**Name: Nikitha C.Y**

**USN: 4NI19IS055**

**Section B**

# Self-learning Exercise

# Assignment-1

## Overview of CI/CD:

CI/CD stands for continuous integration and continuous delivery/continuous deployment. It is the method to frequently deliver apps to customers by introducing automation into the stages of app development. It allows the organization to ship software quickly and efficiently. It facilitates an effective process for getting products to market faster, continuous delivery code into production and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method. It automates much or all of the manual human intervention traditionally needed to get new code from a commit into production such as build ,test and deploy as well as infrastructure provisioning.

## What is Continuous Integration (CI)?

Continuous Integration is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them and automatically build.

With continuous integration, errors and security issues can be identified and fixed more easily, and much earlier in the software development lifecycle.

## What is Continuous Delivery (CD)?

Continuous delivery is a software development practice the works in conjunction with continuous integration to automate the infrastructure provisioning and application release process.Once code has been tested and built, Continuous delivery takes over during the final stages to ensure it can be deployed packaged with everything that needs to deploy to any environment at any time.

# Fundamentals of CI/CD

1. **Single source repository**
   Source code management has all necessary files and scripts to create builds. The repository should contain everything needed for the build. This includes source code, database structure, libraries, properties files and version control. It should also contain test scripts and scripts to build applications.

2. **Frequent check-ins to main branch**
   Integrating code in your trunk, mainline or master branch early and often. Avoid branches and work with the main branch only. Use small segments of code and merge them into the branch as frequently as possible.

3. **Automated builds**
   Scripts should include everything you need to build from a single command. This includes web server files, database scripts and application software.

4. **Self-testing builds**
   Testing script should ensure that the failure of a test results in a failed build. Use static pre-build testing scripts to check code for integrity, quality and security compliance.

5. **Frequent iterations**
   Multiple commits to the repository results

# What are Feature flags and how it is used?

Feature flags, also known as feature toggles, are a technique used in software development and DevOps to enable or disable features in a software application without the need to deploy a new version of the application. This allows developers to work on new features or changes to the application without disrupting the functionality of the current version for users.

Feature flags can be used to test new features with a subset of users before releasing them to the entire user base, or to gradually roll out a feature to all users.

• They can also be used to quickly disable a feature in the event of a problem or to experiment with different versions of a feature.

• In DevOps, feature flags are often used in conjunction with continuous delivery and deployment practices.

• Feature flags are typically implemented using a combination of code and configuration.

• They can be managed through a variety of tools and platforms, including feature flag management platforms, version control systems, and configuration management systems. An example of how feature flags might be used in a DevOps workflow

• A developer writes and tests a new feature on their local machine.

• The developer commits the code changes to a version control system (such as Git) and pushes them to a central repository.

• The code is automatically built and deployed to a staging environment.

• The developer creates a feature flag to toggle the availability of the new feature on or off.

• The developer uses the feature flag to enable the new feature for a small group of users, such as a subset of the staging environment or a group of beta testers.

• The team monitors the usage and performance of the new feature to ensure that it is working as expected.

• If the new feature is stable and performs well, the team can use the feature flag to roll it out to a wider audience. If there are any issues with the feature, the team can use the feature flag to quickly disable it.

# Explain stages of CI/CD Pipeline with block diagram

The continuous integration/continuous delivery (CI/CD) pipeline is an agile DevOps workflow focused on a frequent and reliable software delivery process. The methodology is iterative, rather than linear, which allows DevOps teams to write code, integrate it, run tests, deliver releases and deploy changes to the software collaboratively and in real-time.

A key characteristic of the CI/CD pipeline is the use of automation to ensure code quality. As the software changes progress through the pipeline, test automation is used to identify dependencies and other issues earlier, push code changes to different environments and deliver applications to production environments. Here, the automation's job is to perform quality control, assessing everything from performance to API usage and security. This ensures the changes made by all team members are integrated comprehensively and perform as intended.

The ability to automate various phases of the CI/CD pipeline helps development teams improve quality, work faster and improve other DevOps metrics.

# CICD pipeline phases

Here is a high-level overview of the steps involved in a typical CI/CD pipeline:

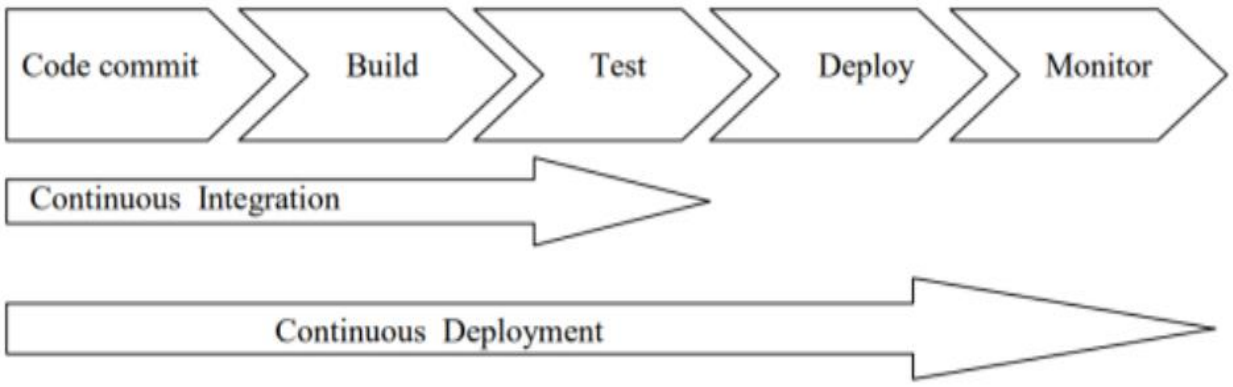
**Code Commit**: Developers commit their code changes to a version control system (such as Git) and push them to a central repository.

**Build:** A build system retrieves the latest code from the repository and compiles it into a deployable package (such as a Docker image). The build process may also include tasks such as running tests and generating documentation.

**Test**: The build is deployed to a staging environment and automated tests are run to ensure that the code is working as expected.

**Deploy:** If the tests pass, the code is deployed to production. If the tests fail, the deployment process is halted and the developers are notified of the issue.

**Monitor:** Once the code is deployed to production, it is monitored for any issues or errors. By automating these steps, a CI/CD pipeline allows development teams to quickly and confidently deliver new code changes to users, while also ensuring that the code is of high quality.



## Advantages of CI/CD pipelines

- Builds and testing can be easily performed manually
- It can improve the consistency and quality of code.
- Improves flexibility and has the ability to ship new functionalities.
- CI/CD pipeline can streamline communication.
- It can automate the process of software delivery.
- Helps you to achieve faster customer feedback.
- CI/CD pipeline helps you to increase your product visibility.
- It enables you to remove manual errors.
- Reduces costs and labour.
- CI/CD pipelines can make the software development lifecycle faster.
- It has automated pipeline deployment.
- A CD pipeline gives a rapid feedback loop starting from developer to client.
- Improves communications between organization employees.
- It enables developers to know which changes in the build can turn to the brokerage and to avoid them in the future.
- The automated tests, along with few manual test runs, help to fix any issues that may arise