Let us move onto continuous integration.

Continuous integration, referred to as CI in DevOps, is the practice of frequently merging code changes done by developers. Traditionally each developer would control separate portions of the codebase, and they might work for weeks or even months before they would try to merge those things altogether into one whole application. This time delay for the application could then run into an issue when merging that bit of code to the source code if updates have rendered the work incompatible.

Continuous integration means constantly merging throughout the day, rather than merging large changes all at once or bringing together a lot of small changes. With the help of build automation, unit tests that can automatically detect errors when merges occur will stop the code from being deployed.

Let's go into what continuous integration looks like. While continuous integration is usually done with the help of a continuous integration server every time a developer commits the code change. This server sees the change and automatically triggers it to execute an automated build. As discussed in the build automation lesson, it will then perform any automated tests against the code. These tests run against every single version of the code, even if deployed multiple times a day.

Another vital component of continuous integration is seeing when there's a problem with the build, whether the code failed to compile or an automated test that fails. The continuous integration server immediately and automatically notifies the developers to acquire urgent information on the problem.

Because of the continuous integration setup, the developers that deployed the code can quickly fix the problem or roll back a previous version if there is no rapid solution. This nimble response ensures developers can continue to work and not work on a broken code while keeping the program operational.

The benefit of continuous integration is that it allows you to detect certain types of deficiencies in their infancy stage, for example, compilation errors caused when bringing code from multiple developers together. If the code doesn't compile or one developer has changed something that generates an automated test to fail, the developers can focus on rectifying the fault and not bringing down the whole team and program.

Think about it it's much easier to fix a bug that came from a change you did just a couple of hours ago than fixing a bug you created last week. The sooner these bugs can be detected, the faster and easier they can be fixed and CI helps to minimise these bugs slipping into deployment.

Continuous integration also eliminates the scramble to integrate just before a big release. Without continuous integration, the developers are usually scrambling to bring all their code together and to get it all to work right before their release or deadline. While if you use continuous integration, that code is constantly merged, which means there's no reason to have an extensive integration effort at the end.

Continuous integration also makes frequent releases possible with the code always kept in a state that can be deployed to production without a whole lot of work to get it ready. It also makes continuous testing possible because the code can always be executed on a server where QA testers can get their hands on it all throughout the development process, not just at the end.

Continuous integration also encourages good coding practices. Frequent commits throughout the day encourage developers to create simple modular code rather than making a giant complex mess that can only be finished after several months of work. I hope you understand the benefits of CI and how breaking things down into smaller module deployments is an efficient and highly advantageous practice.