Git & GitHub In-Depth Integrating Jenkins with Git

Used tools

Git GitHub

AWS EC2

Vs code

Jenkins

Maven

Explain

project focuses on Continuous Integration and Continuous Deployment (CI/CD) using Jenkins, Git, and Maven.

DevOps

DevOps is a software development approach that integrates Development (Dev) and Operations (Ops) to enable continuous integration, continuous deployment (CI/CD), and automation for faster and more reliable software delivery.

Key futures

Collaboration between Dev & Ops.
Continuous Integration & Deployment (CI/CD).
Automation of infrastructure & testing.
Monitoring & feedback loops.

Lyfe cycle:

Plane, develop, build, test, release, deploy, operate, monitor.

DevOps Tools:

- **CI/CD:** Jenkins, GitHub Actions, GitLab CI
- **Configuration Management:** Ansible, Puppet, Chef
- **Containerization:** Docker, Kubernetes
- **Monitoring & Logging:** Prometheus, ELK Stack
- **Version Control:** Git, SVN

Benefits of DevOps:

Faster Software Delivery

Improved Collaboration
Higher Software Quality
Increased Security
Cost & Time Efficiency

LAUNCHING AN EC2 INSTANCE

1. Log into AWS Management Console.

2. Navigate to EC2 Dashboard and select Launch Instance.

3. Choose a Linux AMI (e.g., Amazon Linux 2).

4. Select an instance type (e.g., t2.micro for free tier).

5. Configure security group to allow SSH access (port 22).

6. Launch the instance and download/attach a key pair for SSH access

Git Overview

Git is a distributed version control system (VCS) that helps developers track changes in their code, collaborate efficiently, and manage project versions. It was created by Linus Torvalds.

Why Use Git?

Tracks every change in the code.
Enables multiple developers to work together.
Allows branching and merging for feature development.
Provides a backup of the project.

Basic Git Workflow:

1. **Initialize a Repository** → git init
2. **Clone an Existing Repository** → git clone <repo-url>
3. **Check Status** → git status
4. **Add Changes to Staging** → git add <file> or git add .
5. **Commit Changes** → git commit -m "Commit message"
6. **View Commit History** → git log
7. **Create & Switch Branches** → git branch <branch-name>, git checkout <branch-name>
8. **Merge Branches** → git merge <branch-name>

9. Command used to manage and interact with remote repositories → git remote add origin <remote-url>

10. **Push Changes to Remote** → git push –u origin main or master
11. **Pull Latest Changes** → git pull origin main or master.

#Create GitHub account

GitHub is a cloud-based platform for hosting and managing Git repositories. It allows developers to collaborate on projects, track changes, and manage code versions efficiently. GitHub provides features like:

- **Version Control**: Uses Git for tracking code changes.
- **Collaboration**: Allows multiple developers to work on the same project using branches, pull requests, and code reviews.
- **Repositories**: Stores project files and version history.
- **GitHub Actions**: Automates CI/CD workflows.
- **Issue Tracking**: Helps in bug tracking and feature requests.
- **Security & Code Management**: Offers vulnerability scanning and access controls.
- **Integration**: Supports third-party tools like Jenkins, Maven, and Docker.

Since you are working on a **DevOps CI/CD project**, GitHub can be useful for managing source code and integrating with Jenkins for automation. Do you need help setting up a GitHub repository or integrating it with Jenkins?

Crate a repository like **Puropale project** clone

https://github.com/Bajisai/Puropale-project-.git

Then write code in vs code in pull and push to GitHub account
Git init

Git add .

git pull origin main

Git commit –m “puropale project”

Git remote origin https://github.com/Bajisai/Puropale-project-.git

Git push –u origin main

Then check the GitHub account there push or not.

What is a Git Branch?
A branch in Git is like a separate line of development that allows multiple developers to work on different features or bug fixes without affecting the main codebase.

Create New branch --> git branch <branch-name>

Switch to Another Branch--> git checkout -b <branch-name>

-->git merge is a command used to **combine changes** from one branch into another.

Switch to the Target Branch (Main Branch)

--> git merge <branch-name>

Merge Another Branch into the Current Branch

--> git merge <branch-name>

Maven

Maven is a build automation tool primarily used for Java projects but can manage a variety of project types. It simplifies the process of building, managing dependencies, and publishing documentation for software projects. Here are some of its key features.

PROJECT OBJECT MODEL (POM)

At the heart of Maven is the Project Object Model (POM), an XML file (pom.xml) that defines the project structure, dependencies, build process, and plugins. The POM is essential because it:

Specifies Dependencies: Lists the libraries and frameworks your project requires. Maven automatically downloads and manages these dependencies from online repositories. Configures

Build Settings: Provides settings such as versioning and the Java SDK used for compilation.

REPOSITORIES

Maven uses repositories to store and retrieve project artifacts. There are two primary types of repositories.

Local Repository: A structure on the developer's machine that caches downloaded dependencies.
Remote Repository: A central server (like Maven Central) that hosts libraries which can be accessed by all developers.

PLUGINS

Maven supports plugins to extend its functionality. These plugins can automate tasks such as compiling code, running tests, or packaging applications. Common plugins include.

Ex: pom.xml

```
xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">


  <modelVersion>4.0.0</modelVersion>


  <groupId>com.example</groupId>

  <artifactId>hello-world</artifactId>

  <version>1.0-SNAPSHOT</version>

  <packaging>jar</packaging>
```

```
<name>Hello World</name>
<description>A simple Hello World application for Jenkins</description>

<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
```

```
            <version>3.2.0</version>
            <configuration>
              <archive>
                <manifest>
                  <mainClass>com.example.App</mainClass>
                </manifest>
              </archive>
            </configuration>
          </plugin>
        </plugins>
    </build>

</project>
```

To build the project, run the command.

mvn clean install

Jenkins

Jenkins is an open-source automation server recognized for its key role in Continuous Integration (CI) and Continuous Delivery (CD) in modern software development. It provides developers with the ability to automate the entire lifecycle of software development and deployment through a user-friendly interface.

AUTOMATION THROUGH PIPELINES, JOBS, AND PLUGINS

Automation through Pipelines, Jobs, and Plugins

Jenkins facilitates automation via:

•**pelines**: Defined through a Jenkinsfile, containing the stages and steps required to build, test, and deploy an application.

•**Jobs:** Tasks triggered manually or automatically based on events such as changes in the source code repository. These can include build scripts, test execution, or application deployment.

•**Plugins**: Jenkins has a vast ecosystem of plugins that extend its capabilities, including integrations with Git, Maven, Docker, and cloud services.

Example Jenkins Pipeline

A simple Jenkins pipeline integrating with GitHub and Maven:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/username/repository.git' // Clone GitHub repo
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean install' // Execute Maven build
            }
        }
    }
}
```

This pipeline checks out the code from a GitHub repository and runs the Maven build command.

Automating Deployment

To automate deployment with Jenkins, set up a Jenkins pipeline to monitor your GitHub repository. Configure it to trigger builds automatically when code is pushed, ensuring continuous integration and delivery.

CONCLUSION

By integrating Git, GitHub, Maven, and Jenkins on an AWS EC2 instance, developers can establish a seamless workflow for software development and deployment. This setup enables efficient version control, build automation, continuous integration, and streamlined delivery, ensuring high-quality software development practices.