Benefits of CI/CD

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)

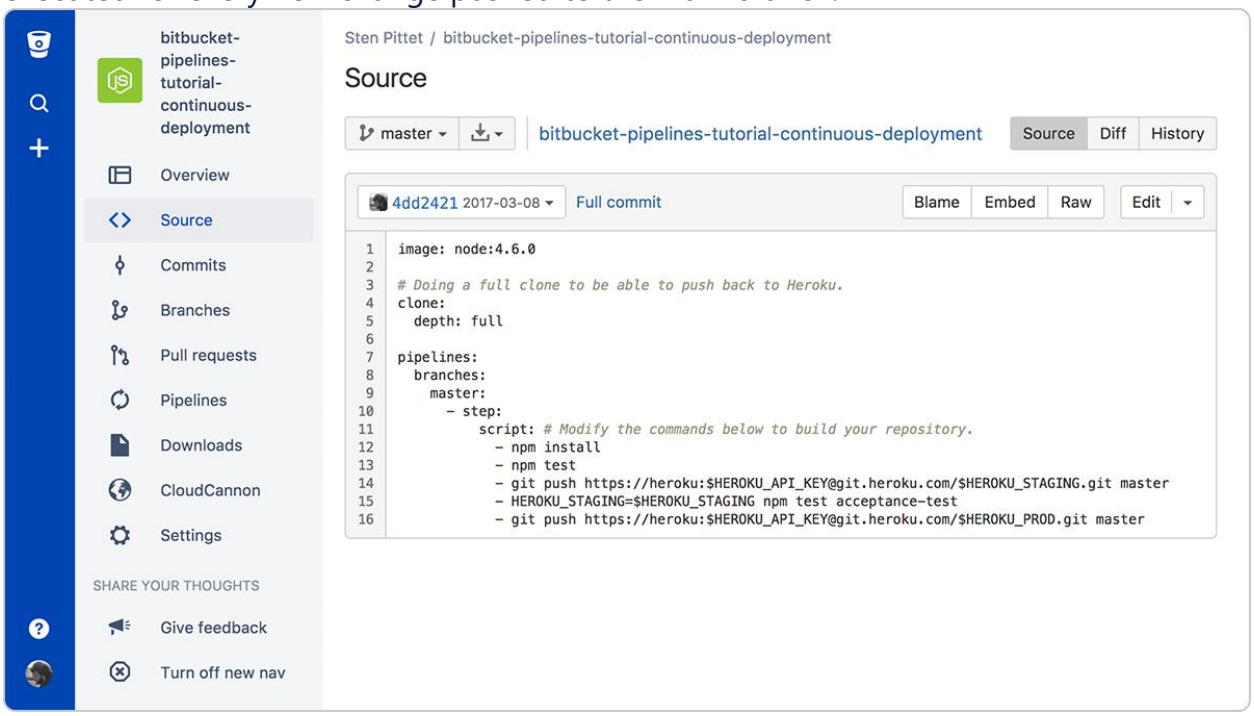
- Your testing culture needs to be at its best. The quality of your test suite will determine the quality of your releases.
- Your documentation process will need to keep up with the pace of deployments.
- Feature flags become an inherent part of the process of releasing significant changes to make sure 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.

One of the traditional cost associated with continuous integration is the installation and maintenance of a CI server. But you can reduce significantly the cost of adopting these practices by using a cloud service like Bitbucket Pipelines which adds automation to every Bitbucket repository. By simply adding a configuration file at the root of your repository you will be able to create a continuous deployment pipeline that gets executed for every new change pushed to the main branch.



Going from continuous integration to continuous deployment

If you're just getting started on a new project with no users yet, it might be easy for you to deploy every commit to production. You could even start by automating your deployments and releasing your alpha version to production with no customers. Then you can ramp up your testing culture and make sure that you increase code coverage as you build your application. By the time you're ready to onboard users, you will have a

great continuous deployment process where all new changes are tested before being automatically released to production.

But if you already have an existing application with customers you should slow things down and start with continuous integration and continuous delivery. Start by implementing basic unit tests that get executed automatically -- there's no need to focus yet on running complex end-to-end tests. Instead, you should try automating your deployments as soon as possible and get to a stage where deployments to your staging environments are done automatically. The reason is, if you have automatic deployments, you can focus your energy on improving your tests rather than periodically stopping things to coordinate a release.

Once you can start releasing software on a daily basis, you can look into continuous deployment. But make sure that the rest of your organization is ready as well: documentation, support, marketing, etc. These functions will need to adapt to the new cadence of releases, and it is important that they do not miss on significant changes that can impact customers.