**What is CI/CD?**

CI/CD stands for Continuous Integration and Continuous Deployment (or Delivery). It's a software development practice that aims to improve the quality, speed, and reliability of software releases.

**Continuous Integration (CI):**

* Involves integrating code changes from multiple developers into a single repository frequently.
* Automated tests are run to ensure the code changes didn't break the existing functionality.
* Feedback is provided to developers quickly, allowing them to fix issues early.

**Continuous Deployment (CD):**

* Involves automating the deployment of software to production after it has been tested and validated.
* Deployment is done automatically, without human intervention.
* Reduces the risk of human error and ensures faster time-to-market.

**What is a CI/CD Pipeline?**

A CI/CD pipeline is a series of automated processes that take code from version control, build, test, and deploy it to production. The pipeline is designed to automate the entire software release process, from code commit to production deployment.

**Automates software release process from:**

1. Source Code Management: Code is stored in a version control system like Git.

2. Build: Code is built into a deployable artifact.

3. Automated Testing: Artifact is tested to ensure it meets quality standards.

4. Deployment: Artifact is deployed to production.

5. Monitoring: Application is monitored for performance and errors.

**What is Deployment?**

Deployment is the process of delivering software to production, making it available to end-users. Deployment can be done manually or automatically, using CI/CD pipelines.

**Types of deployment:**

1. Manual Deployment: Humans manually deploy software to production.

2. Automated Deployment: Software is deployed automatically, using CI/CD pipelines.

3. Continuous Deployment: Software is deployed automatically, after passing automated tests.

**How is CI/CD used in Jenkins?**

Jenkins is a popular CI/CD tool that automates the entire software release process. Here's how CI/CD is used in Jenkins:

1. Job Creation: Create a new job in Jenkins, specifying the source code repository, build script, and deployment details.

2. SCM Polling: Jenkins polls the source code repository for changes, triggering the build process when changes are detected.

3. Build and Test: Jenkins builds the software, runs automated tests, and validates the results.

4. Deployment: Jenkins deploys the software to production, using automated deployment scripts.

5. Monitoring: Jenkins monitors the application for performance and errors, providing feedback to developers.



1. **Developer:** The developer writes code locally and pushes it to GitHub
2. **GitHub:** The source code is stored in a GitHub repository, serving as version control.
3. **Maven:** Maven is triggered to build the project when the developer pushes changes. It compiles the code, runs unit tests, and packages the application (e.g., WAR/JAR file).
4. **Jenkins:** After Maven completes the build and packaging, Jenkins is responsible for deploying the artifact to the Tomcat Server.
5. **Tomcat Server:** The artifact is deployed to Tomcat, where it is hosted for access by **End Users**
6. **End User:** End users access the application via their browser.

**Feedback Loop:**

* If end users report issues, the developer updates the code and pushes changes back to GitHub, triggering the process again.

**Workflow of Github:**

**Step 1: Create a new repository in Github**

1. Create a new directory for your project.
2. Initialize a new Git repository using **git init.**
3. Create a new file called README.md to describe your project.

**Step 2: Add files to the repository**

1. Create new files or modify existing ones.
2. Stage changes using **git add <file> .<extension>**
3. Verify staged changes using **git status**

**Step 3: Commit changes**

1. Commit staged changes using **git commit -m "<commit message>"**
2. Verify commit history using **git log**

**Step 4: Create a new branch**

1. Create a new branch using **git branch -M <branch-name>**
2. Switch to the new branch using **git checkout <branch-name>**

**Step 5: Merge changes into master branch**

1. Switch to the master branch using **git checkout master**
2. Merge changes from the new branch using **git merge <branch-name>.**

**Step 6: Push changes to remote repository**

1. Add a remote repository using **git remote add origin <SSH-url>**
2. Push changes to the remote repository using **git push origin <branch-name>.**

**Step 7: Pull changes from remote repository**

1. Pull changes from the remote repository using **git pull origin <branch-name>.**

**Step 8: Resolve conflicts (if necessary)**

1. Resolve conflicts using **git status** and **git merge --continue**
2. Commit resolved conflicts using **git commit**

**Step 9: Repeat the cycle**

1. Repeat the cycle of making changes, committing, and pushing to the remote repository.

**Git commands:**

* git init: Initialize a new Git repository.
* git add: Stage changes.
* git commit: Commit staged changes.
* git branch: Create a new branch.
* git checkout: Switch to a different branch.
* git merge: Merge changes from one branch into another.
* git push: Push changes to a remote repository.
* git pull: Pull changes from a remote repository.
* git status: Verify staged changes.
* git log: Verify commit history.

**Workflow of Maven:(Build Tool)**

1. Create a New Maven Project : **mvn archetype:create**
2. Navigate to the Project Directory: **cd myproject**
3. Clean the Project: **mvn clean**
4. Validate the project: **mvn validate**
5. Compile the Project: **mvn compile**
6. Test the Project: **mvn test**
7. Package the Project: **mvn package**
8. Verify the Project: **mvn verify**
9. Install the Project: **mvn install**
10. Deploy the Project: **mvn deploy**

**Or can use :**

1. Cleans and packages the project: **mvn clean package**
2. Cleans, packages, and installs the project: **mvn clean install**
3. Cleans, packages, installs,& deploys the project:  
    **mvn clean deploy**
4. Compiles the test source code: **mvn test-compile**
5. Runs the integration tests: **mvn integration-test**

**Complete workflow of Jenkins:(Automation tool)**

**Step 1: Install and Configure Jenkins**

1. Install Jenkins on a server or virtual machine.
2. Configure Jenkins by setting up the administrator account, installing plugins, and configuring the Jenkins URL.

**Step 2: Create a New Job**

1. Log in to the Jenkins dashboard and click on "New Item".
2. Enter a name for the job and select the type of job (e.g., Freestyle project, Maven project, etc.).

**Step 3: Configure the Job**

1. Configure the job by setting up the source code management (SCM) system, such as Git or SVN.
2. Set up the build triggers, such as building the job periodically or after a Git push.
3. Configure the build environment, such as setting up the JDK or Maven versions.

**Step 4: Add Build Steps**

1. Add build steps to the job, such as:

* Building the project using Maven
* Running automated tests using JUnit or TestNG.
* Deploying the application to a server or container.

**Step 5: Configure Post-Build Actions**

1. Configure post-build actions, such as:

* Sending email notifications after a successful or failed build.
* Deploying the application to a production environment.
* Running automated tests or analysis tools.

**Step 6: Run the Job**

1. Run the job manually by clicking the "Build Now" button.
2. Verify that the job runs successfully and that the build artifacts are generated.

**Step 7: Monitor and Analyze the Job**

1. Monitor the job's build history and logs.
2. Analyze the job's performance and identify areas for improvement.
3. Use Jenkins plugins, such as the Jenkins Performance Plugin, to analyze the job's performance.

**Step 8: Maintain and Update the Job**

1. Maintain and update the job by updating the build scripts, plugins, and configurations.
2. Ensure that the job remains stable and reliable.
3. Continuously monitor and analyze the job to identify areas for improvement.

**Jenkins Plugins**

* Git Plugin: Integrates Jenkins with Git.
* Maven Plugin: Integrates Jenkins with Maven.
* Gradle Plugin: Integrates Jenkins with Gradle.
* JUnit Plugin: Integrates Jenkins with JUnit.
* TestNG Plugin: Integrates Jenkins with TestNG.
* Email Extension Plugin: Sends email notifications after a build.
* Deploy to container Plugin: Deploys the application to a container.

**Workflow Docker(Containerization)**

1. Create Dockerfile: **touch Dockerfile**
2. Build Docker Image: **docker build -t myimage** .
3. Pull Image from Registry: **docker pull <image-name>**
4. Check image pulled or not**: docker images**
5. Display pulled image: **docker inspect <imageID>**
6. Create Docker Container: **docker run -itd --name mycontainer -p <host:container-portnum><imageID>**
7. Verify Docker Container:   
   **docker ps -a**  
   **docker logs mycontainer**
8. Push Image to Registry and create tag:   
   **docker tag myimage:latest <username>/en>v:<tag-name>**  
   **docker push <username>/evn>:<tag-name>**
9. Start Docker Container: **docker start mycontainer**
10. Stop Docker Container: **docker stop mycontainer**
11. Remove Docker Container: **docker rm mycontainer**
12. Remove Docker Image: **docker rmi Imagename**

**Jenkins workflow in Tomcat:**

**Step 1: Install Jenkins WAR file in Tomcat**

1. Download the Jenkins WAR file from the official Jenkins website.
2. Copy the WAR file to Tomcat's webapps directory.
3. Restart Tomcat to deploy the Jenkins application.

**Step 2: Access Jenkins Web Interface**

1. Open a web browser and navigate to <http://localhost:8080/jenkins>.
2. Configure Jenkins by setting up the administrator user and password.

**Step 3: Configure Jenkins**

1. Configure Jenkins' global settings, such as the JDK installation, Git plugin, and Maven plugin.
2. Set up Jenkins' security settings, such as authentication and authorization.

**Step 4: Create Jenkins Job**

1. Create a new Jenkins job by clicking on "New Item" in the Jenkins dashboard.
2. Configure the job settings, such as the job name, description, and build triggers.

**Step 5: Configure Build Steps**

1. Configure the build steps for the job, such as:

* Building the project using Maven or Gradle.
* Running unit tests and integration tests.
* Deploying the application to a production environment.

**Step 6: Run Jenkins Job**

1. Run the Jenkins job by clicking on the "Build Now" button.
2. Monitor the job's progress and output in the Jenkins console.

**Step 7: Review Jenkins Job Results**

1. Review the job's results, including any errors or warnings.
2. Analyze the job's performance metrics, such as build time and test coverage.

**Step 8: Maintain and Optimize Jenkins**

1. Regularly maintain and optimize Jenkins by:

* Updating plugins and dependencies.
* Monitoring performance and adjusting settings as needed.
* Implementing security patches and updates.