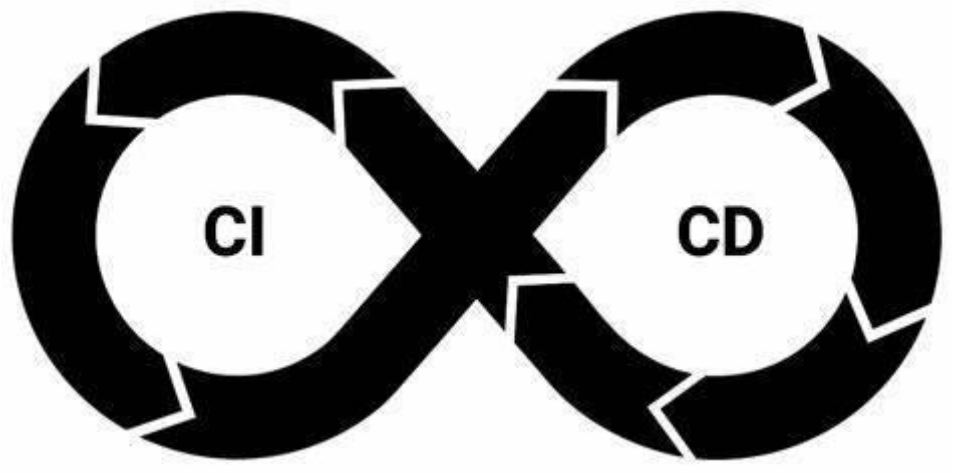
DEVOPS ASSIGNMENT-1

Swathi B J

4NI19IS103 B

1) What is Continuous Integration and Continuous Deployment (CI/CD)

CI/CD is a strategy to automate the development process to speed up turnaround time. CI, or continuous integration, is the practice of frequently merging code changes into a shared branch. CD can refer to continuous delivery or deployment, which both automate the release and rollout of the application after merging. The code is then delivered quickly and seamlessly as a part of the CD process. In the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers' desktops to be delivered quickly and reliably to production.



Why is CI/CD important?

CI/CD automates the process of integrating, releasing, and deploying software while removing traditional roadblocks. It supports the larger goal of agile methodology to accelerate the software development lifecycle, and it supports the DevOps approach of aligning development and operations teams.

In addition to supporting these larger goals, CI/CD enables you to:

- **Ship software quickly and efficiently**: CI/CD pipelines move applications from the coding to deployment phases at scale, ensuring that the pace of development matches the needs of the business.
- **Increase productivity**: By implementing automated processes, development and operations teams are no longer spending time merging, building, testing, releasing, and deploying software manually. Instead, they can focus on writing better code and monitoring deployments for issues.
- **Reduce risk on delivery**: Testing every change before it's deployed ensures that the result will be a higher quality product and lower the rate of bugs in production. Customers will receive a better product, and the development team will spend less time fixing urgent defects discovered after release.
- **Incorporate user feedback faster**: CI/CD removes traditional roadblocks for development and operations teams, enabling the faster release of new features to meet users' needs. This will increase customer satisfaction and provide valuable insights into the capabilities that users value for future projects.
- **Standardize processes**: Automating the merge, test, delivery, and deployment processes means that they will always follow the same structure. This standardizes the pipeline, whereas manual execution of these tasks always comes with the risk of human error, such as executing tests in a different order.

Difference between CI and CD

Continuous integration (CI) is practice that involves developers making small changes and checks to their code. Due to the scale of requirements and the number of steps involved, this process is automated to ensure that teams can build, test, and package their applications in a reliable and repeatable way. CI helps streamline code changes, thereby increasing time for developers to make changes and contribute to improved software.

Continuous Delivery (CD) is the automated delivery of completed code to environments like testing and development. CD provides an automated and consistent way for code to be delivered to these environments.

Continuous Deployment is the next step of continuous delivery. Every change that passes the automated tests is automatically placed in production, resulting in many production deployments.

How CI/CD relate to DevOps

Continuous Integration and Continuous Delivery are often quoted as the pillars of successful DevOps. DevOps is a software development approach which bridges the gap between development and operations teams by automating build, test and deployment of applications. It is implemented using the CICD pipeline. The CI/CD pipeline is part of the broader DevOps/DevSecOps framework. In order to successfully implement and run a CI/CD pipeline, organizations need tools to prevent points of friction that slow down integration and delivery. Teams require an integrated toolchain of technologies to facilitate collaborative and unimpeded development efforts

2) 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.

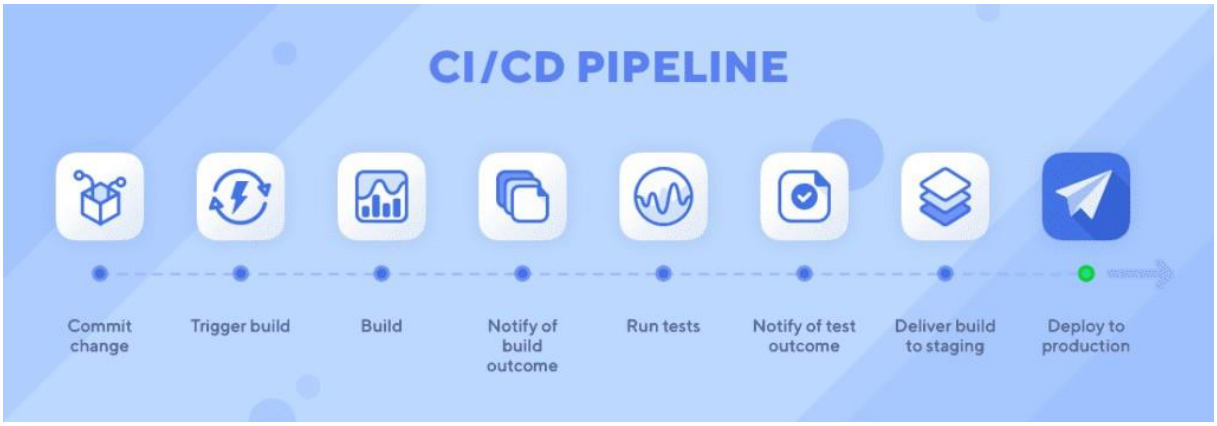
• Feature flags are typically implemented using a combination of code and configuration.

• They can be managed through a variety of tools and platforms, including feature flag management platforms, version control systems, and configuration management systemsExample code:

```
if(activateFeature("addTaxToOrder")==True) {
ordervalue = ordervalue + tax }
else{
 ordervalue = ordervalue }
```

In this example code, the activateFeature function allows us to find out whether the application should add the tax to order according to the addTaxToOrder parameter, which is specified outside the application (such as in a database or configuration file).

How and where Feature flags are used along with CI/CD Pipeline



Continuous Development

Continuous deployment takes the benefits of continuous delivery a step further: now, the code is automatically pushed to a repository and then deployed to production in the same process. This relieves a potential burden on the operations team to manually deploy new releases, which may be numerous depending on the pace of the development team.

However, automating manual deployment processes places an even greater emphasis on effective test automation during the CI phase because this is now the main failsafe before bugs are pushed to production.

Continuous Delivery

Continuous delivery is the next evolution in the automation of the software lifecycle. Once code has been bug tested, it is immediately released to a shared repository like GitHub. Continuous delivery is intended to improve visibility and communication between the development and operations teams by speeding the rate at which production-ready code is delivered. The operations team can then deploy the application with ease.

Continuous delivery is only effective when CI is also implemented. Testing in the CI phase is critical to validate that code is ready for release to the repository. Otherwise, defective code is automatically delivered, leaving bugs that are then pushed to production.

Continuous Testing

Feature flags provide a way to quickly roll back the feature during the canary testing. And the beauty of the feature flag is that both the development team and the non-technical stakeholders, eg. product managers, can handle the “control button”. The non-tech stakeholders usually are at the front line when collecting users’ feedback. Giving them the control of turning on and off new features aligns well with the core concept of DevOps.

Operation Support

If we look at the operation part, feature flags can also play an important role. In addition to the testing in production use cases mentioned in the earlier section, feature flags can also improve the incident management process.For example, usually when the monitoring system detects an abnormal performance that may be related to the new release feature, with integration to an ITSM system, it’ll create an incident. The operation team will respond to the incident by going through incident response process. Even with SLA defined, it’ll still take some time for the incident ticket to land in an operation staff’s hand. By the time they finished their investigation, escalation and roll back, it may have caused certain level of damages.

Feature flags are like “buttons”. We can manually flip them on or off, or build a decision tree to automatically decide when to flip. These “buttons” can be used across DevOps lifecycle, which complement continuous integration and deployment as well as operational excellence. And remember to “clean up after yourself” to avoid building up the technical debt when using these “buttons”.

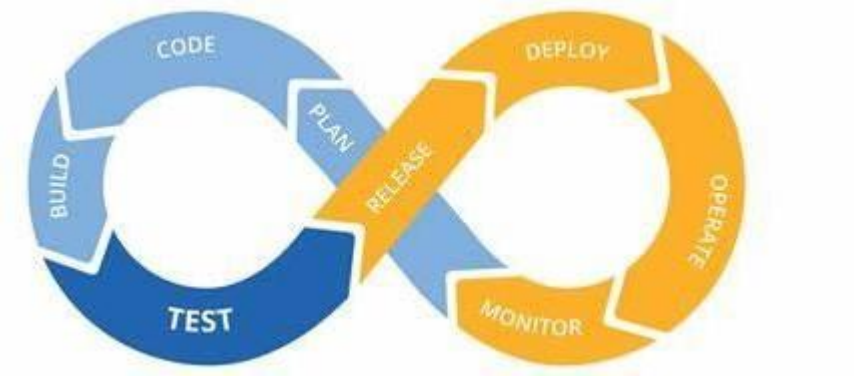
3) Explain CI/CD Pipeline with Block Diagram

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.



CI/CD pipeline phases

From source code to production, these phases make up the development lifecycle and workflow of the CI/CD pipeline:

- **Build:** This phase is part of the continuous integration process and involves the creation and compiling of code. Teams build off of source code collaboratively and integrate new code while quickly determining any issues or conflicts.
- **Test:** At this stage, teams test the code. Automated tests happen in both continuous delivery and deployment. These tests could include integration tests, unit tests, and regression tests.
- **Deliver:** Here, an approved codebase is sent to a production environment. This stage is automated in continuous deployment and is only automated in continuous delivery after developer approval.
- **Deploy:** Lastly, the changes are deployed and the final product moves into production. In continuous delivery, products or code are sent to repositories and then

moved into production or deployment by human approval. In continuous deployment, this step is automated.

Implementation of CI/CD pipeline stages

Plan

Propose a feature that needs to be built, or outline an issue that warrants change.

Code

Turn use case suggestions and flowcharts into code, peer-review code changes, and obtain design feedback.

Test

Verify code changes through testing, preferably automated testing. At this point, unit testing will usually suffice.

Repository

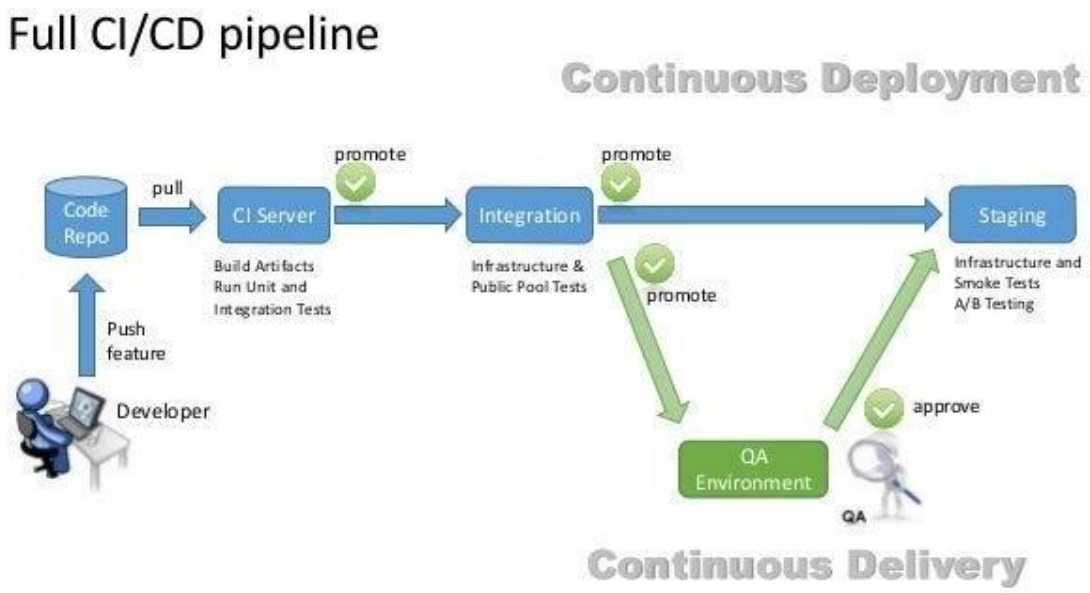
Push code to a shared repository like Github that uses version control software to keep track of changes made. Some consider this as the first phase of the CI/CD pipeline.

Set up an Integration Testing Service

For example, Travis CI to continuously run automated tests, such as regression or integration tests, on software hosted on services like Github and BitBucket.

Set up a Service To Test Code Quality

Better Code Hub can help continuously check code for quality in CI/CD pipeline. This will enable the software development team to spend less time fixing bugs. Better Code Hub uses 10 guidelines to gauge quality and maintainability in order to future proof the code.



Build

This might overlap with compilation and containerization. Assuming the code is written in a programming language like Java, it'll need to be compiled before execution. Therefore, from the version control repository, it'll go to the build phase where it is compiled. Some pipelines prefer to use Kubernetes for their containerization.

Testing Phase

Build verification tests as early as possible at the beginning of the pipeline. If something fails, you receive an automated message to inform you the build failed, allowing the DevOps to check the continuous integration logs for clues.

Smoke testing might be done at this stage so that a badly broken build can be identified and rejected, before further time is wasted installing and testing it. Push the container

(Docker) image created to a Docker hub: This makes it easy to share your containers in different platforms or environments or even go back to an earlier version.

Deploy the App, Optionally To a Cloud-Based Platform

The cloud is where most organizations are deploying their applications. Heroku is an example of a relatively cheap cloud platform. Others might prefer Microsoft Azure or Amazon Web Service (AWS).