CI/CD

Continuous Integration and Continuous Delivery/Deployment

Aryaman Chhikara
Gungun Gungun
Thomas Johnson
Stephen Killough

What is CI / CD

CI - stands for continuous integration

It is the practice of automatically and frequently integrating code changes into a repository.

CD - Stands for continuous delivery

It is the process which refers to the integration testing and delivery of code changes.

CI (Continuous Integration)

CI detects integration issues early on ensuring that all the developers code is seamlessly integrated. It does that by automating the build and test processes.

CI helps identify conflicts bugs and other issues that occur during development, which enhances the overall collaboration among team members.

Some Key component of CI

Version Control System (VCS):

- Centralized (e.g., SVN) or Distributed (e.g., Git)
- Enables tracking changes, managing codebase, and collaboration among developers.

Automated Build:

- Automatic compilation of code into executable or deployable artifacts.
- Ensures consistency and repeatability in the build process.

Automated Testing:

- Includes unit tests, integration tests, and other forms of automated testing.
- Validates code changes and prevents regressions.

Continuous Integration Server:

- Manages the CI process, including triggering builds, running tests, and reporting results.
- Examples include Jenkins, Travis CI, CircleCI.

CD (Continuous Delivery)

Continuous delivery (CD) takes the concept of CI further by enabling the seamless and rapid deployment of code changes to production environments. CD automates all the steps involved in releasing software, including building, packaging, and deploying applications.

It ensures that software is always in a releasable state, allowing organizations to deliver value to end-users promptly

Some Key components of CD

Continuous Monitoring and Feedback:

- Monitors deployed applications and infrastructure in real-time.
- Collects metrics, logs, and alerts to provide feedback on application health and performance.

Rollback Mechanism:

- Provides the ability to rollback deployments in case of failures or issues.
- Ensures quick recovery and minimal downtime in case of deployment failures.

Release Management:

- Manages the release process, including scheduling, versioning, and tracking of deployments.
- Facilitates coordination between development, operations, and other stakeholders.

Some more key components of CD

Compliance and Security:

- Implements security measures and compliance checks throughout the deployment pipeline.
- Ensures that deployments meet regulatory requirements and security standards.

Infrastructure as Code (IaC):

- Treats infrastructure as code, enabling automation and versioning of infrastructure configurations.
- Allows for reproducible and consistent infrastructure deployments.

Integration with CI:

- Integrates seamlessly with the Continuous Integration (CI) process to automate the end-to-end software delivery pipeline.
- Ensures that changes validated in CI are seamlessly deployed to production environments.

Scope for CI and CD

CI focuses on integrating code changes into a shared repository frequently, usually several times a day. It emphasizes automating the build and test phases.

CD extends CI by automating the entire software release process. It encompasses deploying code changes to production environments after passing through the CI pipeline

Objectives of CI and CD

The primary goal of CI is to catch integration errors quickly by integrating code changes into the main branch and running automated tests. It ensures that the codebase remains in a deployable state at all times.

CD aims to automate the deployment process further beyond CI, enabling frequent, reliable, and low-risk releases to production environments. It emphasizes delivering changes to end-users rapidly and continuously.

Deployment Strategies

CI does not involve deploying code changes to production environments automatically. It focuses on validating code changes through automated testing.

CD encompasses various deployment strategies, including continuous delivery and continuous deployment. Continuous delivery involves deploying code changes to staging or pre-production environments for further testing and validation before manual approval for production deployment. Continuous deployment automates the deployment of code changes to production environments after passing through the CI/CD pipeline without manual intervention.

Why would we want CI/CD?

Faster time to market

CI/CD automates the software delivery pipeline, enabling rapid and frequent releases of new features, enhancements, and bug fixes. This agility allows organizations to respond quickly to market demands and stay ahead of competitors.

Improved Software quality

By automating build, testing, and deployment processes, CI/CD reduces the likelihood of introducing bugs or errors into the codebase. Early detection of issues through automated testing ensures higher overall software quality.

Benefits of CI/CD

Accelerated Delivery

- Rapid feedback loops: CI/CD enables developers to receive immediate feedback on code changes, allowing them to identify and address issues early in the development process, thereby accelerating the delivery of features and updates.
- Reduced time-to-market: By automating the build, test, and deployment processes, CI/CD shortens the development lifecycle, enabling organizations to release new features and updates to customers faster, gaining a competitive edge in the market.

Enhanced Quality

- Continuous testing and validation: CI/CD pipelines automate the execution of various tests, including unit tests, integration tests, and end-to-end tests, ensuring that code changes meet quality standards and functional requirements.
- Early bug detection and resolution: With CI/CD, bugs and defects are detected early in the development cycle, making them easier and less costly to fix, resulting in higher-quality software products and improved customer satisfaction.

Improved Collaboration

- Seamless integration among teams: CI/CD promotes collaboration and teamwork by providing a centralized platform where developers, testers, and other stakeholders can collaborate on code changes, share insights, and coordinate efforts effectively.
- Enhanced communication and transparency: CI/CD pipelines offer visibility into the development process, allowing team members to track progress, monitor changes, and communicate effectively, leading to improved transparency and alignment across the organization.

Increased Productivity

- Automation of repetitive tasks: CI/CD automates time-consuming and repetitive tasks such as code compilation, testing, and deployment, freeing up developers to focus on more value-added activities, such as innovation and problem-solving.
- Focus on value-added activities: By automating routine tasks, CI/CD enables developers to spend more time on creative and strategic activities, such as designing new features, improving user experience, and addressing customer feedback, thereby increasing overall productivity.

Higher Customer Satisfaction

- Rapid feature delivery: CI/CD allows organizations to deliver new features and updates to customers quickly and frequently, addressing their needs and preferences in a timely manner, leading to higher levels of customer satisfaction and loyalty.
- Reduced downtime and bugs: With CI/CD, software updates are deployed incrementally and with minimal disruption, reducing the risk of downtime and minimizing the occurrence of bugs and performance issues, ensuring a seamless and reliable user experience.

Competitive Advantage

- By embracing CI/CD practices, organizations can differentiate themselves in the market by delivering innovative features and updates to users faster and with higher quality.
- The ability to respond quickly to customer feedback and market changes gives companies a competitive edge and positions them as leaders in their industry.

Consistency Across Environments

- CI/CD ensures that development, testing, staging, and production environments remain consistent and in sync.
- By using infrastructure as code and automated deployment pipelines, organizations can replicate environments easily, reducing configuration drift and minimizing discrepancies between different stages of the deployment pipeline.

Principles of CI/CD

Automated Testing

- Automated tests (unit tests, integration tests, etc.) are an integral part of the CI process.
- Tests are executed automatically as part of the build process to ensure that changes haven't introduced any regressions or bugs.
- Testing should cover different levels (unit, integration, end-to-end) to provide comprehensive code coverage.

Infrastructure as Code (IaC)

- CI/CD pipelines often rely on Infrastructure as Code principles, where infrastructure (servers, databases, networking, etc.) is defined in code and managed programmatically.
- Tools like Terraform, CloudFormation, or Ansible are used to define and provision infrastructure, ensuring consistency and reproducibility across different environments.

Version Control

- Version control systems (such as Git) are at the core of CI/CD practices, enabling developers to collaborate effectively and track changes to the codebase over time.
- All code changes, configurations, and scripts used in the CI/CD pipeline should be version-controlled to maintain a reliable history and facilitate collaboration.

Feedback Loop

- CI/CD emphasizes quick feedback loops, providing developers with immediate feedback on the quality and correctness of their code.
- This feedback loop helps identify and address issues early in the development process, reducing the time and effort required to fix them.

Visibility and Monitoring

- CI/CD pipelines should incorporate robust monitoring and logging mechanisms to provide visibility into the health and performance of the software delivery process.
- Monitoring helps identify bottlenecks, failures, and performance issues in the pipeline, allowing teams to continuously optimize and improve their workflows.

Security and Compliance

- Security and compliance considerations should be integrated into every stage of the CI/CD pipeline to ensure that software releases meet regulatory requirements and security standards.
- Automated security scans, vulnerability assessments, and compliance checks should be part of the deployment pipeline to identify and address potential risks early in the development process.

Environment Parity

- Ensuring parity between development, testing, staging, and production environments is crucial for reliable and predictable deployments.
- By maintaining consistent configurations, dependencies, and infrastructure across environments, teams can minimize the risk of issues arising due to environmental differences.

Thomas’s Section

Continuous Delivery

Once code has gone through the CI process, the Continuous Delivery process ensures that it's packaged correctly and ready to be deployed to an environment.

Continuous Delivery can get changes into a testing environment in a safe, quick, and sustainable way.

Continuous Delivery practices develop code in such a way that it can be deployed at any time.

- Production-like test environments
- Requires human intervention to deploy

Continuous Delivery cont.

Benefits:

- **Lower Risk** - makes delivery painless by using repeatable patterns
- **Faster** - integration and delivery can take a significant amount of time if done manually
- **Higher Quality** - automated and continuous delivery means more time available for fixes and improvements, as well as a higher likelihood of finding errors
- **Lower Costs** - less time and money spent on fixes

Continuous Deployment

Continuous Deployment allows automatic deployment of applications and changes. Once the CI/CD process determines that the criteria have been met, code can be deployed to the production environment.

Continuous Deployment takes continuous delivery a step further by trusting the developers and tests enough such that it can deploy automatically.

- Developers and tests trusted
- Once tests pass, deployment automated without human approval

Natural outcome of Continuous Delivery. Eventually, human approval slows the process down.

Ex: Japan’s AN/TPY-2 Radar

Continuous Deployment Tools

Version Control - track revisions and aid collaboration

Code Review - test the current source code

Continuous Integration - automate integration and testing at least once per day

Configuration Management - ensure software and hardware maintain a consistent state

Release Automation - automate activities for continuous deployment

Infrastructure Monitoring - impact of changes to performance of system

Continuous Deployment Cont.

Continuous Integration is a necessary part of Continuous Deployment.

Developers often work off a copy of a master branch. The longer that developers work independently of each other, the greater the risk of failure.

Continuous Integration ensures that code gets tested and integrated at least once per day, so that this risk is mitigated.

The Role of Automated Testing

As the creator, you may not know the impact of a change, or forget to test some aspect of the system. Testing is also time-consuming and costly.

Extreme Programming Explained: Embrace Change, Kent Beck, Cynthia Andres, 1999

- Cornerstones of modern software development
  - Rapid feedback loops
  - Safe and easy ways to make change
  - Happy and satisfied team

Automated testing plays a key role for all three.

Automated Testing Cont.

Rapid Feedback Loops

- Testing is about feedback. The test should return results quickly. The faster the results come, the sooner you can progress, and the sooner you know if your tests work as expected.

Safe and easy way to make changes

- Automated testing means a repeatable and easy testing process. Making changes easy means quicker and more reliable test results.

Happy and satisfied team

- Automated testing makes testing much easier and more reliable. Teams can be confident in the results and use less time testing.

Automated Testing Mistakes

semaphorci.com - Automated Testing

Testing slows progress

- “Give me six hours to chop down a tree and I will spend the first four sharpening the axe.”
  - Abraham Lincoln
- Forces the developers to think about the problem and develop a “safety net”
- Allows for quick changes once safety net is in place

Testing is only for finding bugs

- Also makes bugs easier to detect and fix (safety net)

We must achieve 100% coverage

- Coverage only means that a test puts a portion of code under scrutiny. Instead, we should compromise between coverage and quality.

Writing an Automated Test

The first tests should have enough code to give some value, but little enough to be approachable.

End-to-end tests that emulate user behavior

Use tools, but don’t limit to only using tools.

Write your test with automation in mind.

Writing an Automated Test Cont.

semaphorci.com - The Six Principles of Test Automation

Improve Quality - Don’t let bad code pass

Reduce Risk - minimize risk of failure; find errors while not giving false positives

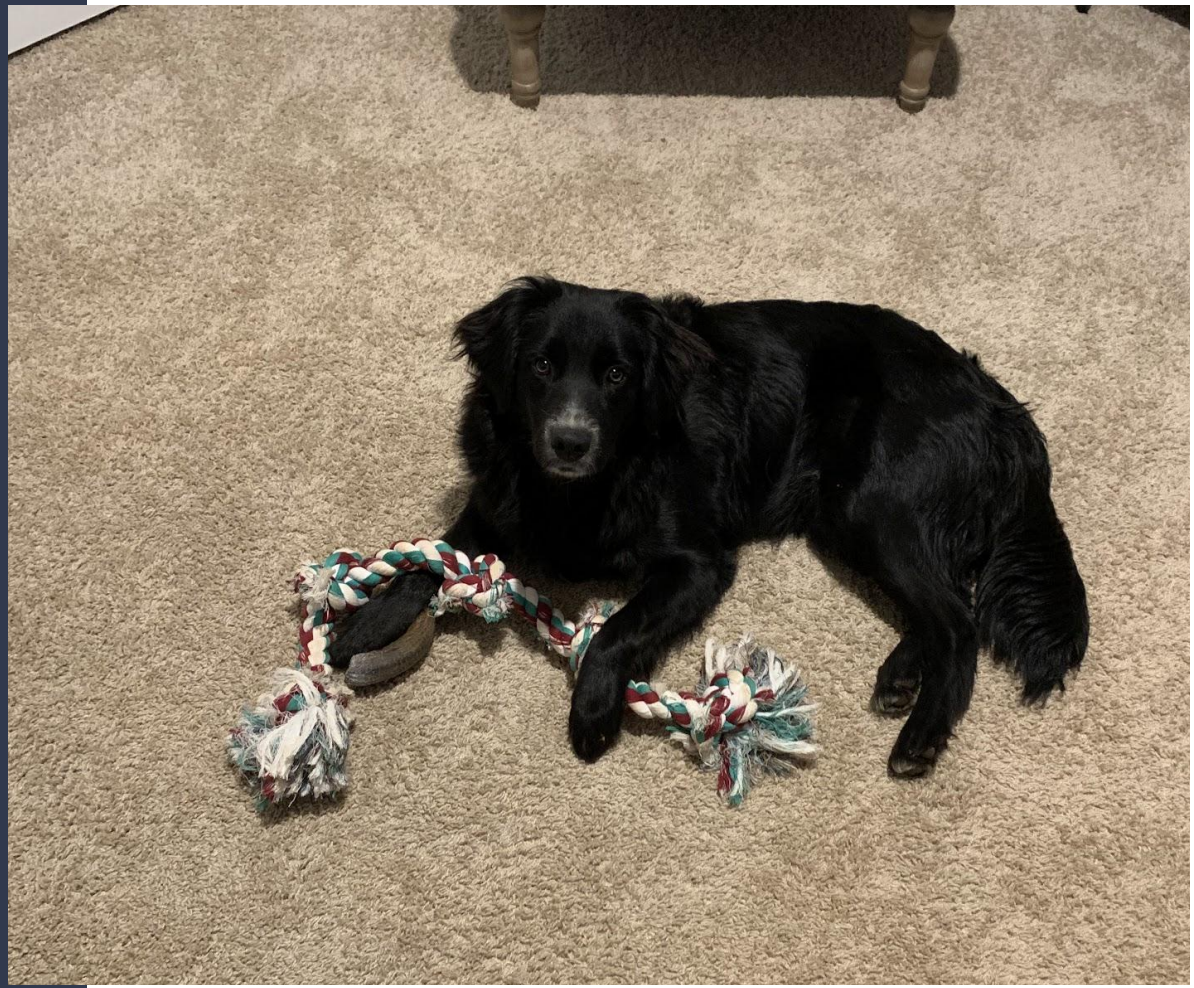
Understand the Code - have an expectation for the code’s behavior

Easy to Write - Impractical to test your test

Easy to Run - Should be able to start automatically

Minimal Maintenance - Don’t tie so closely to code that a change will break the test

Stephen’s Section



Introduction to CI/CD Pipelines:

CI/CD pipelines are automated processes in software development that enable code changes made by developers to be automatically integrated, tested, and prepared for release to production

Automate the integration, testing, and deployment processes.

Facilitate a seamless flow from development to deployment.

Reduce manual intervention, improving efficiency.

Ensure software quality through automated testing.

Benefits of CI/CD Pipelines

Discuss the advantages of implementing CI/CD pipelines, such as faster delivery times, improved software quality, and enhanced developer productivity.

Faster Delivery Times: Automated processes lead to quicker deployments.

Improved Software Quality: Early bug detection and consistent testing environments.

Enhanced Developer Productivity: Focus on high-value tasks with reduced integration hassles.

Key Components of a CI/CD Pipeline

Break down the pipeline into its essential components, including source control, build automation, testing, and deployment

Source Control: Foundation for collaboration and code management.

Build Automation: Compiles code, ensuring consistency and early problem detection.

Automated Testing: Identifies defects early, improving quality.

Deployment: Automates release processes, enhancing speed and reliability

The Role of Automated Testing

Discuss how automated testing is integrated into CI/CD pipelines and its impact on software quality

Triggered automatically by code commits.

Covers unit, integration, functional, and performance tests.

Provides real-time feedback to developers.

Ensures only validated changes are deployed.

Overview of CI/CD Tools

Introduce the tools that facilitate CI/CD processes, including Jenkins, GitLab CI/CD, and GitHub Actions.

Jenkins: Extensive plugin library, supports complex workflows.

GitLab CI/CD: Integrated environment, configuration as code.

GitHub Actions: Deep integration with GitHub, marketplace for shared workflows.

Jenkins: An In-depth Look

Provide an overview of Jenkins, its ecosystem, and how it supports CI/CD pipelines.

Extensive collection of plugins for broad capability extension.

"Jenkins Pipeline" for continuous delivery as code.

"Jenkinsfile" for pipeline version control.

GitLab CI/CD and GitHub Actions

Compare these tools, focusing on their integration with source code repositories and built-in CI/CD capabilities

GitLab CI/CD: Integrated CI/CD with Auto DevOps, visibility, and built-in security.

GitHub Actions: Workflow automation within GitHub, supports a marketplace of actions.

Both emphasize simplicity and direct repository integration.

Emerging CI/CD Tools and Technologies

Highlight new and emerging tools in the CI/CD landscape, discussing their unique features or approaches to automation.

CircleCI: Known for fast execution times and "Orbs".

Travis CI: Simplicity in setup, integrates seamlessly with GitHub.

Docker/Kubernetes: Enhances CI/CD with containerization and orchestration.

Tekton/Argo CD: Kubernetes-native CI/CD solutions.

Choosing the Right CI/CD Tool

Offer criteria or considerations for selecting a CI/CD tool, such as scalability, community support, and integration capabilities

Consider scalability, community support, and integration capabilities.

Ensure flexibility for pipeline configuration and workflow customization.

Prioritize security features and access control.

Evaluate ease of use, setup, and cost implications.

Best Practices and Future Trends in CI/CD

Summarize best practices for implementing CI/CD pipelines and tools, and speculate on future trends in CI/CD technologies

Automate fully, maintain code quality, encourage small frequent commits.

Future trends include AI/ML integration, serverless CI/CD pipelines, infrastructure as code, shift-left security, and the rise of GitOps.

References

https://www.redhat.com/en/topics/devops/what-is-ci-cd

https://zeet.co/blog/ci-cd-pipeline-examples

https://continuousdelivery.com/

https://www.redhat.com/en/topics/devops/what-is-ci-cd#:~:text=CI%2FCD%2C%20which%20stands%20for,a%20shared%20source%20code%20repository

https://about.gitlab.com/topics/ci-cd/

https://www.ibm.com/topics/continuous-deployment#:~:text=Continuous%20deployment%20is%20a%20strategy,directly%20to%20the%20software's%20users

https://semaphoreci.com/blog/automated-testing-cicd

https://semaphoreci.com/blog/test-automation

https://www.jenkins.io/

https://about.gitlab.com/topics/ci-cd/

https://github.com/features/actions

https://circleci.com/

https://www.travis-ci.com/

https://www.docker.com/products/kubernetes/