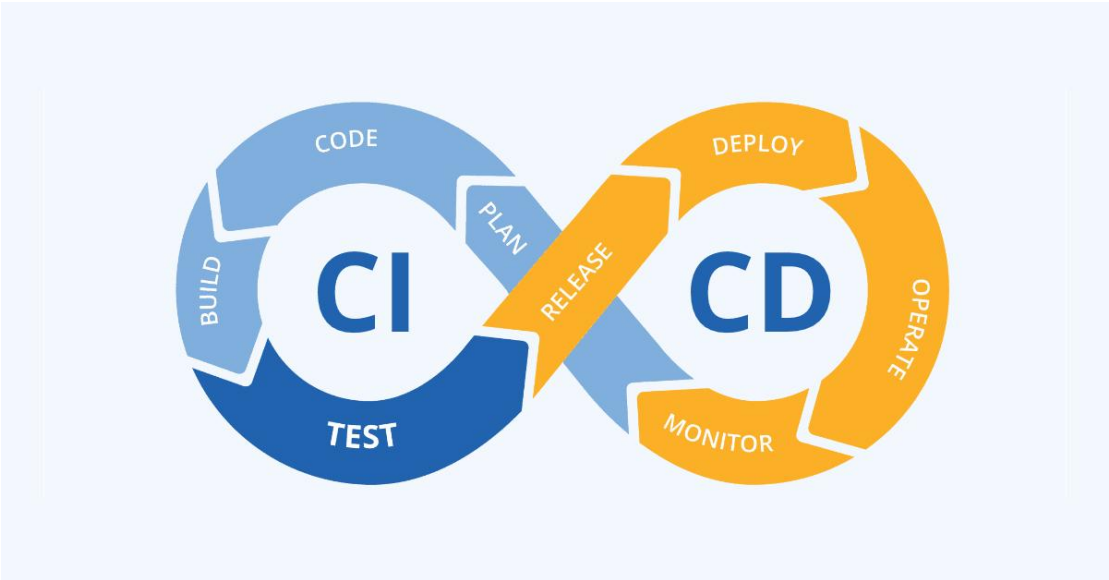

Continuous integration is the practice of merging all developer's working copies to a shared mainline several times a day and continuous delivery is an engineering practice in which teams produce and release value in short cycles. While continuous deployment is a software engineering approach in which the value is delivered frequently through automated deployments.

BENEFITS OF CI/CD TO ACHIEVE, BUILD, AND DEPLOY AUTOMATION FOR CLOUD-BASED SOFTWARE PRODUCTS.

Continuous integration, delivery and deployment (CI/CD) have enabled many organizations to release on a more frequent basis without compromising on quality.

FASTER TIME TO MARKET: The primary goal of a CI/CD pipeline is to deliver working software to users quickly and frequently. Tech giants may have led the way, adopting Agile and DevOps techniques to transform their development processes and deliver constant improvements to their users, but with many smaller organizations following suit the landscape is becoming increasingly competitive.

Understanding your users' needs, coming up with innovative features, and turning them into robust code is not necessarily enough if your competition is moving more quickly. With an automated CI/CD pipeline you can ship changes weekly, daily or even hourly.

New features can be launched faster, with deployment strategies giving you the option to experiment and collect feedback, which you can then incorporate into the next update. Being able to push changes out quickly and with confidence means you can respond to new trends and address pain-points as they emerge.

REDUCED RISK: Being able to test your innovations with users early and often – either with test participants in a pre-production environment or with real users in live – means you can validate your approach before investing months or even years working on a feature that doesn't actually solve a problem for your users.

BETTER CODE QUALITY: Automating tests ensures they are performed consistently, making the results more reliable. Because automated tests are quicker to run then their manual equivalents, it becomes feasible to test much more frequently.

Testing your code regularly and thoroughly means you'll discover bugs sooner, making it easier to fix them as less functionality has been built on top of them. Over time this results in better quality code. Once you've invested in a first layer of automated tests, the time you save on running those tests manually can be spent developing additional layers of automated tests – such as end-to-end or performance tests – and on manual exploratory testing.

SMOOTHER PATH TO PRODUCTION:
Adding automation for builds, tests, environment creation and deployments makes each step consistent and repeatable. Having broken it down, you can keep optimizing each stage to make your process more efficient. From being a significant event that

occupies multiple teams for several days, with CI/CD releasing matures into a familiar and predictable occurrence.

EFFICIENT INFRASTRUCTURE: Taking an infrastructure-as-code approach involves automating the creation of those environments. Rather than managing individual servers manually, their configuration is scripted and stored in version control so that new environments can be brought online quickly without the risk of inadvertent changes and inconsistencies.
This not only makes the continuous delivery stage faster and more robust, but also allows you to respond quickly to requests for additional preview and training environments with minimal interruption to development work.

TIGHTER FEEDBACK LOOP: Rapid feedback is a key part of the DevOps approach with applications throughout the pipeline. It starts with automated build and test steps to inform you of immediate problems, helping you to work more efficiently and effectively than if there is a long delay between the original work and the results. Feeding insights into a cycle of continuous deployment allows you to see how your changes perform soon after you've made them. That means you can keep iterating and tweaking without the loss of context that results from a long delay between coding and release.

COLLABORATION AND COMMUNICATION: Aligning around the overarching aim of delivering a product that meets user needs and understanding all the steps involved in reaching that goal helps everyone to focus on what needs to be achieved rather than being limited by their team's remit. Many of the tools available to help manage your CI/CD pipeline also make it easier for non-developers to see what is in train, while access to staging environments allows them to engage with and provide feedback on what is being built. Sharing details of what is being released, usage metrics and the results of experiments opens the door to more communication which in turn fosters innovation.

MAXIMIZED CREATIVITY: As we've seen, building a CI/CD pipeline eliminates waste and helps create a leaner, more efficient software development and release process. By using computers to perform repetitive tasks, an automated process also frees up individuals to be creative. Instead of following manual test scripts, refreshing environments, or deploying updates, you can focus on solving problems and experimenting with solutions.

In conclusion, the benefits of an automated CI/CD pipeline range from practical considerations like code quality and rapid bug fixes, to ensuring you're building the right thing for your users and improving your entire software development process.