5. Introduction to CI/CD

Overview:

- stands for continuous integration and continuous delivery/deployment
- aims to streamline and accelerate the software development lifecycle
- **Continuous integration (CI)** refers to the practice of automatically and frequently integrating code changes into a shared source code repository.
- **Continuous delivery and/or deployment (CD)** is a 2-part process that refers to the integration, testing, and delivery of code changes.
- Continuous delivery stops short of automatic production deployment,
- Continuous deployment automatically releases the updates into the production environment.

- these connected practices are often referred to as a "CI/CD pipeline"

- supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach.

Importance of CI/CD:

- **Avoids Bugs and Code Failures:**
  CI/CD helps identify bugs and issues early in the development process, reducing the chances of code failures in production

- **Maintains Continuous Software Development:**
  Ensures a continuous cycle of development, integration, testing, and deployment, enabling faster updates and improvements.

- **Decreases Complexity in Large Projects:**
  As applications grow, CI/CD helps manage and reduce complexity, making it easier to handle larger codebases.

- **Increases Efficiency:**
  Automates repetitive tasks, allowing developers to focus on writing code and improving features, thereby increasing overall efficiency.

- **Streamlines Workflows:**

Creates a smooth and streamlined workflow from code commit to production deployment, enhancing collaboration and productivity.

- **Minimizes Downtime:**
  By automating the deployment process, CI/CD reduces the risk of human error, leading to minimized downtime during code releases.

- **Faster Code Releases:**
  Automates the release process, enabling faster and more frequent code releases, which helps in responding quickly to market demands and user feedback.

- **Improves Code Quality:**
  Continuous testing and integration ensure that only high-quality code is deployed, maintaining the stability and reliability of the application.

- **Enhances User Feedback Integration:**
  Facilitates quick incorporation of user feedback, leading to more user-centric improvements and higher customer satisfaction.

- **Positive Outcomes for End Users:**
  Faster updates, better quality, and more reliable software led to positive experiences for end users, resulting in greater customer satisfaction.

Continuous Integration:

- The "CI" in CI/CD always refers to continuous integration
- Facilitates frequent merging of code changes back to a shared branch or trunk.
- Triggers automated testing steps to ensure the reliability of merged code changes.
- Supports multiple developers working simultaneously on different features of the same application.
- Eliminates the need for a manual, time-intensive process of merging all branching source code together on a designated "merge day."
- Reduces the chances of code conflicts that arise when developers make isolated changes that may clash with others.
- Encourages the use of consistent development environments, often through cloud-based IDEs, to minimize conflicts from personalized local setups.
- Addresses the problem of having too many branches in development simultaneously that might conflict with each other.
- Ensures that once a developer’s changes are merged, they are validated by automatically building the application and running automated tests.
- Validates changes through unit and integration tests, ensuring changes haven’t broken the application.
- Enables quicker bug detection and resolution, as conflicts between new and existing code are discovered and fixed promptly.

CD in the CI/CD:

- The "CD" in CI/CD refers to continuous delivery and/or continuous deployment
- related concepts that sometimes get used interchangeably
- Both are about automating further stages of the pipeline, but they're sometimes used separately to illustrate just how much automation is happening
- The choice between continuous delivery and continuous deployment depends on the risk tolerance and specific needs of the development teams and operations teams.

Continuous Delivery:

- Continuous delivery automates the release of validated code to a repository after the automation of builds and unit and integration testing in CI.
- An effective continuous delivery process requires CI to be already built into the development pipeline.
- Every stage from code changes to delivery of production-ready builds involves test automation and code release automation.
- At the end of the process, the operations team can swiftly deploy an app to production.
- Developer changes to an application are automatically bug-tested and uploaded to a repository (like GitHub or a container registry).
- The changes can then be deployed to a live production environment by the operations team.
- **Purpose:**
    - Addresses poor visibility and communication between dev and business teams.
    - Ensures that the codebase is always ready for deployment to a production environment.
    - Aims to minimize the effort required to deploy new code.

Continuous Deployment:

- Continuous deployment is the final stage of a mature CI/CD pipeline and refers to automating the release of a developer's changes from the repository to production, where it is usable by customers.
- It extends continuous delivery by automating the release of code changes to production.
- Addresses the problem of overloading operations teams with manual processes.
- Automates the next stage in the pipeline after continuous delivery.
- Allows a developer's change to go live within minutes of writing it, assuming it passes automated testing.
- Facilitates continuous incorporation of user feedback.
- Makes the deployment process less risky by releasing changes to apps in small pieces rather than all at once.
- Easier to manage incremental updates.
- Requires well-designed test automation due to the absence of a manual gate before production.
- Needs significant upfront investment to write automated tests for various testing and release stages in the CI/CD pipeline.

CI/CD vs DevOps:

1. **CI/CD and DevOps Relationship**:
    - CI/CD is an essential part of the DevOps methodology.
    - Both aim to foster collaboration between development and operations teams.

2. **Focus on Automation**:
    - CI/CD and DevOps focus on automating processes of code integration.
    - They speed up the process of moving an idea (new feature, enhancement request, or bug fix) from development to deployment in a production environment.

3. **Value Delivery**:
    - The goal is to provide value to the user by ensuring quicker and more reliable deployment.

4. **Security Integration**:
    - In DevOps, security is a shared responsibility integrated end-to-end.
    - This mindset has led to the term "DevSecOps" to emphasize the need for a security foundation in DevOps initiatives.

5. **DevSecOps Definition**:
    - DevSecOps (development, security, and operations) integrates security as a shared responsibility throughout the entire IT lifecycle.
    - It is an approach to culture, automation, and platform design focused on security.

6. **Cultural and Practical Integration**:
    - DevOps and DevSecOps emphasize the cultural and practical integration of development, security, and operations teams to enhance collaboration and efficiency.

CI/CD Security:

- **Purpose**: CI/CD security safeguards code pipelines with automated checks and testing to prevent vulnerabilities in software delivery.

- **Security Integration**:

  Incorporates security into the pipeline using methods like shift left and shift right security.

  Protects code from attacks, prevents data leaks, ensures compliance with policies, and guarantees quality assurance.

- **Risks of Rapid Development and Deployment**:

  Exposure of sensitive data to outside sources.

Use of insecure code or third-party components.

Unauthorized access to source code repositories or build tools.

- **Vulnerability Management**:

  Identifies and mitigates vulnerabilities throughout the software development cycle.

  Ensures code changes are thoroughly tested and adhere to security standards before deployment to production.

CI/CD Tools:

- CI/CD tools can help a team automate their development, deployment, and testing.
- Some tools specifically handle the integration (CI) side
- Some manage development and deployment (CD)
- Others specialize in continuous testing or related functions.
- Tekton Pipelines is a CI/CD framework for Kubernetes platforms that provides a standard cloud-native CI/CD experience with containers.
- Other open source CI/CD tools you may wish to investigate include:
  - **Jenkins**, designed to handle anything from a simple CI server to a complete CD hub
  - **Spinnaker**, a CD platform built for multicloud environments.
  - **GoCD**, a CI/CD server with an emphasis on modeling and visualization.
  - **Concourse**, "an open-source continuous thing-doer."
  - **Screwdriver**, a build platform designed for CD
- Teams may also want to consider managed CI/CD tools, which are available from a variety of vendors.
- The major public cloud providers all offer CI/CD solutions, along with GitLab, CircleCI, Travis CI, Atlassian Bamboo, and many others.
- Additionally, any tool that's foundational to DevOps is likely to be part of a CI/CD process.
- Tools for configuration automation (such as Ansible, Chef, and Puppet), container runtimes (such as Docker, rkt, and cri-o), and container orchestration (Kubernetes) aren't strictly CI/CD tools, but they'll show up in many CI/CD workflows.
- There are many different ways you can implement CI/CD based on your preferred application development strategy and cloud provider.
- Red Hat OpenShift Service on AWS has several options available to make your own CI/CD workflow easier like Tekton and OpenShift Pipelines.
- By using Red Hat OpenShift, organizations can employ CI/CD to automate building, testing, and deployment of an application across multiple on-premises and cloud platforms.

<u>GitHub Actions:</u>

Overview:

- GitHub Actions is a CI/CD tool integrated within GitHub.

- Automates workflows for building, testing, and deploying code directly from GitHub repositories.

Features:

- **Integration:** Seamless integration with GitHub repositories.
- **Customization:** Define workflows using YAML configuration files.
- **Marketplace:** Access to a wide range of pre-built actions in the GitHub Marketplace.
- **Matrix Builds:** Test code across multiple environments and configurations.
- **Triggers:** Trigger workflows based on events such as pushes, pull requests, and issue creation.

Benefits:

- **Ease of Use:** Simplifies setup and management of CI/CD pipelines within GitHub.
- **Visibility:** Provides detailed logs and insights into workflow runs within the GitHub interface.
- **Cost:** Free for public repositories, with usage limits for private repositories.

Jenkins:

Overview:

- Jenkins is an open-source automation server widely used for CI/CD.
- Known for its extensive plugin ecosystem and flexibility.

Features:

- **Extensibility:** Over 1,500 plugins available to support various tasks and integrations.
- **Pipeline as Code:** Define build pipelines using Jenkinsfile (supports both declarative and scripted syntax).
- **Distributed Builds**: Distribute build workloads across multiple machines for scalability.
- **Integration:** Integrates with many tools and platforms, including Git, Docker, Kubernetes, and more.

Benefits:

- **Flexibility:** Customizable to suit a wide range of CI/CD needs.
- **Community Support:** Strong community with regular updates and extensive documentation.
- **Scalability:** Suitable for projects of all sizes, from small teams to large enterprises.

Buddy:

Overview:

- Buddy is a user-friendly CI/CD tool designed to simplify and speed up deployment processes.
- Emphasizes a visual approach to pipeline creation and management.

Features:

- **Visual Editor:** Drag-and-drop interface for building and managing pipelines.
- **Pre-Configured Actions:** Ready-to-use actions for building, testing, and deploying code.
- **Integrations:** Supports integrations with GitHub, Bitbucket, GitLab, Docker, AWS, Google Cloud, and more.
- **Parallelism:** Run multiple actions in parallel to speed up workflows.
- **Monitoring and Alerts:** Built-in monitoring and alerting to track pipeline status.

Benefits:

- **User-Friendly:** Intuitive interface suitable for developers of all experience levels.
- **Speed:** Optimized for fast execution with support for parallel actions.
- **Cost-Effective:** Various pricing plans, including a free tier for small teams and projects.

Netlify CI/CD:

Overview:

- Netlify CI/CD is a part of the Netlify platform, primarily used for deploying static websites and modern web applications.
- Automates the build and deployment process, integrating with various version control systems.

Features:

- **Automated Builds:** Automatically builds and deploys sites when changes are pushed to the repository.
- **Deploy Previews:** Creates deploy previews for pull requests, enabling preview of changes before merging.
- **Branch Deploys:** Supports branch-specific deployments, useful for staging and testing environments.
- **Serverless Functions:** Integrates with Netlify Functions for deploying serverless functions alongside static assets.
- **Integrations:** Connects with GitHub, GitLab, Bitbucket, and other version control systems.

Benefits:

- **Ease of Use:** Simplifies the process of deploying static sites and modern web applications.
- **Preview Capabilities:** Deploy previews enhance collaboration and review processes.
- **Comprehensive:** Combines CI/CD with other Netlify features like serverless functions and edge handlers.
- **Free Tier:** Offers a free tier suitable for personal projects and small teams.

These tools provide various features and benefits to cater to different needs in the CI/CD pipeline, from simple static site deployments to complex, scalable build and deployment processes.