JENKINS PLUGINS AND CI/CD PIPELINE

JENKINS PLUGINS

What are Jenkins Plugins?

Jenkins plugins are extensions that enhance Jenkins functionality by integrating with various tools, improving automation, and adding new features. They allow customization based on project needs.

Why are Jenkins Plugins Used?

- **Extend Jenkins capabilities**: Add new integrations, build steps, and reporting features.
- **Automate tasks**: Reduce manual interventions in CI/CD pipelines.
- **Integrate with external tools**: Such as Git, Docker, Kubernetes, and AWS.
- **Enhance monitoring and security**: With plugins for logs, security scanning, and notifications.

Methods to Install Plugins in Jenkins

1. **Using the Jenkins Plugin Manager**
    - Navigate to **Manage Jenkins > Manage Plugins**.
    - Search for the required plugin.
    - Click **Install without restart** or **Install and restart**.
2. **Manually Installing a Plugin (.hpi/.jpi File)**
    - Download the .hpi or .jpi plugin file from the Jenkins plugin repository.
    - Upload it via **Manage Jenkins > Manage Plugins > Advanced > Upload Plugin**.
    - Restart Jenkins to apply changes.
3. **Using the Jenkins CLI**
    - Use the command:
    - java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin <plugin-name>
    - Restart Jenkins to complete the installation.

Build Triggers in Jenkins

Build triggers define how and when Jenkins should start a job. Below are three commonly used triggers:

1. **Poll SCM** (Source Control Management)
    - Jenkins checks the version control system (e.g., Git) at regular intervals to detect code changes.
    - **Steps to Enable:**
        - Go to **Job Configuration > Build Triggers**.
        - Select **Poll SCM**.
        - Specify a schedule (e.g., H/5 * * * * to check every 5 minutes).
2. **Build Periodically**
    - Schedules builds at fixed intervals, independent of code changes.
    - **Steps to Enable:**
        - Go to **Job Configuration > Build Triggers**.
        - Select **Build Periodically**.
        - Define a cron schedule (e.g., @daily for once a day).
3. **Webhooks**
    - Triggers a Jenkins build when a push or pull request occurs in a Git repository (e.g., GitHub, GitLab).
    - **Steps to Enable:**
        - Install the **GitHub Plugin** in Jenkins.
        - In GitHub, go to **Repository > Settings > Webhooks**.
        - Add a webhook with the Jenkins URL: http://your-jenkins-server/github-webhook/.
        - Enable **Push events** to trigger builds.

CI/CD Process

What is CI/CD?

CI/CD (Continuous Integration/Continuous Delivery or Deployment) is a set of practices in software development that automate code integration, testing, and deployment.

CI (Continuous Integration) Process

- Developers push code to a shared repository.
- Jenkins fetches the latest code and triggers a build.
- Unit tests and other automated tests run to check code quality.
- If tests pass, the build artifact is stored (e.g., in a Docker registry or artifact repository).

CD (Continuous Delivery/Deployment) Process

- **Continuous Delivery**: Ensures the software is always ready for deployment but requires manual approval.
- **Continuous Deployment**: Fully automates the deployment process, making new changes live without manual intervention.

Pipeline Procedure in Jenkins

1. **Code Commit**: Developer pushes code to Git.
2. **Build Stage**: Jenkins pulls the latest code and compiles it.
3. **Test Stage**: Runs unit and integration tests.
4. **Artifact Storage**: Stores the build output in a repository (e.g., AWS S3, Docker Hub).
5. **Deployment**: Deploys to testing, staging, or production environments.
6. **Monitoring & Feedback**: Logs and reports are generated to improve future builds.

Feedback Feature in CI/CD

- **Automated Testing Feedback**: Helps developers identify issues early.
- **Build and Deployment Logs**: Logs from Jenkins, Docker, and cloud services provide insights into failures.
- **Alerts & Notifications**: Slack, email, or monitoring tools notify teams of build failures, ensuring quick response times.

Traditional Software Development vs. CI/CD Approach

Feature	Traditional Development	CI/CD Approach
Deployment Frequency	Few times per year	Multiple times per day/week
Error Detection	Late-stage, after integration	Early detection through continuous testing
Manual Effort	High, with long approval cycles	Automated, reducing human intervention
Risk	High due to batch releases	Low, with small incremental releases

Traditional Development Process (Before CI/CD)

Code Development ---> Manual Testing ---> Manual Deployment ---> Production Release

Slow, error-prone process with long release cycles and delayed feedback.

CI/CD Pipeline Process (After CI/CD)

Developers ---> Code Commit ---> Build – Test – Deploy

Automated, fast, and reliable deployment process with continuous feedback loops.