Chapter 3 : Continuous Integration Using Jenkins (06)

Introduction to CI/CD, Install & Configure Jenkins, Jenkins Architecture Overview, Build Jobs and Configurations, Jenkins Plugins, Jenkins Integration with other Tools.

C3.3. About Title of Chapter

The title "Continuous Integration Using Jenkins" can be broken down into its components and explained as follows:

- Continuous Integration (CI): This refers to a software development practice where code changes are automatically built, tested, and integrated into a shared repository multiple times a day. The goal is to detect and address integration issues early in the development process.
    - Explanation: Continuous Integration is a methodology that emphasizes frequent integration of code changes to minimize conflicts and identify issues sooner rather than later. It ensures that new code additions do not break existing functionality and maintains code quality.
- Using: In this context, "using" indicates the tool or platform that will be employed to implement Continuous Integration practices.
    - Explanation: The term "using" suggests that Jenkins, a specific tool or platform, will be utilized to facilitate the CI process. Jenkins is an automation server widely used for CI/CD purposes.
- Jenkins: Jenkins is an open-source automation server that helps automate various aspects of the software development process, including building, testing, and deploying code changes.
    - Explanation: Jenkins is the central focus of this title. It is the tool that will be featured in the chapter or content to explain how Continuous Integration practices are implemented.

In summary, the title "Continuous Integration Using Jenkins" indicates that the content will cover the implementation of Continuous Integration practices, with a specific emphasis on the utilization of Jenkins as the automation server to achieve this goal. It highlights the importance of automation and integration in modern software development.

C3.4 About Central Idea of the Chapter

The central idea of the chapter is to equip readers with the knowledge and practical skills required to implement Continuous Integration using Jenkins. It will serve as a foundational guide for those looking to streamline their software development processes, increase code quality, and accelerate the delivery of software projects through automation and integration.

C3.5 Importance of the Chapter in Subject Syllabus

The chapter on Continuous Integration Using Jenkins is not only relevant but also highly valuable in the subject syllabus. It equips students with practical skills and knowledge that are directly applicable

in real-world software development scenarios, making them more competitive and well-prepared for their future DevOps Engineer careers.

C3.6 Chapter Objective / Outcome

Create and Configure CI/CD Build Jobs using Jenkins.

Outcome : Student will able to Create and Configure CI/CD Build Jobs using Jenkins.

Introduction to CI/CD

Continuous Integration/Continuous Deployment (CI/CD) is a set of practices and tools used in software development to automate the building, testing, and deployment of code changes. CI/CD is aimed at improving the efficiency, quality, and speed of software delivery by breaking down the development process into smaller, manageable increments. Here's an introduction to CI/CD:

Continuous Integration (CI):

- Automation: CI involves the automation of code integration from multiple developers into a shared repository on a frequent basis, typically several times a day.
- Code Quality: Automated tests, code reviews, and static code analysis tools are often used to ensure that the integrated code is of high quality.
- Early Detection of Issues: CI detects integration issues, bugs, and conflicts between different code branches at an early stage, making it easier to address them.
- Immediate Feedback: Developers receive immediate feedback on code changes, allowing them to fix issues before they escalate.
- Consistency: CI promotes consistency in the development process and helps maintain a stable codebase.

Continuous Deployment (CD):

- Automation: CD extends CI by automating the deployment of code changes to various environments, such as development, testing, staging, and production.
- Frequent Releases: CD enables frequent releases of software, allowing new features and bug fixes to be delivered to users rapidly.
- Reduced Manual Intervention: Manual steps in the deployment process are minimized or eliminated, reducing the risk of human error.
- Rollback Capability: CD systems often include rollback mechanisms that allow for quick recovery in case of deployment issues.

CI/CD Pipeline:

- Pipeline Definition: A CI/CD pipeline is defined as a series of automated steps that code changes go through, from integration and testing to deployment.
- Continuous Integration Stage: This stage involves building, testing, and validating code changes.
- Continuous Deployment Stage: Once code changes pass CI, they can be automatically deployed to various environments.
- Monitoring and Feedback: The pipeline includes monitoring and feedback mechanisms to track the status of each stage and provide notifications in case of issues.

Benefits of CI/CD:

- Speed: CI/CD accelerates the software development lifecycle, reducing time-to-market.
- Quality: It improves code quality by automated testing and early issue detection.
- Consistency: CI/CD ensures consistent and reliable builds and deployments.
- Efficiency: It streamlines the development process, reducing manual tasks and errors.
- Collaboration: Teams can collaborate effectively, as CI/CD encourages frequent code integration.
- Risk Reduction: Rollback capabilities and automated testing mitigate deployment risks.
- User Satisfaction: Frequent releases deliver new features and fixes to users faster.

In summary, CI/CD is a crucial aspect of modern software development that emphasizes automation, collaboration, and efficiency. It enables teams to deliver high-quality software more quickly and with greater confidence.

Install & Configure Jenkins

To install and configure Jenkins on Ubuntu, follow these steps:

Installation:

Update System Packages:

Open a terminal window and run the following commands to update the package list and upgrade existing packages:

sudo apt update

sudo apt upgrade

Install Java:

Jenkins requires Java to run. You can install OpenJDK using the following command:

sudo apt install openjdk-11-jdk

Add Jenkins Repository and Key:

Add the Jenkins repository and its GPG key to your system:

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

Add Jenkins Repository:

Add the Jenkins repository to your list of sources:

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Install Jenkins:

Update the package list and install Jenkins:

sudo apt update

sudo apt install jenkins

Start Jenkins:

Start the Jenkins service and enable it to start on boot:

sudo systemctl start jenkins

sudo systemctl enable jenkins

Retrieve Initial Administrator Password:

Jenkins generates an initial password for the admin user. Retrieve it using this command:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Access Jenkins Dashboard:

Open a web browser and navigate to http://localhost:8080 (or the IP address of your server). You will be prompted to enter the initial administrator password.

Configuration:

Unlock Jenkins:

Paste the initial admin password that you retrieved earlier into the Jenkins unlock screen.

Follow the on-screen instructions to complete the setup.

Customize Jenkins:

You can choose to install suggested plugins or select specific plugins based on your requirements during the initial setup.

Create an admin user and set up its credentials.

Jenkins URL:

Confirm or set the Jenkins URL based on your server's domain or IP address.

Start Using Jenkins:

Once configured, you can start using Jenkins for continuous integration and continuous delivery tasks.

That's it! Jenkins is now installed and configured on your Ubuntu server, and you can begin using it for your CI/CD workflows.

Jenkins Architecture Overview

Jenkins follows a distributed architecture that allows for flexibility and scalability. The key components of Jenkins architecture include:

1. Jenkins Master: The Jenkins master server is the core component that controls and manages the entire Jenkins environment. It handles the following tasks:
    a. Scheduling and triggering jobs.
    b. Monitoring and managing agents (nodes/agents).
    c. Storing configurations and job details.
    d. Serving the Jenkins web interface.
2. Jenkins Agents (Nodes): Jenkins agents are worker machines that perform the actual build and deployment tasks. These agents can run on different operating systems and architectures. The master server distributes work to agents and collects results from them. Agents can be set up as:
    a. Master (controller) and agent on the same machine: In this configuration, the master node can also act as a worker node.
    b. Distributed agents: Multiple worker machines (agents) can be connected to the Jenkins master, allowing for parallel execution of jobs on different environments.
3. Job/Project: A job or project in Jenkins represents a task that needs to be executed, such as building a software project, running tests, or deploying an application. These jobs are configured and defined by users in Jenkins.
4. Executor: Executors are execution slots on Jenkins agents that allow jobs to run. Each agent can have one or more executors, depending on the available resources. Jobs are assigned to executors for execution.
5. Workspace: Each executor on a Jenkins agent has its own workspace, which is a directory where a job's files and configurations are stored during its execution. Workspaces are isolated from each other to prevent conflicts.
6. Plugins: Jenkins has a rich ecosystem of plugins that extend its functionality. Plugins can be installed to add features like version control system integration, build tools support, notifications, and more.
7. Jenkins Controller: This is the central management component that controls the overall Jenkins environment. It manages the distribution of jobs to agents, stores configurations, and provides a web-based interface for users.

8. Jenkins Web Interface: Jenkins provides a user-friendly web interface that allows users to configure jobs, view build results, and manage the Jenkins environment. Users can access Jenkins via a web browser.
9. Build History and Artifacts: Jenkins keeps a history of build results, including logs and artifacts generated during job execution. Users can review past builds and access build artifacts for further analysis.
10. Global Configuration: Jenkins allows global configurations for tools, version control systems, and various settings that can be used across multiple jobs.

Jenkins' architecture is highly customizable, enabling users to set up and scale their CI/CD pipelines according to their specific needs. It provides the flexibility to distribute workloads across multiple agents and provides extensive plugin support for integration with various tools and services, making it a popular choice for automating software development processes.

Build Jobs and Configurations

In Jenkins, a "job" or "project" represents a task or process that needs to be executed as part of your Continuous Integration (CI) or Continuous Delivery (CD) pipeline. Creating and configuring jobs is a fundamental aspect of using Jenkins to automate software development and deployment processes. Below are the key components and steps involved in building jobs and configuring them in Jenkins:

1. **Job Creation:**
   - Log in to the Jenkins web interface.
   - Click on "New Item" or "Create New Jobs" to start creating a new job.
   - Enter a name for the job, select the type of job you want to create (e.g., Freestyle project, Pipeline), and click "OK."
2. **General Configuration:**
   - In the job configuration page, you can define general settings for the job.
   - This includes specifying the source code management system (e.g., Git, Subversion) and providing repository details.
   - You can set build triggers, such as polling for changes in the repository or manual triggering by users.
3. **Build Environment:**
   - Configure the environment in which your job will run.
   - This includes specifying the JDK version, setting environment variables, and defining build tools like Maven or Gradle.
   - You can also configure workspace settings, build discard options, and custom build directories.
4. **Build Steps:**

- Define the actual build steps or actions that Jenkins should perform as part of the job.
- Depending on your project requirements, these steps may include compiling code, running tests, packaging artifacts, or deploying applications.
- You can use build steps like "Execute shell" (for running shell commands) or specific build tools steps (e.g., "Invoke Ant" or "Invoke Gradle script").

5. **Post-Build Actions:**
   - After the build steps are completed, you can specify post-build actions.
   - Common post-build actions include archiving build artifacts, sending notifications (e.g., emails), and triggering downstream jobs.
   - You can configure actions based on build results, such as success or failure.
6. **Build Triggers:**
   - Decide how the job should be triggered. Jenkins offers various build triggers, such as:
     - Poll SCM: Periodically check the source code repository for changes and trigger builds if changes are detected.
     - Webhooks: Configure webhooks in your version control system to trigger builds on code commits.
     - Manual Trigger: Allow users to manually trigger builds through the Jenkins web interface.
7. **Authentication and Authorization:**
   - Ensure that the job's configuration includes the necessary authentication and authorization settings to control who can access and execute the job.
   - You can use Jenkins' built-in user management and role-based access control.
8. **Save and Build:**
   - Once you have configured the job settings, save the job configuration.
   - You can manually trigger the job to run immediately or wait for an automatic trigger based on your defined build triggers.
9. **View Build History:**
   - Jenkins maintains a build history for each job, allowing you to view the status, logs, and artifacts of previous builds.
   - You can access this information through the Jenkins web interface.
10. **Job Maintenance:**
    - Periodically review and update job configurations as needed to accommodate changes in your project or build process.
    - Consider using version control for your job configurations to track changes over time.

By creating and configuring jobs in Jenkins, you can automate various tasks related to building, testing, and deploying your software projects. Jenkins provides a highly customizable and extensible platform for building and managing complex CI/CD pipelines.

Jenkins Plugins

Jenkins plugins are essential extensions that enhance the functionality and capabilities of the Jenkins automation server. They allow you to customize and extend Jenkins to support various tools, technologies, and processes. Here's an overview of Jenkins plugins:

1. **What Are Jenkins Plugins?**
    - Jenkins plugins are software modules that can be installed into Jenkins to add new features or integrate Jenkins with other tools and systems.
2. **Types of Jenkins Plugins:**
    - There are thousands of Jenkins plugins available, covering a wide range of categories. Some common types of plugins include:
        - Source Code Management (SCM) plugins: Enable integration with version control systems like Git, Subversion, and Mercurial.
        - Build Tool plugins: Provide support for build tools such as Apache Maven, Gradle, and Ant.
        - Notification plugins: Send notifications and alerts via email, Slack, or other messaging platforms.
        - Deployment plugins: Facilitate deployment to various environments, including cloud services like AWS and Azure.
        - Testing and Quality Assurance plugins: Integrate with testing frameworks and static code analysis tools.
        - Reporting plugins: Generate reports, charts, and visualizations for build and test results.
        - Authentication and Authorization plugins: Enhance security by supporting different authentication methods and access control.
        - Monitoring and Metrics plugins: Monitor Jenkins performance and collect metrics for analysis.
3. **How to Install Jenkins Plugins:**
    - You can install plugins through the Jenkins web interface:
        - Go to "Manage Jenkins" > "Manage Plugins."
        - In the "Available" tab, you can search for and select the plugins you want to install.
        - Click "Install" or "Download now and install after restart" to add the selected plugins.
4. **Plugin Configuration:**
    - After installing a plugin, you may need to configure it to integrate with other tools or customize its behavior.
    - Configuration options vary depending on the plugin's purpose.
5. **Plugin Updates:**

- Jenkins plugins are actively maintained and updated. It's important to keep your plugins up to date to ensure compatibility with Jenkins and other tools.
- Jenkins provides a "Check Now" button in the plugin manager to check for available updates.

6. **Plugin Compatibility:**
   - It's crucial to ensure that installed plugins are compatible with your Jenkins version.
   - Jenkins may display warnings or errors if there are compatibility issues.
7. **Writing Custom Plugins:**
   - If you have specific requirements that are not met by existing plugins, you can write custom Jenkins plugins in Java or other supported languages.
   - Jenkins provides an extensible framework for creating custom plugins.
8. **Managing Plugin Dependencies:**
   - Some plugins may have dependencies on other plugins. Jenkins will automatically manage these dependencies when you install a plugin.
9. **Security Considerations:**
   - Ensure that the plugins you install are from trusted sources.
   - Regularly review and audit installed plugins to maintain the security of your Jenkins environment.

In summary, Jenkins plugins are a fundamental part of Jenkins' flexibility and extensibility. They enable you to tailor Jenkins to your specific CI/CD needs, integrate it with your existing tools, and automate various aspects of software development and delivery. Properly chosen and configured plugins can significantly streamline your CI/CD pipelines and improve productivity.

Jenkins Integration with other Tools

Jenkins integration with other tools is a key aspect of its functionality, enabling seamless automation and orchestration of various stages in the software development and delivery process. Here's an overview of how Jenkins can integrate with other tools:

1. **Source Code Management (SCM) Tools:**
   - Jenkins can integrate with popular SCM tools like Git, Subversion, Mercurial, and others.
   - Integration allows Jenkins to monitor repositories for changes and trigger builds or deployments automatically.

2. **Build Tools:**
    - Jenkins supports integration with build tools like Apache Maven, Gradle, Ant, and more.
    - These integrations enable Jenkins to compile, package, and build applications according to your project's requirements.
3. **Testing Frameworks:**
    - Jenkins can integrate with various testing frameworks, including JUnit, TestNG, Selenium, and others.
    - Test results can be automatically collected and displayed within Jenkins for analysis.
4. **Artifact Repositories:**
    - Artifactory, Nexus, and other artifact repositories can be integrated with Jenkins to store and manage build artifacts.
    - Jenkins can publish and retrieve artifacts from these repositories during the build and deployment process.
5. **Container Orchestration Platforms:**
    - Jenkins can integrate with container orchestration platforms like Kubernetes and Docker Swarm.
    - Integration enables Jenkins to build, push, and deploy containerized applications to container clusters.
6. **Cloud Services:**
    - Jenkins can be integrated with cloud platforms such as AWS, Azure, Google Cloud, and others.
    - Integration allows Jenkins to provision and manage cloud resources, deploy applications, and scale infrastructure.
7. **Continuous Delivery (CD) Tools:**
    - Jenkins can work in conjunction with CD tools like Ansible, Puppet, and Chef.
    - Integration allows for automated configuration management and infrastructure provisioning.
8. **Deployment Tools:**
    - Tools like Ansible, Terraform, and Docker can be integrated with Jenkins for automated deployments.
    - Jenkins pipelines can trigger deployment processes using these tools.
9. **Monitoring and Logging Tools:**
    - Jenkins can integrate with monitoring and logging tools such as Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), and more.
    - Integration allows for real-time monitoring and analysis of application and infrastructure health.
10. **Notification and Collaboration Tools:**
    - Jenkins can send notifications and alerts to collaboration tools like Slack, Microsoft Teams, and email.
    - Team members can be informed about build and deployment statuses.

11. **Static Code Analysis Tools:**
    - Integration with tools like SonarQube and Checkmarx enables Jenkins to perform static code analysis during the build process.
    - Code quality reports can be generated and shared.
12. **Security Scanning Tools:**
    - Jenkins can be integrated with security scanning tools like OWASP ZAP and Nessus.
    - Vulnerability scans can be automated as part of the CI/CD pipeline.
13. **Version Control and Issue Tracking:**
    - Jenkins can interact with version control systems like GitHub, Bitbucket, and GitLab.
    - Integration facilitates tracking of issues, pull requests, and code changes.
14. **Database and Database Migration Tools:**
    - Integration with database tools like Liquibase and Flyway allows Jenkins to manage database schemas and migrations as part of deployments.
15. **Custom Scripts and APIs:**
    - Jenkins can execute custom scripts and interact with external systems via APIs.
    - This flexibility enables integration with virtually any tool or system.
16. **Plugin Ecosystem:**
    - Jenkins offers a vast plugin ecosystem, making it easy to extend its capabilities and integrate with a wide range of tools.

Jenkins' ability to integrate with diverse tools and technologies makes it a powerful automation and orchestration platform for implementing Continuous Integration and Continuous Delivery (CI/CD) pipelines tailored to your specific requirements. This integration streamlines development, testing, and deployment processes, ultimately leading to faster and more reliable software delivery.