CI/CD

Definition

First, a quick definition. All that continuous delivery actually means is that you always are in a position to deploy a working software build at any point in time. It implies that you have the capability to press a single button and have your software tested and deployed to a *production-like* environment.

Continuous delivery is often confused with Continuous **Deployment** which is the practice of actually deploying every change to production. This means that as soon as a developer commits a change to the source control repository, that change will automatically be deployed to your real users. This doesn't always make sense or might not be practical for every organization, but if continuous deployment is your goal, it is important to understand that continuous delivery is a prerequisite.

The Benefits of Continuous Delivery

Now let's talk about some of the benefits of adopting Continuous Delivery in your organization.

Speed

Having a solid continuous delivery pipeline in place means that you can ship faster. If you have a critical bug fix or an important new feature that must get out as quickly as possible, your investment in continuous delivery will allow you to do so confidently and with as little pain and stress as possible.

Your customers will not be annoyed by the same bug for months at a time. You will be able to adapt and iterate based on the feedback of your customers much

more quickly. If you find that your application is not quite meeting the needs of your customers, you will be able to shift direction more easily. Your ability to fix problems and deliver improved features will not be constrained by complicated and bureaucratic deployment processes. In short, continuous delivery provides you a competitive advantage by making your organization more nimble.

Flexibility

The value of getting new features out to real users and adapting those features based on their feedback cannot really be overstated. It can be transformative in the way you develop your application and, potentially, to your entire business.

One of the components of a robust continuous delivery practice are feature toggles. This is the ability to enable or disable certain parts of an application with a simple configuration change. You may want to disable features that are not yet fully baked or even finished at all. Or, you might enable new features for only a small subset of your users while you gauge their response and gather their feedback. With this capability in place it means that releases do not have to be held up while developers work on a big new feature. And, it means you can move away from long running feature branches. Ideally, you would be breaking up large features into smaller pieces but this is not always practical and feature toggles allow you to introduce and integrate these larger features into the code-base much more quickly.

This kind of flexibility makes it much easier for your team to experiment with new ideas. You can test out a new feature for a small group of users and quickly iterate on it based on their usage and feedback. If a particular feature is not well-received or even working at all, you can more easily rollback. All in all, having this capability will again contribute to making your software better and give you an advantage over your competition.

Risk

When you do not have the practices that make up continuous delivery in place, such as automated testing and deployment amongst other things, you will find that every release is a risk. Especially, if you are waiting weeks or months between

releases. With each deployment, in the back of your mind, you are going to be wondering what will break and how long it will take you and your team to fix.

Non-automated deployments typically require extensive coordination and communication between various teams. Often these procedures are bureaucratic nightmares that would make Kafka tremble. Picture interminable pre-release meetings, never-ending checklists, and indecipherable email chains. Even with all of this ceremony, something will usually go wrong. And when it does, prepare for another round of inefficient communication, finger pointing and exhausting postmortems.

In a continuous delivery environment, software is released frequently (even if only to testing or staging environments). Therefore, all of the processes that make up a release are continuously refined and improved. Since you will be deploying so often, there is a large incentive to iron out all problems with the deployment process itself. Manual steps are automated. Flaky processes become more resilient. Over time, your release process will become rock solid and your releases will become predictable and uneventful.

Quality

As you introduce continuous delivery into your organization and start releasing more frequently, you will start to notice the quality of your software improve. Bugs will be shorter lived. You will discover them faster and be able to fix them more quickly. If you are in an organization that is used to broken, buggy releases, you may think that by releasing more frequently, you are just going to be causing more stress. After all, if you release broken software once a week instead of once every three months, you are just going to be scrambling to put out fires every week instead of every three months. But, if you take the time to implement solid continuous delivery practices, you will find that you will be releasing better quality software and you will not have as many late nights cleaning up after your broken deployment.

Focus

When working on an application that has long, infrequent release cycles, it is common for developers to work on several features or bug fixes before seeing

any of them released to production. This can be a problem. Let's say a developer finished a particularly difficult feature. Now, they have moved on to other things. It takes several weeks before that feature is released to actual customers and, inevitably, your team will quickly learn that something is either broken or could just be improved. But, now it is going to be more of a challenge for that developer to address those issues. The developer is going to be less productive because they likely haven't looked at that code in a long time. Or, even worse, perhaps that developer is on vacation or has left the company?

On the contrary, with more frequent deployments, the time between when a developer works on code and when it is delivered to customers is much shorter. Problems are discovered more quickly and the code will be fresh in the minds of developers so that they will be much more productive if they have to change it.

Morale

Developers want to see the code they have spent their time and energy on in the hands of users. This is an important feedback loop that keeps them motivated and involved. It is a completely demoralizing experience to work hard on something and then having to wait seemingly forever for it to actually get used. Shorter release cycles will help keep your development team fired up.

In addition, continuous delivery practices help to ensure that software releases are no longer stressful events. This also has a positive effect on the dedication and enthusiasm of both development and operations teams.

Cost Savings

In some organizations, the number of people and the hours involved in preparing for a new software release can be truly staggering. Consider how much money is being spent on meetings, release committees, multiple layers of management, documentation and checklists, human labor, and other inefficient processes. Imagine if all of those manual, redundant and ineffective procedures were eliminated with an automated continuous delivery pipeline that actually results in better quality software. That is the achievable dream of continuous delivery.