**CI/CD (Continues Integration and Continues delivery or deployment)**

1. **What is CI/CD process?**

CI/CD combines two major practices―continuous integration (CI) and continuous delivery/continuous deployment (CD). Together, they form an approach to developing and updating software in which repeatable actions are automated, and human decision-making is excluded as much as possible to minimize errors and tedious processes.

1. **What is continuous Integration?**

Continuous integration means that “members of a team integrate their work frequently” by collaborating on a shred codebase, merging disparate code changes into a version control system, and automatically creating and testing builds

1. **What is continuous delivery/continuous deployment?**

Continuous delivery means that “The software can be released to production at any time”

Automatically deploying builds to a non-production environment after the continuous integration process completes

Though the term is sometimes used interchangeably with continuous delivery, continuous deployment takes things one step further, meaning every change “automatically gets put into production without any operator intervention”

1. **The CI/CD Pipeline?**

The CI/CD pipeline is a series of stages that run in a sequence. When a developer pushes code to the feature branch, it automatically triggers the pipeline execution. On the feature branch, only the CI part is executed. Once the developer merges the feature branch into the mainline branch, the full CI/CD pipeline is executed.

Each part of CI/CD breaks down into specific stages. The CI pipeline includes three major stages: Build, Unit Testing, and Static Code Analysis. The whole cycle is completely automated and takes anywhere from a few seconds to several minutes.

Right after continuous integration comes continuous delivery. The CD pipeline includes four essential stages: Deploy to QA, Automated Testing, Deploy to Staging, and Deploy to Production. The first two stages are automated and should take no longer than a couple of hours.

Diagram

Description automatically generated

**Feature Implementation**

**Code**  
Developers write code on their local machines to implement an app’s functionality.

**Commit**  
Once developers complete a code change, they commit it to the shared source code repository.

**Continuous Integration**

**Build**  
Source code from the repository is compiled into a new build called an “artifact.”

**Unit Testing**A unit test suite runs to check the input and output of every method and function (unit) in code.

**Static Code Analysis**Automated static code analysis runs in parallel with unit testing to examine code for syntax errors, duplications, violations of coding guidelines, security problems, or excessive code complexity.

# **What are the Benefits of CI/CD?**

**Faster Feedback, Fewer Errors:** CI/CD is designed to yield feedback as quickly as possible, ideally within minutes after pushing code to the VCS. This rapid response guarantees that developers pay attention to the result and that they can fix any problems quickly because the changes they just committed will be fresh in their minds. Faster feedback and fewer errors mean less downtime.

**Less Manual Effort:** With automated tests that can find issues in each integration, CI helps eliminate the problem of comparing and merging blocks of code that diverge from the main branch over a long period—a necessary but tedious task in the traditional software release cycle.

CD removes all human intervention from a continuous flow to the production environment. Automation eliminates manual tasks and allows developers to focus on developing software.

**Faster Root Cause Identification:** CI/CD makes the processes of testing and deploying transparent. All developers can see which commits

 are included in each build, issues are visible almost immediately, and developers can quickly find the source of the problem (or which code change introduces the issue). This reduces the guesswork in identifying the root cause of any problems and holds developers accountable. The developer who worked on a section of code with problems is typically the best person to resolve them.

**Faster Time-to-Market**

CI/CD encompasses the tools and processes needed to facilitate frequent, incremental software and configuration updates. It facilitates rolling out well-tested features in minutes or hours, rather than days or weeks.

**Actionable Visibility**

The CI/CD system—which includes both the practices and their implementation with appropriate tools—enables visibility inside the pipeline at every stage of the release cycle, giving stakeholders and managers the awareness required to make better decisions and meet goals efficiently. They can easily identify code changes in the pipeline, understand the current status of those changes, and determine which activities have happened—or are happening—within the CI/CD pipeline.

**More Auditability**

The CI/CD system includes a detailed history of every software release, including its progression from development to production. This auditability ensures that the code passed all security checks and only authorized builds are deployed. The event timelines related to each environment can be verified.

# **Following the Basic Rules of CI/CD?**

**Use Version Control**

Track all code in a version control system. Make sure the VCS is the single source of truth. Nothing can be managed manually or off the record.

**Integrate Frequently in Small Batches**

Merge code in small pieces as frequently as possible. Because automated tests are run on every commit, it’s immediately clear which set of changes caused the issue. Developers can easily trace a broken build back to the exact lines of code that caused the problem. The more frequently developers commit, the smaller the change is. The smaller the change is, the faster and easier it is to find and fix issues.

Frequent integrations also minimize merge conflicts―the main root cause of integration issues. The more frequently developers merge code, the fewer merge conflicts arise, the easier testing is, and the fewer defects there are to fix.

**Build Once, Deploy Often**

Once a specific build (also known as the “artifact”) is created, run all tests against that build to confirm it is ready to go to production. Deploy the same artifact to at least one or two environments that match production as closely as possible.

**Automate End-to-End**

Building, testing, and deploying code without manual intervention is the key to reliable and reproducible updates. Even if you’re not planning to deploy code continuously to production, the final production deployment step should run fully unattended after being triggered by a human.

**Share Responsibility**

Fix the pipeline when something goes wrong. Do not push any new code until the problem is resolved—think of it like halting the assembly line in a factory. Fixing the build before resuming development work is the entire team’s responsibility.

**Build Fast, Fix Fast**

Slow-build processes are counterproductive. Strive to eliminate redundant and time-consuming steps. Ensure that your build system has enough agents—environments in which builds of the CI/CD pipeline are run—and that the agents have sufficient system resources to build quickly**.**