Jenkins for Beginners (Introduction)

Understand with Real Life Examples

DigiTalk

https://digitalksystems.com/

Reach us at digitalk.fmw@gmail.com
DigiTalk Channel: https://www.youtube.com/channel/UCCGTnI9vvF_ETMhGUXGdFWw
Playlists: https://www.youtube.com/@digitalk.middleware/playlists
Weblogic Server Architecture: https://youtu.be/gNqeIfLjUqw

DigiTalk Udemy Courses and Coupon Code

SOA Suite Administration
https://www.udemy.com/course/mastering-oracle-soa-suite-12c-administration/?couponCode=739A60915F86847014EB
Coupon Code: 739A60915F86847014EB

JBoss 8 Administration
https://www.udemy.com/course/mastering-jboss-eap-8-administration-from-intro-to-advanced/?couponCode=BF65EB008CFE16686BD2
Coupon Code:BF65EB008CFE16686BD2

OHS Administration
https://www.udemy.com/course/mastering-oracle-ohs-http-12c-web-server-administration/?couponCode=8E990556B21AF3E1A316
Coupon Code: 8E990556B21AF3E1A316

Weblogic Server Administration
https://www.udemy.com/course/oracle-weblogic-server-12c-and-14c-administration/?couponCode=87BC1314AC7690FD5294
Coupon Code:87BC1314AC7690FD5294

You can write us on digitalk.fmw@gmail.com if coupon code expired.

Introduction to Jenkins

Jenkins is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) in software development. It helps automate various stages of the software development lifecycle, such as building, testing, and deploying code. Jenkins is widely used in DevOps practices to streamline the software development process, ensure code quality, and speed up the release cycle.

Confused in CI & CD. Let's understand

Continuous Delivery (CD) and Continuous Integration

Concept: Continuous Delivery (CD) extends CI by automating the process of deploying code changes to production or staging environments. The goal is to ensure that code can be deployed at any time with minimal manual intervention.

Basic Example: Baking and Selling the Cake

Scenario: Once the cake mixture (code) is successfully tested and meets quality standards, the bakery (CD pipeline) moves on to preparing the cake for sale.

Steps in CD:

Preparation: The bakery prepares the cake for delivery (deploys the application to a staging environment) to ensure it meets all customer requirements.

Final Testing: The cake undergoes a final quality check (user acceptance testing) to ensure it's ready for sale.

Approval: If everything is satisfactory, the bakery (CD pipeline) approves the cake for sale (deploys the application to production).

Delivery: The cake is packaged and delivered to customers (code is deployed to live servers).

Benefits of CD:

Faster Release Cycles: Automating deployment allows for quicker release of new features and bug fixes.

Consistent and Reliable Deployments: Automated deployment processes reduce the risk of human error and ensure consistency across environments.

Integrated Example: A Simple Web Application

Scenario: You're developing a simple web application with a team of developers. You decide to implement CI and CD to streamline your development and deployment processes.

CI Process

Developers Commit Code: Each developer commits their changes to the shared code repository (e.g., GitHub).

Automated Build: A CI server (e.g., Jenkins) detects the new commits and triggers an automated build process.

Automated Testing: After building, the CI server runs a suite of automated tests (unit tests, integration tests) to verify the changes.

Feedback: If tests fail, developers receive notifications to fix the issues. If tests pass, the build is considered successful.

CD Process:

Deploy to Staging: Once the build is successful, the CD pipeline automatically deploys the application to a staging environment for further testing.

User Acceptance Testing: Testers review the application in the staging environment to ensure it meets requirements.

Approval: If the application passes user acceptance testing, the CD pipeline promotes it to production.

Production Deployment: The application is deployed to the production environment, making it available to users.

Summary

- CI focuses on the early integration of code changes and automated testing to maintain code quality and detect issues early.
- CD extends this by automating the deployment process, ensuring that code changes can be released to production reliably and frequently.
- By using CI and CD, development teams can streamline their workflows, enhance collaboration, and deliver high-quality software more efficiently.

Key Features of Jenkins:

- **Automation:** Jenkins automates repetitive tasks such as building and testing code, which improves efficiency and reduces manual effort.
- **Integration:** It integrates with a wide range of tools and technologies, including version control systems, build tools, and deployment platforms.
- **Extensibility:** Jenkins is highly extensible through plugins, allowing it to support various functionalities and workflows.
- **Scalability:** It can handle large-scale projects by distributing workloads across multiple build agents (executors).

Components of Jenkins and Real-Life Examples

1. Jenkins Server (Bakery)

Concept: The Jenkins server is the central hub where all automation tasks are managed and executed. It orchestrates various processes and oversees the overall operation of Jenkins.

Real-Life Example: Think of the Jenkins server as the central bakery where all the baking activities are organized. This bakery oversees the entire baking process, from ingredient preparation to final packaging. It ensures that every step is completed efficiently and that all products meet quality standards.

2. Build Executors (Bakers)

Concept: Build executors are the worker nodes that execute tasks defined in Jenkins jobs. Each executor runs a specific job or a set of jobs.

Real-Life Example: In our bakery, build executors are like bakers who are responsible for different baking tasks. Some bakers might focus on cakes, while others handle cookies or bread. Each baker (executor) has specific tools and recipes to follow, ensuring that each type of baked good is prepared correctly.

3. Jobs (Recipes)

Concept: Jobs in Jenkins are tasks or processes that define what needs to be done. They can include building code, running tests, or deploying applications.

Real-Life Example: Jobs are like recipes in the bakery. Each recipe outlines the steps needed to produce a specific baked good, such as a chocolate cake or a batch of muffins. Just as recipes provide detailed instructions for baking, Jenkins jobs specify the tasks and commands to execute in the automation process.

4. Pipelines (Order Management System)

Concept: Pipelines define a sequence of stages and steps for automating the software delivery process. They represent the workflow of tasks from code commit to deployment.

Real-Life Example: The order management system in the bakery is analogous to Jenkins pipelines. This system tracks orders, organizes tasks (baking, decorating, packaging), and ensures that each step is completed in the correct sequence to fulfill customer orders.

5. Source Code Repository (Ingredient Inventory)

Concept: A source code repository is where the source code is stored and managed. Jenkins pulls code from this repository to build and test it.

Real-Life Example: The ingredient inventory in the bakery is similar to the source code repository. Just as the bakery keeps an inventory of ingredients needed for baking, the source code repository holds the code required for software development. Jenkins accesses this repository to retrieve the latest code for processing.

6. Artifact Repository (Packaging Station)

Concept: An artifact repository stores build artifacts, such as compiled code or binaries, produced by Jenkins jobs.

Real-Life Example: The packaging station in the bakery is where baked goods are packaged and prepared for delivery. Similarly, the artifact repository stores the final products (build artifacts) after Jenkins has completed the build process. This ensures that the products are organized and ready for deployment.

7. Continuous Integration (Quality Control Team)

Concept: Continuous Integration (CI) is a practice where code changes are continuously integrated and tested to ensure they do not introduce defects.

Real-Life Example: The quality control team in the bakery inspects each batch of baked goods to ensure they meet quality standards. This team checks for taste, appearance, and consistency. In Jenkins, CI performs automated tests and checks on code changes to maintain code quality and detect issues early.

8. Plugins (Automation Tools)

Concept: Plugins extend Jenkins' functionality, allowing it to integrate with various tools and services.

Real-Life Example: Automation tools in the bakery, such as mixers and ovens, enhance the efficiency of the baking process. Similarly, Jenkins plugins add extra features and capabilities, such as integrating with version control systems, managing build triggers, or generating reports.

9. Administrators (Shift Supervisors)

Concept: Administrators manage the Jenkins server, configure settings, and handle any issues that arise.

Real-Life Example: Shift supervisors in the bakery oversee operations, manage staff, and ensure that the baking process runs smoothly. In Jenkins, administrators perform similar roles by configuring the server, managing user access, and troubleshooting issues.

Detailed Example: A Jenkins Pipeline for Deploying a Web Application

Scenario

Imagine you are developing a web application that needs to be built, tested, and deployed automatically every time a developer commits code changes to the repository. Here's how Jenkins handles this process:

1. Source Code Commit

A developer makes changes to the web application code and commits these changes to the source code repository (e.g., Git).

2. Triggering the Pipeline

The Jenkins server is configured to monitor the source code repository. When a new commit is detected, Jenkins triggers a pipeline job.

3. Pipeline Stages

a. Build Stage:

ask: Jenkins pulls the latest code from the repository and runs a build job.

Process: The build executor (baker) compiles the code, packages it into an artifact (e.g., a WAR file), and stores it in the artifact repository (packaging station).

b. Test Stage:

Task: Jenkins runs automated tests on the built artifact.

Process: The quality control team (CI) runs unit tests, integration tests, and performs code quality checks. Any test failures are reported, and the pipeline may be halted if issues are detected.

c. Deploy Stage:

Task: Jenkins deploys the tested artifact to a staging environment.

Process: The artifact is deployed to a staging server where further testing or user acceptance testing can occur. This deployment is managed by Jenkins, which ensures that the application is correctly installed and configured.

d. Approval Stage (Optional):

Task: If configured, Jenkins can include a manual approval stage before deploying to production.

Process: An administrator or stakeholder reviews the deployment and approves it before moving to production.

e. Production Deployment:

Task: Jenkins deploys the application to the production environment.

Process: The application is installed on production servers, making it available to end-users. Jenkins ensures that the deployment is completed successfully and monitors the application for any issues.

4. Feedback and Monitoring

Jenkins provides feedback on the build, test, and deployment processes through logs and reports. Administrators and developers can review this feedback to address any issues or make improvements.

5. Final Steps

Once the deployment is successful, Jenkins may trigger additional jobs or pipelines for related tasks, such as notifying team members or updating documentation.

Summary

In this detailed example, Jenkins acts as a highly efficient bakery, automating the entire process of building, testing, and deploying a web application. From handling source code commits to managing deployment pipelines, Jenkins ensures that each step is completed efficiently and reliably. Just as a bakery uses recipes, quality control, and packaging to produce high-quality baked goods, Jenkins uses jobs, pipelines, and automation tools to deliver high-quality software consistently.

Detailed Example: Using Jenkins in an IT Environment

Scenario

Imagine you are working on a web application project in a large IT environment. Your development team follows best practices for continuous integration and continuous delivery (CI/CD). You decide to use Jenkins to automate the build, test, and deployment processes. The environment includes various tools and systems such as a version control system, build tools, testing frameworks, and deployment platforms.

Components and Integrations

Version Control System (VCS)

Tool: GitHub

Role: Stores the source code and tracks changes. Developers commit code changes to the GitHub repository.

Integration with Jenkins:

Webhooks: Jenkins is configured with webhooks to trigger jobs whenever there's a new commit or pull request in the GitHub repository. This ensures that Jenkins starts the CI/CD pipeline automatically when code changes are detected.

Jenkins Server

Tool: Jenkins

Role: Orchestrates the CI/CD pipeline. Manages jobs, builds, tests, and deployments.

Configuration:

Jenkins Plugins: Various plugins are installed to integrate Jenkins with other tools, such as GitHub for source code management, Maven for build automation, and Docker for containerization.

Build Tool

Tool: Apache Maven

Role: Handles the build process of the web application, including compiling code, packaging artifacts, and managing dependencies.

Integration with Jenkins:

Maven Plugin: The Jenkins Maven plugin is used to execute Maven goals (e.g., compile, test, package) as part of the build job. Jenkins pulls the latest code from GitHub and triggers Maven to build the application.

Testing Framework

Tool: JUnit (for Java applications) or Jest (for JavaScript applications)

Role: Executes automated tests to verify the correctness of the code. Ensures that new changes do not break existing functionality.

Integration with Jenkins:

JUnit Plugin: For Java applications, Jenkins uses the JUnit plugin to report test results. After the build, Jenkins collects test results from JUnit and displays them in the Jenkins dashboard.

Jest Plugin: For JavaScript applications, Jenkins uses the Jest plugin to run tests and report results.

Artifact Repository

Tool: Nexus Repository or Artifactory

Role: Stores built artifacts (e.g., binaries, libraries) that are used for deployment.

Integration with Jenkins:

Artifact Plugin: Jenkins is configured to publish build artifacts to the artifact repository. After a successful build, Jenkins uploads the generated artifacts to Nexus or Artifactory.

Deployment Platform

Tool: Kubernetes (for container orchestration) or AWS Elastic Beanstalk (for deploying web applications)

Role: Deploys the application to staging or production environments.

Integration with Jenkins:

Kubernetes Plugin: Jenkins uses the Kubernetes plugin to deploy Docker containers to a Kubernetes cluster. Jenkins triggers deployment jobs that use Kubernetes configurations to deploy the application.

AWS Plugin: For AWS Elastic Beanstalk, Jenkins uses the AWS plugin to deploy the application. Jenkins uses AWS credentials to interact with Elastic Beanstalk and deploy the application.

Monitoring and Notifications

Tool: Slack (for team notifications) and Prometheus/Grafana (for monitoring)

Role: Provides real-time feedback and monitors application performance.

Integration with Jenkins:

Slack Plugin: Jenkins is configured with the Slack plugin to send notifications about build status, test results, and deployment progress to a Slack channel. This keeps the team informed about the CI/CD pipeline.

Prometheus/Grafana: Jenkins can be integrated with Prometheus and Grafana to monitor application performance and health metrics. Jenkins jobs can be configured to send metrics to Prometheus, which are then visualized in Grafana dashboards.

End-to-End CI/CD Pipeline Example

1. Code Commit:

- Developers commit code changes to the GitHub repository.
- A webhook triggers Jenkins to start the CI pipeline.

2. Build:

- Jenkins pulls the latest code from GitHub.
- Jenkins uses Maven to compile the code and package it into a deployable artifact (e.g., a JAR or WAR file).

3. Test:

- Jenkins runs automated tests using JUnit (or Jest).
- Test results are collected and reported in Jenkins.
- If tests fail, Jenkins notifies the team via Slack and stops the pipeline.

4. Artifact Publishing:

- After a successful build and test, Jenkins publishes the built artifact to the artifact repository (e.g., Nexus).

5. Deployment:

Jenkins triggers a deployment job.

- **For Kubernetes:** Jenkins uses the Kubernetes plugin to deploy the Docker container to the Kubernetes cluster.
- **For AWS Elastic Beanstalk:** Jenkins uses the AWS plugin to deploy the application to Elastic Beanstalk.

6. Monitoring and Notifications:

- Jenkins sends deployment status updates to a Slack channel.
- Prometheus collects performance metrics, and Grafana visualizes them on dashboards.

7. Feedback:

- The team monitors Slack notifications and Grafana dashboards to ensure the deployment is successful and the application is performing well.

Summary

In this detailed example, Jenkins acts as the automation server that coordinates various aspects of the CI/CD pipeline. It integrates with tools for source control, build automation, testing, artifact management, deployment, and monitoring. By orchestrating these components, Jenkins helps streamline the development process, ensuring that code changes are integrated, tested, and deployed efficiently and reliably.

DISCLAIMER AND CONSENT

This document is being provided by DigiTalk as part of its effort to assist users in understanding and working with Jenkins. While every effort has been made to ensure the accuracy and reliability of the information presented in this document, there is a possibility of typographical errors or inaccuracies. DigiTalk does not guarantee the correctness or completeness of the content provided in this document.

Users of this document are encouraged to cross-reference the information presented here with official documentation available on their website or other authoritative sources. Any discrepancies or inaccuracies found in this document should be reported to us at digitalk.fmw@gmail.com.

By using this document, you acknowledge and consent to the following:

This document is not officially endorsed or verified by any other third party organization..

The Company makes no claims or guarantees about the accuracy or suitability of the information contained in this document.

Users are responsible for verifying and validating any information presented here for their specific use case.

DigiTalk disclaims any liability for any errors, omissions, or damages that may result from the use of this document.

If you discover any inaccuracies or errors in this document, please report them to digitalk.fmw@gmail.com, and the Company will endeavor to correct them as necessary.

This consent statement is provided to ensure transparency and understanding of the limitations of the information contained in this document. By using this document, you agree to abide by the terms and conditions outlined herein.