Jenkins

Introduction to Jenkins :

Jenkins is an open-source automation server that enables the automation of software development processes. It provides a platform for building, testing, and deploying applications in an efficient and automated manner. Jenkins is widely used in the industry due to its flexibility, extensibility, and large community support. It supports various programming languages and integrates with popular tools and frameworks.

Continuous Integration and Delivery (CI/CD): CI/CD is a software development practice that involves automating the process of integrating code changes, building applications, running tests, and deploying them to production. The primary goals of CI/CD are to increase productivity, improve software quality, and enable rapid and frequent releases. Continuous Integration (CI) focuses on integrating code changes frequently, while Continuous Delivery (CD) ensures that applications can be released at any time.

CI/CD Pipeline Overview: A CI/CD pipeline is a series of automated steps that code goes through from version control to production deployment. It consists of multiple stages, each representing a specific task or set of tasks. Common stages include code compilation, unit testing, code analysis, artifact creation, and deployment. Jenkins allows you to define and manage these stages, ensuring that the entire process is automated, consistent, and reproducible.

Installation of Jenkins

Best site: https://www.jenkins.io/doc/book/installing/linux/

- Update your ubuntu

  Sudo apt update

- Install java

  Sudo apt install Openjdk-17-jre

- Check java installation

  Java -version

- Add keys

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io 2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

```
echo deb [signed-by=/usr/share/keyrings/jenkins keyring.asc] \
https://pkg.jenkins.io/debian binary/ | sudo tee \ /etc/apt/sources.list.d/jenkins.list >
/dev/null

sudo apt-get update

sudo apt-get install Jenkins
```

Important points for Jenkins Declarative Pipelines :

Introduction to Jenkins Declarative Pipelines:

- Overview of Jenkins and its role in continuous integration and delivery.
- Explanation of declarative pipelines and their benefits.
    - Declarative pipelines provide a more structured and simplified approach to defining pipelines in Jenkins.
    - They offer better readability, maintainability, and reusability compared to scripted pipelines.

Structure of a Declarative Pipeline: Stages and steps:

- Defining the different stages of the pipeline and the tasks performed in each stage.
    - Stages represent logical divisions in the pipeline, such as build, test, and deploy.
    - Steps define the individual tasks within each stage, such as code checkout, compilation, testing, and deployment.
- Agent: Specifying the execution environment for the pipeline, such as a specific Jenkins node or a container.
    - Agents allow pipelines to run on designated resources, providing control over where each stage or step executes

Syntax and Configuration:

- Declarative pipeline syntax: Understanding the syntax used to define pipelines in Jenkins.
    - Discuss the use of the pipeline block and its sections (agent, stages, steps, etc.).
    - Explain defining and using variables, loops, conditionals, and function calls within the pipeline script.
- Pipeline script: Defining the pipeline stages, steps, and other configuration options using the script.
    - Explain how to define stages using the stage block and steps using various built-in and custom steps.

Building Blocks of Declarative Pipelines:

- Steps: Overview of commonly used built-in steps for actions like code checkout, building, testing, and deployment.
    - Discuss steps like git, sh, npm, docker, JUnit, archive, deploy, etc.
- Environment variables: Using environment variables to store and access information throughout the pipeline.
    - Explain how to define and use environment variables within the pipeline script.
    - Discuss the use of environment variables for storing credentials, build numbers, and other useful data.
- Parameters: Defining parameters to make pipelines more flexible and customizable.
    - Discuss different parameter types (e.g., string, boolean, choice) and their usage.
    - Explain how to prompt users for input and use parameter values within the pipeline script.
- Post actions: Configuring post-build actions, such as sending notifications or publishing reports.
    - Explain how to use the post block to define actions that execute after the pipeline completes.
    - Discuss actions like always, success, failure, unstable, email, junit, slack, etc.

Pipeline Visualization and Logs:

- Blue Ocean: Introduction to the Blue Ocean plugin for Jenkins, providing a graphical visualization of pipeline execution.
    - Discuss the benefits of using Blue Ocean to visualize pipelines and monitor their progress.
    - Explain how to navigate the Blue Ocean interface and interpret the pipeline visualization.
- Pipeline logs: Understanding how to view and analyze the logs generated by pipeline executions.
    - Explain how to access and review console output and log files generated during pipeline execution.
    - Discuss strategies for troubleshooting and debugging pipeline issues using the logs/

Best Practices and tips for Jenkins Declarative Pipelines

- Keeping pipelines modular: Splitting complex pipelines into reusable components.

    - Discuss techniques for defining shared libraries, functions, and templates to promote reusability.
    - Highlight the benefits of modular pipelines in terms of maintainability and code organization.
    - Interview question: How would you structure a pipeline to promote modularity and code reuse?

- Version control and code review: Leveraging version control systems and performing code reviews for pipeline scripts.
    - Discuss the importance of storing pipeline scripts in a version control system for change tracking.
    - Explain the benefits of code reviews to ensure best practices, identify errors, and improve pipeline quality.
    - Interview question: How would you integrate Jenkins declarative pipelines with a version control system like Git?

- Error handling and recovery: Implementing error handling mechanisms and retry strategies in pipelines.
    - Discuss techniques like try-catch blocks, error handling steps (e.g., catchError, error, timeout), and retries.
    - Highlight the importance of handling errors gracefully and recovering from failures in the pipeline.
    - Interview question: How would you handle a failing step in a pipeline and ensure proper error reporting?

- Testing pipelines: Strategies for testing and validating pipeline changes before production deployment.
    - Discuss techniques like unit testing pipeline components, running pipeline simulations, and using test environments.
    - Explain the benefits of testing pipelines to catch errors, validate changes, and ensure expected behaviour.
    - Interview question: How would you approach testing a complex Jenkins declarative pipeline?