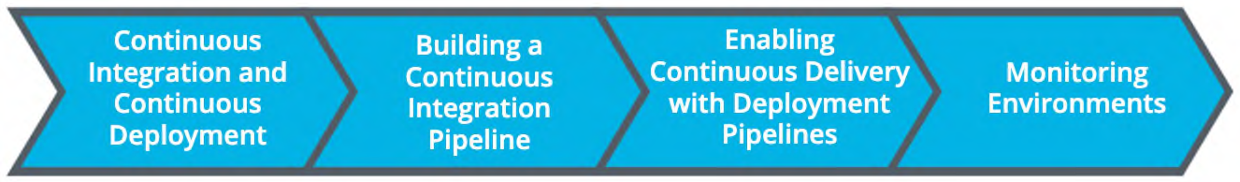
CI CD Pipelines

Summary: the software lifecycle process has daily tasks to maintain the project, but sometime it affect the project schedule, like the code merge, fixing bugs, blaming developers for corrupted commit, the deployment to different environments process consume a lot of time from the team, and some configuration need to be documented every time inn order to deploy smoothly to production environment, which actually fail sometimes for misconfiguration or release files not copied correctly.

CICD pipeline : this phrase is short cut for Continues Integration and Continues deployment, that lead to continues Delivery and less cost in money and time, its like workflow to automate the release lifecycle.


Continuous Integration and Continuous Deployment
Building a Continuous Integration Pipeline
Enabling Continuous Delivery with Deployment Pipelines
Monitoring Environments

What is CI?

Continuous Integration is the practice of automating the integration of code changes from multiple contributors into a single software project *(definition from Atlassian)*.

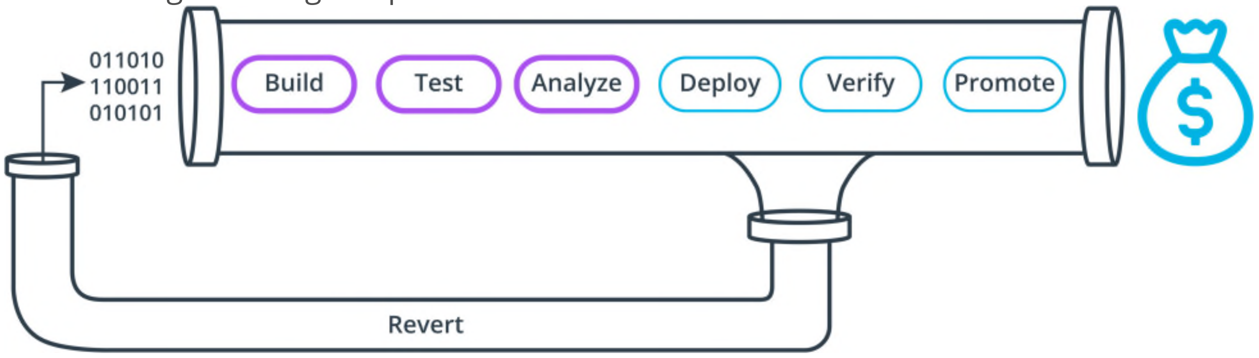
CI Opens Doors

- Singular, Unified, Consistent Build Process
- Automatically Packaged and Downloadable "Binaries"
- Faster Recovery from botched Integration (If You Break It, You Fix It!)
- Unit Test Suite Gets Some Respect
- Greater Transparency and Communication
- More Time Adding Value (aka Developing Software)

Differentiating between CI and CD

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.



Where Does CI/CD Fit In?

Stage	*Before* CI/CD	*After* CI/CD
Coding	*Human*	*Human*
Code Review	*Human*, Subjective, Inconsistent	*Human*/CI - Static Analysis
Compile/Lint	*Human*	CI

Stage	***Before* CI/CD**	***After* CI/CD**
Merge/Integrate	*Human*	CI
Run Unit Tests	*Human*, Hit or Miss, Easily Bought Off with Pressure	CI
Run Integration Tests	*Human*, Hit or Miss, Easily Bought Off with Pressure	CI
Verify Dependency Security	*Human*, Often Not Done	CI
Deploy to Test Env	*Human*, Problematic, Missed Steps	CD
Team Test	*Human*, Time Consuming	CD - Automated Acceptance Tests

Best Practices for CI/CD:

Fail Fast

Set up your CI/CD pipeline to find and reveal failures as fast as possible. The faster you can bring your code failures to light, the faster you can fix them.

Measure Quality

Measure your code quality so that you can see the positive effects of your improvement work (or the negative effects of technical debt).

Only Road to Production

Once CI/CD is deploying to production on your behalf, it must be the only way to deploy. Any other person or process that meddles with production after CI/CD is running will inevitably cause CI/CD to become inconsistent and fail.

Maximum Automation

If it can be automated, automate it. This will only improve your process!

Config in Code

All configuration code must be in code and versioned alongside your production code. This includes the CI/CD configuration files!

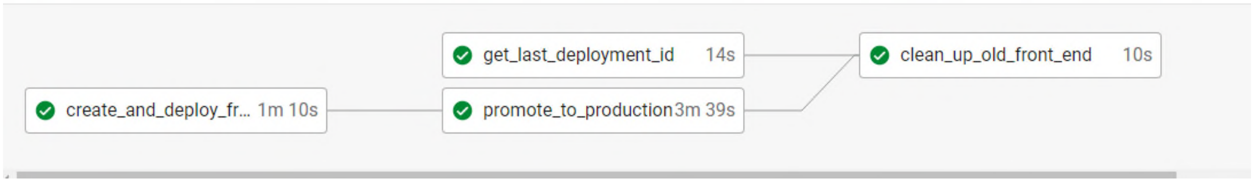
Deployment Strategy	**Description**
Big-Bang	Replace A with B all at once.
Blue Green	Two versions of production: Blue or previous version and Green or new version. Traffic can still be routed to blue while testing green. Switching to the new version is done by simply shifting traffic from blue to green.
Canary	Aka Rolling Update, After deploying the new version, start routing traffic to new version little by little until all traffic is

Deployment Strategy	Description
	hitting the new production. Both versions coexist for a period of time.
A/B Testing	Similar to Canary, but instead of routing traffic to new version to accomplish a full deployment, you are testing your new version with a subset of users for feedback. You might end up routing all traffic to the new version, but that's always the goal.

Workflow Example:

The workflow below describe the steps for deploying to production,

1- The workflow start by creating new environment.
2- Check for the current production info.
3- Deploy the solution to the new environment.
4- Change the production configuration to the new environment


create_and_deploy_fr... 1m 10s
get_last_deployment_id 14s
promote_to_production 3m 39s
clean_up_old_front_end 10s