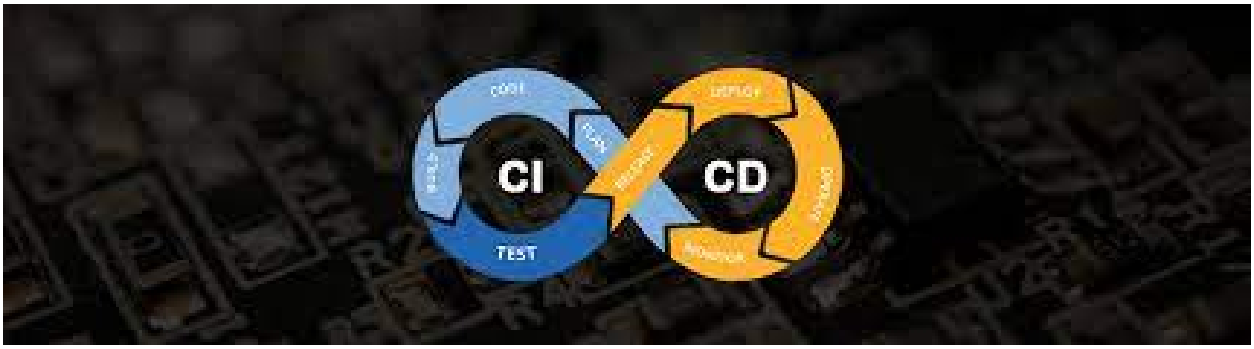
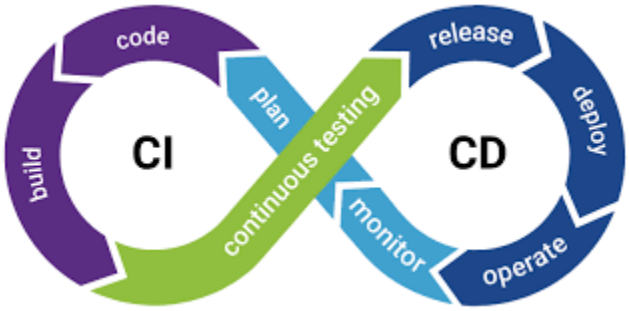


CI/CD - Interview questions - PART 1

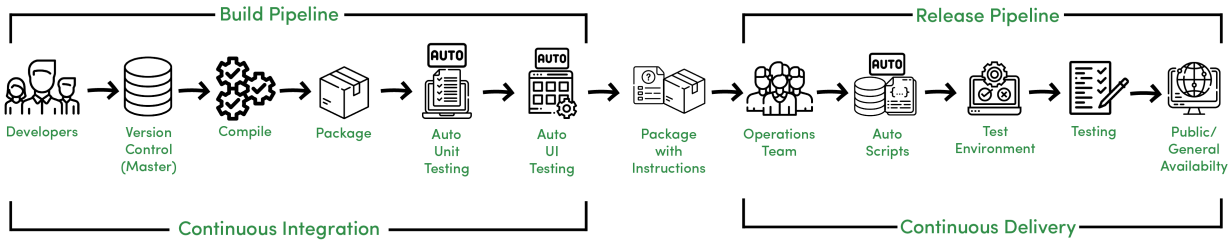
Created	@November 5, 2023 1:14 PM
Tags	CI/CD DevOps Interview Q&A
Topic	CI/CD
Youtube channel	DevOps Motivation



1. What is Continuous Integration?

A development practice where developers integrate code into a shared repository frequently. It can range from a couple of changes every day or a week to a couple of changes in one hour in larger scales.

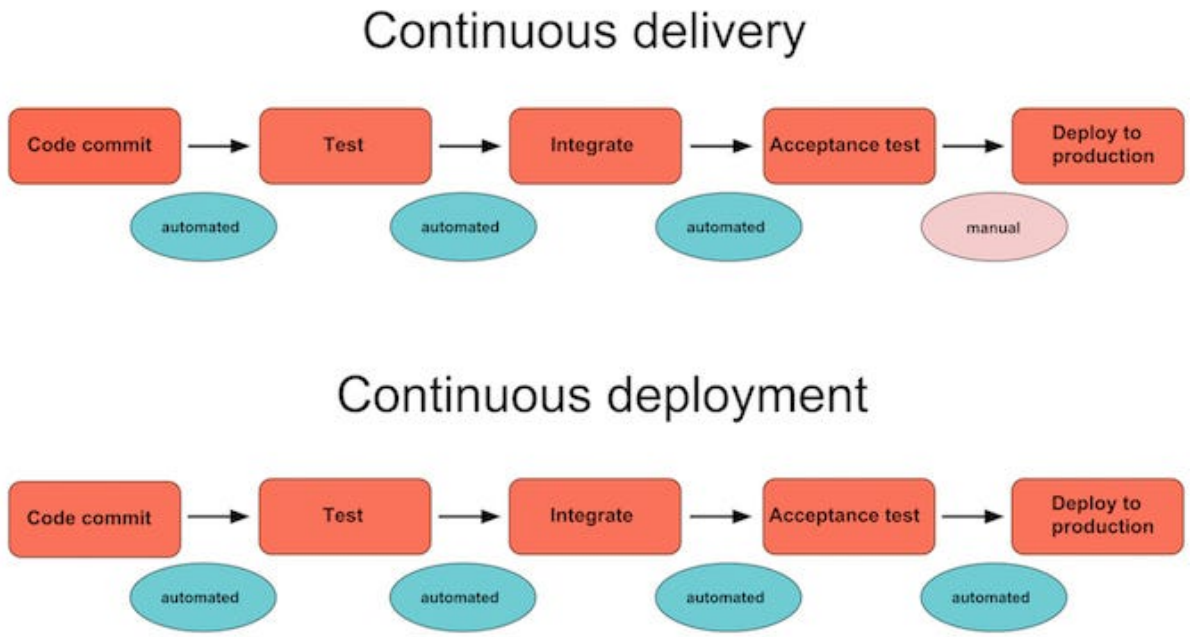
Each piece of code (change/patch) is verified, to make the change is safe to merge.
Today, it's a common practice to test the change using an automated build that makes sure the code can be integrated.



2. What is Continuous Deployment?

A development strategy used by developers to release software automatically into production where any code commit must pass through an automated testing phase. Only when this is successful is the release considered production worthy. This eliminates any human interaction and should be implemented only after production-ready pipelines have been set with real-time monitoring and reporting

of deployed assets. If any issues are detected in production it should be easy to rollback to previous working state.

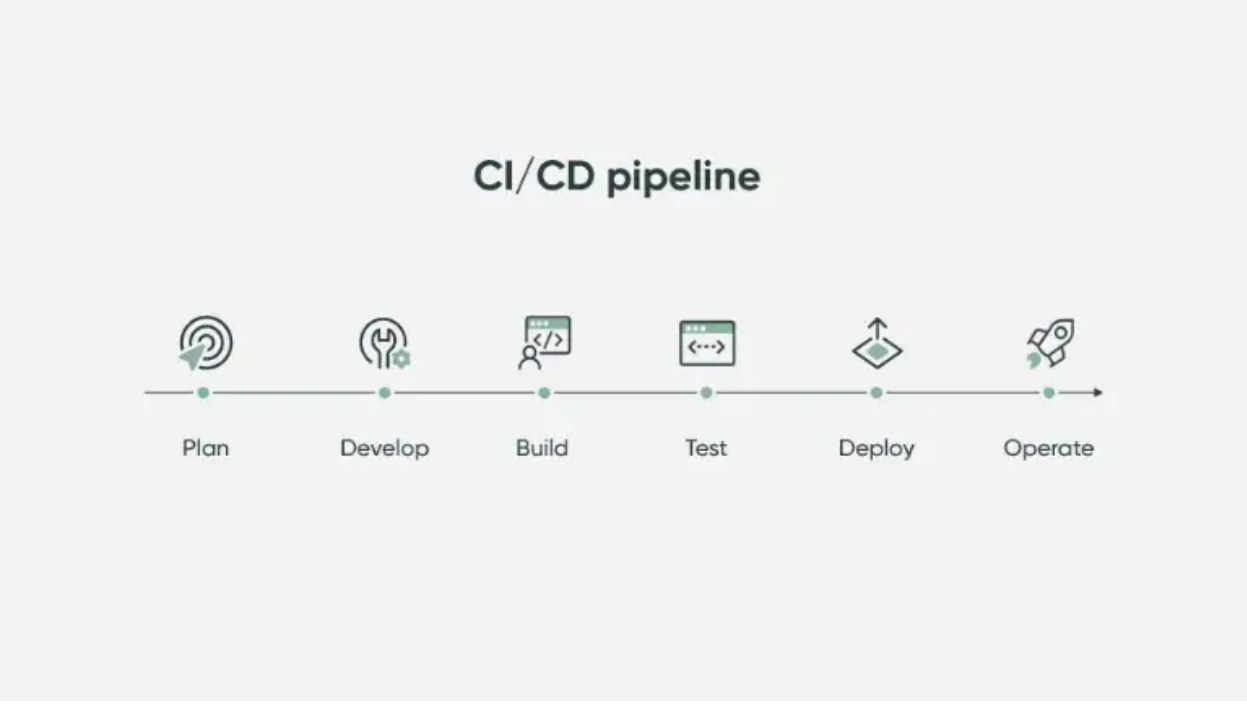


3. Can you describe an example of a CI (and/or CD) process starting the moment a developer submitted a change/PR to a repository?

There are many answers for such a question, as CI processes vary, depending on the technologies used and the type of the project to where the change was submitted. Such processes can include one or more of the following stages:

1. **Checkout**
2. **Build**
3. **Test**
4. **Static Analysis**
5. **Review Apps (Optional)**

6. **Review and Approval**
7. **Merge to Main**
8. **Artifact Storage**
9. **Staging Deployment (Optional)**
10. **Automated Testing in Staging (Optional)**
11. **Manual Testing (Optional)**
12. **Security Scans (Optional)**
13. **Production Deployment (Optional)**
14. **Monitoring and Alerts**
15. **Rollback (Optional)**
16. **Reporting and Notifications**
17. **Completion and Cleanup**



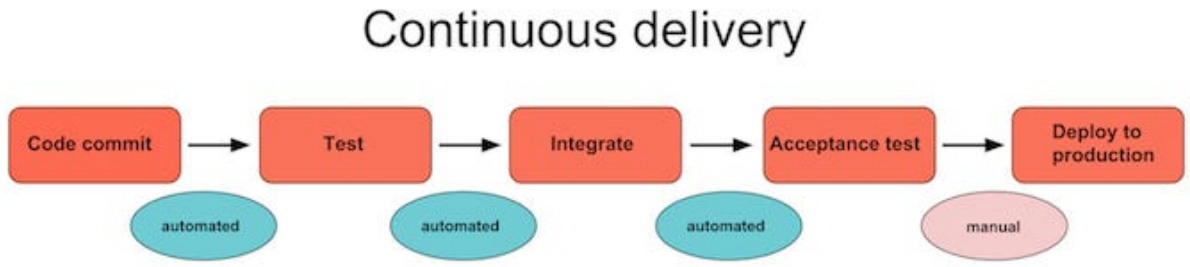
An example of one possible answer:

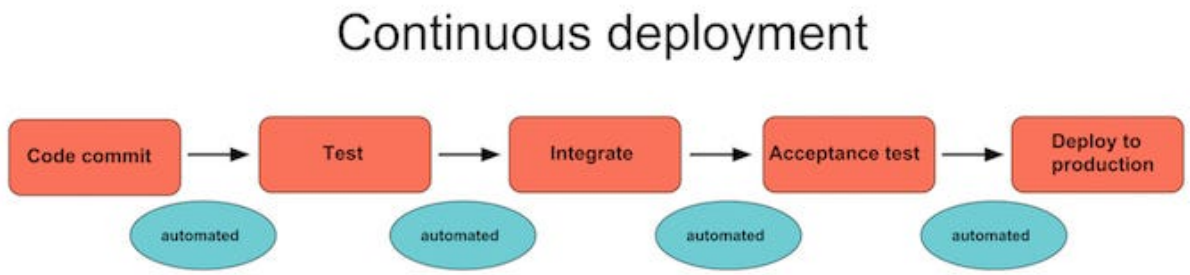
A developer submitted a pull request to a project. The PR (pull request) triggered two jobs (or one combined job). One job for running lint test on the change and the second job for building a package which includes the submitted change, and running multiple api/scenario tests using that package. Once all tests passed and the change was approved by a maintainer/core, it's merged/pushed to the repository. If some of the tests failed, the change will not be allowed to merged/pushed to the repository.

A complete different answer or CI process, can describe how a developer pushes code to a repository, a workflow then triggered to build a container image and push it to the registry. Once in the registry, the k8s cluster is applied with the new changes.

4. What is Continuous Delivery?

A development strategy used to frequently deliver code to QA and Ops for testing. This entails having a staging area that has production like features where changes can only be accepted for production after a manual review. Because of this human entanglement there is usually a time lag between release and review making it slower and error prone as compared to continuous deployment.





5. What is difference between Continuous Delivery and Continuous Deployment?

Both encapsulate the same process of deploying the changes which were compiled and/or tested in the CI pipelines.The difference between the two is that Continuous Delivery isn't fully automated process as opposed to Continuous Deployment where every change that is tested in the process is eventually deployed to production. In continuous delivery someone is either approving the deployment process or the deployment process is based on constraints and conditions (like time constraint of deploying every week/month/...)

6. What CI/CD best practices are you familiar with? Or what do you consider as CI/CD best practice?

Commit and test often.

Testing/Staging environment should be a clone of production environment.

Clean up your environments (e.g. your CI/CD pipelines may create a lot of resources. They should also take care of cleaning up everything they create)

The CI/CD pipelines should provide the same results when executed locally or remotely

Treat CI/CD as another application in your organization. Not as a glue code.

On demand environments instead of pre-allocated resources for CI/CD purposes

Stages/Steps/Tasks of pipelines should be shared between applications or microservices (don't re-invent common tasks like "cloning a project")

7. You are given a pipeline and a pool with 3 workers: virtual machine, baremetal and a container. How will you decide on which one of them to run the pipeline?

The decision on which type of worker (virtual machine, bare-metal, or container) to use for running a pipeline would depend on several factors, including the nature of the pipeline, the requirements of the software being built, the available resources, and the specific goals and constraints of the development and deployment process. Here are some considerations that can help in making the decision:

1. Pipeline requirements
2. Resource availability
3. Scalability and flexibility
4. Deployment and isolation requirements
5. Security considerations
6. Development and operational workflows
7. Cost considerations

Based on these considerations, the appropriate choice of worker (virtual machine, bare-metal, or container) for running the pipeline would be determined by weighing the pros and cons of each option and aligning with the specific requirements, resources, and goals of the development and deployment process. It may also be useful to consult with relevant stakeholders, such as developers,

operations, and infrastructure teams, to gather input and make an informed decision.

8. Where do you store CI/CD pipelines? Why?

There are multiple approaches as to where to store the CI/CD pipeline definitions:

App Repository - store them in the same repository of the application they are building or testing (perhaps the most popular one)
Central Repository - store all organization's/project's CI/CD pipelines in one separate repository (perhaps the best approach when multiple teams test the same set of projects and they end up having many pipelines)
CI repo for every app repo - you separate CI related code from app code but you don't put everything in one place (perhaps the worst option due to the maintenance)

The platform where the CI/CD pipelines are being executed (e.g. Kubernetes Cluster in case of Tekton/OpenShift Pipelines).

9. How do you perform plan capacity for your CI/CD resources? (e.g. servers, storage, etc.)

Capacity planning for CI/CD resources involves estimating the resources required to support the CI/CD pipeline and ensuring that the infrastructure has enough capacity to meet the demands of the pipeline. Here are some steps to perform capacity planning for CI/CD resources:

1. Analyze workload
2. Monitor current usage
3. Identify resource bottlenecks
4. Forecast future demand
5. Plan for growth
6. Consider scalability and elasticity
7. Evaluate cost and budget
8. Continuously monitor and adjust

By following these steps, you can effectively plan the capacity for your CI/CD resources, ensuring that your pipeline has sufficient resources to operate efficiently and meet the demands of your development process.

10. How would you structure/implement CD for an application which depends on several other applications?

Implementing Continuous Deployment (CD) for an application that depends on several other applications requires careful planning and coordination to ensure smooth and efficient deployment of changes across the entire ecosystem. Here are some general steps to structure/implement CD for an application with dependencies:

1. Define the deployment pipeline
2. Automate the deployment process
3. Version control and dependency management
4. Continuous integration and testing
5. Rolling deployments
6. Monitor and manage dependencies
7. Testing across the ecosystem
8. Rollback and recovery strategies
9. Security and compliance
10. Documentation and communication

Implementing CD for an application with dependencies requires careful planning, coordination, and automation to ensure efficient and reliable deployments. By following best practices such as automation, version control, testing, monitoring, rollback strategies, and effective communication, you can ensure a smooth and successful CD process for your application ecosystem.

11. How do you measure your CI/CD quality? Are there any metrics or KPIs you are using for measuring the quality?

Measuring the quality of CI/CD processes is crucial to identify areas for improvement, ensure efficient and reliable software delivery, and achieve continuous improvement. Here are some commonly used metrics and KPIs (Key Performance Indicators) to measure CI/CD quality:

1. Build Success Rate: This metric measures the percentage of successful builds compared to the total number of builds. A high build success rate indicates that the majority of builds are successful and the CI/CD pipeline is stable.

2. Build and Deployment Time: This metric measures the time it takes to build and deploy changes from code commit to production. Faster build and deployment times indicate shorter feedback loops and faster time to market.

3. Deployment Frequency: This metric measures the frequency of deployments to production within a given time period. Higher deployment frequency indicates faster release cycles and more frequent updates to production.

4. Mean Time to Detect (MTTD): This metric measures the average time it takes to detect issues or defects in the CI/CD pipeline or production environment. Lower MTTD indicates faster detection and resolution of issues, leading to higher quality and more reliable deployments.

5. Mean Time to Recover (MTTR): This metric measures the average time it takes to recover from issues or incidents in the CI/CD pipeline or production environment. Lower MTTR indicates faster recovery and reduced downtime, leading to higher availability and reliability.

6. Feedback Loop Time: This metric measures the time it takes to receive feedback on code changes, including code reviews, test results, and other feedback mechanisms. Faster feedback loop times enable quicker iterations and faster improvements in the CI/CD process.

7. Customer Satisfaction: This metric measures the satisfaction of end-users or customers with the quality and reliability of the deployed software. Higher customer satisfaction indicates that the CI/CD process is delivering high-quality software that meets customer expectations.

These are just some examples of metrics and KPIs that can be used to measure the quality of CI/CD processes. It's important to choose metrics that align with the goals and objectives of your organization and regularly track and analyze them to continuously improve the CI/CD process and ensure high-quality software delivery.

SUBSCRIBE TO OUR CHANNEL FOR MORE INTERESTING CONTENT !!

— Devops Motivation —