTIALS = credentials('github-personal-token')
    }

    tools {
        maven 'Maven 3.9.9'
        jdk 'JDK 17'
    }

    stages {
        stage('Checkout') {
            steps {
                echo 'Cloning repository...'
                // Clone the repository from GitHub
                git branch: 'master', url: 'https://github.com/rohanthapa123/Jenkins.git', credentialsId: 'github-personal-token'
            }
        }
    }
}
```

# Complete CI/CD Example

```
stage('Test') {
    steps {
        echo 'Running tests...'
        sh 'mvn clean test'
    }
}

stage('Package'){
    steps{
        echo "Packaging the application"
        sh "mvn clean package"
    }
}

stage('Build Docker Image') {
    when {
        // Only build the Docker image if the tests passed
        expression {
            return currentBuild.resultIsBetterOrEqualTo('SUCCESS')
        }
    }
    steps {
        echo 'Building Docker image...'
        // Build Docker image
        script {
            dockerImage = docker.build("${env.DOCKER_IMAGE}:latest")
        }
    }
}

stage('Deploy to Server') {
    when {
        // Only deploy if the image was built successfully
        expression {
            return currentBuild.resultIsBetterOrEqualTo('SUCCESS')
        }
    }
    steps {
        echo 'Deploying Docker image...'
        sh '''
        docker login -u $DOCKERHUB_CREDENTIALS_USR -p $DOCKERHUB_CREDENTIALS_PSW
        docker push $DOCKER_IMAGE:latest
        '''
    }
}
```

# Complete CI/CD Example

```
stage('Notify') {
    steps {
        script {
            slackSend(
                channel: '#jenkinsdemo',
                tokenCredentialId: 'slackjenkinsci',
                color: 'good',
                message: 'Build was successful!',
                username: 'Jenkins' // Change this to your desired bot name
            )
        }
    }
}

post {
    success {
        script {
            slackSend(
                channel: '#jenkinsdemo',
                tokenCredentialId: 'slackjenkinsci',
                color: 'good',
                message: 'Build was successful!',
                username: 'JenkinsBot' // Change this to your desired bot name
            )
        }
    }
    failure {
        script {
            slackSend(
                channel: '#jenkinsdemo',
                tokenCredentialId: 'slackjenkinsci',
                color: 'danger',
                message: 'Build failed! Please check the Jenkins console output.',
                username: 'JenkinsBot' // Change this to your desired bot name
            )
        }
    }
}
```

# Complete CI/CD Example

## Checkout:

- This stage pulls your code from GitHub.

## Test:

- Runs automated tests .
- If tests pass, the pipeline moves forward.

## Build Docker Image:

- Uses Docker to build an image from your Dockerfile.
- Only executes if the previous stage (Test) was successful.
- The Docker image is tagged as latest and stored locally.

# Complete CI/CD Example

## Deploy to Server:

- This stage pushes the built Docker image to DockerHub.
- Then it SSHs into your remote server and pulls and runs the new image.

## Post:

- Sends a Slack notification on both success and failure. If successful, it sends a green message; if the build fails, it sends a red message.

# Set up Webooks for Github

To trigger the pipeline whenever code is pushed to **GitHub**, you need to set up a webhook.

1. Go to your GitHub repository settings.
2. Select Webhooks and click Add webhook.
3. For the Payload URL, use  
**http://<JENKINS\_URL>:<PORT>/github-webhook/**
4. Set Content type to **application/json**.
5. Choose Let me select individual events and **select Push**.

Now, every time code is pushed to GitHub, Jenkins will be notified and trigger the pipeline.

# Set up Webooks for Github

The screenshot shows the GitHub settings interface for managing webhooks. On the left, a sidebar lists various configuration sections like General, Access, Collaborators, Code and automation, and Webhooks (which is currently selected). The main area is titled "Webhooks / Manage webhook". It contains tabs for "Settings" and "Recent Deliveries". A descriptive text explains that GitHub will send POST requests to the specified URL with event details. The "Payload URL \*" field contains a placeholder URL. The "Content type \*" dropdown is set to "application/json". There's a "Secret" input field and an "SSL verification" section with a note about verifying SSL certificates. Under "SSL verification", there are two radio buttons: "Enable SSL verification" (selected) and "Disable (not recommended)". Below this, a section asks "Which events would you like to trigger this webhook?", with three options: "Just the push event." (selected), "Send me everything.", and "Let me select individual events."

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Actions

**Webhooks**

Codespaces

Pages

Security

Code security

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

Webhooks / Manage webhook

Settings Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

**Payload URL \***

`https://e6ce-2400-1a00-b020-b37e-1d05-6e7a-fb7c-639b.ngrok-free.app`

**Content type \***

`application/json`

**Secret**

**SSL verification**

By default, we verify SSL certificates when delivering payloads.

Enable SSL verification    Disable (not recommended)

**Which events would you like to trigger this webhook?**

Just the push event.  
 Send me everything.  
 Let me select individual events.

**Format:** `https://jenkins-url/github-webhook/`

# Dockerfile Example

Your **Dockerfile** should be configured to build and containerize your application. Here's a simple example for a Spring Boot application:

```
FROM openjdk:17

# Set the working directory
WORKDIR /app

# Copy the application JAR file to the container
COPY target/jenkins.jar /app/jenkins.jar

# Expose the application port
EXPOSE 8085

# Run the application
CMD ["java", "-jar", "/app/jenkins.jar"]
```

# Deployment Setup

- If uploading to cloud server:
- Ensure you have Docker installed and running on your remote server.
- Set up SSH keys so Jenkins can SSH into your server without a password.
- In Jenkins, store the SSH credentials and DockerHub login credentials in the Jenkins credentials store.
- Your deployment can either pull the latest Docker image and run it or do a more sophisticated deployment using Kubernetes, Docker Swarm, or a similar orchestration tool.

# Configure for slack Notification

- Install the Slack Notification Plugin from the **Jenkins Plugin Manager**.
- Configure the Slack plugin with your Slack workspace and channel.
- Create a Slack Webhook (go to Slack > Apps > Custom Integrations > Jenkins CI).
- Store your Slack workspace name and credentials Jenkins credentials (slack-credentials).

# Result

## Push code to Github

```
> git push
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 734 bytes | 183.00 KiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:rohanthapa123/Jenkins.git
  c3d0fad..5d27e48 master -> master
```

## Github push trigger jenkins

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

<a href="https://873d-2400-1a00-b020-611...">https://873d-2400-1a00-b020-611...</a> (push)	<a href="#">Edit</a>	<a href="#">Delete</a>
Last delivery was successful.		

## Jenkins performing ci/cd

Build History

trend ▾

Filter...

#1 | Sep 26, 2024, 3:10 PM

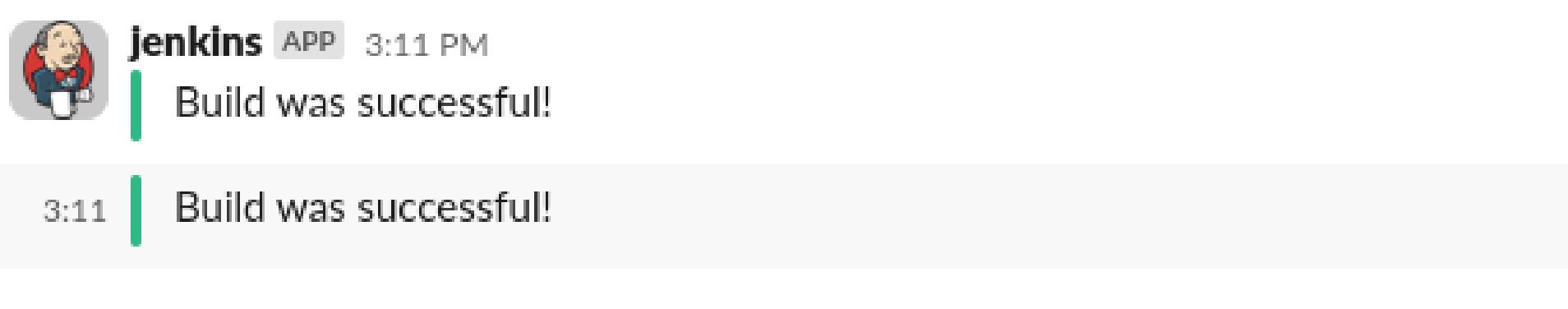
Atom feed for all Atom feed for failures

# Result

Uploaded to docker hub

Tags				
Tag	OS	Type	Pulled	Pushed
latest		Image	14 minutes ago	14 minutes ago
<a href="#">See all</a>				

Notification sent to slack channel



A screenshot of a Slack message from the Jenkins app. The message was sent at 3:11 PM and reads: "Build was successful!". The Jenkins logo is shown next to the message.

# Jenkins with Kubernetes

**Jenkins** can be integrated with **Kubernetes** to dynamically create Jenkins agents (**pods**) in **Kubernetes clusters**, making your **CI/CD** pipeline more scalable.

# Advantages of Jenkins

- **Extensibility:** With over 1,800 plugins, Jenkins can integrate with almost every tool.
- **Automation:** Completely automates the CI/CD pipeline.
- **Flexibility:** Highly customizable and supports any kind of project (Java, Node.js, Python, etc.).
- **Community Support:** Jenkins has a large, active community providing documentation, tutorials, and plugins.
- **Cross-Platform:** Can run on any operating system (Windows, Linux, macOS).

# Tips and Best Practices for Jenkins

- **Use Pipelines as Code:** Always prefer Jenkins Pipelines (via `Jenkinsfile`) over Freestyle Projects.
- **Leverage Parallelism:** Run different stages of the **pipeline in parallel** for faster builds.
- **Use Credentials Safely:** Store sensitive information (like **tokens, passwords**) using **Jenkins credentials**.
- **Build on Separate Agents:** Distribute your builds on different nodes to avoid overloading the master node.

# Tips and Best Practices for Jenkins

- **Regularly Backup Jenkins Config:** Ensure you backup Jenkins configurations (including pipelines, plugins, and credentials).
- **Monitor Jenkins Performance:** Use monitoring tools and plugins to keep an eye on Jenkins performance and system load.

# Conclusion

By **mastering Jenkins**, you can **automate** nearly **every part** of your **software delivery process**—from **building** and **testing** to **deploying** applications across environments.

Jenkins' flexibility, combined with its extensive **plugin ecosystem**, makes it a critical tool for any DevOps workflow. Whether you're working with **Docker**, **Kubernetes**, or deploying **Spring Boot applications**, Jenkins provides the foundation for robust CI/CD pipelines.

# **Thank You**

**Rohan Thapa**

**thaparohan2019@gmail.com**