How to get started with Continuous Integration

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

BY STEN PITTET

Continuous integration (CI) 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. Since it automates deployment, it helps teams meet business requirements, improve code quality, and increase security.

One of the primary benefits of adopting CI is that it will save you time during your development cycle by identifying and addressing conflicts early. It's also a great way to reduce the amount of time spent on fixing bugs and regression by putting more emphasis on having a good test suite. Finally, it helps share a better understanding of the codebase and the features that you're developing for your customers.

The first step on your journey to continuous integration: setting up automated testing.

Getting started with automated testing

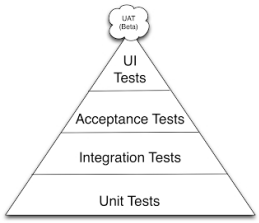
Understanding the different types of tests

To get the full benefits of CI, you will need to automate your tests to be able to run them for every change that is made to the main repository. We insist on running tests on every branch of your repository and not just focus on the main branch. This way you will be able to capture issues early and minimise disruptions for your team.

There are many types of tests implemented, but it is not necessary to do everything at once if you're just getting started. You can start small with unit tests and work on extending your coverage over time.

- **Unit tests** are narrow in scope and typically verify the behaviour of individual methods or functions.
- **Integration tests** make sure that multiple components behave correctly together. This can involve several classes as well as testing the integration with other services.
- **Acceptance tests** are similar to the integration tests but they focus on the business cases rather than the components themselves.
- **UI tests** will make sure that the application functions correctly from a user perspective.

Not all the tests are equal, and you can visualize the tradeoffs that you will make with the test pyramid developed by Mike Cohn.



Source

Unit tests are fast and cheap to implement as they're mostly doing checks on small pieces of code. On the other hand UI tests will be complex to implement and slow to run as they often require to get a full environment started as well as multiple services to emulate browser or mobile behaviours. Because of that, you may want to limit the

number of complex UI tests and rely on good Unit testing at the base to have a fast build and get feedback to developers as soon as possible.

Running your tests automatically

To adopt continuous integration, you will need to run your tests on every change that gets pushed back to the main branch. To do so, you will need to have a service that can monitor your repository and listen to new pushes to the codebase. There are many solutions that you can choose from both on-premise and in the Cloud. You'll need to consider the following to pick your server:

- **Where is your code hosted?** Can the CI service access your codebase? Do you have a special restriction on where the code can live?
- **What OS and resources do you need for your application?** Is your application environment supported? Can you install the right dependencies to build and test your software?
- **How much resource do you need for your tests?** Some Cloud applications might have restrictions on the resources that you can use. If your software consumes a lot of resources, you might want to host your CI server behind your firewall.
- **How many developers are on your team?** When your team practice CI you will have many changes pushed back to the main repository every day. For the developers to get the fast feedback, you need to reduce the amount of queue time for the builds, and you will want to use a service or server that gives you the right concurrency.

In the past, you typically had to install a separate CI server like Bamboo or Jenkins, but now you can find solutions on the Cloud that are much simpler to adopt. For instance, if your code is hosted on Bitbucket Cloud you can use the Pipelines feature in your repository to run tests on every push without the need to configure a separate server or build agents, and with no restriction on concurrency.

```
image: node:4.6.0 pipelines:   default:     - step:
```

Example of configuration to test a Javascript repository with Bitbucket Pipelines.

Use code coverage to find untested code

Once you adopt automated testing it is a good idea to couple it with a test coverage tool that will give you an idea of how much of your codebase is covered by your test suite.

It is good to aim a coverage above 80% but be careful not to confuse high percentage of coverage with a good test suite. A code coverage tool will help you find untested code but it is the quality of your tests that will make the difference at the end of the day.

If you're just getting started don't rush in attaining 100% coverage of your codebase, but rather than that use a test coverage tool to find out critical parts of your application, that do not yet have tests and start there.

Refactoring is an opportunity to add tests

If you are about to make significant changes to your application you should start by writing acceptance tests around the features that may be impacted. This will provide you with a safety net to ensure that the original behavior has not been affected after you've refactored code or added new features.

Adopting continuous integration

While automating your tests is a key part of CI it is not enough by itself. You may need to change your team culture to make sure that developers do not work days on a feature without merging their changes back to the main branch and you'll need to enforce a culture of a green build

Integrate early and often

Whether you're using trunk-based development or feature branches, it is important that developers integrate their changes as soon as possible on the main repository. If you

changes as soon as possible on the main repository. If you let the code sitting on a branch or the developer workstation for too long, then you expose yourself to the risk of having too many conflicts to look at when you decide to merge things back to the main branch.

By integrating early, you reduce the scope of the changes which makes it easier to understand conflicts when you have them. The other advantage is to make it easier to share knowledge among developers as they will get more digestible changes.

If you find yourself making some changes that can impact an existing feature you can use feature flags to turn off your changes in production until your work is completed.

Keep the build green at all time

If a developer breaks the build for the main branch, fixing it becomes the main priority. The more changes get into the build while it's broken, the harder it will be for you to understand what broke it - and you also have the risk of introducing more failures.

It is worth spending time on your test suite to make sure that it can fail fast and give feedback to the developer that pushed the changes as soon as possible. You can split your tests so that the fast ones (Unit Tests for example) run before the long-running tests. If your test suite always takes a long time to fail, you will waste a lot of developer time as they'll have to switch context to go back to their previous work and fix it.

Don't forget to set notifications to make sure that developers are alerted when the build breaks, and you can also go a step further by displaying the state of your main branches on a dashboard where everyone can see it.

Write tests as part of your stories

Finally, you'll need to make sure that every feature that gets developed has automated tests. It may look like you will slow down development but in fact, this is going to reduce drastically the amount of time that your team spends on fixing regression or bugs introduced in every iteration. You will also be able to make changes to your codebase with confidence as your test suite will be able to rapidly make sure that all the previously developed features work as expected.

To write good tests, you will need to make sure that developers are involved early in the definition of the user stories. This is an excellent way to get a better-shared understanding of the business requirements and facilitate the relationship with product managers. You can even start by writing the tests before implementing the code that will fulfill them.

Write tests when fixing bugs

Whether you have an existing codebase or you just getting started, it is certain that you will have bugs occurring as part of your releases. Make sure that you add tests when you solve them to prevent them from occurring again.

CI will enable your QA Engineers to scale quality

Another role that will change with the adoption of CI and automation is the role of the QA Engineer. They no longer need to test manually trivial capabilities of your application and they can now dedicate more time to providing tools to support developers as well as help them adopt the right testing strategies.

Once you start adopting continuous integration, your QA Engineers will be able to focus on facilitating testing with better tooling and datasets as well as help developers grow in their ability to write better code. There will still be some exploratory testing for complex use cases, but this should be a less prominent part of their job.

Continuous integration in 5 steps

You should now have a good idea of the concepts behind continuous integration, and we can boil it down to this:

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

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.

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.

Our recommendation is 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.

continuous delivery

What Is Continuous Deployment?

Microservices and Microservices Architecture

Bitbucket CI/CD tutorials

Continuous Delivery articles

STEN PITTET

I've been in the software business for 10 years now in various roles from development to product management. After spending the last 5 years in Atlassian working on Developer Tools I now write about building software. Outside of work I'm sharpening my fathering skills with a wonderful toddler.

TUTORIAL

Continuous Integration Tutorial

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

Try this tutorial →

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.

Read this article →

Up Next

How infrastructure as a service empowers the modern enterprise →