NAME - PRAJWAL C
USN NO - 4NI19IS064

DEVOPS
ASSIGNMENT - 01

1. WHAT IS CI/CD ?

--> CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams

--> Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline" and are supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach.

2. What is CI/CD PIPELINE ?

--> The continuous integration/continuous delivery (CI/CD) pipeline is an agile DevOps work flow 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.

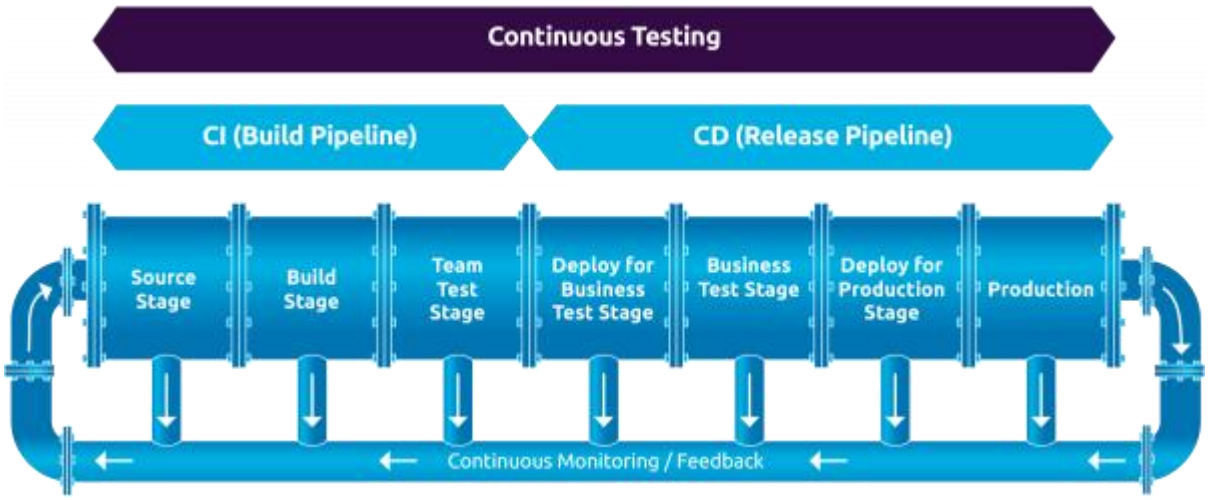
The ability to automate various phases of the CI/CD pipeline helps development teams improve quality, work faster and improve other DevOps metrics.

3. USES OF CI/CD PIPELINE ?

--> Automation of software releases — from initial testing to the final deployment — is a significant benefit of the CI/CD pipeline. Additional benefits of the CI/CD process for development teams include the following:

- **Reducing time to deployment through automation:** Automated testing makes the development process more efficient, reducing the length of the software delivery process. In addition, continuous deployment and automated provisioning allow a developer's changes to a cloud application to go live within minutes of writing them.
- **Decreasing the costs associated with traditional software development:** Fast development, testing and production (facilitated by automation) means less time spent in development and, therefore, less cost.
- **Continuous feedback for improvement:** The CI/CD pipeline is a continuous cycle of build, test and deploy. Every time code is tested, developers can quickly take action on the feedback and improve the code.
- **Improving the ability to address error detection earlier in the development process:** In continuous integration, testing is automated for each version of code built to look for issues integration. These issues are easier to fix the earlier in the pipeline that they occur.
- **Improving team collaboration and system integration**. Everyone on the team can change code, respond to feedback and quickly respond to any issues that occur.

4. STEPS INVOLVED IN CI/CD PIPELINE WITH EXAMPLE ?



A CI/CD pipeline resembles the various stages software goes through in its lifecycle and mimics those stages:

Source stage: Implement a code repository and version control system

This initial CI/CD stage involves storing and managing source code in a repository with a version control system (VCS) that supports collaboration and tracking changes across a distributed team. The VCS can trigger a pipeline execution based on various events, such as a branch push or pull request validation. These pipeline runs also can be scheduled or initiated by a user.

Actions to perform in this stage:

1. Select a repository and VCS, such as Git or SVN.
2. Determine if you'll host the VCS yourself or use a provider such as GitHub, Bitbucket, Azure DevOps and others.
3. Create repositories to house the application source code along with the pipelines.

Build stage: Choose a CI engine, compile code, run checks

The key to this CI/CD stage is to provide early feedback to developers and maintain the application in a state where it can be released to an environment.

It is imperative to provide feedback as soon as a developer checks in new code changes to flag and resolve any issues such as syntax or compilation. Also, compilation ensures that the code can successfully generate artifacts for eventual release into environments further down the CI/CD pipeline.

Actions to perform in this stage:

1. Determine the build/CI server to use, whether self-hosted such as Jenkins or Jenkins X, or a third-party Jenkins alternative such as GitHub Actions, CircleCI or Azure Pipelines.
2. Ideally, set up a pipeline-as-code This can be stored in version control with appropriate triggers identified to release software, such as branch pushes and pull requests.
3. Implement a stage/job in the pipeline that compiles (builds) the application source code. For modern cloud-native applications, this step can generate a Docker image.
4. Run static analysis and style checks to ensure there are no "code smells" – indications the existence of deeper problems to explore -- and that the coding style is consistent with

organizational guidelines. A CI engine may support various plugins to perform these static analyses and generate warnings.

5. The pipeline then generates a versioned and built (compiled) artifact or a container image. Publish this build artifact to a store or feed (or container image to a registry), from where it is readily available for testing or deployment.

Testing stage: Test the build, publish results, release for production

The goal of this stage is to ensure that the changes do not break any logic or functionality and that the code is safe to release. In short, testing provides a safety net to release the code. Among the many tests that occur in this stage are unit, integration and functional tests.

Unit tests examine small units of application code and only test the logic in isolation without any dependencies, although if required, you can simulate them through mocking. For code developed in object-oriented languages such as Java or C#, etc., these small units can be class methods.

Integration tests analyze individual units of code together in a group. This builds confidence that individual modules integrated to build the entire application won't break under test.

An end-to-end *functional test* introduces the software into an environment to mimic a production deployment. This step is often automated with tools such as Selenium.

Actions to perform in this stage:

1. Choose from a vast ecosystem of plugins, such as xUnit or JUnit, that enable integration of the test runner of choice within the pipeline.
2. Publish the test report and code coverage reports so that the results of the testing are easily available in the pipeline run.
3. Maintain a predetermined benchmark of code coverage to release a software build. If the code coverage drops below a certain threshold, fail the stage.

Deploy stage: Deliver and deploy the final build

Once the software is built and tested and artifacts are generated, it is ready to be released into an environment. Ideally, there are multiple environments through which the built artifacts are released and then tested, and if the release passes all the tests it progresses through to production deployment. This stage also handles adding the required resources to host the application in the cloud.