Introduction to CI/CD in DevOps

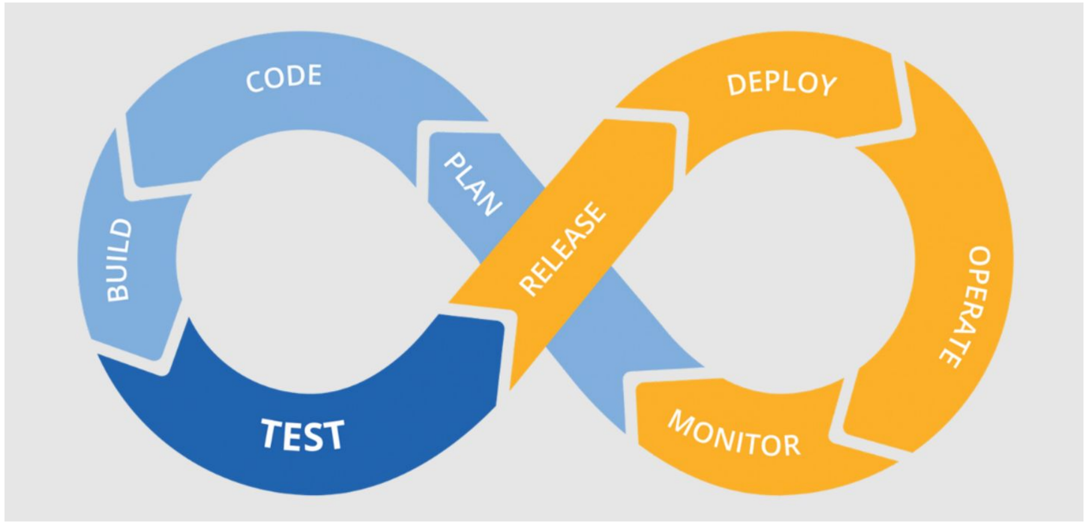
Streamlining Software Delivery with Continuous Integration and Continuous Deployment

Traditional Software Development Process

- Sequential Process
- Manual Integration and Testing
- Slow Release Cycles
- Limited Automation
- Feedback Loop Issues

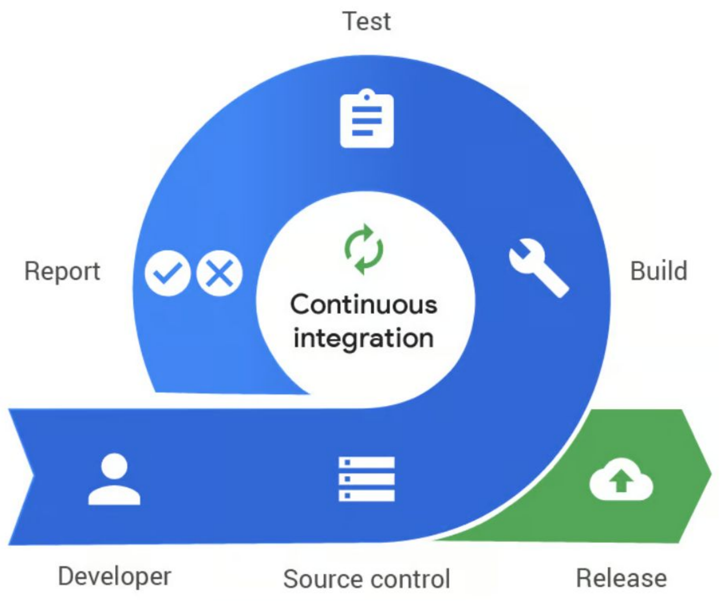
What is CI/CD?

CI/CD, which stands for continuous integration and continuous delivery/deployment, aims to streamline and accelerate the software development lifecycle.



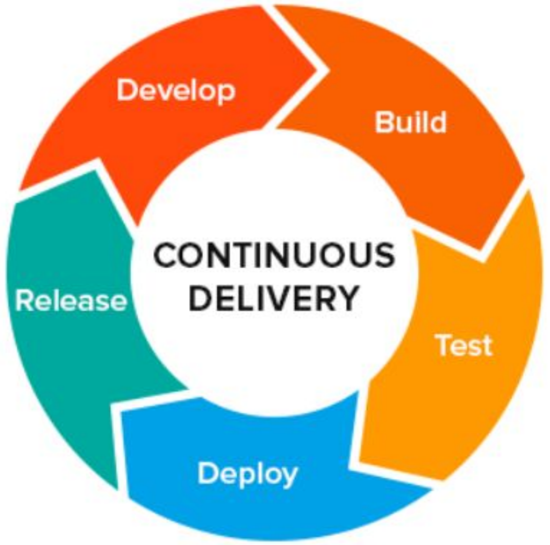
Continuous Integration (CI)

Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.



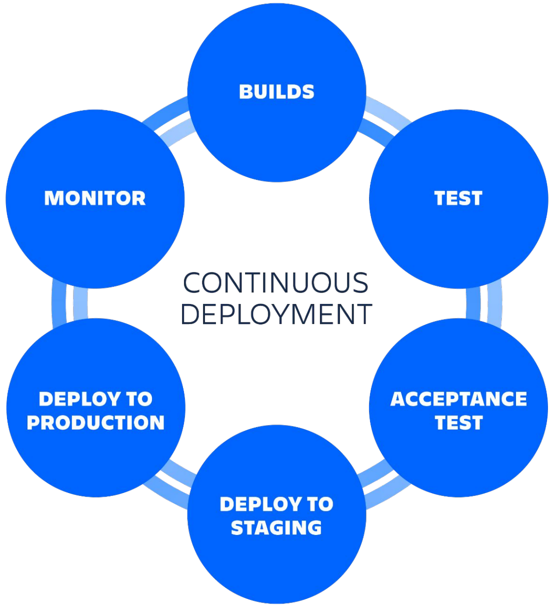
Continuous Delivery (CD)

Continuous delivery is a software development practice where code changes are automatically prepared for a release to production.



Continuous Deployment (CD)

Continuous deployment is an extension of continuous delivery, and can refer to automating the release of a developer’s changes from the repository to production, where it is usable by customers.



Continuous Delivery vs Continuous Deployment

Key Differences:

- **Deployment to Production**
  a. **Continuous Delivery**: Requires manual approval for deployment to production.
  b. **Continuous Deployment**: Automates the deployment to production without manual intervention.

- **Risk and Control**
  a. **Continuous Delivery**: Allows for a final check before deployment, reducing risk.
  b. **Continuous Deployment**: Requires high confidence in automated tests and processes to mitigate risks.

Continuous Delivery vs Continuous Deployment

Use Cases:

- **Continuous Delivery**:
  a. Suitable for applications that require periodic updates and can benefit from frequent, but controlled releases.

- **Continuous Deployment**:
  a. Ideal for applications where rapid delivery of new features and fixes is critical, such as web services and SaaS products.

Benefits after implementation of CI/CD

- Improved Quality: Early detection of integration issues and bugs leads to higher software quality.
- Faster Time-to-Market: Shortens release cycles, allowing faster delivery of features and fixes.
- Reduced Risks: Smaller, incremental changes reduce the risk of deployment failures.
- Enhanced Collaboration: Encourages collaboration and teamwork across development and operations teams.

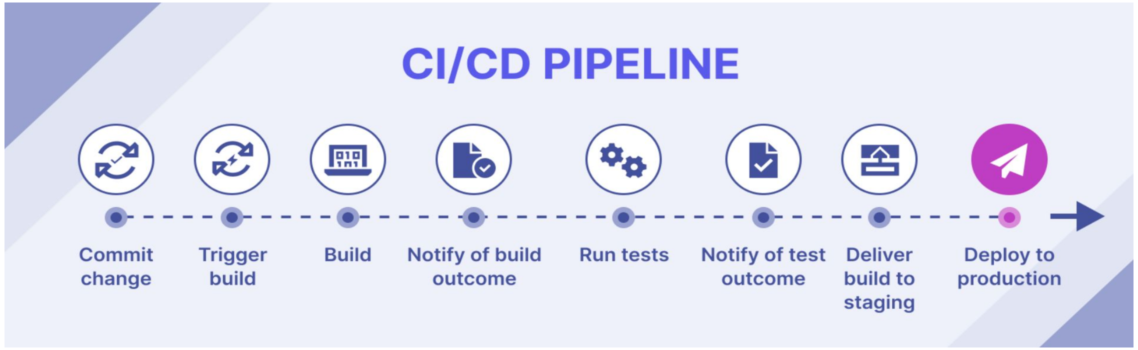
Building Blocks of a CD Pipeline

- Version Control System - Git
- Public Cloud Provider - AWS, GCP, Azure
- Continuous Integration Tool - Jenkins, Gitlab CI, Circle CI
- Continuous Delivery Tool - AWS CodePipeline, Azure DevOps, Google Cloud Build

CI/CD Pipeline Workflow

Workflow Overview:

1. **Commit**: Developer commits code changes to the repository.
2. **Build**: Automated build process triggered on commit.
3. **Test**: Automated tests (unit, integration, acceptance) are executed.
4. **Deploy**: Application artifacts are deployed to staging or production.
5. **Monitor**: Continuous monitoring of application and infrastructure.
6. **Feedback Loop**: Feedback to developers based on monitoring and testing results.



Drawbacks of CI/CD

Complexity in Setup and Maintenance:

- Setting up CI/CD pipelines and maintaining infrastructure can be complex and time-consuming.
- Requires expertise in automation tools and continuous integration practices.

Cultural Shift and Adoption:

- Requires a cultural shift towards automation and collaboration across development and operations teams.
- Resistance to change and lack of buy-in from stakeholders can hinder adoption.

Initial Investment:

- Initial setup costs for CI/CD infrastructure, automation tools, and training.
- Long-term benefits may outweigh costs, but requires upfront investment.

Drawbacks of CI/CD

Integration with Legacy Systems:

- Integration with existing legacy systems and processes can be challenging.
- Compatibility issues may arise, requiring careful planning and phased implementation.

Security and Compliance:

- Automated deployments increase the frequency of changes, potentially impacting security and compliance.
- Requires robust security measures and compliance checks in automated pipelines.

Monitoring and Maintenance:

- Continuous monitoring of CI/CD pipelines and automation tools to ensure reliability and performance.
- Maintenance of pipelines to incorporate updates, patches, and changes in software and infrastructure.

Case Study: Netflix

Netflix’s Approach to Continuous Delivery

Continuous Integration & Continuous Deployment:

- Automated build and test processes ensure rapid integration of code changes from multiple developers.
- Allows Netflix to maintain a fast pace of development without compromising on quality.
- Automated deployment pipelines enable Netflix to deploy new features and updates to production quickly and reliably.
- Continuous deployment ensures that improvements and new features reach users rapidly, enhancing user experience

Key Benefit:

- Agility & Speed
- Quality Assurance

Impact & Success:

- Market Leadership

Case Study: Amazon

Amazon’s Approach to Continuous Integration and Continuous Delivery

CI/CD Implementation:

- Automates the integration of code changes into a central repository multiple times a day.
- Ensures early detection of integration issues and maintains code quality across development teams.
- Automates the deployment process, enabling rapid and reliable releases of new features and updates to production.
- Enhances agility and responsiveness to market demands and customer feedback.

Key Benefit:

- Speed & Efficiency
- Reliability & Scalability

Impact & Success:

- Innovation Leadership

Conclusion

- Key Takeaways
- Future Directions

Thanks!

Do you have any questions?

Jay Sanghani, Software Engineer