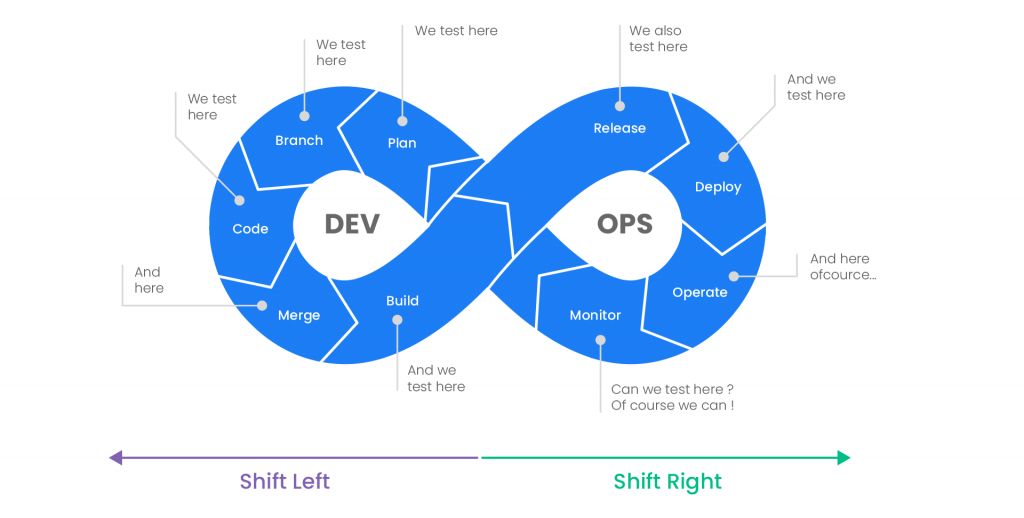
**DEVOPS**

**Assignment-1**

**What is 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.



CI/CD is a method to frequently deliver [apps](https://www.redhat.com/en/topics/cloud-native-apps) to customers by introducing [automation](https://www.redhat.com/en/topics/automation) into the stages of [app development](https://www.redhat.com/en/topics/cloud-native-apps/why-choose-red-hat-cloud-native). The main concepts attributed to CI/CD are continuous integration, [continuous delivery](https://www.redhat.com/en/topics/devops/what-is-continuous-delivery), and continuous deployment. CI/CD is a solution to the problems [integrating](https://www.redhat.com/en/topics/integration) new code can cause for development and operations teams (AKA "integration hell").

Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the [lifecycle of apps](https://www.redhat.com/en/topics/devops/what-is-application-lifecycle-management-alm), from integration and testing phases to delivery and [deployment](https://www.redhat.com/en/topics/automation/what-is-deployment-automation). Taken together, these connected practices are often referred to as a "[CI/CD pipeline](https://www.redhat.com/en/topics/devops/what-cicd-pipeline)" and are supported by development and operations teams working together in an agile way with either a [DevOps](https://www.redhat.com/en/topics/devops) or [site reliability engineering (SRE)](https://www.redhat.com/en/topics/devops/what-is-sre) approach.

**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](https://www.synopsys.com/glossary/what-is-continuous-integration.html) helps streamline code changes, thereby increasing time for developers to make changes and contribute to improved software.

[Continuous delivery](https://www.synopsys.com/glossary/what-is-continuous-delivery.html) (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](https://www.synopsys.com/glossary/what-is-continuous-development.html) 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.

## 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](https://www.synopsys.com/software-integrity/solutions/devsecops.html" \t "_self)*.

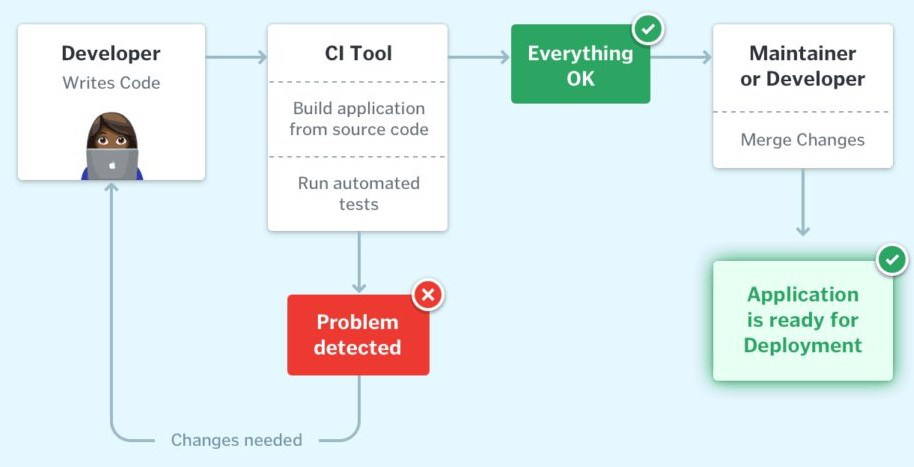
The key practice of DevSecOps is integrating security into all DevOps workflows. By conducting security activities early and consistently throughout the software development life cycle ([SDLC](https://www.synopsys.com/glossary/what-is-sdlc.html)), organizations can ensure that they catch vulnerabilities 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 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.



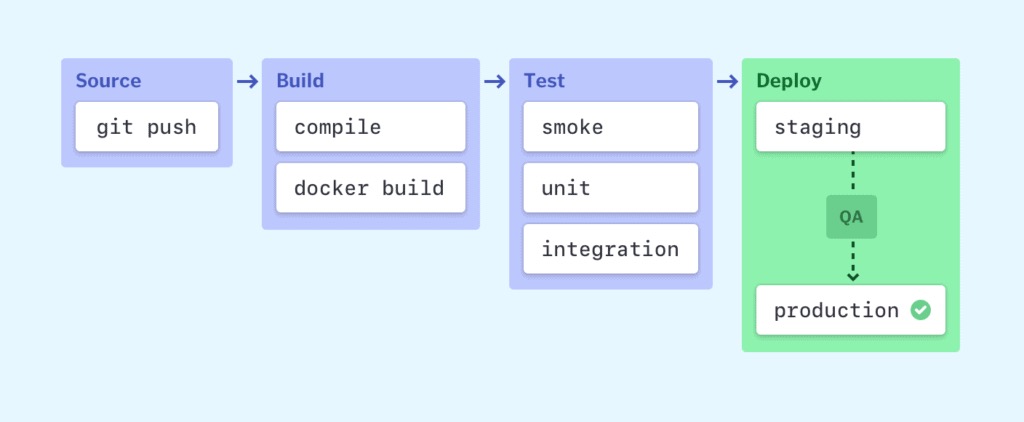
A continuous integration and continuous deployment ([CI/CD](https://www.redhat.com/en/topics/devops/what-is-ci-cd)) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.

By automating CI/CD throughout development, testing, production, and monitoring phases of the software development lifecycle, organizations are able to develop higher quality code, faster. Although it’s possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through automation.

A CI/CD pipeline may sound like overhead, but it isn’t. It’s essentially a runnable specification of the steps that any developer needs to perform to deliver a new version of a software product. In the absence of an automated pipeline, engineers would still need to perform these steps manually, and hence far less productively.

Some popular CI/CD tools include: **Jenkins**: An open source CI automation server, is one of the leading continuous delivery and continuous integration tools on the market. The leading open source automation server, Jenkins provides hundreds of plugins to support building and automating any project.

Most software releases go through a couple of typical stages:



Failure in each stage typically triggers a notification—via email, Slack, etc.—to let the responsible developers know about the cause. Otherwise, the whole team receives a notification after each successful deployment to production.

### 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](https://semaphoreci.com/blog/build-stage) 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](https://semaphoreci.com/resources/cicd-docker-kubernetes).

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](https://semaphoreci.com/blog/automated-testing-cicd) 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](https://semaphoreci.com/blog/20-types-of-testing-developers-should-know) is to do so as we write new code in [test- or behavior-driven development](https://semaphoreci.com/community/tutorials/behavior-driven-development).

Depending on the size and complexity of the project, this phase can last from seconds to hours. Many large-scale projects run tests in multiple stages, starting with [smoke tests](https://semaphoreci.com/community/tutorials/smoke-testing) that perform quick sanity checks to end-to-end integration tests that test the entire system from the user’s point of view. An extensive test suite is typically parallelized to reduce run time.

Failure during the test stage exposes problems in code that developers didn’t foresee when writing the code. It’s essential for this stage to produce feedback to developers quickly, while the problem space is still fresh in their minds and they can [maintain the state of flow](https://semaphoreci.com/blog/2016/11/03/how-bdd-and-continuous-delivery-help-developers-maintain-flow.html).

## Examples of CI/CD pipelines

Pipelines reflect the complexity of a project. Configuring even the simplest pipeline with one job that runs on every code change will save a team many headaches in the future.

On Semaphore, pipelines can easily be extended with multiple sequential or parallel [blocks of jobs](https://docs.semaphoreci.com/essentials/concepts/). Pipelines can also be extended using promotions that are triggered manually or automatically, based on custom conditions.

### A pipeline for a simple program

A pipeline can start very simple. Here’s [an example of a Go project pipeline](https://github.com/semaphoreci-demos/semaphore-demo-go) that:

* Compiles the code,
* Checks code style, and
* Runs automated tests in two parallel jobs:

