1. Explain the architecture of Jenkins and discuss how it facilitates continuous integration and continuous delivery (CI/CD) pipelines. In your answer, describe the role of Jenkins master and agents, plugins, and how Jenkins integrates with various development tools. Provide examples of how Jenkins can automate the stages of a typical software development lifecycle.

Jenkins Architecture and Its Role in CI/CD Pipelines

Jenkins is an open-source automation server widely used to implement **continuous integration (CI)** and **continuous delivery (CD)** pipelines. Its architecture is designed to support the automation of various stages in the software development lifecycle, ensuring faster delivery, higher quality, and reliable deployments.

🏗️ 1. Jenkins Architecture Overview

The Jenkins architecture follows a **master-agent** model that ensures scalability, flexibility, and efficient resource utilization.

🔹 **Jenkins Master**

- **Role:**
  - Orchestrates the entire Jenkins environment.
  - Manages build schedules, assigns jobs to agents, and monitors execution.
  - Provides the web-based user interface (UI) for configuring jobs and pipelines.
- **Functions:**
  - Handles job configurations and job queue management.
  - Triggers builds based on events (e.g., Git commits).
  - Stores build results and logs.

🔹 **Jenkins Agents**

- **Role:**
  - Perform the actual tasks like building, testing, and deploying applications.
- **Functions:**
  - Allow distributed builds by running on different machines or environments.
  - Execute tasks based on instructions from the master.
- **Types of Agents:**
  - **Static Agents:** Always connected and available.
  - **Dynamic Agents:** Created on-demand using Docker, Kubernetes, or cloud services like AWS.

🔌 2. Role of Plugins in Jenkins

Plugins extend Jenkins's functionality, allowing it to integrate with various tools and frameworks.

- **Examples of Essential Plugins:**
  - **Git Plugin:** For source code management.
  - **Pipeline Plugin:** To define and run CI/CD pipelines.
  - **JUnit Plugin:** For running and reporting test results.
  - **Docker Plugin:** To run builds in Docker containers.
- **Impact of Plugins:**
  - Enable Jenkins to work seamlessly with third-party tools.
  - Provide customizable build, test, and deployment workflows.

🔗 3. Jenkins Integration with Development Tools

Jenkins integrates with numerous development tools, making it highly adaptable in different environments.

- **Version Control Systems (VCS):** Git, GitHub, GitLab, Bitbucket.
- **Build Tools:** Maven, Gradle, Ant.
- **Testing Frameworks:** JUnit, Selenium, TestNG.
- **Deployment Tools:** Docker, Kubernetes, Ansible.
- **Notification Systems:** Slack, Email, Microsoft Teams.

🔄 4. Jenkins in Automating CI/CD Pipelines

Jenkins automates the CI/CD process by managing each stage of the software development lifecycle.

Example Pipeline Automation:

1. **Code Commit (Continuous Integration):**
   - Developers push code to a Git repository.
   - Jenkins detects the commit via webhooks and triggers a build.
2. **Build Stage:**
   - Jenkins pulls the latest code and compiles it using tools like Maven or Gradle.
   - Any build failures are immediately reported to the development team.
3. **Testing Stage:**
   - Automated tests (unit, integration, and UI tests) run using frameworks like JUnit or Selenium.
   - Test results are published in the Jenkins UI for review.

4. **Artifact Management:**
    - Successful builds produce deployable artifacts.
    - Artifacts are stored in repositories like Nexus or Artifactory.
5. **Deployment Stage (Continuous Delivery):**
    - Jenkins deploys the application to a staging environment for further testing.
    - Post-deployment tests (such as smoke tests) ensure deployment success.
6. **Production Deployment (Continuous Deployment):**
    - After approval, Jenkins can automatically deploy the application to production environments.
    - Blue-green or canary deployment strategies can be used for zero-downtime releases.

⚡ 5. Benefits of Jenkins in CI/CD

- **Faster Feedback Loop:** Jenkins provides quick feedback on code changes, reducing the time to detect and fix issues.
- **Increased Automation:** Reduces manual effort and human error in the build and deployment process.
- **Scalability:** Distributed architecture allows multiple builds and tests to run in parallel.
- **Flexibility:** Supports a wide range of plugins and integrations for diverse development needs.

2. How can Jenkins be integrated with Git for automated builds?

Integrating Jenkins with Git for Automated Builds

Jenkins can be seamlessly integrated with Git to automate the process of building code whenever changes are pushed to a Git repository. This integration ensures continuous integration (CI) by providing immediate feedback on code quality and build status.

⚡ Steps to Integrate Jenkins with Git for Automated Builds:

1️⃣Install the Git Plugin in Jenkins

- Go to **Jenkins Dashboard** → **Manage Jenkins** → **Manage Plugins**.
- In the **Available** tab, search for **Git Plugin** and install it (if not already installed).
- The Git plugin allows Jenkins to pull code from Git repositories.

2️⃣ Configure Global Git Settings

- Navigate to **Manage Jenkins → Global Tool Configuration**.
- Under **Git**, specify the path to the Git executable installed on the Jenkins server.
- Jenkins uses this Git installation to execute Git commands.

3️⃣ Create a New Jenkins Job

- From the **Jenkins Dashboard**, click **New Item**.
- Enter a name for the job, select **Freestyle project** or **Pipeline**, and click **OK**.

4️⃣ Configure Git Repository in Job

- In the job configuration page:
  - Go to the **Source Code Management** section.
  - Select **Git** and provide the **Repository URL** (e.g., `https://github.com/username/repo.git`).
  - If the repository is private, provide **credentials** (username/password or SSH key).

5️⃣ Set Up Build Triggers for Automation

- In the **Build Triggers** section, choose one of the following:
  - ✅ **Poll SCM:** Jenkins will periodically check for changes in the repository (e.g., `H/5 * * * *` for every 5 minutes).
  - ✅ **GitHub hook trigger for GITScm polling:** Use this for real-time builds when connected with GitHub webhooks.

6️⃣ Configure Build Steps

- In the **Build** section, define how Jenkins should build the project. For example:
  - **Execute Shell:** For shell commands (`mvn clean install`).
  - **Invoke Gradle/Maven Targets:** If using build tools like Maven or Gradle.

7️⃣ Set Up Webhooks for Real-Time Builds (GitHub Example)

- In the **GitHub repository**, go to **Settings → Webhooks → Add webhook**.

- Enter the Jenkins URL followed by `/github-webhook/` (e.g., `http://your-jenkins-url/github-webhook/`).
- Select **application/json** as the content type and choose **Just the push event**.
- Jenkins will now trigger a build automatically whenever new code is pushed to the repository.

Write about poll SCM trigger. What will be the configuration poll SCM if you want to trigger the build after two days periodically? What is the advantage of poll SCM over git hook?

Already done in class.