Dev-ops assignment-1

Name: Naveen R **Topic – Continuous integration and continuous deployment (CI/CD)**

USN: 4NI20IS402

Section: A

Continuous Integration and continuous deployment

Continuous integration (CI) and continuous delivery (CD), also known as CI/CD, embodies a culture, operating principles, and a set of practices that application development teams use to deliver code changes more frequently and reliably.

CI/CD is a best practice for devops teams. It is also a best practice in agile methodology. By automating integration and delivery, CI/CD lets software development teams focus on meeting business requirements while ensuring code quality and software security.

CI/CD tools help store the environment-specific parameters that must be packaged with each delivery. CI/CD automation then makes any necessary service calls to web servers, databases, and other services that need restarting. It can also execute other procedures following deployment.

Because the objective is to deliver quality code and applications, CI/CD also requires continuous testing. In continuous testing, a set of automated regression, performance, and other tests are executed in the CI/CD pipeline.

A mature devops team with a robust CI/CD pipeline can also implement *continuous deployment*, where application changes run through the CI/CD pipeline and passing builds are deployed directly to the production environment. Some teams practicing continuous deployment elect to deploy daily or even hourly to production, though continuous deployment isn't optimal for every business application.

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.

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

Tools for CI

CI is mainly a cultural shift, but some tools could help you to get the job done quickly. And you can even start on a dollar a day, according to James Shore. All you need is an old computer, a rubber chicken (really!), and a desk bell. Shore's post is hilarious, and I recommend you read it. He makes a valid point clear: lack of tools isn't an excuse. You can do CI without them.

Nonetheless, tools can help. Here's a list of common tools that you can start using today.

- **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-sourceprojects.
- **Cloud Build**—the managed service offering from Google Cloud Platform.
- **Travis CI**—a popular tool for GitHub open-source projects that offers a hosted orself-hosted solution.
- **GitLab CI**—a free tool from GitLab that can also integrate with other tools via theAPI.
- **CircleCI**—a tool that's popular for GitHub projects and has a hosted and self-hosted solution. You can start for free.
- **CodeShip**—a self-hosted-only solution. You can start with the free version, but it'sa paid tool.

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.

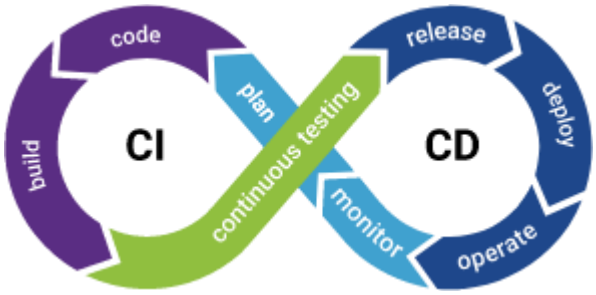

Fig1.1 Cl/Cd Flow Diagram

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.

What AppSec tools are required for CI/CD pipelines?

One of the largest challenges faced by development teams using a CI/CD pipeline is adequately addressing security. It is critical that teams build in security without slowing down their integration and delivery cycles. Moving security testing to earlier in the life cycle is one of the most important steps to achieving this goal. This is especially true for DevSecOps organizations that rely on automated security testing to keep up with the speed of delivery.

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