How to get to Continuous Deployment

In this guide, we will assume that your continuous integration and continuous delivery workflow is already in place and that your next step is to set up your continuous delivery pipeline.

BY STEN PITTET

Browse topics

Continuous Delivery Principles

Continuous Delivery Pipeline 101

What is Continuous Integration

Software testing for continuous delivery

What Is Continuous to Continu Deploy

Microser and Microser Architect

Bitbucke CI/CD tutorials

Continuc Delivery articles

> Also, check out our tutorial on how to get started with Continuous Deployment on Bitbucket Pipelines.

"Nobody touch production – I'm about to release!"

That phrase may be familiar to you. Deploying to production can be a risky and costly exercise that sometimes requires putting all development on hold. This causes release cycles to be slow and changes to stagnate in the development environment. It's the start of an awkward (and vicious) circle in which the more commits are made to the repository, the more risks get introduced into the next deployment to production, and the less likely your team is to make *that* release.

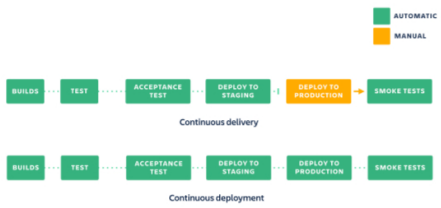
Continuous deployment solves this problem by automatically shipping every change pushed to the main repository to production. It's a radical approach – very different from spending days preparing and controlling each release – but there are several benefits to continuous deployment:

- Releases become smaller and easier to understand.
- No one is required to drop their work to make a deployment because everything is automated.
- The feedback loop with your customers is faster: new features and improvements go straight to production when they're ready.

It can be quite daunting to go from approved releases to continuous deployment, but this guide is here to help you make that transition. We'll cover the foundations so that you can get started as soon as possible and start accelerating your release cycle.

Continuous deployment vs. continuous delivery

If you want to implement a continuous deployment pipeline, the best place to start with continuous delivery. These two practices are similar, but differ in their approaches to production deploys. In continuous delivery, every change pushed to the main repository is ready to be shipped, but the production release process still requires human approval. In continuous deployment, the release to production is done automatically for every change that passes the test suite.



Before you start shipping changes automatically to production, you'll need to have a good continuous integration (CI) culture, and it's highly recommended to start with continuous delivery first. You can read more about the two practices in the articles below:

- Continuous Integration guide
- Continuous Delivery guide

Moving from continuous delivery to continuous deployment

Emphasize a culture of continuous integration

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.

Make sure you have good test coverage (and good tests too!)

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.

Adopt real-time monitoring

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.

Review your post-deployment tests

After each deployment to production, you should run some simple smoke tests to make sure that the application is up and running.

Make sure that these tests do more than just hitting a static page, waiting for a success response. It's good practice to have a test that requires that all your production services (database, micro-services, 3rd parties...) are working properly.

Get your QA team to work upstream

As every commit goes now straight to production, it means that there's no traditional buffer for your QA to review and approve new features. Instead, they should be working closely with the product manager and the development team to define the risks associated with new improvements.

This is a great opportunity for the QA team as it will result over time in better quality for the releases. With continuous deployment, your team will be naturally inclined to have good test coverage of the codebase. Failing to do so means frequent disruptions due to bugs slipping out to production where they're discovered by your users.

Drop the traditional release notes

Continuous deployment makes it hard for you to keep up with releases – and that's a good thing! Your goal should be to have daily deployments to production, even multiple deployments per day. It may be bug fixes, small improvements, new features... whatever. As soon as a developer finishes their work, that work should be running in production in a matter of minutes.

In this new world it's no longer possible to write release notes that get sent to your customers with each new version. Instead, embrace this new flow and limit your announcements to key features that matter to your customers. Bug reports and small improvement requests can simply be handled in your ticketing system – JIRA, for instance, can update all watchers of a ticket as soon as you close or update it.

A different approach for new projects: deploy to production before coding

There's something interesting that you can do if you're getting started on a brand new codebase. You can start by creating your continuous deployment pipeline before having any customers and any feature ready (not unlike test-driven development). At the beginning, you'll be shipping an empty platform. Then, as development progresses, new capabilities will start appearing automatically. You just need to keep your production environment closed until you're ready to open it up for your customers.

This approach may seem counter-intuitive at first. But it's an easy way to take the stress out of going from a manual release approval to continuously deploying to production. It's also an excellent way to make sure that you have a good test coverage and continuous integration culture at launch since they necessary conditions for continuous deployment.

Build your knowledge and share your experience!

It's understandably hard to take the leap and switch to continuous deployment. All of a sudden the gates are lifted, and code goes straight to the customers in a matter of minutes or hours. (Eep!) So it's important to invest in the tests and automations that give you confidence in your builds. This will be easy if your codebase is new, but it may require adopting a different strategy for a complex ledgacy codebase.

Start small and build up your continuous deployment knowledge and experience. Once you get there for one project, it will be easy to replicate your success in other parts of your organization.

What is Continuous Integration

Software testing for continuous delivery

What Is Continuous Deployment?

Overview

- **How to get to 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 Deployment Tutorial

This tutorial will show you how to get started with continuous deployment with Bitbucket Pipelines.

Try this tutorial →

ARTICLE

Microservices and Microservices Architecture

The guiding principle of microservices is to build applications by splitting business components into services that can be deployed and operated independen

Read this article →

Subscribe

Up Next
Microservices and Microservices Architecture →