Much like with [agile development](https://www.sitepoint.com/entrepreneur/business/agile-development-business/), DevOps isn’t a particular tool or technique that you can implement and you’re done. Rather, it’s a culture — even a mindset — that your team and organization can adopt and that will make processes smoother.

But what is it exactly? Think of developers being able to do some of the tasks that are normally assigned to system administrators, such as creating new servers, making updates to production sites, deploying apps. Also think of sysadmins sharing some duties with developers, and being able to manage multiple servers at once.

**Benefits**

Companies embracing a DevOps culture report significant improvements, and different surveys conducted among some of them seem to confirm these claims (see “[Continuous Delivery: Huge Benefits, but Challenges Too](http://ieeexplore.ieee.org/document/7006384/?reload=true&arnumber=7006384)“).

Some of the these improvements include:

* **Accelerated**[time-to-market](https://en.wikipedia.org/wiki/Time_to_market): shorten the time it takes from when an idea is conceived until it’s available for sale or in production.
* **Building the right product**: developers get faster feedback from users with more frequent releases and live testing of ideas (more on this later on the “A/B Testing” section).
* **Cost reduction**: reports average a 20% reduction.
* **Improved productivity**: with continuous delivery, developers and testers save time setting up and fixing their working environments. Also, deployments are significantly faster (more on this later in the “Continuous Integration with Jenkins” section).
* **Reliable releases**: with smaller and more frequent releases, the changes in code — and therefore the bugs introduced and their impact — are also smaller.
* **Improved product quality**: companies report a very significant reduction in open bugs and other issues (in some cases, by more than 90%).
* **Improved customer satisfaction**: this is, not surprisingly, a byproduct of all of the previous improvements.

The conventional wisdom goes that one shouldn’t mix production with development environments — that these two should be carefully separated in order to avoid disasters. And while it’s true that making changes to a live platform without testing and rollback mechanisms is, in fact, a formula for disaster, having multiple environments comes at a high operational cost.

**Automating the creation of environments** is key in DevOps, and one technology that enables this is [Docker](https://www.docker.com/). We won’t get into the full details here, but the principle is that, with Docker, you can create a well-tested and functional environment (called an *image*) that developers and engineers in the operations side will be able to launch as *software containers*. Whenever an update or patch is required, the QA team will just patch the image, and the changes will be replicated in every new container that’s launched.

This gives independence to programmers and designers to launch environments that are constantly updated and known to work (and as many of them as they need), and even deploy changes to live platforms, because the QA people are maintaining these environments and have already set the appropriate tests. It also provides a communication channel with the server administrators, because they’re all basically working together in having a working environment that’s good for both production and development.

Notice that this is also valid if development and deployment are carried out by the same people — as in a single freelance developer — because these concerns for testing and implementing a live application in different environments are still valid regardless of who’s in charge of doing it.

The principle is that you can **test variants of an idea on live environments** to make improvements in usability and other areas, based on popularity or users feedback. The reckless part is that you perform these tests with your actual audience — who are the ones best qualified for feedback anyway — which is something that Facebook, for example, [does on a big scale](http://www.pnas.org/content/111/24/8788.full) to conduct social experiments. In the web dev field, think of [experiments](https://support.google.com/analytics/answer/1745147?hl=en) for [testing landing pages with Google Analytics](https://support.google.com/adwords/answer/2404033?hl=en).

How is all of this related to DevOps? Well, with **automated deployment processes** you give the parties involved — be they designers, developers, or even scientists with no background whatsoever in programming or engineering — the autonomy they need to conduct the experiments — again, with the safeguard mechanisms in place to test, pull or rollback changes.

[Jenkins](https://jenkins.io/) is a [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration) (CI) and [continuous delivery](https://en.wikipedia.org/wiki/Continuous_delivery) (CD) software — an orchestration system with hundreds of plugins to automate everything from building an application and testing it, to the final deployment. These plugins can integrate building software from source code platforms such as Git or Mercurial, to cloud services such as Amazon Web Services (AWS) or the Google Cloud Platform (GCP), passing through all of the scheduled tests. It is, essentially, **a pipeline from source to delivery** that’s [becoming the engine of DevOps](http://www.infoworld.com/article/3046038/application-development/why-jenkins-is-becoming-the-engine-of-devops.html).

Again, we see how everyone involved (developers, QA and sysadmins) work together on an effective and automated workflow:

* Do **developers need a new working environment**? Jenkins will fire up Docker to launch it.
* Do **sysadmins require a new QA test to pass before an update can go live**? They just add it to the Jenkins pipeline.
* Do **developers want to pull changes to a production site and launch new servers with it**? No problem: sysadmins have already instructed Jenkins what tests to run, and if everything goes fine, how to launch the servers.

As you can see, there’s a lot of automation in the process, and a fair amount of trust between the parties — all of which requires a lot of work to set up, but which, once in place, also gives everyone a lot of room to operate on their own.

Switching to a DevOps culture may bring a number of benefits, but it also comes with certain challenges to overcome.

The first one is at an **organization** level. As development and deployment constraints are removed and programmers and sysadmins have more independence, the people involved will need to adopt a different mindset, and the appropriate mechanisms for a *feedback loop* from ops to dev will need to be set in place — such as issue trackers, boards of discussions (think of [Slack](https://slack.com/)), and the like.

The second key challenge is regarding **processes**. You no longer want developers and server administrators themselves using their time to individually test after changes. What you’ll want is to automate the testing processes, so that you can allow different teams to make changes and quickly check that things are still in place, and reverse those changes should problems arise.

Finally, there’s a **technology** challenge. As mentioned, once your organization has carefully reviewed the processes from end to end, you may need either to adopt or create a technology that addresses the kind of automation and feedback loop that better suits your processes and organization.

It isn’t trivial: **you can’t simply copy/paste** the processes and technologies others are using. These issues need to be addressed individually, and there has to be a fair level of trust between the parties involved.