What is Continuous Integration?

Build your team's agility with faster feedback. Because you only move as fast as your tests.

Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. It's a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run. Automated tools are used to assert the new code's correctness before integration.

A source code version control system is the crux of the CI process. The version control system is also supplemented with other checks like automated code quality tests, syntax style review tools, and more.

READ ON BELOW

Continuous integration articles

ARTICLE

How to get to Continuous Integration

Learn about how to adopt continuous integration and automated testing in 5 steps.

ARTICLE

3 Git Hooks for Continuous Integration

An intro to Git hooks, plus 3 hooks you can use to support your continuous integration and continuous delivery efforts.

ARTICLE

5 Tips for CI-Friendly Git Repos

Five tips to make the best out of Git and your continuous integration tool!

ARTICLE

Continuous Integration Tools

Five tips to make the best out of Git and your continuous integration tool!

ARTICLE

Trunk-based Development

Learn about trunk-based development, a version control management practice where developers merge small, frequent updates to a core "trunk" or main branch

TUTORIAL

Continuous Integration Tutorial

This tutorial will show you how to get started with continuous integration in three simple steps.

Try this tutorial →

[CONTINUED]

The importance of continuous integration

In order to understand the importance of CI, it's helpful to first discuss some pain points that often arise due to the absence of CI. Without CI, developers must manually coordinate and communicate when they are contributing code to the end product. This coordination extends beyond the development teams to operations and the rest of the organization. Product teams must coordinate when to sequentially launch features and fixes and which team members will be responsible.

The communication overhead of a non-CI environment can become a complex and entangled synchronization chore, which adds unnecessary bureaucratic cost to projects. This causes slower code releases with higher rates of failure, as it requires developers to be sensitive and thoughtful towards the integrations. These risks grow exponentially as the engineering team and codebase sizes increase.

Without a robust CI pipeline, a disconnect between the engineering team and the rest of the organization can form. Communication between product and engineering can be cumbersome. Engineering becomes a black box which the rest of the team inputs requirements and features and maybe gets expected results back. It will make it harder for engineering to estimate time of delivery on requests because the time to integrate new changes becomes an unknown risk.

What CI does

What CI does

CI helps to scale up headcount and delivery output of engineering teams. Introducing CI to the aforementioned scenario allows software developers to work independently on features in parallel. When they are ready to merge these features into the end product, they can do so independently and rapidly. CI is a valuable and well-established practice in modern, high performance software engineering organizations.

How CI can be used

CI is generally used alongside an agile software development workflow. An organization will compile list of tasks that comprise a product roadmap. These tasks are then distributed amongst software engineering team members for delivery. Using CI enables these software development tasks to be developed independently and in parallel amongst the assigned developers. Once one of theses tasks is complete, a developer will introduce that new work to the CI system to be integrated with the rest of the project.

CI vs Continuous Deployment vs Continuous Delivery

Continuous integration, deployment, and delivery are three phases of an automated software release pipeline, including a DevOps pipeline. These three phases take software from idea to delivery to the end-user. The integration phase is the first step in the process. Continuous integration covers the process of multiple developers attempting to merge their code changes with the main code repository of a project.

Continuous delivery is the next extension of continuous integration. The delivery phase is responsible for packaging an artifact together to be delivered to end-users. This phase runs automated building tools to generate this artifact. This build phase is kept 'green,' which means that the artifact should be ready to deploy to users at any given time.

Continuous deployment is the final phase of the pipeline. The deployment phase is responsible for automatically launching and distributing the software artifact to end-users. At deployment time, the artifact has successfully passed the integration and delivery phases. Now it is time to automatically deploy or distribute the artifact. This will happen through scripts or tools that automatically move the artifact to public servers or to another mechanism of distribution, like an app store.

Benefits and challenges of continuous integration

Continuous integration is an essential aspect of DevOps and high-performing software teams. Yet CI benefits are not limited to the engineering team but greatly benefit the overall organization. CI enables better transparency and insight into the process of software development and delivery. These benefits enable the rest of the organization to better plan and execute go to market strategies. The following are some of the overall organizational benefits of CI.

Enable scaling

CI enables organizations to scale in engineering team size, codebase size, and infrastructure. By minimizing code integration bureaucracy and communication overhead, CI helps build DevOps and agile workflows. It allows each team member to own a new code change through to release. CI enables scaling by removing any organizational dependencies between development of individual features. Developers can now work on features in an isolated silo and have assurances that their code will seamlessly integrate with the rest of the codebase, which is a core DevOps process.

Improve the feedback loop

Faster feedback on business decisions is another powerful side effect of CI. Product teams can test ideas and iterate product designs faster with an optimized CI platform. Changes can be rapidly pushed and measured for success. Bugs or other issues can be quickly addressed and repaired.

Enhance communication

CI improves overall engineering communication and accountability, which enables greater collaboration between development and operations in a DevOps team. By introducing pull request workflows tied to CI, developers gain passive knowledge share. Pull requests allow developers to observe and comment on code from other team members. Developers can now view and collaborate on feature branches with other developers as the features progress through the CI Pipeline. CI can also be used to help QA resource expenses. An efficient CI pipeline with high-confidence automated test coverage will safeguard from regressions and ensure that new features match a specification. Before new code is merged it must pass the CI test assertion suite which will prevent any new regressions.

The benefits of CI far outweigh any challenges in adoption. That said, it is important to be aware of the challenges of CI. The real challenges of CI arise when transitioning a project from no CI to CI. Most modern software projects will adopt CI from early inception stages and alleviate the challenges of later adoption.

Adoption and installation

The challenges of continuous integration are primarily around team adoption and initial technical installation. If a team doesn't currently have a CI solution in place, it can require some effort to pick one and get started. Thus, considerations need to be made around the existing engineering infrastructure when installing a CI pipeline.

Technology learning curve

CI functionality comes with a list of supportive technologies that may be learning curve investments for the team to undertake. These technologies are version control systems, hosting infrastructure, and orchestration technologies.

CI best practices

Test Driven Development

Once a project has established a CI pipeline with automatic test coverage, it is a best practice to constantly develop and improve the test coverage. Each new feature coming down the CI pipeline should have an accompanying set of tests to assert that the new code is behaving as expected.

Test Driven Development (TDD) is the practice of writing out the test code and test cases before doing any actual feature coding. Pure TDD can closely involve the product team to help craft an expected business behavior specification, which can then be transformed into the test cases. In a pure TDD scenario, developers and product team will meet and discuss a spec or list of requirements. This list of requirements will then be converted into a checklist of code assertions. The developers will then write code that matches these assertions.

Pull requests and code review

Most modern software development teams practice a pull request and code review workflow. Pull requests are a critical practice to effective CI. A pull request is created when a developer is ready to merge new code into the main codebase. The pull request notifies other developers of the new set of changes that are ready for integration.

Pull requests are an opportune time to kick off the CI pipeline and run the set of automated approval steps. An additional, manual approval step is commonly added at pull request time, during which a non-stakeholder engineer performs a code review of the feature.. This allows for a fresh set of eyes to review the new code and functionality. The non-stakeholder will make edit suggestions and approve or deny the pull request.

Pull requests and code review are a powerful tool to foster passive communication and knowledge share among an engineering team. This helps guard against technical debt in the form of knowledge silos, where specific engineers are the only stakeholders for certain features of a code base.

Optimize pipeline speed

Given that the CI pipeline is going to be a central and frequently used process, it is important to optimize its execution speed. Any small delay in the CI workflow will compound exponentially as the rate of feature releases, team size, and codebase size grows. It is a best practice to measure the CI pipeline speed and optimize as necessary.

A faster CI pipeline enables a faster product feedback loop. Developers can rapidly push changes and experiment with new feature ideas to help improve the user experience. Any bug fixes can be quickly patched and resolved as discovered. This increased execution speed can offer both an advantage over other competitors and an overall higher-quality experience to your customers.

Getting started with continuous integration

The foundational dependency of CI is a version control system (VCS). If the target code base for a CI install does not have a VCS, step one is installing a VCS. The absence of a VCS should be very unlikely on modern codebases. Some popular VCSs are Git, Mercurial, and Subversion.

Once version control is in place, finding a version control hosting platform is the next move. Most modern version control hosting tools have support and features built in for CI. Some popular version control hosting platforms are Bitbucket, Github, and Gitlab.

After version control has been established on the project, integration approval steps should be added. The most valuable integration approval step to have in place is automated tests. Adding automated tests to a project can have an initial cost overhead. A testing framework has to be installed, then test code and test cases must be written by developers.

Some ideas for other, less expensive CI approval mechanisms to add are syntax checkers, code style formatters, or dependency vulnerability scans. Once you have a version control system setup with some merge approval steps in place, you've established continuous integration!

CI is not purely an engineering specific business process. The rest of the organization, marketing, sales, and product teams will also benefit from a CI pipeline. Product teams will need to think how to parallelize execution of simultaneous streams of development. Product and engineering will work closely to determine the qualifying business functionality expectations that will make up the automated test suite.

Marketing and sales will be able to reference the CI pipeline to coordinate with customer facing communications efforts and events. CI gives a level of transparency to the rest of the organization on how engineering execution is progressing. This transparency and communication utility integrates gracefully with an agile project development workflow.

In conclusion...

If your organization strives to reap the benefits of a DevOps approach or simply has a multiple-developer software team, CI is important. It will help your engineering organization execute quicker and more effectively.

CI is a standard fixture of modern high efficiency software development organizations. The best companies have robust CI pipelines and don't think twice about further efficiency investments. The benefits of CI are not limited to the engineering team and applicable to the whole organization.

Many third-party tools exist to aid in CI management and installation. Some popular options are Codeship, Bitbucket Pipelines, SemaphoreCI, CircleCI, Jenkins, Bamboo, Teamcity, and many others. These tools have their own in-depth setup guides and documentation to help get started.

Some of the best CI tools are provided by Atlassian. Bitbucket pipelines and Bamboo are great utilities to bring a project up to speed with modern CI features. Jira is one of the world's most

popular agile and DevOps project management tools. Jira tightly integrates with other Bitbucket projects and when coupled with a CI pipeline, can give a very transparent view into the execution health of an organization.

MAX REHKOPF

As a self-proclaimed "chaos muppet" I look to agile practices and lean principles to bring order to my everyday. It's a joy of mine to share these lessons with others through the many articles, talks, and videos I make for Atlassian

TUTORIAL

Continuous Integration Tutorial

This tutorial will show you how to get started with continuous integration in three simple steps.

Try this tutorial →

UP NEXT

How to get to Continuous Integration

Learn about how to adopt continuous integration and automated testing in 5 steps.

Read this article →

CI/CD Topics

Continuous Delivery
Continuous Integration
Continuous Deployment
Pipelines
Software Testing
Microservices
Tutorials

Sign up for more CI/CD articles and tutorials.

Email

email@example.com

Subscribe

Up Next
How to get to Continuous Integration →