

**SPRING BOOT**

**CI/CD WITH**

**JENKINS**

**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**.

**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).

**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.

**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

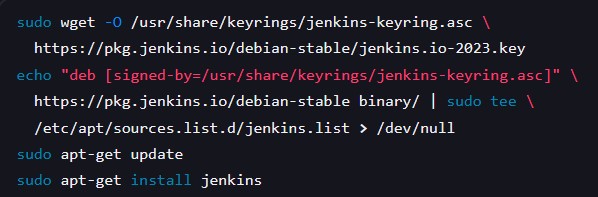
**Installation**

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):



# 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:



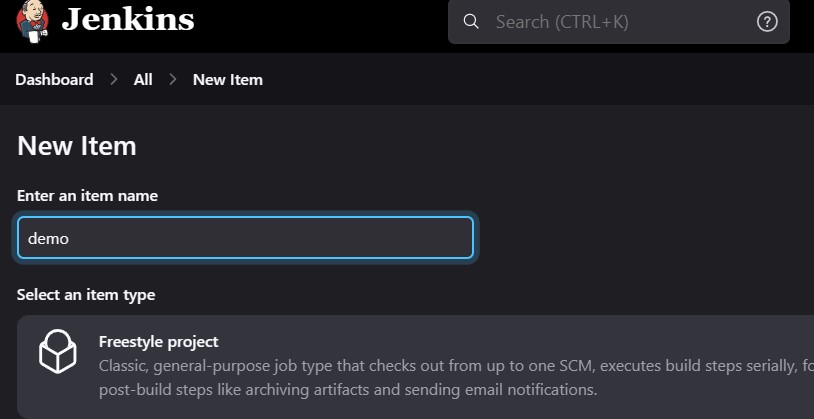
Complete the setup wizard and install recommended plugins.

**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.

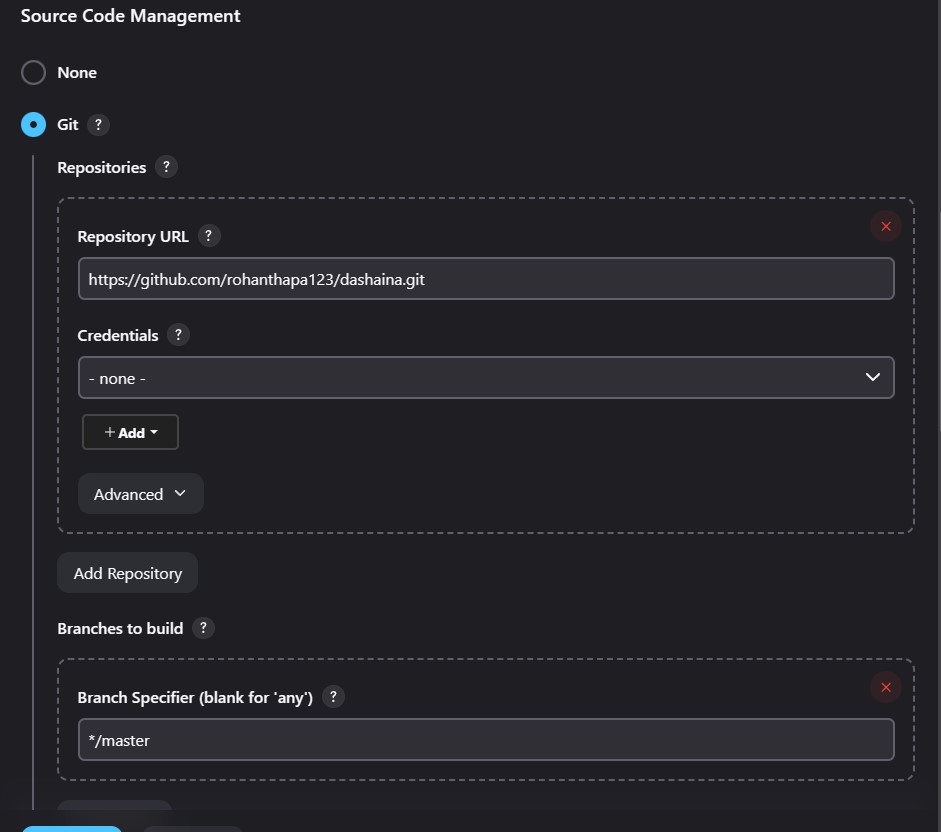
**Steps to create a Freestyle Project:**

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

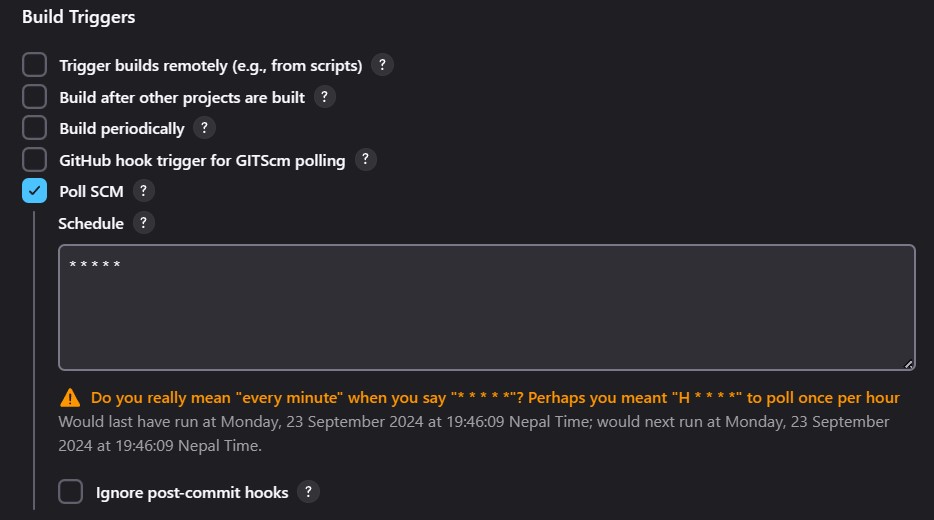


1. Define your **Source Code Management**

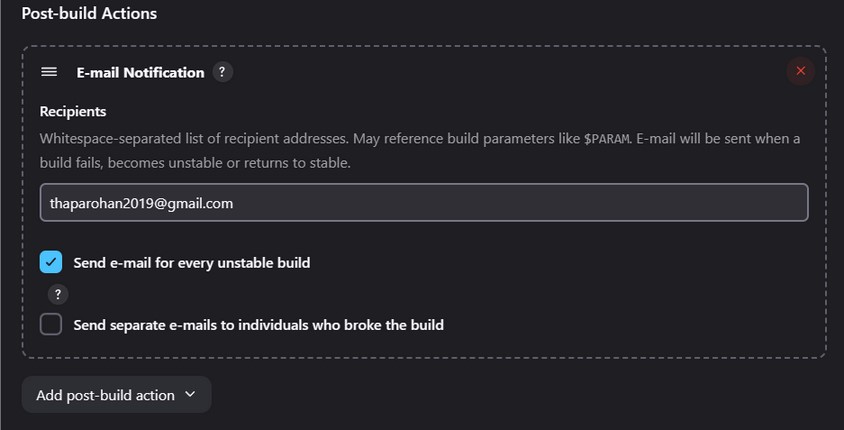
(SCM) system (e.g., **Git**).

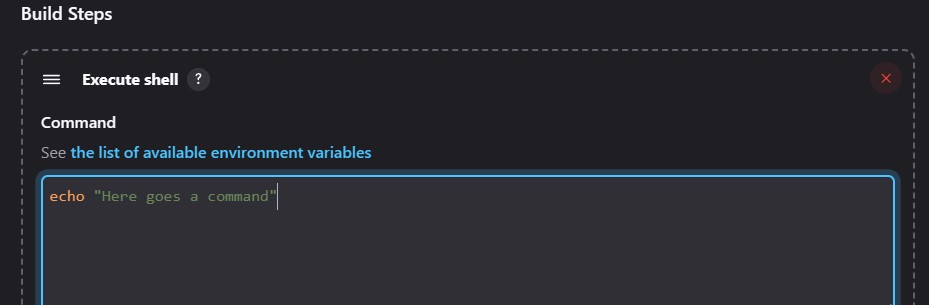


1. Define Build Triggers (when you want the job to run—manually, periodically, on a SCM change, etc.).



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

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



1. Define Post-Build Actions (e.g., **sending notifications, archiving artifacts**).

**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.

**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.



# 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 ondemand (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:



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.

**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.

**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)

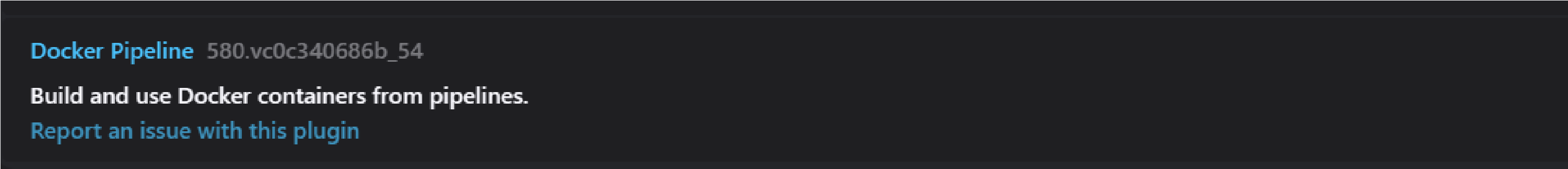
b.**Pipeline Plugin** (for using Jenkins pipelines as code)

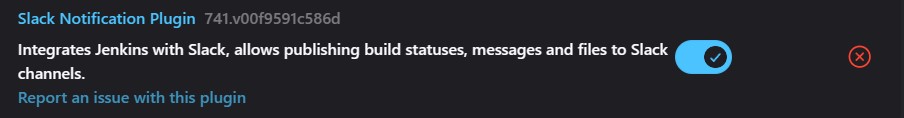
c.**Docker Pipeline Plugin** (for Docker integration)

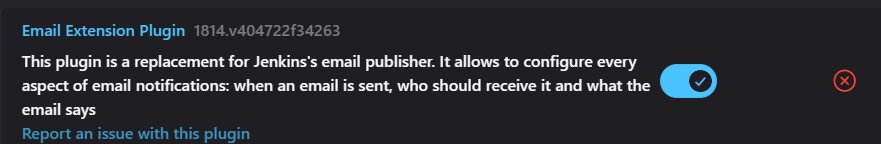
d.**Slack Notification Plugin** or **Email**

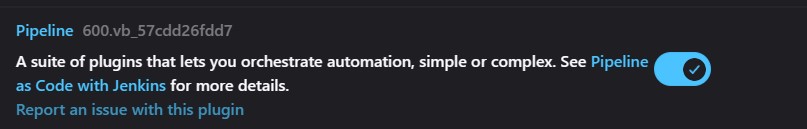
**Extension Plugin** (for notifications)









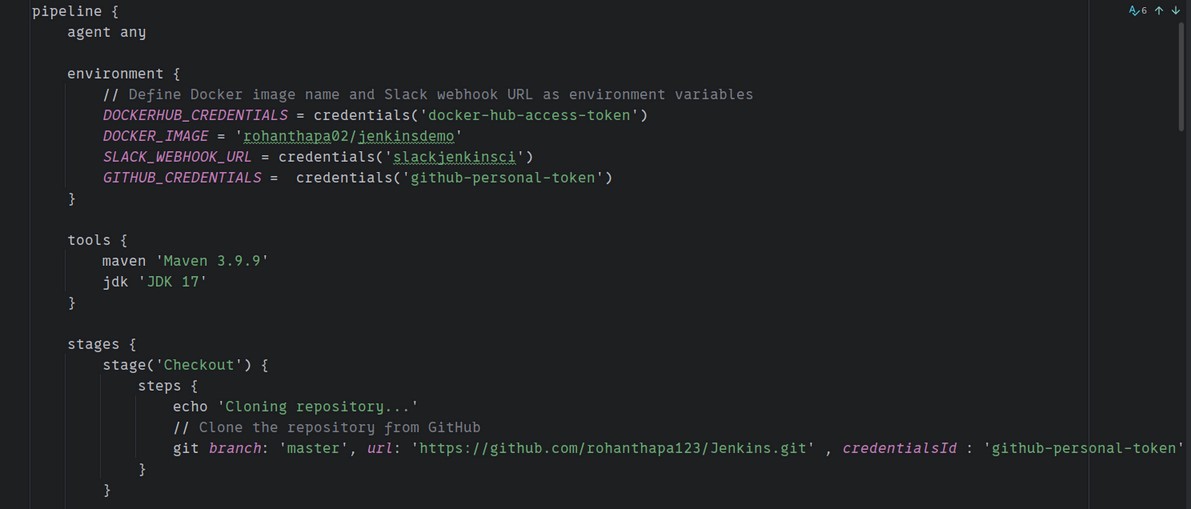


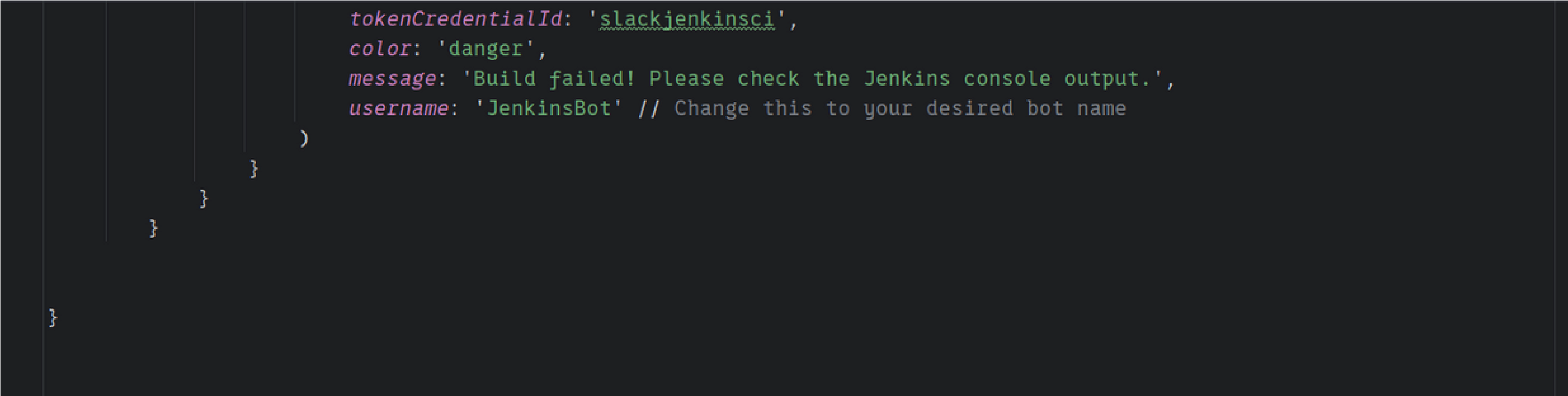
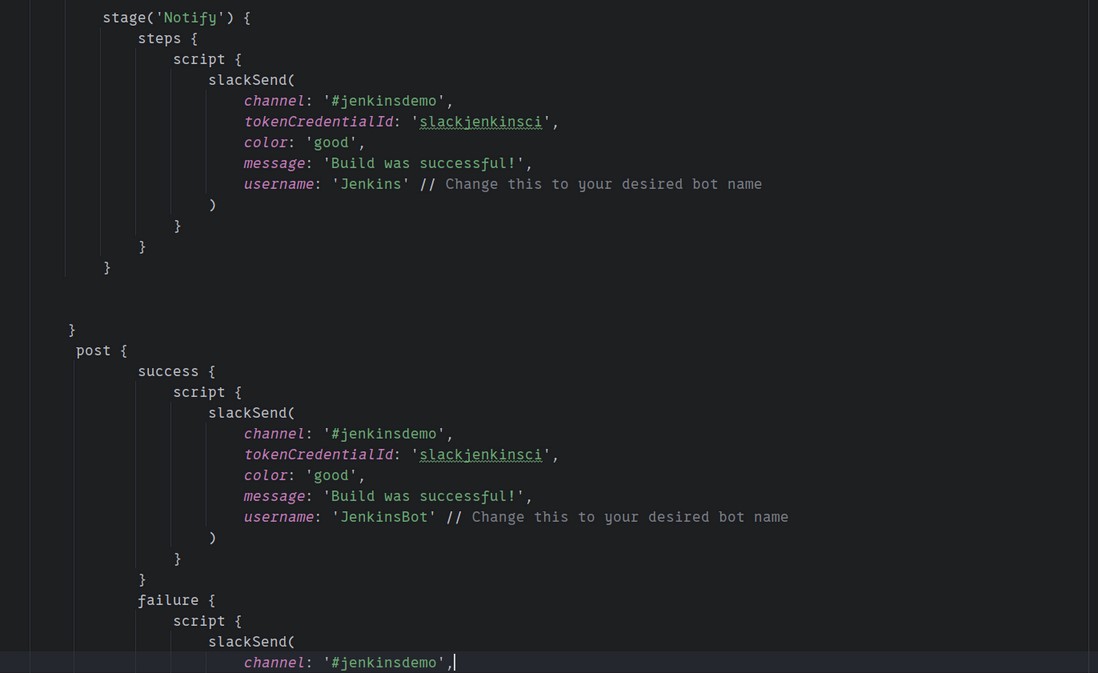
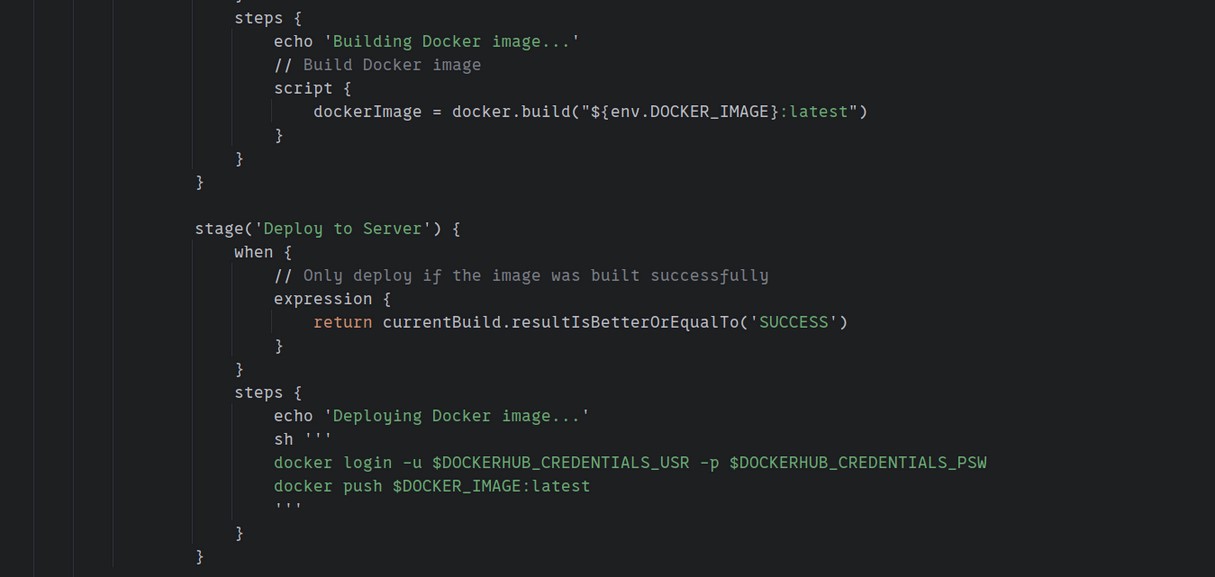
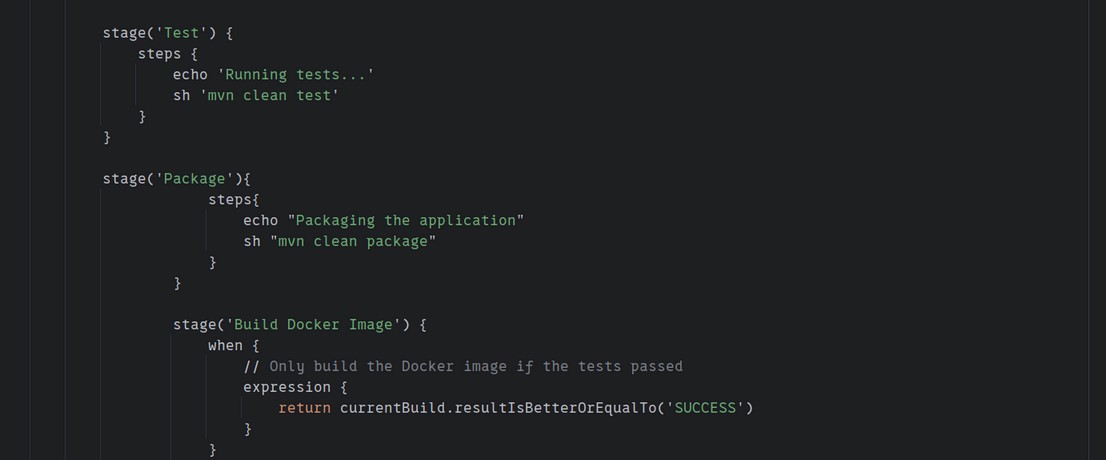
**GitHub Repository Setup**

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



**Create a Jenkins Pipeline (Jenkinsfile)** In the root of your GitHub repository, create a file named Jenkinsfile to define your pipeline.





**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.

**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 Wekbooks 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.

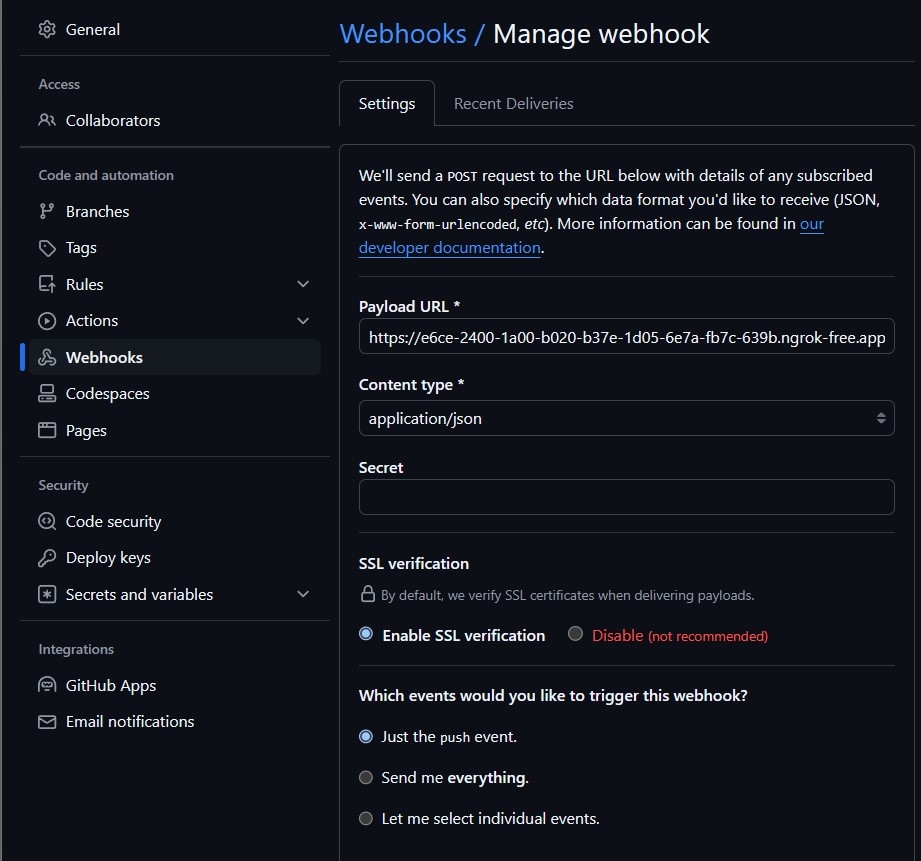
3.For the Payload URL, use **http://<JENKINS\_URL>:<PORT>/githubwebhook/**

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.

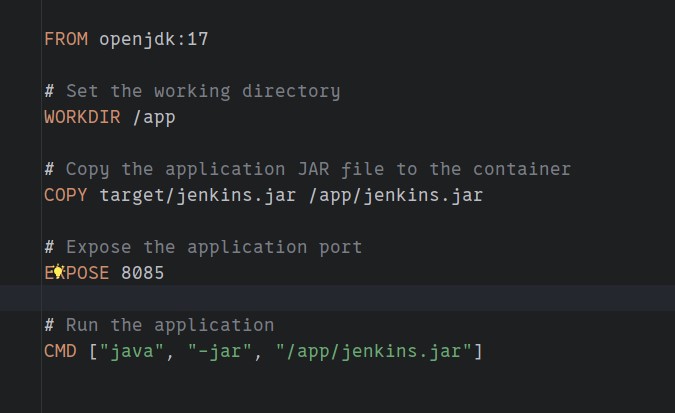
# Set up Wekbooks for Github



***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:



# 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**.

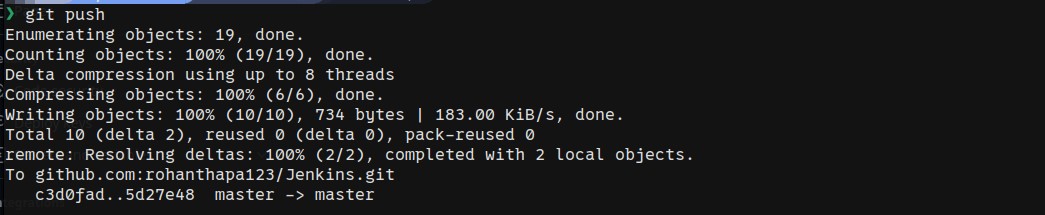
Configure the Slack plugin with your Slack workspace and channel.

Create a Slack Webhook (go to Slack > Apps > Custom Integrations > Jenkins CI).

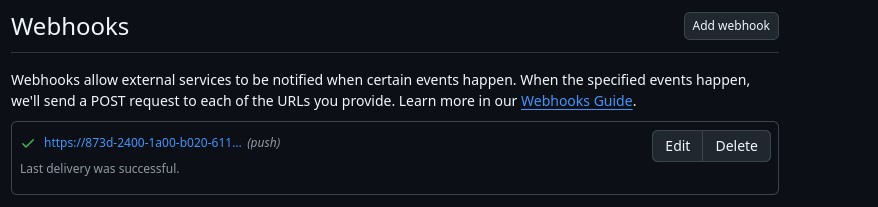
Store your Slack workspace name and credentials Jenkins credentials (slackcredentials).

# Result

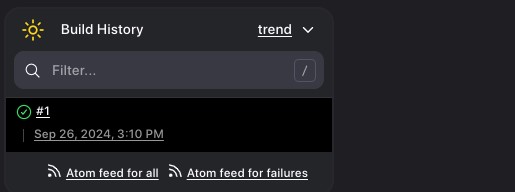
Push code to Github



Github push trigger jenkins



Jenkins perfoming ci/cd

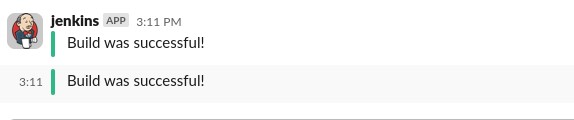


# Result

Uploaded to docker hub



Notification sent to slack channel



# 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.