UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TOPIC:

Preparation of the environment for development and integration continue

STUDENT:

Maciel Leyva Ariana Lizeth

GROUP:

10A

SUBJECT:

Comprehensive Mobile Development

TEACHER:

Ray Brunett Parra Galaviz

Tijuana, Baja California, January 8th of 2025

“Preparation of the environment for development and integration continue”

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) refers to the practice of automatically and frequently integrating code changes into a shared source code repository. Continuous delivery and/or deployment (CD) is a two-part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.

What is continuous integration?

The "CI" in CI/CD always refers to continuous integration, an automation process for developers that facilitates more frequent merging of code changes back to a shared branch, or “trunk.” As these updates are made, automated testing steps are triggered to ensure the reliability of merged code changes.

In modern application development, the goal is to have multiple developers working simultaneously on different features of the same app. However, if an organization is set up to merge all branching source code together on one day (known as “merge day”), the resulting work can be tedious, manual, and time-intensive.

What is continuous delivery?

Continuous delivery automates the release of validated code to a repository following the automation of builds and unit and integration testing in CI. So, in order to have an effective continuous delivery process, it’s important that CI is already built into your development pipeline.

In continuous delivery, every stage from the merger of code changes to the delivery of production-ready builds involves test automation and code release automation. At the end of that process, the operations team is able to swiftly deploy an app to production.

Continuous delivery usually means a developer’s changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment by the operations team. It’s an answer to the problem of poor visibility and communication between dev and business teams. To that end, the purpose of continuous delivery is to have a codebase that is always ready for deployment to a production environment, and ensure that it takes minimal effort to deploy new code.

Continuous deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a "release day" anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

What is the “CD” in CI/CD?

The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they’re sometimes used separately to illustrate just how much automation is happening. The choice between continuous delivery and continuous deployment depends on the risk tolerance and specific needs of the development teams and operations teams.

What are the benefits of each practice?

We've explained the difference between continuous integration, continuous delivery, and continuous deployments but we haven't yet looked into the reasons why you would adopt them. There's an obvious cost to implementing each practice, but it's largely outweighed by their benefits.

Continuous integration

What you need (cost)

- Your team will need to write automated tests for each new feature, improvement or bug fix.
- You need a continuous integration server that can monitor the main repository and run the tests automatically for every new commits pushed.
- Developers need to merge their changes as often as possible, at least once a day.

What you gain

- Less bugs get shipped to production as regressions are captured early by the automated tests.
- Building the release is easy as all integration issues have been solved early.
- Less context switching as developers are alerted as soon as they break the build and can work on fixing it before they move to another task.
- Testing costs are reduced drastically – your CI server can run hundreds of tests in the matter of seconds.
- Your QA team spends less time testing and can focus on significant improvements to the quality culture.

Continuous delivery

What you need (cost)

- You need a strong foundation in continuous integration and your test suite needs to cover enough of your codebase.
- Deployments need to be automated. The trigger is still manual but once a deployment is started there shouldn't be a need for human intervention.
- Your team will most likely need to embrace feature flags so that incomplete features do not affect customers in production.

What you gain

- The complexity of deploying software has been taken away. Your team doesn't have to spend days preparing for a release anymore.
- You can release more often, thus accelerating the feedback loop with your customers.
- There is much less pressure on decisions for small changes, hence encouraging iterating faster.

Continuous deployment

What you need (cost)

- Your testing culture needs to be at its best. The quality of your test suite will determine the quality of your releases.
- Your documentation process will need to keep up with the pace of deployments.
- Feature flags become an inherent part of the process of releasing significant changes to make sure you can coordinate with other departments (support, marketing, PR...).

What you gain

- You can develop faster as there's no need to pause development for releases. Deployments pipelines are triggered automatically for every change.
- Releases are less risky and easier to fix in case of problem as you deploy small batches of changes.
- Customers see a continuous stream of improvements, and quality increases every day, instead of every month, quarter or year.

Preparation of the Environment for Development and Continuous Integration

The preparation of the development and continuous integration environment is a fundamental step in the software development lifecycle. It ensures that the team can work efficiently, collaborate seamlessly, and deliver high-quality software. Here's an outline of the key considerations and steps involved:

1. Development Environment Setup

A well-configured development environment is crucial for individual productivity and consistency across the team.

- **Hardware Requirements:** Ensure the necessary hardware resources (RAM, storage, CPU) for smooth development.
- **Operating System:** Standardize the OS (e.g., Windows, macOS, or Linux) across the team if possible, or ensure cross-platform compatibility.
- **Development Tools:**

  - **Text Editors/IDEs:** Install IDEs like Visual Studio Code, IntelliJ IDEA, or Android Studio.
  - **Version Control:** Set up Git and link it to the project's repository.
  - **Dependency Management:** Configure tools like npm, pip, or Maven for managing libraries and frameworks.
  - **Containerization:** Use Docker to create consistent development environments.

2. Setting Up the Integration Environment

Continuous integration (CI) ensures that code changes are regularly merged, tested, and validated automatically.

- **Version Control System (VCS):** Use a Git repository hosted on platforms like GitHub, GitLab, or Bitbucket.
- **CI Tools:** Select a CI platform, such as Jenkins, GitHub Actions, GitLab CI/CD, or CircleCI.
- **Automated Testing:**
    - Create unit, integration, and end-to-end tests.
    - Integrate test execution into the CI pipeline.
- **Build Automation:**
    - Use tools like Gradle, Maven, or Webpack to automate the build process.
    - Configure build scripts to include dependency installation, compilation, and packaging.

3. Environment Configuration

- **Environment Variables:** Use .env files or secret managers to store sensitive data like API keys and database credentials.
- **Configuration Management:** Tools like Ansible, Puppet, or Chef can help automate environment setup.
- **Staging and Production:** Define separate environments for development, staging, and production to ensure isolation and reliability.

4. Collaboration and Workflow

A smooth workflow ensures that the team can collaborate effectively and integrate code without conflicts.

- **Branching Strategy:** Adopt a branching model like GitFlow or trunk-based development.
- **Code Reviews:** Set up mandatory code reviews and approval processes.
- **Pull Requests (PRs):** Use PRs to integrate changes after review and testing.

5. Monitoring and Feedback

Feedback loops are vital to ensure the environment runs smoothly.

- **Monitoring Tools:** Use tools like Prometheus, Grafana, or ELK stack to monitor CI pipelines and environment health.
- **Error Tracking:** Tools like Sentry or Bugsnag can help track errors in development or integration pipelines.

6. Documentation and Training

- Provide clear documentation for setting up the development environment.
- Conduct training sessions to familiarize the team with CI/CD workflows and tools.

By preparing a robust development and continuous integration environment, teams can ensure streamlined workflows, consistent quality, and faster delivery cycles.

Sources

What is CI/CD? (s. f.). Retrieved on January 8th, 2025, of https://www.redhat.com/en/topics/devops/what-is-ci-cd

Atlassian. (s. f.-a). *Continuous integration vs. delivery vs. deployment*. Retrieved on January 8th, 2025, of https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

GitLab. (2023, April 13th). *What is CI/CD? | GitLab*. GitLab. Retrieved January 8th, 2025, of https://about.gitlab.com/topics/ci-cd/

Palmer, J. (2024, May 28th). What is Continuous Integration (CI) in DevOps? | Padok. *Theodo.cloud*. Retrieved on January 8th, 2025, of https://cloud.theodo.com/en/blog/devops-continuous-integration

Team, S. (2024, July 1st). How to implement continuous delivery. *Statsig*. Retrieved on January 8th, 2025, of https://www.statsig.com/perspectives/how-to-implement-continuous-delivery