Development Operations: DevOps.

“Software Development ("dev") and IT Operations ("ops")”

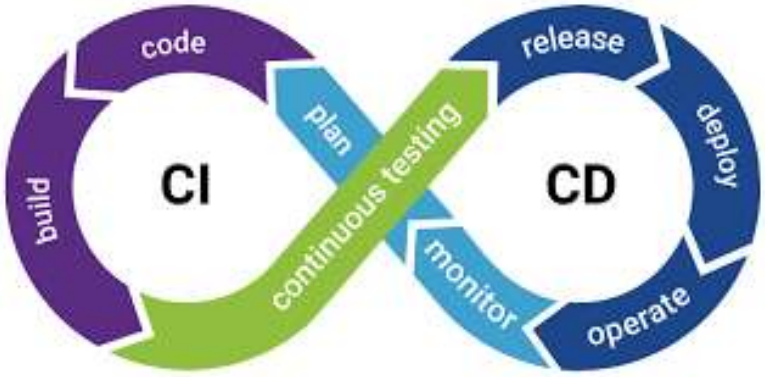
DevOps.

It is a working mechanism through which development and operations teams are combined into one, where both work across the entire application lifecycle from development and testing to deployment and operations. When QA and security teams are added to this process, this mechanism becomes integrated, and this mechanism is referred to as DevSecOps.

Continuous integration And Continuous delivery are a DevOps software development best practice. In the continuous integration developers regularly merge their code changes into a central repository, after which automated builds and tests are run.The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When properly implemented, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

In short, it is a set of processes that help software development teams deliver code changes more frequently and reliably. CI/CD is part of DevOps, which helps shorten the software development lifecycle.



Why is Continuous Integration and Continuous Delivery Needed?

Without continuous integration, developer collaboration is a tedious manual process of coordinating code updates and merges. In the past, developers on a team might work in isolation for an extended period of time and only merge their changes to the master branch once their work was completed. This made merging code changes difficult and time-

consuming, and also resulted in bugs accumulating for a long time without correction. These factors made it harder to deliver updates to customers quickly.

Continuous Delivery lets developers automate testing beyond just unit tests so they can verify application updates across multiple dimensions before deploying to customers. These tests may include UI testing, load testing, integration testing, API reliability testing, etc. This helps developers more thoroughly validate updates and pre-emptively discover issues. With the cloud, it is easy and cost-effective to automate the creation and replication of multiple environments for testing, which was previously difficult to do on-premises.

How Does Continuous Integration Work?

With continuous integration, developers frequently commit to a shared repository using a version control system such as Git. A continuous integration service automatically builds and runs unit tests on the new code changes to immediately surface any errors.

Continuous Integration Tools:-

1. Bitbucket Pipelines.
2. Jenkins.
3. AWS Code Pipeline.
4. Circle CI.
5. Azure Pipelines.
6. GitLab.
7. Atlassian Bamboo.

Continuous Delivery vs. Continuous Deployment?

With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment. There can be multiple, parallel test stages before a production deployment. The difference between continuous delivery and continuous deployment is the presence of a manual approval to update to production. With continuous deployment, production happens automatically without explicit approval.

The Organizations that have successfully implemented CI/CD pipelines?

1. Amazon.
2. Netflix.
3. Target
4. Walmar.
5. Nordstrom.
6. Facebook
7. Etsy.
8. Adobe.
9. Sony Pictures Entertainment
10. Fidelity Worldwide Investment

Disadvantages of DevOps:

Even though DevOps has many advantages, there are also some disadvantages. Let's discuss the cons of DevOps in this section.

Initial Implementation Challenges:

1 - Cultural resistance occurs if the team isn't used to functioning as a single unit.

Security Concerns:

1 - Vulnerabilities and breaches can occur since the security team still works in silos.

2 - Access control and permissions are assigned to particular people, denying others the same rights.

Cost Considerations:

1 - Investments in tools and training required for complexities can be expensive.

2 - Ongoing maintenance expenses have to be considered.

Continuous Monitoring and Maintenance:

1 - DevOps is resource and time-intensive since a lot of people and money goes into this.

2 - Dependency on automation can be harmful.

Toolchain Complexity:

1 - The learning curve is high due to new skills being acquired.

2 - Integration challenges can occur due to multiple teams being merged into one.

Access Control and Permissions:

1 - Implementing Role-Based Access Control (RBAC) is often not applied, hence access isn't specified.

2 - Periodic permission audits aren't implemented, lacking deep checks of who has permission.

Strategies for Overcoming the Disadvantages of DevOps:

Cultural Transformation:

Promoting a DevOps culture through leadership support can result in a cultural transformation. A top-down approach is needed through which DevOps practices can be slowly integrated into the work culture.

Initial Implementation:

The implementation can happen in stages. First, pilot projects can be done in a DevOps style, and then education and training programs can be put in place. This helps teams to stay calm with a sudden shift in work culture.

Security Measures:

A way to circumvent the issue of security measures by introducing more regular audits for deep checking. Additionally, rather than the security team working in silos, they can be integrated from the start.

Cost Management:

Costs can be managed through resource optimization. Automation and team integration can help reduce costs. A quick cost-benefit analysis will show that DevOps leads to faster software turnout in the long term, minimizing the harms of high short-term costs.

Continuous Monitoring and Maintenance:

Since manual tasks will be automated, continuous monitoring and maintenance become easily manageable. With the addition of regular process reviews, maintenance, and monitoring become much easier.

Toolchain Simplification:

The software delivery toolchain can be simplified with careful tool selection and extremely comprehensive documentation of the toolchain.

Access Control:

Access control can be improved by implementing RBACs (role-based access control), to monitor who gets access to what. Additionally, periodic permission audits can help improve security as well.