CI/CD

**1. What is continuous integration?**

[Continuous Integration](https://semaphore.io/continuous-integration) (CI) is a software development methodology where developers — following the trunk-based model — merge their changes to the main branch many times per day.

CI is supported by automated tests and a build server that runs them on every change. As a result, [failures are made visible as soon as they are introduced](https://qameta.io/blog/test-automation-as-a-service/?utm_source=semaphore&utm_medium=blog&utm_campaign=semaphore-cicd-questions&utm_content=2022-06-07) and can be fixed within minutes.

**2. How do CI and version control relate to one another?**

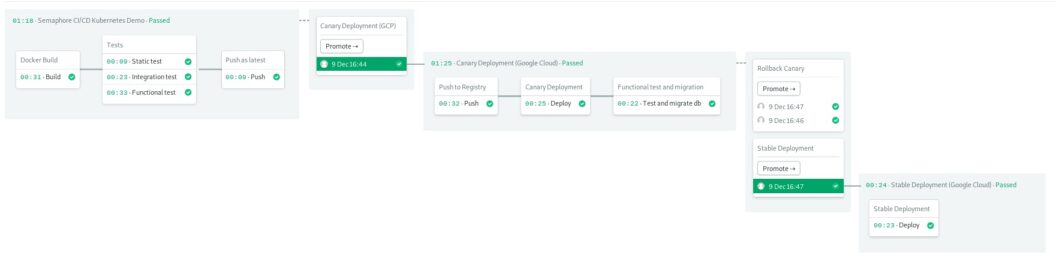
Every change in the code must trigger a continuous integration process. This means that a CI system must be connected with a Git repository to detect when changes are pushed, so tests can be run on the latest revision.

**3. What’s the difference between continuous integration, continuous delivery, and continuous deployment?**

**Continuous integration** (CI) executes the sequence of steps required to build and test the project. CI runs automatically on every change committed to a shared repository, offering developers quick feedback about the project’s state.

[**Continuous delivery**](https://semaphore.io/blog/2017/07/27/what-is-the-difference-between-continuous-integration-continuous-deployment-and-continuous-delivery.html)(CD) is an extension of CI. Its goal is to automate every step required to package and release a piece of software. The output of a continuous delivery pipeline takes the form of a deployable binary, package, or container.

**Continuous deployment** is an optional step-up from continuous delivery. It is a process that takes the output from the delivery pipeline and deploys it to the production system in a safe and automated way.



**4. Name some benefits of CI/CD**

* **Less risk**: automated tests reduce the chance of introducing bugs, creating a safety net that increases the developer’s confidence in their code.
* **More frequent releases**: the automation provided by continuous delivery and continuous deployment allows developers to release and deploy software safely many times per day.
* **Improved productivity**: freed from the manual labor of building and testing the code, developers can focus on the creative aspects of coding.
* **Elevated quality**: CI acts as a quality gate, preventing code that is not up to standards from getting released.
* **Better design**: the iterative nature of continuous integration lets developers work in small increments, allowing a higher degree of experimentation, which leads to more innovative ideas.

**5. What are the most important characteristics in a CI/CD platform?**

* **Reliability**: the team depends on the CI server for testing and deployment, so it must be reliable. An unreliable CI/CD platform can block all development work.
* **Speed**: the platform should be fast and scalable to obtain results in a few minutes.
* **Reproducibility**: the same code should always yield the same results.
* **Ease of use**: easy to configure, operate, and troubleshoot.

**6. What is the build stage?**

The [build stage](https://semaphore.io/blog/build-stage) is responsible for building the binary, container, or executable program for the project. This stage validates that the application is buildable and provides a testable artifact.

**7. What’s the difference between a hosted and a cloud-based CI/CD platform?**

A hosted CI server must be managed like any other server. It must be first installed, configured, and maintained. Upgrades and patches must be applied to keep the server secure. Finally, failures in the CI server can block development and stop deployments.

On the other hand, a cloud-based CI platform does not need maintenance. There’s nothing to install or configure, so organizations can immediately start using them. The cloud provides all the machine power needed, so scalability is not a problem. Finally, the reliability of the platform is guaranteed by SLA.

**8. How long should a build take?**

Developers should get results from their CI pipeline in [less than 10 minutes](https://semaphore.io/blog/2017/03/02/what-is-proper-continuous-integration.html). That’s the longest time that’s practical to wait for results.

**9. Is security important in CI/CD? What mechanisms are there to secure it?**

Yes. CI/CD platforms have access to all kinds of sensitive data such as API keys, private repositories, databases, and server passwords. An improperly secured CI/CD system [is a prime target for attacks](https://research.nccgroup.com/2022/01/13/10-real-world-stories-of-how-weve-compromised-ci-cd-pipelines/) and can be exploited to release compromised software or to get unauthorized access. A CI/CD platform must support mechanisms to securely manage secrets, and control access to logs and private repositories.

**10. Can you name some deployment strategies?**

* **Rolling Deployment:** This is a phased approach where the new version of the application gradually replaces the old one. It updates instances one by one or in small batches. This strategy minimizes downtime, but it requires that both the old and new versions of the application can coexist and function simultaneously. If a deployment fails, it can be stopped, and the remaining instances will continue to run the old version.
* **Blue-Green Deployment:** This strategy uses two identical, separate production environments: a **"blue"** environment (the old version) and a **"green"** environment (the new version). While the blue environment serves live traffic, the new version is deployed and tested in the green environment. Once a new version is validated, traffic is instantly switched from blue to green. The old blue environment is kept as a backup for a quick rollback if something goes wrong with the new version.
* **Canary Release:** A canary release is a way to deploy a new version to a small, randomly selected subset of users. This small group of users detect any issues with the new version before it's rolled out to everyone. If the new version performs well, it gradually rolls out to more users. If it fails, the traffic is routed back to the old version with minimal impact. This is a great way to test new features with real traffic.

**11. What is version control?**

Version control is a set of practices and tools for managing codebases. Developers use version control to keep track of every line of code, and share, review, and synchronize changes among a team.

**12. What is a CI/CD pipeline? Describe its key stages.**

A pipeline is an automated workflow that guides code from version control to production. Key stages for a Java application typically include:

* **Source:** Developers commit code to a Git repository.
* **Build:** A tool like Maven or Gradle compiles the Java code, manages dependencies, and packages the application into an artifact (e.g., a .jar or .war file).
* **Test:** Automated tests (unit, integration, and end-to-end) are executed against the application artifact.
* **Deploy:** The artifact is deployed to a staging or production environment.

**13. Why is automated testing crucial for CI/CD? What types of tests are essential?**

Automated testing is the backbone of CI/CD. It ensures that new code changes don't introduce regressions and provides rapid feedback on the application's stability. Essential test types include:

* **Unit Tests:** Verify individual components or methods. These are fast and run on every commit.
* **Integration Tests:** Ensure different modules or services work together correctly.
* **Static Code Analysis:** Tools like SonarQube analyze code quality and security without executing the code.

**14. What is a "Quality Gate" in a CI/CD pipeline?**

A Quality Gate is a set of predefined conditions that must be met for a build to be allowed to proceed to the next stage. A typical quality gate might check for:

* A minimum level of code coverage (e.g., 80%).
* Zero new bugs or security vulnerabilities.
* A successful run of all unit and integration tests.

**15. What is the core purpose of Continuous Delivery (CD)?**

The core purpose of Continuous Delivery is to ensure that a codebase is always in a state where it can be deployed to production at any time. It automates the process of building, testing, and packaging the application, so the decision to release is a simple, manual action, not a complex.

**16.** **What is a deployment pipeline, and what are its key stages?**

A deployment pipeline is the automated workflow for taking a built artifact and deploying it through a series of environments. The key stages are:

* **Staging/QA:** The artifact is deployed to an environment that mimics production for final testing and validation.
* **Pre-Production/UAT (User Acceptance Testing):** An optional stage for business users to test the new features.
* **Production:** The artifact is deployed to the live environment.

**17. What is a "rollback" and why is it important in CD?**

A rollback is the process of reverting a deployed application to a previous, stable version. It's a critical safety net in a CD pipeline. If a new deployment introduces a critical bug, an automated rollback allows the team to quickly revert to a working state, minimizing downtime and user impact.

**18. When would you choose a Blue-Green deployment over a Rolling deployment?** Choose Blue Green when:

* You need zero downtime and a quick, safe rollback.
* The new version is not backward compatible with the old version.
* You want to fully test the new version in a production-like environment before any live traffic hits it.

**19. Explain the concept of "immutable infrastructure.**

Immutable infrastructure is the practice of never changing a server or its configuration after it has been deployed. If you need to update the application or configuration, you don't modify the existing server; you create a new server with the updated version and replace the old one.

**20. What is a "release gate" or "approval gate" in a CD pipeline?**

A release gate is a manual or automated checkpoint in the pipeline that a deployment must pass before it can proceed to the next stage (e.g., production). This is a feature of Continuous Delivery, not Continuous Deployment, and is used for things like manual sign-offs or final security audits.

**21. How does monitoring "shift left" in a mature CD pipeline?**

Shifting left means bringing practices from later stages of development to earlier ones. In CD, this means that monitoring isn't just for production. You should also be monitoring your staging and testing environments to catch performance and stability issues before they ever reach production.

**22. What role do containers play in Continuous Delivery?**

Containers (e.g., Docker) are a used for CD because they:

* **Package everything:** The application and all its dependencies (like the JDK) are bundled into a single, portable unit.
* **Provide consistency:** The container runs the same way on a developer's machine as it does in staging and production.
* **Simplify deployment:** The deployment process is reduced to simply running the containerized application.

**23. What role does Docker play in Continuous Delivery?**

Docker allows you to package your application and all its dependencies into a single, portable container. This ensures that the application runs consistently across different environments (from a developer's machine to production), simplifying the deployment process.

**24. How would you monitor a new deployment in production?**

We can monitor key metrics like application performance, error rates, and resource utilization. Tools like Prometheus and Grafana are often used for this.

**25. What are the key differences between a monolith and a microservices architecture in a CD pipeline?**

* **Monolith:** A single, large application. The CD pipeline is simpler, as there's one codebase and one artifact to build and deploy. However, every change requires full redeployment of the entire application.
* **Microservices:** An application composed of small, independent services. The CD pipeline is more complex, as each microservice has its own pipeline. This allows for independent and faster deployments but requires a more sophisticated orchestration system.