DevOps Assignment – 1

Name: Aman Narayan Singh

USN: 4NI19IS119

Branch: ISE

Section: ‘B’

Q.1.) What is CI/CD?

CI and CD are two acronyms frequently used in modern development practices and DevOps. CI stands for continuous integration, a fundamental DevOps best practice where developers frequently merge code changes into a central repository where automated builds and tests run. But CD can either mean continuous delivery or continuous deployment.

Continuous integration

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid integration challenges that can happen when waiting for release day to merge changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

Benefits of Continuous Integration:

- Our team will need to write automated tests for each new feature, improvement or bug fix.
- We need a continuous integration server that can monitor the main repository and run the tests automatically for every new commits pushed.

- Developers need to merge their changes as often as possible, at least once a day.
- Less bugs get shipped to production as regressions are captured early by the automated tests.
- Building the release is easy as all integration issues have been solved early.
- Less context switching as developers are alerted as soon as they break the build and can work on fixing it before they move to another task.
- Testing costs are reduced drastically – your CI server can run hundreds of tests in the matter of seconds.
- Your QA team spends less time testing and can focus on significant improvements to the quality culture.

Continuous delivery

Continuous Delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.
This means that on top of automated testing, you have an automated release process and you can deploy your application any time by clicking a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

Benefits of Continuous Delivery:

- We need a strong foundation in continuous integration and your test suite needs to cover enough of your codebase.
- Deployments need to be automated. The trigger is still manual but once a deployment is started there shouldn't be a need for human intervention.

- Our team will most likely need to embrace feature flags so that incomplete features do not affect customers in production.
- The complexity of deploying software has been taken away. Our team doesn't have to spend days preparing for a release anymore.
- We can release more often, thus accelerating the feedback loop with your customers.
- There is much less pressure on decisions for small changes, hence encouraging iterating faster.

Continuous deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a "release day" anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

Benefits of Continuous deployment:

- Our testing culture needs to be at its best. The quality of your test suite will determine the quality of your releases.
- Our documentation process will need to keep up with the pace of deployments.
- Feature flags become an inherent part of the process of releasing significant changes to make sure you can coordinate with other departments (support, marketing, PR...).
- We can develop faster as there's no need to pause development for releases. Deployments pipelines are triggered automatically for every change.

- Releases are less risky and easier to fix in case of problem as you deploy small batches of changes.
- Customers see a continuous stream of improvements, and quality increases every day, instead of every month, quarter or year.

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.

Q.2.) What are feature flags and how it is used?

Feature flags (also commonly known as feature toggles) is a software engineering technique that turns select functionality on and off during runtime, without deploying

new code. This enables teams to make changes without pushing additional code and allows for more controlled experimentation over the lifecycle of features. Because of this, feature flags enable many novel workflows that are incredibly useful to an agile management style and CI/CD environments.

During development, software engineers wrap desired code paths in a feature flag. The following is an example of a basic feature flag written in Javascript:

```
if(featureFlags[‘new-cool-feature’] == true){
    renderNewCoolFeature();
}
```

This code demonstrates a simple statement that checks if a “new-cool-feature” is enabled. Even with advanced frameworks and tools that help manage flag data or injection and removal of the new logic path, feature flags are essentially just "if statements". Therefore, this binary switch is what all the synonyms have in common.

The following are some popular applications of feature flags in an agile environment:

- **Product testing**

  Feature flags can be used to gradually release new product features. It can be unclear upfront if a proposed new feature will be adopted by users and is worth the return on investment. Teams can release a new product feature or a partial idea of the feature in a feature flag and deploy it to a subset of users to gather feedback. The subset of users may be vocal power users who are happy to beta test and review. If the new product idea proves to be a hit, the development team can then roll out the feature flag to a larger user base. If instead the new product idea turns out to be a flop, the development team can easily disable the feature flag and later remove it from the codebase.

- **Conducting experiments**

  Experiments or A/B testing are a primary feature flag example. In their simplest form, feature flags act as a toggle of “on” and “off” state for a feature. Advanced feature flags utilize multiple flags at once to activate different experiences for subsets of users. For example, imagine you divide your user base into thirds. Each third receives its unique

flag and user experience. You can then measure the performance of these three flags against each other to determine the final committed version.

- **Migrations**

  There are times when an application needs a data migration that requires dependent application code changes. These scenarios are sensitive, multi-phase deployment tasks. A database field may be modified, removed, or added in an application database. If the application code is not prepared for this database change, it causes failures and errors. If this happens, it requires a coordinated deploy between the database changes and the application code. Feature flags help to ease the complexity of this scenario by allowing teams to prepare in advance application changes in a feature flag. Once the team makes the changes to the database, they can immediately switch toggle the feature flag to match the application code. This removes the risk and delay of waiting to deploy the new application code and possibly having the deployment fail and desync the application from the database.

- **Canary launches**

  A canary in this context refers to an old, morbid practice where coal miners brought canary birds into the coal mine to detect carbon monoxide. The birds have higher metabolism rates and rapid breathing rates, which led them to succumb to carbon monoxide before the miners. Canary launches in software development occur when a new feature or code change is deployed to a small subset of users to monitor its behavior before releasing it to the full set of users. If the new feature shows any indication of errors or failure, it is automatically rolled back. Feature flags are essential to this process since they restrict the audience pool and can toggle off features easily.

- **System outage**

  A feature flag can also be used as a system outage tool. A web application may utilize a feature flag to “switch off” the entire website for maintenance or downtime. The feature flag can be instrumented throughout the codebase to push sensitive transactions and display outage content to the end users. This can be incredibly helpful when doing sensitive deployments or if an unexpected issue is found and needs to be urgently

resolved. It gives teams the confidence and capability to take a controlled outage if deemed necessary.

- **Continuous deployment**

  Feature flags can be used as an integral component to build a truly continuous deployment system. Continuous deployment is an automated pipeline that takes new code from developers and automatically deploys it to production end users. Continuous deployment depends on layers of automated tests that verify new code behaves as expected against a matching specification as it moves through the pipeline. Feature flags make it safer to deploy continuously by separating code changes from revealing features to users. New code can automatically merge and deploy to production and then wait behind a feature flag. The continuous deployment system can monitor user behavior and traffic, then automatically activate the feature flag. Inversely, the continuous deployment system can monitor the new feature flag code to see if it behaves as expected, and roll it back if not.

Q.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.

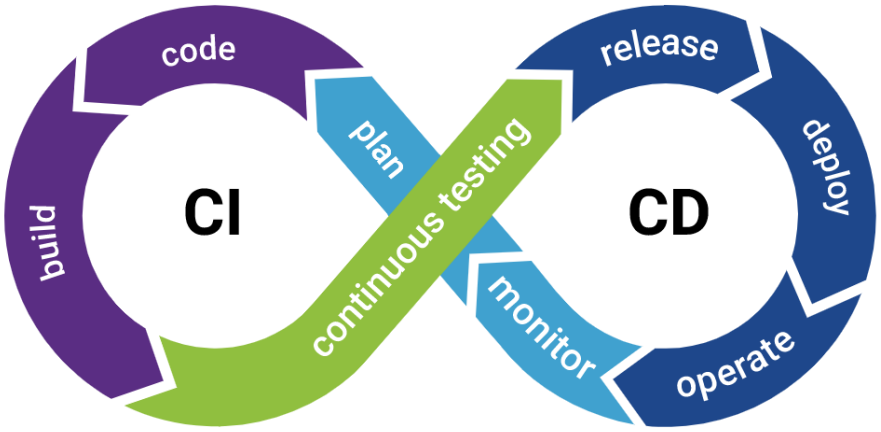

Figure 1: CI/CD Pipeline Phases

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.

Figure 2: CI/CD Processes

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.

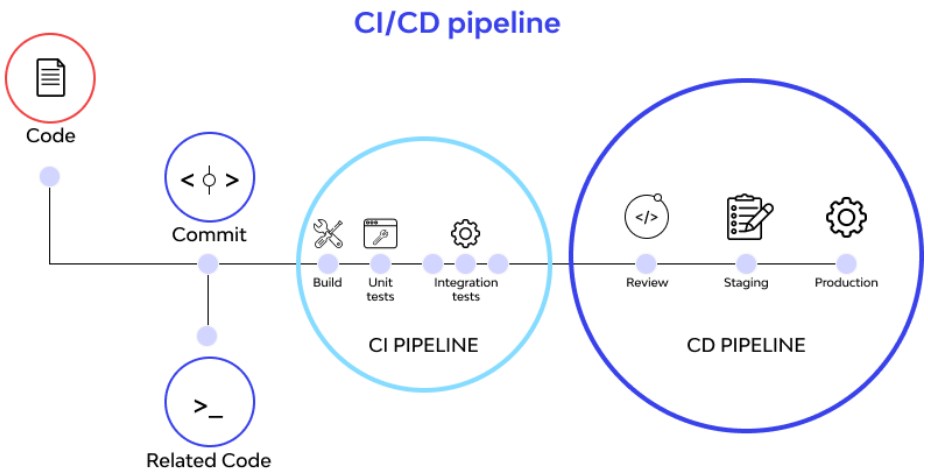

Figure 3: CI/CD pipeline block diagram

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).

Advantages of CI/CD pipelines:

- Builds and testing can be easily performed manually.
- It can improve the consistency and quality of code.
- Improves flexibility and has the ability to ship new functionalities.
- CI/CD pipeline can streamline communication.
- It can automate the process of software delivery.
- Helps you to achieve faster customer feedback.
- CI/CD pipeline helps you to increase your product visibility.
- It enables you to remove manual errors.
- Reduces costs and labour.
- CI/CD pipelines can make the software development lifecycle faster.
- It has automated pipeline deployment.
- A CD pipeline gives a rapid feedback loop starting from developer to client.
- Improves communications between organization employees.
- It enables developers to know which changes in the build can turn to the brokerage and to avoid them in the future.
- The automated tests, along with few manual test runs, help to fix any issues that may arise.