Now let’s talk about the DevOps practices known as continuous delivery and continuous deployment.

Continuous delivery, sometimes referred to as CD, is an ongoing DevOps practice of building, testing, and delivering code in an always ready and deployable state. Breaking it down, the DevOps team doesn't care that the code is being worked at any particular moment as the version of the code is always in a state that can be deployed with a click of a button.

This allows the team to determine whether deploying the code at a certain point is purely a business determination. This ability to deploy the code at any point will not come up with a developer saying, 'No, we are not ready', or 'It needs to be tested'. It will be a management decision on whether enough additional features are ready to deploy the new version.

The reason in this lesson that it has both continuous delivery and continuous deployment is that some people see the terms as interchangeable. It is even worse when both are called CD, so try to avoid the acronym CD if you can. Continuous delivery and continuous deployment are two separate concepts, even though they're very closely related.

So what is continuous deployment? Continuous deployment is the practice of frequently deploying small code changes to production automatically after each committed and built code. Think, continuous delivery is to keep the code in a state that is able to be deployed whenever you hit the commit button.

In comparison, continuous deployment means that the code is producing frequent and automated deployments. Keep in mind there is no standard on how many times you deploy each day, but some companies deploy continuously to production multiple times a day. If possible, the more you deploy the code, the better and more user-friendly the product is.

While we mention continuous deployment, let us go into why it is essential to practice. When deploying code to production, whether in continuous delivery or continuous deployment, it creates a culture where any code deployment to production is not a big, scary event that rarely happens.

So what does it look like when you do continuous delivery and continuous deployment? Well, each version of the code goes through a lot of automation. It will go through a series of stages,

such as automated build, automated testing, perhaps automated packaging, and maybe some manual phases, too, such as manual acceptance testing.

This process results in a version of the code becoming an artefact or a package that is ready to be deployed in an automated way. When the decision is made to deploy, the process of actually deploying that artefact is also automatic and can happen without any additional human interaction. What this automated deployment looks like will depend on the architecture and what type of code is being deployed.

But no matter what that architecture is, there's always one thing in common with continuous deployment: the deployment is always automated. If a deployment causes a problem, it can be quickly and reliably rolled back again, using an automated process with continuous delivery and continuous deployment. Ideally, this can be done before the customer even notices the problem.

But rollbacks aren't a big deal for developers because they can redeploy a fixed version as soon as they have one available. This means rollbacks are not an obstacle for developers getting their features in front of customers because they can quickly deploy their code once they have a fix. Having this function allows the developers not to be worried about the deployment because they occur frequently. Even if the worst-case scenario plays out and the deployment does cause a problem, a rollback is readily available to prevent an outage. Hence, deployments are commonplace, everyday events that happen all the time, and it's not a big deal.

But what do you gain should you do continuous delivery and continuous deployment? Well, continuous delivery and continuous deployment are excellent practices for achieving a faster time to market. They let you get features in the hands of customers quickly, rather than waiting for a lengthy and often cumbersome deployment process that only happens once every couple of weeks or once a quarter.

This constant deployment process will cause fewer problems. Often, the most complex IT processes are rarely used because the IT team doesn't have an incentive to fix the underlying issues with those processes. But when you have deployments frequently, that process is commonly used and often exercised, and any problems with the process are easily discovered and quickly fixed.

Continuous deployments are much lower risk. The more changes you deploy at once, the higher the risk.

Frequent deployments involved deploying only a small number of changes at a time. So with fewer changes in that one deployment, there is less risk that the particular deployment will cause a problem.

Another benefit of continuous delivery and continuous deployment is reliable rollbacks. Because these processes require robust automation, rollbacks benefit from that robust automation and become more reliable.

Rollbacks become a great tool to ensure stability for customers. They're not seen as 'the end of the world'. They're a good thing because the rollbacks prevent customers from experiencing errors or downtime. Rollbacks also don't hurt the developers because the developers can deploy the fix as soon as they have one.

A quick summary of all that is: continuous delivery and continuous deployment will give you fearless deployments. When coupled with the ability to roll back quickly and reliably, this robust automation means that deployments are commonplace, everyday events. They're not this big scary event that has everyone gripping their desk in fear, hoping that nothing goes wrong.