What is CI/CD Pipeline?

This video provides a comprehensive explanation of CI/CD pipelines, crucial for automating the software development lifecycle.

What is CI/CD?

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**. It automates various stages of software development, such as building, testing, and deploying applications. This automation helps overcome the challenges of manual processes, which are prone to errors and time-consuming.

Key Stages of Software Development in CI/CD:

- **Develop:** Writing code for new features or bug fixes.
- **Commit & Push:** Saving changes to a version control system like Git, usually on a dedicated branch.
- **Build:** Creating deployable artifacts (e.g., APK files for Android) from the application's source code and its dependencies.
- **Test:** Performing various types of tests, including:
  - **Unit Tests:** Simple tests usually done by developers.
  - **Integration Tests:** Checking if different components work together correctly.
  - **Regression Tests:** Ensuring new changes don't break existing functionalities.
  - **End-to-End Testing:** Includes security tests, smoke tests, and alpha/beta tests, typically performed after deployment.
- **Release & Deploy:** Making the application available to end-users. Deployment often involves different environments:
  - **Development (Dev) Environment:** For developers only.
  - **Staging Environment:** A pre-production environment for team demos and final checks.
  - **Production (Prod) Environment:** Where the application is used by all end-users.

Continuous Integration (CI):

CI automates the **build and test steps** of the development lifecycle. This means that every time a developer commits a change, the code is automatically built and tested to ensure it's working correctly and that no new errors have been introduced. If tests fail, the developer is immediately notified to fix the issues.

Continuous Delivery (CD) vs. Continuous Deployment (CD):

- **Continuous Delivery:** Automates the **release and deployment steps** up to the staging environment. Before deploying to production, a **manual approval step** is involved, where a team leader or manager must approve the changes. This is the most common way to implement CI/CD.
- **Continuous Deployment:** This is a more advanced form of CD where the application is automatically deployed to production **without any manual approval** once all tests are successful. This is often used in organizations with well-established DevOps teams and less sensitive data (e.g., streaming services). For sensitive data like banking applications, Continuous Delivery with manual approval is preferred.

Why CI/CD is Needed:

Manual processes lead to:

- **Integration Hell:** Conflicts and issues when multiple developers merge their code manually, leading to significant time loss and missed errors.
- **Error-prone and Time-consuming Testing:** Manual testing of numerous test cases is inefficient and can result in bugs reaching users.
- **Infrequent Releases:** Code freezes and manual processes delay new releases to users.
- **Difficult Rollbacks:** Manually restoring previous versions in case of production errors is time-consuming and can lead to application downtime.

CI/CD addresses these problems by providing **reliability and speed** throughout the development process.

How to Implement CI/CD Pipelines:

CI/CD pipelines are implemented by writing automation scripts, often using YAML files (Yet Another Markup Language). These scripts define a workflow of commands and instructions that are executed automatically with every code change.

Tools for CI/CD Pipelines:

Several tools are available to set up CI/CD pipelines, including:

- GitHub Actions
- Jenkins
- CircleCI
- Travis CI
- GitLab CI/CD

- Bamboo

GitHub Actions (released in 2019) is highly popular and native to GitHub, making it very intuitive for projects already on GitHub. **Jenkins** (released in 2011) is an older but very popular job orchestrator with many customization options, though it can be harder to maintain. The recommendation is to start with GitHub Actions for first-time CI/CD setup due to its ease of use and familiarity with Git.

Deployment Strategies with CI/CD:

To achieve near-zero downtime and handle potential bugs in production, various deployment strategies are used with CI/CD:

- **Blue-Green Deployment:** Involves two identical but separate environments: a "Blue" environment with the current application version and a "Green" environment with the new version. All traffic is redirected to the new version, and if anything goes wrong, it can be quickly switched back to the old version.
- **Canary Deployment:** Gradually redirects a small percentage of user traffic to the new version. If the new version performs well, more users are progressively redirected. This allows for early detection of issues with minimal impact.
- **Rolling Deployment:** Deploys the new version to a subset of instances in a single environment, while the old version runs on others. Traffic is then gradually shifted to the updated instances.

This video emphasizes how CI/CD, especially when integrated with tools like Docker and Kubernetes, simplifies work for scalable applications and large teams.

You can watch the full video here: What is CI/CD Pipeline? | Simply Explained by Shradha Ma'am