DevOps Assignment 1

Name:Neha R

USN: 4NI19IS054

7TH SEM ISE

A section

NIE Mysuru

Continuous integration and continuous deployment (CI/CD)

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

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily.

With CI, a developer practices integrating the code changes continuously with the rest of the team. The integration happens after a "git push," usually to a master branch—more on this later. Then, in a dedicated server, an automated process builds the application and runs a set of tests to confirm that the newest code integrates with what's currently in the master branch.

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.

What is the 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.

In short, CI is a set of practices performed *as developers are writing* code, and CD is a set of practices performed *after* the code is completed.

CI works in three simple stages: push, test, and fix. But despite this simplicity, CI might become challenging if only a few members of the team practice it. Consequently, CI also requires a change in culture and support from management.

The three stages of the CI workflow:

Push to master every day Of all the three stages of CI, committing to master is the easiest technically, but the hardest culturally. And the reason that's true is that integrating code daily doesn't mean that a developer will push the code into a feature branch. No, the CI practice is about pushing code into the master branch, because that's the branch that's going to be used to release software. The "push to master" stage is also known as trunk-based development, and there's a dedicated site that explains this technique in much more detail. When you practice CI, it doesn't mean you'll no longer use branches. You still will. The only difference is that because you amplify feedback when you integrate code continually, branches become temporary. A branch might live only for the day; then it's integrated into the master branch

But what about incomplete changes? Well, you can integrate incomplete changes by using feature flags. Feature flags are an if condition determining whether to run the new code or not. If a change isn't complete yet, the flag is off by default. That way, when you integrate the code, the rest of the team has a chance to review it. The same technique applies if the new code has bugs

Rely on automated reliable tests

To validate each time a developer integrates new code, CI relies on an automated and reliable suite of tests. If you need to compile the code, the first test is that the code compiles. Then, you can include as many tests as you consider critical.

Prioritize fixing a broken build

When the build is broken, fixing it should be the priority for the team. And the importance of fixing it should be a shared mindset in the culture. Fixing the build, as a principle, comes from the Toyota Andon cord that the car manufacturer used to produce cars

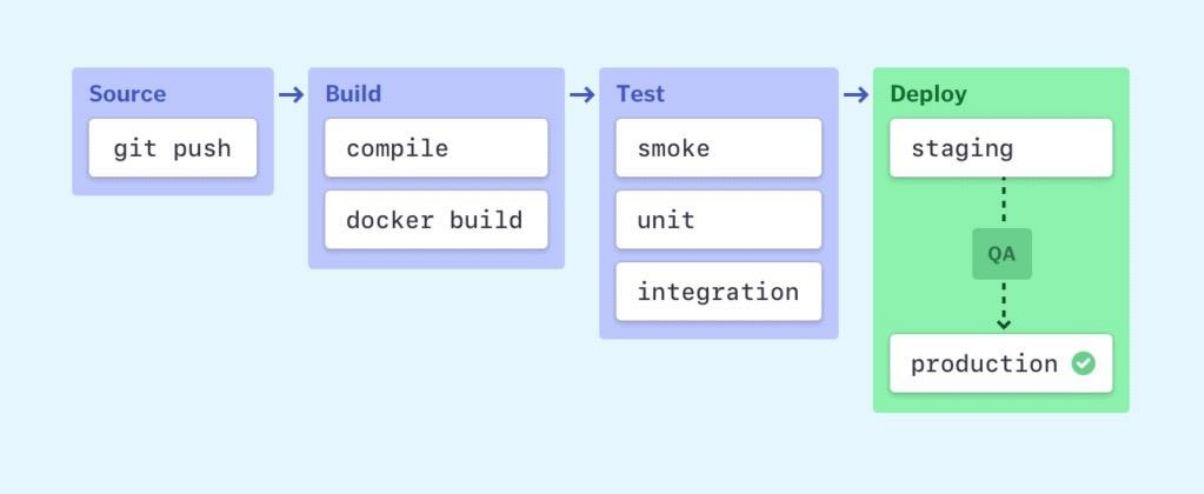
What is CI/CD pipeline?

A pipeline is a process that drives software development through a path of building, testing, and deploying code, also known as CI/CD. By automating the process, the objective is to minimize human error and maintain a consistent process for how software is released. Tools that are included in the pipeline could include compiling code, unit tests, code analysis, security, and binaries creation. For containerized environments, this pipeline would also include packaging the code into a container image to be deployed across a hybrid cloud.

CI/CD is the backbone of a DevOps methodology, bringing developers and IT operations teams together to deploy software. As custom applications become key to how companies differentiate, the rate at which code can be released has become a competitive differentiator.

A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.

Most software releases go through a couple of typical stages:



Source stage

In most cases, a pipeline run is triggered by a source code repository. A change in code triggers a notification to the CI/CD tool, which runs the corresponding pipeline. Other common triggers include automatically scheduled or user-initiated workflows, as well as results of other pipelines.

Build stage

We combine the source code and its dependencies to build a runnable instance of our product that we can potentially ship to our end users. Programs written in languages such as Java, C/C++, or Go need to be compiled, whereas Ruby, Python and JavaScript programs work without this step.

Regardless of the language, cloud-native software is typically deployed with Docker, in which case this stage of the CI/CD pipeline builds the Docker containers.

Failure to pass the build stage is an indicator of a fundamental problem in a project's configuration, and it's best to address it immediately.

Test stage

In this phase, we run automated tests to validate our code's correctness and the behavior of our product. The test stage acts as a safety net that prevents easily reproducible bugs from reaching the end-users.

The responsibility of writing tests falls on the developers. The best way to write automated tests is to do so as we write new code in test- or behavior-driven development.

Tools for CI

• Jenkins—a free, open-source, Java-based tool that gives you a lot of flexibility

• Azure Pipelines—a Microsoft product free for up to five users and open-source projects.

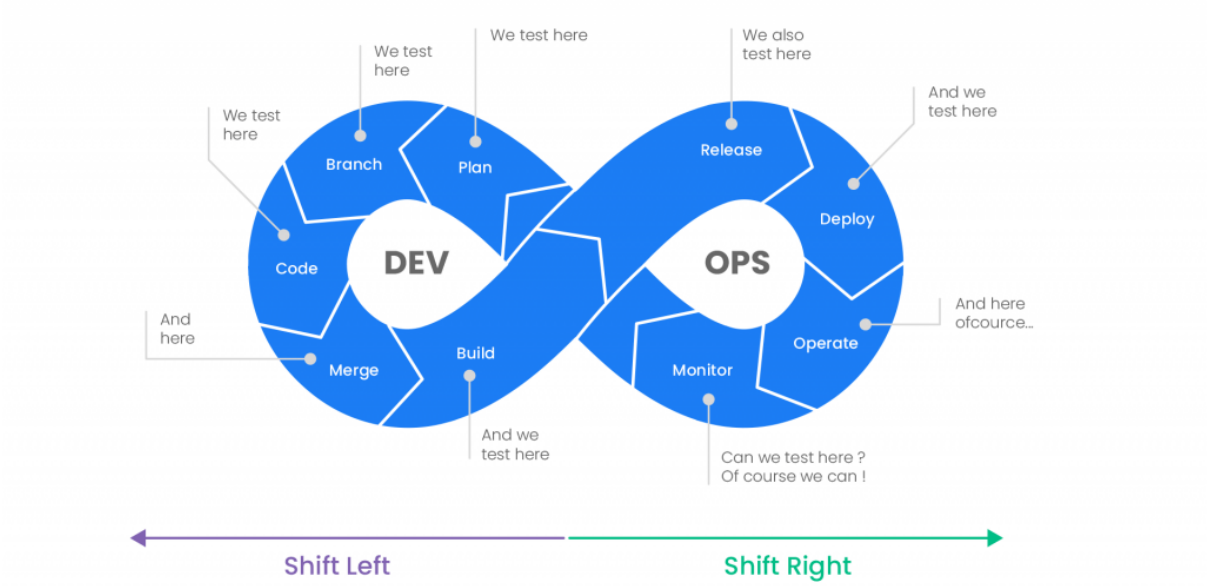
• Cloud Build—the managed service offering from Google Cloud Platform.

• Travis CI—a popular tool for GitHub open-source projects that offers a hosted or self-hosted solution.

• GitLab CI—a free tool from GitLab that can also integrate with other tools via the API

. • CircleCI—a tool that's popular for GitHub projects and has a hosted and selfhosted solution. You can start for free

. • CodeShip—a self-hosted-only solution. You can start with the free version, but it's a paid tool.



CI//CD FLOW DIAGRAM

How does 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 key practice of DevSecOps is integrating security into all DevOps workflows. By conducting security activities early and consistently throughout the software development life cycle (SDLC), organizations can ensure that they catch vulnerabilities as early as possible, and are better able to make informed decisions about risk and mitigation. In more traditional security practices, security is not addressed until the production stage, which is no longer compatible with the faster and more agile DevOps approach. Today, security tools must fit seamlessly into the developer workflow and the CI/CD pipeline in order to keep pace with DevOps and not slow development velocity

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.

What are the benefits of CI/CD?

- Automated testing enables continuous delivery, which ensures software quality and security and increases the profitability of code in production.
- CI/CD pipelines enable a much shorter time to market for new product features, creating happier customers and lowering strain on development.
- The great increase in overall speed of delivery enabled by CI/CD pipelines improves an organization's competitive edge.
- Automation frees team members to focus on what they do best, yielding the best end products.
- Organizations with a successful CI/CD pipeline can attract great talent. By moving away from traditional waterfall methods, engineers and developers are no longer bogged down with repetitive activities that are often highly dependent on the completion of other tasks.

Benefits of the 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