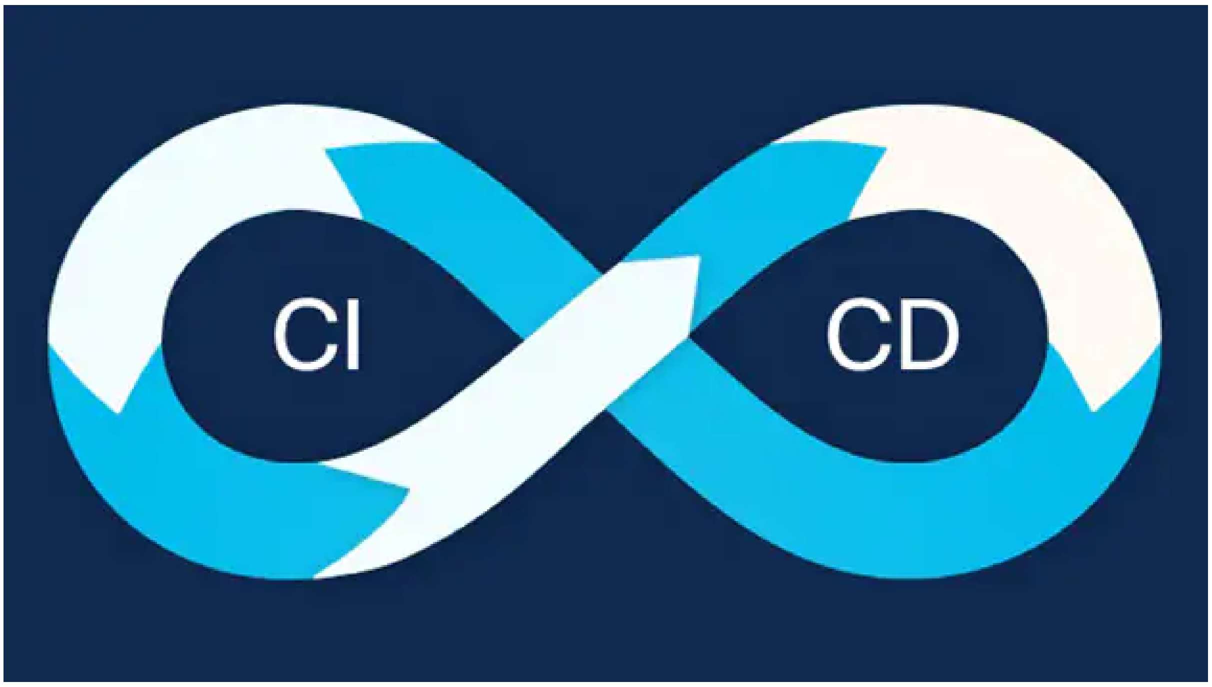
ASSIGNMENT 1

Vishal Rai

4NI19IS113 'B' Section

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

CI and CD stand for continuous integration and continuous delivery/continuous deployment. In very simple terms, CI is a modern software development practice in which incremental code changes are made frequently and reliably. Automated build-and-test steps triggered by CI ensure that code changes being merged into the repository are reliable. 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 allows organizations to ship software quickly and efficiently. CI/CD facilitates an effective process for getting products to market faster than ever before, continuously delivering code into production, and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method.

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.

Continuous deployment should be the goal of most companies that are not constrained by regulatory or other requirements.

How CI/CD relate to DevOps

DevOps is a set of practices and tools designed to increase an organization’s ability to deliver applications and services faster than traditional software development processes. The increased speed of DevOps helps an organization serve its customers more successfully and be more competitive in the market. In a DevOps environment, successful organizations “bake security in” to all phases of the development life cycle, a practice called DevSecOps.

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?

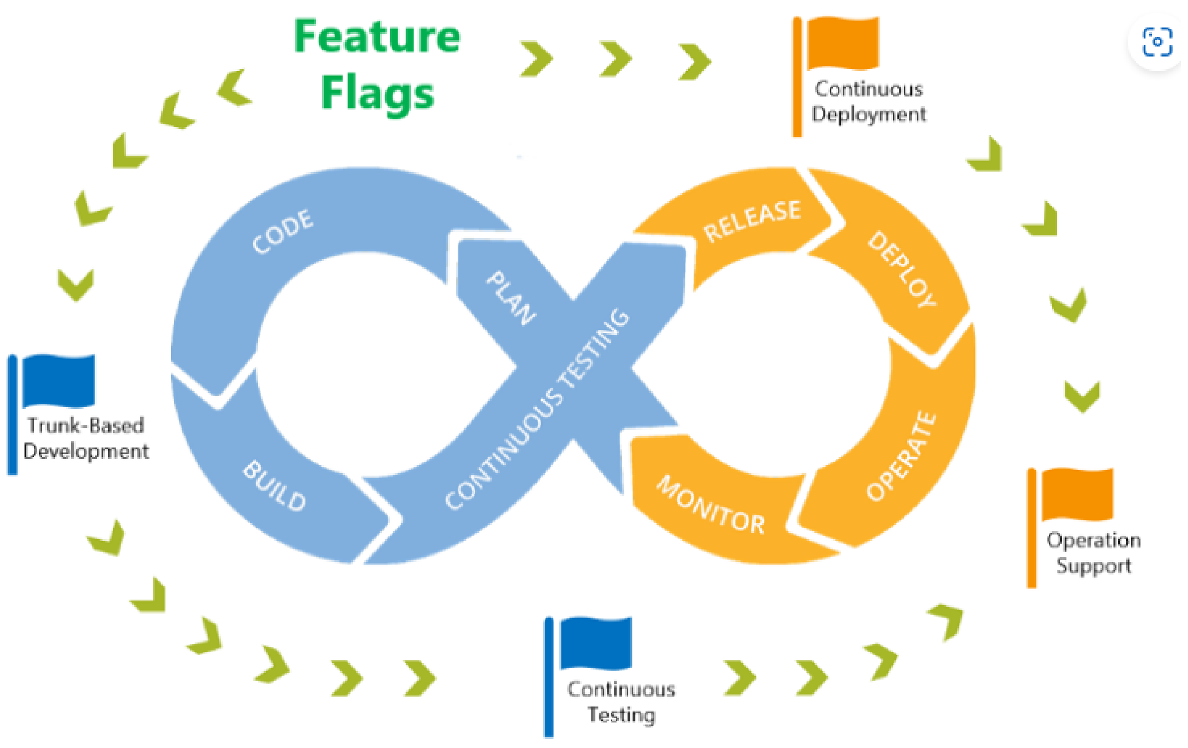
Feature flags are system of codes that allows conditional features to be accessed only when certain conditions are met. In other words, if a flag is on, new code is executed if the flag is off, the code is skipped. It is also referred to as release toggles, feature flags are a best practice in DevOps, often occurring within distributed version control systems.

Example 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



Trunk-Based Development

In the Dev bucket of the DevOps lifecycle, there are two popular development styles: Git Flow and Trunk-Based Development. Trunk-Based Development on the encourages short-lived feature branches. It's like a lean version of a tree that only has the trunk (e.g. the master branch) and some short and direct connected branches. Developers can merge their code directly to the master branch once their code are "ready". We'll explain why we put quotation marks around the word "ready".

Continuous Delivery

In Continuous Delivery, the deploy to production part is a manual process, whereas Continuous Deployment automates all the way through. The automate-all-the-way-through process seems to be more streamlined. However, it may raise concerns in scenarios like companies who have well-established product or large development teams with a lot of junior developers. The production environment is less tolerant for imperfect code.

With feature flags introduced in development, it enables this final automation part. DevOps teams can continuously deploy solutions to their production environment with no sweat. The production users won't be able to interact with the flagged code.

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.

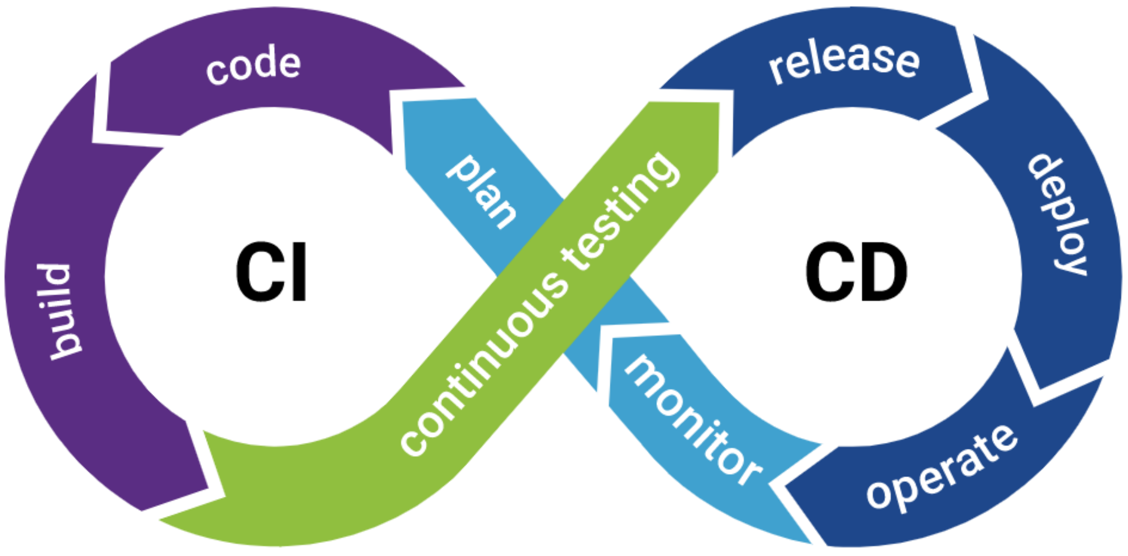
Feature flags are like "buttons". We can manually flip them on or off, or build a decision tree to automatically decide when to flip. These "buttons" can be used across DevOps lifecycle, which complement continuous integration and deployment as well as operational excellence. And remember to "clean up 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.

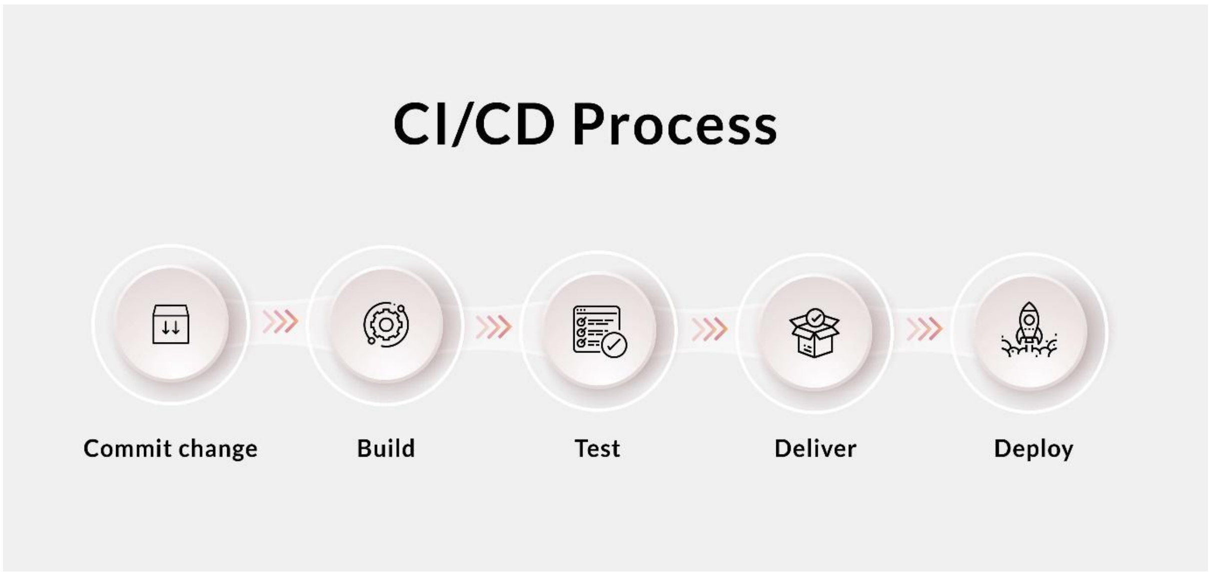
The ability to automate various phases of the CI/CD pipeline helps development teams improve quality, work faster and improve other DevOps metrics.



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

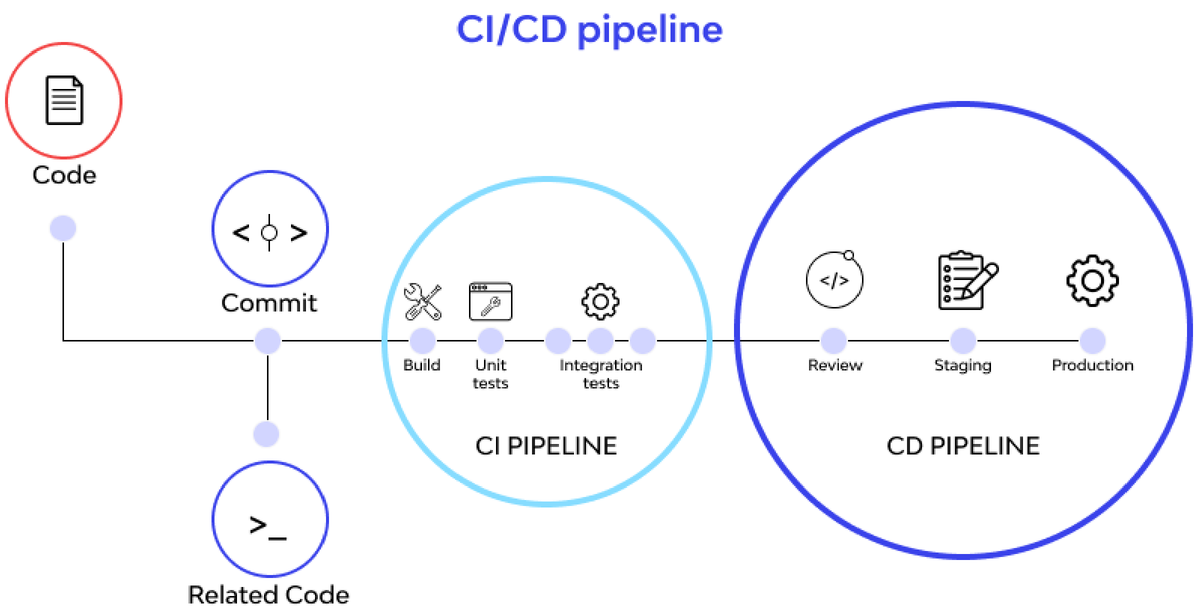
Push code to a shared repository like Github that uses version control software to keep track of changes made. Some consider this as the first phase of the CI/CD pipeline.

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).