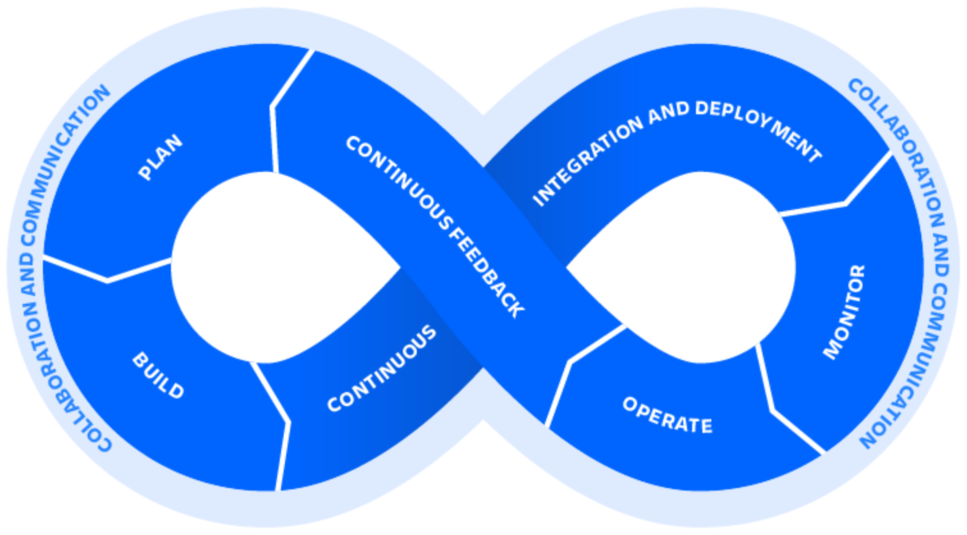
## I. Introduction to Cloud DevOps

DevOps is a set of practices that works to automate and integrate the processes between software development  and IT teams, so they can build, test, and release software faster and more reliably. The term DevOps was formed by combining the words “development” and “operations” and signifies a cultural shift that bridges the gap between development and operation teams, which historically functioned in siloes. This introduction takes us to one of the most powerful DevOps Pillars which is CICD (Continuous Integration Continuous Deployment)


COLLABORATION AND COMMUNICATION
PLAN
BUILD
CONTINUOUS
CONTINUOUS FEEDBACK
INTEGRATION AND DEPLOYMENT
MONITOR
OPERATE
COLLABORATION AND COMMUNICATION

## II. What is CICD?

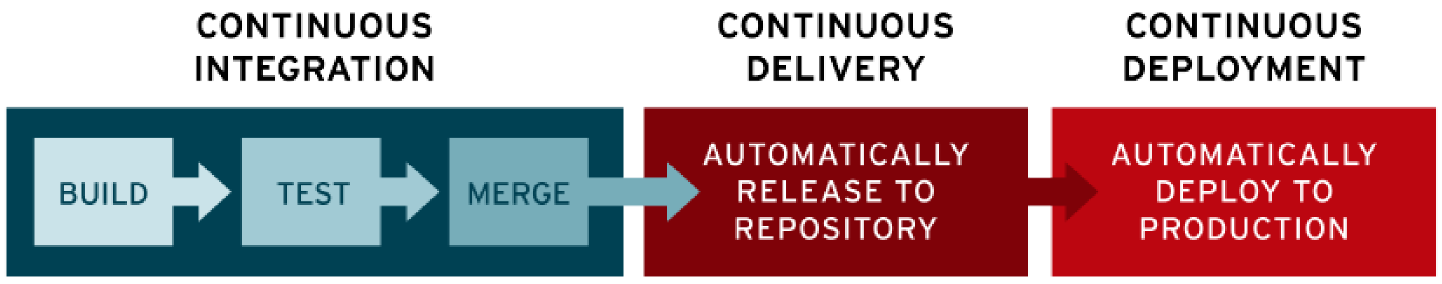
As per Redhat.com definition , CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams . Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline" and are supported by development and operations teams working together in an agile way with either a DevOps or Site reliability engineering (SRE) approach.

CI (Continuous Integration): In modern application development, the goal is to have multiple developers working simultaneously on different features of the same app. However, if an organization is set up to merge all branching source code together on one day (known as “merge day”), the resulting work can be tedious, manual, and time-intensive. Continuous integration (CI) helps developers merge their code changes back to a shared branch, or “trunk,” more frequently—sometimes even daily. Once a developer’s changes to an application are merged, those changes are validated by automatically building the application and running different levels of automated testing, typically unit and integration tests, to ensure the changes haven’t broken the app.

CD (Continuous Delivery) : Following the automation of builds and unit and integration testing in CI, continuous delivery automates the release of that validated code to a repository. So, in order to have an effective continuous delivery process, it’s important that CI is already built into your development pipeline. The goal of continuous delivery is to have a codebase that is always ready for deployment to a production environment.

Continuous Deployment The final stage of a mature CI/CD pipeline is continuous deployment. As an extension of continuous delivery, which automates the release of a production-ready build to a code repository, continuous deployment automates releasing an app to production. Because there is no manual gate at the stage of the pipeline before production, continuous deployment relies heavily on well-designed test automation.

In practice, continuous deployment means that a developer's change to a cloud application could go live within minutes of writing it (assuming it passes automated testing).



## III. Technical CICD Benefits

From the point of view of our powerful DevOps Team, please have a quick look on the below top 10 advantages they will be benefiting when using CICD:

1. Smaller Code Changes
2. Fault Isolations
3. Faster Mean Time To Resolution (MTTR)
4. More Test Reliability
5. Faster Release Rate
6. Smaller Backlog
7. Easy Maintenance and Updates

## IV. CICD Business Impact

Now let us translate the above slides on actual , real-world benefits , below you can find a very quick summary of how CICD will play a huge role in the business growth , cost reduction and revenue Protection

**1. A large portion of IT budget is spent on undifferentiated engineering**

Opportunity costs play a much larger role in the development process than we realize. Organizations can only afford so many engineers at one time, and systems that require extensive maintenance means fewer engineers are working on revenue-generating projects. This will lead to slower innovation and slower growth in the long term. Undifferentiated engineering means too many individuals are having to focus on one thing – maintenance.

**2. Delayed (and even unrealized) revenue**

This is the impact of lost opportunity costs. When there are too many dependencies, too many handoffs, and too many manual tasks, it causes delays between when code is written and when the business gets value from that code. In worst cases, code is written and the business never gets any value from it at all. Code can sit in limbo waiting for others to manually test it, and by the time it's finally reviewed it's already irrelevant. The opportunity cost essentially doubles: Engineers were paid to work on code that never deployed, and the business loses out on revenue the code could have generated.

**3. Lower developer productivity, lower developer happiness, and less reliable software**

Downtime = lost revenue. To avoid that dreaded downtime, developers are spending time working on infrastructure and configuration, and they're also not spending that time delivering business logic. In both cases, they're being less productive and working outside of their core competencies. Developer hiring and retention will inevitably suffer. Uptime and resiliency are also affected because people who aren't domain experts are put in charge of determining infrastructure. It's a self-fulfilling prophecy.

**4. Increased speed of innovation and ability to compete in the marketplace**

Two identical companies: One implements CI/CD technology and the other doesn't. Who do you think deploys applications faster? While this seems like a silly comparison, because of course the company with more automation deploys faster, there are organizations out there still convinced they don't need CI/CD because they're not looking at their competition. Organizations that understand the importance of CI/CD are setting the pace of innovation for everyone else.

## V. Conclusion

CICD is not something that can be simply described in a quick presentation, it is much more deep and wide, however, I tried to highlight some of many other benefits and tempting features that is making the software development take a historical drift towards a process that will appeal all sides participating in the process, Company Owner, Management Team, IT Teams and Developers and of course the customer. Needless to mention how these parties satisfaction will be reflected on revenue increase and cost reduction.