<epam> | CD/CD foundations for JavaScript engineers

CONTINIOUS INTEGRATION, DELIVERY AND DEPLOYMENT CONCEPTS OVERVIEW

EPAM Systems Inc.
CD/CD foundations for JavaScript engineers
Learn & Development
www.epam.com

Table of Contents

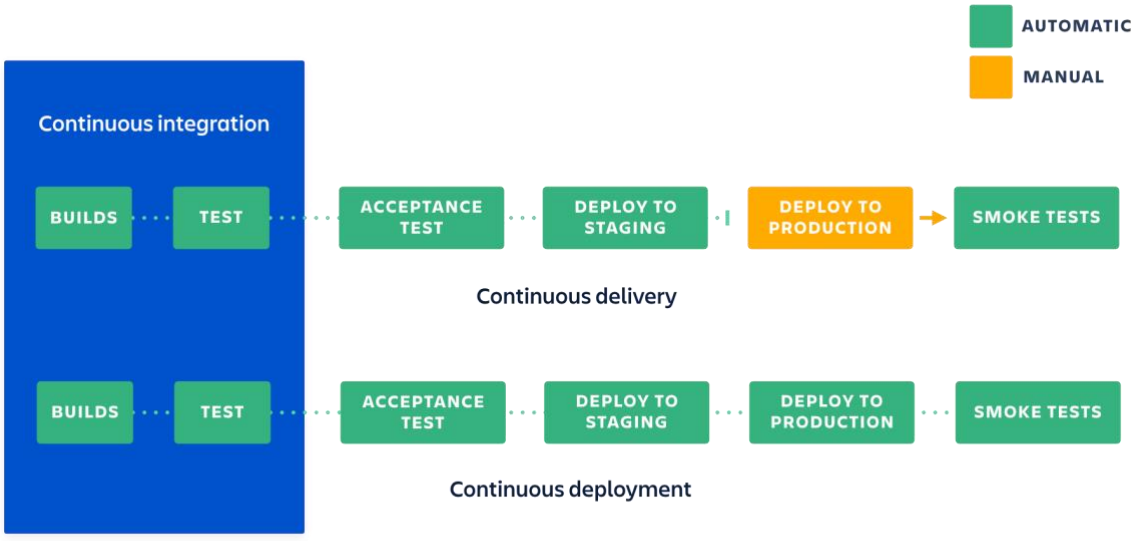
CONTINIOUS INTEGRATION, DELIVERY AND DEPLOYMENT CONCEPTS OVERVIEW 1
1. OVERVIEW 3
1.1. HOW THE PRACTICES RELATE TO EACH OTHER 3
1.2. GLOSSARY 4
1.2.1. CI/CD Pipeline 4
1.2.2. Repository 4
1.2.3. Build 4
1.2.4. Artifact 5
1.2.5. Release 5
1.2.6. Deployment 5
1.2.7. Product Increment 5
1.2.8. Cloud vs On-premise 6
2. CONTINIOUS INTEGRATION (CI) 7
2.1. OVERVIEW 7
2.2. COSTS AND GAINS 7
2.3. ADOPTION PROCESS 8
3. CONTINIOUS DELIVERY (CD) 9
3.1. OVERVIEW 9
3.2. COSTS AND GAINS 9
4. CONTINIOUS DEPLOYMENT (CD) 11
4.1. OVERVIEW 11
4.2. COSTS AND GAINS 11
4.3. HOW TO MOVE FROM CI TO CONTINIOUS DELIVERY 12
4.3.1. Utilize CI 12
4.3.2. Measure test coverage 12
4.3.3. Monitor releases performance 12
4.3.4. Embrace post-deployment testing 13
5. CI AND CD TOOLS 14
5.1. BRIEF SELECTION 14
5.1.1. Jenkins 14
5.1.2. AWS CodePipeline 14
5.1.3. Bitbucket Pipelines 15
5.1.4. CircleCI 15
5.1.5. Azure Pipelines 16
5.1.6. GitLab 16
5.1.7. Atlassian Bamboo 16
5.2. CHOICE FACTORS 17
5.2.1. Version control systems support 17
5.2.2. On-premises vs cloud 17
5.2.3. Containerization support 17
5.2.4. Plugins and third-party integrations 18
6. ADDITIONAL RESOURCES 19

1. OVERVIEW

CI and CD are two acronyms frequently used in modern development practices and DevOps. CI stands for **continuous integration**, a fundamental DevOps best practice where developers frequently merge code changes into a central repository where automated builds and tests run, while CD can either mean **continuous delivery** or **continuous deployment**.

1.1. HOW THE PRACTICES RELATE TO EACH OTHER

Figure below demonstrates that continuous integration is part of both continuous delivery and 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.



1.2. GLOSSARY

This section provides the description of the most commonly spelled terms in this lesson.

1.2.1. CI/CD Pipeline

A pipeline is a set of ordered automated processes that allow developers and DevOps professionals to reliably and efficiently compile, build, and deploy their code to production compute platforms. Whithin a pipeline you can also perform unit, functional, integrational, and (end-to-end) e2e testing, as well as do automated code quality, performance, and security scans. Pipelines' processess are split into *Pipeline* Stages. You may see CI/CD pipelines depicted on the figure in the previous chapter.

1.2.2. Repository

A repository is a file storage location. It is used by version control systems (VCS) to store multiple versions of files. While a repository can be configured on a local machine for a single user, it is often stored on a server (such as GitLab, GitHub, Bitbucket), which can be accessed by multiple users. By committing changes to a repository, developers can quickly revert to a previous version of a program if a recent update causes bugs or other problems. Many version control systems support side-by-side comparisons of different versions of files saved in the repository, which can be helpful for debugging source code.

1.2.3. Build

"Build" may refer to the process by which source code is converted into a stand-alone form that can be run on a computer or to the result of such process. The term build can have a slightly different meaning depending on whether it is used as a noun or a verb. A developer might "do a build", which means to run the build process, but then might also refer to the end result as "build number 175", to differentiate it when talking to their colleagues. The process of building software is usually managed by a build tool. Builds are created when a certain point in development has been reached or the code has been deemed ready for implementation, either for testing or release. A build is also known as a software build or code build.

1.2.4. Artifact

An artifact is a by-product of software development. It's anything that is created so a piece of software can be developed or ran. This might include things like data models, configuration files, diagrams, setup scripts, reports, etc. I.e., an artifact is generated by our pipeline whenever its execution status is successful.

1.2.5. Release

A release is a deployable software package. *Major Software Releases* include a number of changes in the existing functionality or the addition of entirely new functionality in the software. Most of them resolve the issues present in the preceding version of the software. The small improvements and bugs are taken care of in *Minor Software Releases*. To provide large scale point updates, the installation of these improvements and issues generally take place on top of major releases. *Emergency Fix Releases* are the corrections of the errors and issues in the software applications that lower performance. The end-users face difficulties in using the software and thereby the issues need to be resolved immediately and emergency fixes are made.

1.2.6. Deployment

Deployment process includes all of the steps, processes, and activities that are required to make a software system or update available to its intended users. Today, most IT organizations and software developers deploy software updates, patches and new applications with a combination of manual and automated processes. Some of the most common activities of software deployment include software release, installation, testing, deployment, and performance monitoring.

1.2.7. Product Increment

In the context of agile software product development, using a framework such as Scrum, a software product increment is what gets produced at the end of a development period (i.e. PI) or a Sprint. More specifically, product increment is the sum of all the items completed during a sprint and the value of the increments of all previous sprints.

1.2.8. Cloud vs On-premise

The fundamental difference between cloud vs on-premise software is where it resides. On-premise software is installed locally, on your business' computers and servers, whereas cloud software is hosted on the vendor's server and accessed via a web browser.

2. CONTINIOUS INTEGRATION (CI)

2.1. OVERVIEW

Continuous integration is an agile and DevOps best practice where developers integrate their code changes early and often to the main branch or code repository. The goal is to reduce the risk of seeing “integration hell” by waiting for the end of a project or a sprint to merge the work of all developers.

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, they avoid integration challenges that can happen when waiting for release day to merge changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

2.2. COSTS AND GAINS

Costs:

- 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.

Gains:

- 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 spend less time testing and can focus on significant improvements to the quality culture.

2.3. ADOPTION PROCESS

While it may look easy, it will require true commitment from your team to be effective. You will need to slow down your releases at the beginning, and you need buy-in from the product owners to make sure that they do not rush developers in shipping features without tests.

CI adoption process boils down to these 5 points:

1. Start writing tests for the critical parts of your codebase.

2. Get a CI service to run those tests automatically on every push to the main repository.

3. Make sure that your team integrates their changes everyday.

4. Fix the build as soon as it's broken.

5. Write tests for every new story that you implement.

It is recommended to start small with simple tests to get used to the new routine before moving on to implementing a more complex test suite that may be hard to manage.

3. CONTINIOUS DELIVERY (CD)

3.1. OVERVIEW

Continuous delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

This means that on top of automated testing, you have an automated release process and you can deploy your application any time by clicking a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

3.2. COSTS AND GAINS

Costs:

- 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.

Gains:

- 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.

4. CONTINIOUS DEPLOYMENT (CD)

4.1. OVERVIEW

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 automatically. 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.

4.2. COSTS AND GAINS

Costs:

- 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...).

Gains:

- 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.

4.3. HOW TO MOVE FROM CI TO CONTINIOUS DELIVERY

4.3.1. Utilize CI

At the core of continuous deployment is a great CI culture. The quality of your test suite will determine the level of risk for your release, and your team will need to make automating testing a priority during development. This means implementing tests for every new feature, as well as adding tests for any regressions discovered after release.

Fixing a broken build for the main branch should also be first on the list. Otherwise, you'll be exposing yourself to the same risks you faced prior to adopting continuous deployment: changes are accumulated in the development environment, and developers can't be sure that their work is finished as they don't know if their changes passed the acceptance tests.

4.3.2. Measure test coverage

Start using test coverage tools to make sure that your application is adequately tested. A good goal is to aim for 80% coverage.

Be careful not to mistake good coverage for good testing. You could have 100% test coverage with tests that don't really challenge your codebase. Make sure to spend time during code reviews to check how tests are written and don't hesitate to point out gaps in coverage or weak tests. Your test suite acts as the gate to production – make it a strong one.

4.3.3. Monitor releases performance

If you're going to deploy every commit automatically to production, you need to make sure that you have a good way to be alerted if something goes wrong. Sometimes a new change won't break production right away, but it'll cause your CPU or memory consumption to go to dangerous level. This is why it's useful to implement real-time monitoring on your production servers to be able to track irregularities in your metrics. Tools like Nagios or services like New Relic can help

you track basic performance metrics and alert you whenever there's a disturbance in your systems.

4.3.4. Embrace post-deployment testing

If you're going to deploy every commit automatically to production, you need to make sure that you have a good way to be alerted if something goes wrong. Sometimes a new change won't break production right away, but it'll cause your CPU or memory consumption to go to dangerous level. This is why it's useful to implement real-time monitoring on your production servers to be able to track irregularities in your metrics. Tools like Nagios or services like New Relic can help you track basic performance metrics and alert you whenever there's a disturbance in your systems

5. CI AND CD TOOLS

5.1. BRIEF SELECTION

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 automatically. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

5.1.1. Jenkins

Jenkins is a veteran CI Tool with a long proven track record. It is open source and driven by community updates. Jenkins is primarily intended to an on-premise installation. Jenkins is a great option when your organization needs on-prem support for handling sensitive customer like HIPAA compliance data.

Features:

- On-premise
- Open source
- Robust addon and plugin ecosystem

5.1.2. AWS CodePipeline

Amazon Web Services is one of the most dominant cloud infrastructure providers in the market. They offer tools and services for all manner of infrastructure and code development tasks. CodePipeline is their CI Tool offering. CodePipeline is one part of a multitude of tools and can directly interface with other existing AWS tools (such as CodeBuild or CodeDeploy) to provide a seamless AWS experience as well with a list of third-party services.

Features:

- Fully cloud

- Integrated with Amazon Web services
- Custom plugin support
- Robust access control

5.1.3. Bitbucket Pipelines

Bitbucket Pipelines is a CI tool directly integrated into Bitbucket, a cloud version control system offered by Atlassian. Bitbucket Pipelines is an easy next step to enable CI if your project is already on Bitbucket. Bitbucket Pipelines are managed as code so you can easily commit pipeline definitions and kick off builds. Bitbucket Pipelines, additionally offers CD. This means projects built with Bitbucket Pipelines can be deployed to production infrastructure as well.

Features:

- Easy setup and configuration
- Unified Bitbucket experience
- Cloud by 3rd party

5.1.4. CircleCI

CircleCI is a CI Tool that gracefully pairs with Github, one of the most popular version control system cloud hosting tools. CircleCi is one of the most flexible CI Tools in that it supports a matrix of version control systems, container systems, and delivery mechanisms. CircleCi can be hosted on-premise or used through a cloud offering.

Features:

- Notification triggers from CI events
- Performance optimized for quick builds
- Easy debugging through SSH and local builds

- Analytics to measure build performance

5.1.5. Azure Pipelines

Azure is Microsoft's cloud infrastructure platform, the Microsoft equivalent to Amazon Web Services. Like the aforementioned AWS CodePipeline, Azure Pipelines is a CI Tool offering from Microsoft that is fully integrated into the Azure suite of hosting tools.

Features:

- Azure platform integration
- Windows platform support
- Container support
- Github integration

5.1.6. GitLab

GitLab is a CI Tool which offers a full DevOps experience. GitLab was created with intentions to improve Github's overall experience. GitLab offers a modern UX with container support.

Features:

- On-prem or cloud hosting
- Continuous security testing
- Easy to learn UX

5.1.7. Atlassian Bamboo

Another CI offering from Atlassian. Whereas Bitbucket Pipelines is purely a cloud hosted option, Bamboo offers a self hosted alternative.

Features:

- Best integration with Atlassian product suite
- An extensive market place of add-ons and plugins
- Container support with Docker agents
- Trigger API for IFTTT functionality

5.2. CHOICE FACTORS

5.2.1. Version control systems support

The core pillar of a CI / CD system is the support and integration of the underlying Version Control System (VCS). The most popular VCS's are Git, Subversion, Mercurial and Perforce. Cloud CI tools may offer support for some or all of these VCS's. It is critical to choose a CI tool that offers support for your projects VCS.

5.2.2. On-premises vs cloud

Some of the aforementioned CI Tools like Jenkins can be installed on-premises. This means your team is responsible for configuring and managing the CI system on your own infrastructure. This can be beneficial for privacy and security reasons. For example, if you have customer data privacy concerns to meet compliance standards, on-premises may be a requirement. Additionally, on-premises instances may offer deeper customization and configuration options.

On-premises can be extra work that is not in line with your core business needs. Cloud options outsource the management of the CI Tool to a 3rd party vendor. The cloud hosting company then handles uptime, support, and scaling of the CI Tool allowing your team to focus on core business needs. This can be a huge benefit for tight budget teams or smaller companies that need aggressive focus on product market fit goals.

5.2.3. Containerization support

Containerization enables the distribution of an immutable, repeatable, isolated copy of an application. Containerization is enabled by tools like Docker, and Kubernetes. Modern CI Tools

will offer support for integrating containers into the CI/CD process. Containers solve the “works on my machine” problem. Ensuring that application code is packaged in a frozen snapshot of system level dependencies. This provides guarantees that when your team's code is executed on the CI Tool it is a replicate of the local environment. This eliminates a whole class of environment parity troubleshooting issues that arise without containers.

5.2.4. Plugins and third-party integrations

CI Tools become even more useful when integrated with the rest of your tech stack. Analytics about engineering team efficiency and performance can be collected from CI tools. Sprint planning applications can be tied to CI tools to automatically update sprint status when the code has been delivered. These integrations can be used to guide engineering team KPI’s and roadmaps.

6. ADDITIONAL RESOURCES

These are some additional resources that are worth checking out if you want a deeper dive on the discussed topics:

- Atlassian CI/CD essay: https://www.atlassian.com/continuous-delivery/principles
- Continuous Delivery handbook: https://continuousdelivery.com/