1. Introduction to CI & CD

* Continuous Integration (CI):
  + Frequent merging of code changes into a central repository.
  + Automated builds and tests triggered upon each merge.
  + Goal: Early detection of integration errors.
* Continuous Delivery (CD):
  + Extends CI by automating the release process.
  + Code is always in a deployable state.
  + Goal: Frequent, reliable software releases.

2. Introduction to Jenkins

* Open-source automation server.
* Used for implementing CI/CD pipelines.
* Extensible through plugins.
* Automates build, test, and deployment processes.

3. Advantages of CI/CD

* Faster feedback loops.
* Reduced integration risks.
* Improved code quality.
* Increased deployment frequency.
* Enhanced team collaboration.

4. Use Cases of CI/CD

* Software development.
* Web application deployment.
* Mobile app development.
* Infrastructure as code (IaC).
* Data science pipelines.

5. Jenkins Installation

* VM Installation:
  + Install Jenkins on a virtual machine for dedicated resources.
  + Provides greater control over the environment.
* Container Installation (Docker):
  + Use Docker to run Jenkins in a container.
  + Simplified deployment and portability.
  + Scalability.
* Setup Options:
  + Configure security settings, user management, and plugin installation.
  + Setup Java path.
  + Setup initial admin user.

6. Jenkins Architecture

* Master-Slave (or Master-Agent) Architecture:
  + Master: Controls and schedules builds.
  + Agents: Execute build jobs.
  + Enables distributed builds and parallel execution.

7. Jenkins UI Description

* Dashboard: Overview of build status, jobs, and system information.
* Job Configuration: Settings for individual build jobs.
* Build History: Records of past build executions.
* Plugin Manager: Interface for installing and managing plugins.
* System configuration: tools, paths, security.

8. Configuring Jenkins Jobs

* Freestyle Projects: General-purpose jobs with flexible configurations.
* Pipeline Projects: Jobs defined using Groovy scripts for complex workflows.
* Source code management configuration.
* Build triggers.
* Build steps.
* Post-build actions.

9. Plugins

* Introduction: Extend Jenkins' functionality with plugins.
* Installation: Install plugins through the Plugin Manager.
* Commonly Used Plugins:
  + Git: Integration with Git repositories.
  + Maven: Build automation for Maven projects.
  + JUnit: Test result reporting.
  + Docker: Docker container integration.
  + Pipeline: Used for creating CI/CD pipelines.

10. Features and Functionality

* Build Triggers:
  + SCM polling.
  + Scheduled builds.
  + Webhooks.
* Build Steps:
  + Execute shell commands.
  + Invoke Maven goals.
  + Run scripts.
* Post-Build Actions:
  + Send email notifications.
  + Archive artifacts.
  + Deploy to servers.

11. Configuring Jenkins with Java, Git, and Maven

* Java: Configure the JDK path in Jenkins' global tool configuration.
* Git: Install the Git plugin and configure Git credentials.
* Maven: Install the Maven Integration plugin and configure Maven installations.
* Configure Maven settings files.
* Setup repository url's.

1. Jenkins Build and Jenkins Workspace

* Jenkins Build:
  + A single execution of a Jenkins job.
  + Each build has a unique number.
  + Represents a specific instance of the automated process.
* Jenkins Workspace:
  + A directory on the Jenkins agent (or master) where the build is executed.
  + Contains the source code, build artifacts, and temporary files.
  + Cleaned or reused between builds based on job configuration.

2. Jenkins Directory Structure

* JENKINS\_HOME: The root directory for Jenkins data.
  + jobs/: Stores job configurations.
  + plugins/: Contains installed plugins.
  + users/: User data.
  + workspace/: where the workspaces are stored.
  + secrets/: stores sensitive data.
  + config.xml: main jenkins configuration file.

3. Webhooks and Auto-Detecting Changes

* Webhooks:
  + HTTP callbacks triggered by events in source code repositories (e.g., Git).
  + Jenkins can be configured to receive webhooks, automatically triggering builds.
  + Enables real-time CI.
* Auto-Detecting Changes:
  + Jenkins can poll source code repositories for changes at regular intervals.
  + If changes are detected, a build is triggered.
  + This is less realtime than webhooks, but still very useful.

4. Maven Build Jobs with Options

* Maven Build Jobs:
  + Specifically designed for building Maven projects.
  + Integrates with Maven's build lifecycle.
  + Options:
    - Maven goals (e.g., clean, install, deploy).
    - Settings files.
    - Profiles.
    - Passing properties.

5. Build Jobs

* Creating Freestyle Build Jobs:
  + Highly flexible jobs for various tasks.
  + Configurable build steps, triggers, and post-build actions.
* Build Triggers and Build Steps:
  + Build Triggers:
    - SCM polling.
    - Scheduled builds (cron).
    - Upstream/downstream job triggers.
    - Webhooks.
  + Build Steps:
    - Execute shell commands.
    - Invoke Maven goals.
    - Run scripts (e.g., Groovy).
    - Run batch commands.
* Pre and Post Build Actions:
  + Pre-build actions:
    - Setup environment variables.
    - Run scripts before the build.
  + Post-build actions:
    - Archive artifacts.
    - Send notifications.
    - Deploy artifacts.
    - Run scripts after the build.

6. Properties and Properties File Description

* Properties:
  + Variables used in build jobs.
  + Can be defined in the job configuration or in properties files.
* Properties Files:
  + External files containing key-value pairs.
  + Used to manage configuration settings.
  + Useful for parameterized builds.

7. Executing Build Jobs and Identification of Success and Failures

* Executing Build Jobs:
  + Manual triggering.
  + Automatic triggering (triggers).
* Success and Failures:
  + Build status (success, failure, unstable).
  + Console output (build logs).
  + Test results.

8. Parameterized Builds

* Parameterized Builds:
  + Jobs that accept input parameters.
  + Allows for dynamic configuration.
  + Use cases:
    - Deploying to different environments.
    - Running tests with different configurations.

9. Distributed Builds

* Distributed Builds:
  + Using multiple Jenkins agents to execute builds.
  + Improves build performance and scalability.
  + Master/Agent architecture.

10. Failure and Success Email Notifications

* Email Notifications:
  + Jenkins can send email notifications for build events.
  + Configurable triggers (e.g., failure, success).
  + Customizable email content.
  + Requires email server configuration.

1. Security in Jenkins

* Importance:
  + Jenkins handles sensitive data (credentials, code).
  + Requires robust security measures to prevent unauthorized access.
* Authentication and User Levels:
  + Authentication: Verifying user identity (e.g., username/password, LDAP, SAML).
  + User Levels: Defining user roles and permissions (e.g., administrator, developer, read-only).
* Access Types and Administration of Access:
  + Access Control: Restricting access to Jenkins resources (jobs, nodes, plugins).
  + Authorization Strategies:
    - Matrix-based security.
    - Role-based access control (RBAC).
  + Credential Management: Securely storing and managing credentials.
  + Protecting against Cross Site Request Forgery(CSRF) attacks.

2. Jenkins Maintenance

* Regular Maintenance:
  + Plugin updates.
  + System resource monitoring.
  + Log file management.
  + Workspace cleanup.
* Maintenance and Troubleshooting:
  + Identifying and resolving issues.
  + Monitoring system health.
  + Keeping the Java Virtual Machine(JVM) up to date.
* Filesystem:
  + Monitoring disk space usage.
  + Managing JENKINS\_HOME directory.
  + Cleanup of old build artifacts.
* Troubleshooting:
  + Analyzing build logs.
  + Debugging plugin issues.
  + Investigating system errors.
  + Checking the system logs.

3. Jenkins Setup Backup and Recovery

* Backup Strategy:
  + Regular backups of JENKINS\_HOME.
  + Automated backups (e.g., cron jobs).
  + Backups of configuration files.
* Recovery Process:
  + Restoring backups to a new Jenkins instance.
  + Testing recovery procedures.
* Disaster Recovery:
  + Having a plan to restore Jenkins in case of a server failure.

4. Jobs Backup and Recovery

* Job Configuration Backup:
  + Individual job configurations are stored as XML files.
  + These files can be backed up separately.
  + Jenkins also has plugins that aid in backing up jobs.
* Job Recovery:
  + Restoring job configurations from backups.
  + Copying job directories.

5. Migrating Jenkins from One Server to Another

* Migration Methods:
  + Copying JENKINS\_HOME to the new server.
  + Using Jenkins backup and restore plugins.
  + Containerized migration (Docker).
* Migration Steps:
  + Install Jenkins on the new server.
  + Copy or restore JENKINS\_HOME.
  + Verify plugin compatibility.
  + Test build jobs.
  + DNS changes if needed.
  + Testing of security configurations.
* Considerations:
  + Operating system compatibility.
  + Java version compatibility.
  + Plugin compatibility.
  + Network configurations.