ci/cd
No more Tears

Difference between ci and cd

What is CI?

Continuous Integration is the practice of automating the integration of code changes from multiple contributors into a single software project .

Continuous integration is all about the source code.

New changes to the code need to be validated, verified, exercised, worked over, massaged and squeezed to see if there are leaks. We do this by compiling, transpiling, linting, running unit tests, performing static analysis, checking dependencies for security vulnerabilities and other things.

Continuous deployment is all about built code and deployment.

Once the source code has been built in CI, we're ready to ship it to servers and devices either in the same network or elsewhere. Depending on your team's delivery process and deployment strategy, you might deploy to a staging or pre-production server for final testing or you might deploy to production right away. Before doing so, CD can run scripts to prepare the infrastructure, run smoke tests, and handle rollbacks and reverts if something doesn't go as planned.

What are the signals that you need CI/CD?

- **Investing more time in a release cycle than delivering value**
- **Going through integration hell every time we finish a feature**
- **Code gets lost because of botched merges**
- **Unit test suite hasn't been green in ages**
- **Deployments contribute to schedule slip**
- **Friction between ops and development departments**
- **Only one engineer can deploy a system**
- **Deployments are not cause for celebration**

And that what we’re rally suffer from.

Continuous delivery

What you need (cost)

You need a strong foundation in continuous integration and your test suite needs to cover enough of your codebase.

Deployments need to be automated. The trigger is still manual but once a deployment is started there shouldn't be a need for human intervention.

Your team will most likely need to embrace feature flags so that incomplete features do not affect customers in production.

What you gain

The complexity of deploying software has been taken away. Your team doesn't have to spend days preparing for a release anymore.

You can release more often, thus accelerating the feedback loop with your customers.

There is much less pressure on decisions for small changes, hence encouraging iterating faster.

Continuous deployment

What you need (cost)

Your testing culture needs to be at its best. The quality of your test suite will determine the quality of your releases.

Your documentation process will need to keep up with the pace of deployments.

Feature flags become an inherent part of the process of releasing significant changes to make sure you can coordinate with other departments (support, marketing, PR...).

What you gain

You can develop faster as there's no need to pause development for releases. Deployments pipelines are triggered automatically for every change.

Releases are less risky and easier to fix in case of problem as you deploy small batches of changes.

Customers see a continuous stream of improvements, and quality increases every day, instead of every month, quarter or year.

Continuous integration

What you need (cost)

Your team will need to write automated tests for each new feature, improvement or bug fix.

You need a continuous integration server that can monitor the main repository and run the tests automatically for every new commits pushed.

Developers need to merge their changes as often as possible, at least once a day.

What you gain

Less bugs get shipped to production as regressions are captured early by the automated tests.

Building the release is easy as all integration issues have been solved early.

Less context switching as developers are alerted as soon as they break the build and can work on fixing it before they move to another task.

Testing costs are reduced drastically – your CI server can run hundreds of tests in the matter of seconds.

Your QA team spends less time testing and can focus on significant improvements to the quality culture.

The following table showing up
The translation from technical language
To cost and revenue.

Technical Language	Value	Translation
Catch Compile Errors After Merge	Reduce Cost	Less developer time on issues from new developer code
Catch Unit Test Failures	Avoid Cost	Less bugs in production and less time in testing
Detect Security Vulnerabilities	Avoid Cost	Prevent embarrassing or costly security holes
Automate Infrastructure Creation	Avoid Cost	Less human error, Faster deployments
Automate Infrastructure Cleanup	Reduce Cost	Less infrastructure costs from unused resources
Faster and More Frequent Production Deployments	Increase Revenue	New value-generating features released more quickly
Deploy to Production Without Manual Checks	Increase Revenue	Less time to market
Automated Smoke Tests	Protect Revenue	Reduced downtime from a deploy-related crash or major bug
Automated Rollback Triggered by Job Failure	Protect Revenue	Quick undo to return production to working state