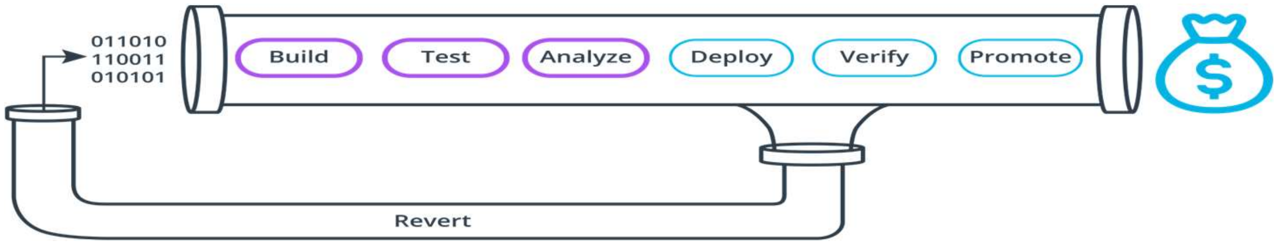
CI/CD — A better way to build and ship products to market.

Fundamentals and Benefits of CI/CD to Achieve, Build, and Deploy Automation.

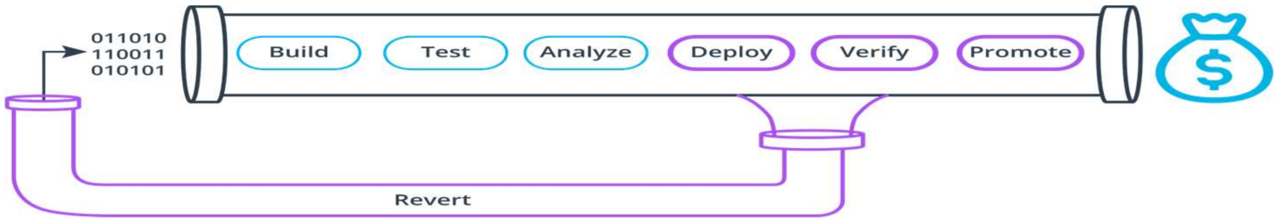
Continuous Integration

- The practice of merging all developers' working copies to a shared mainline several times a day.
- It's the process of "Making". Everything related to the code fits here, and it all culminates in the ultimate goal of CI: *a high quality, deployable artifact!*
- *Some of the steps in this stage include compiling, testing, running static analysis, checking for vulnerabilities in our dependencies and storing the code artifacts.*



Continuous Deployment

- A software engineering approach in which the value is delivered frequently through automated deployments.
- Everything related to deploying the artifact fits here.
- It's the process of "Moving" the artifact from the shelf to the spotlight
- Some steps in this stage include setting up infrastructure, provisioning servers, copying files, smoke testing, promoting to production and even rolling back a change if something did not look right.



Identifying The Need For CI/CD

How do you know 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

What CI/CD asks in return ?

- No more manual deploying to environments
- No more modifying environment settings in GUI's
- No more neglecting the unit tests
- No more leaving broken code in place
- Requires a high level of discipline
- Requires additional skills to maintain and extend automation

Benefits of CI/CD

- **Automate Infrastructure Creation and clean up:**
Eliminating human errors and *avoid unnecessary cost* of unused or invalid infrastructure
- **Faster to production:**
By automating the pipeline to production this way, we can deploy features as soon as created *Without Manual Checks* which would help increase revenue
- **Automated Rollback Triggered by Job Failure:**
Automate the process of rolling back and cleaning any infrastructure left which would help in protecting revenue and lower down time
- **Catch Compile Errors After Merge:**
Discover errors as soon as the developer make his commit which would help reduce the time of developers and *avoid unnecessary* cost
- **Catch Unit Test Failures:**
Unit tests are not neglected with CICD which will increase code quality and catch errors early before production which would reduce cost
- **Automated Smoke Tests:**
Automate smoke test after deployment and automatic rollback in case of failure which would help in protecting revenue and lower down time