Fundamentals and Benefits of CI/CD

Achieve, Build, and Deploy Automation for Cloud-Based Software Products

What is Continuous Delivery?

Continuous Delivery is an engineering practice in which teams produce and release value in short cycles.

Continuous Integration + Continuous Deployment = Continuous Delivery

CI + CD = C. Delivery

- 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 common CI-related phases might include:*
  - Compile
  - Unit Test
  - Static Analysis
  - Dependency vulnerability testing
  - Store artifact
- 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 common CD-related phases might include:*
  - Creating infrastructure
  - Provisioning servers
  - Copying files
  - Promoting to production
  - Smoke Testing (aka Verify)
  - Rollbacks

Why Continuous Delivery?

The following problems can be fixed with 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 Business value brings CI/CD?

CI/CD Benefit	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

https://blackswanfarming.com/value-a-framework-for-thinking/

Principles of Continuous Delivery.

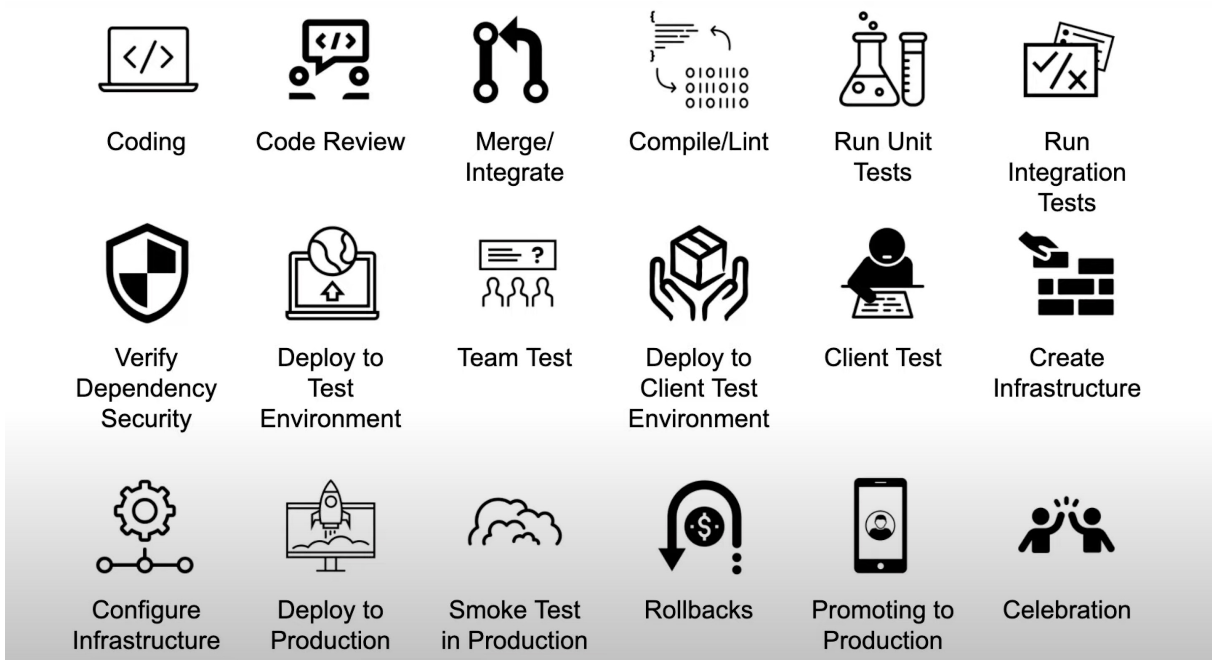
1. **Repeatable Reliable Process** of Releasing/Deploying SW.
1. **Automate Everything**: No manual process necessary.
2. **Version Control Everything**: including Infrastructure, Configuration, Test Cases, Acceptance Tests, E2E Tests.
3. **Bring the Pain Forward**: if something is difficult, do that thing more often until it is not difficult anymore.
4. **Build-in Quality**: verify quality, keep metrics to alert the team if expected quality drops (Coverage, Complexity, Code Smells Code Style). Process should motivate team to improve metrics over time.
5. **"Done" Means Released**: A feature is only done, when it is running in production.
6. **Everyone is Responsible**: No Silos, when there is a problem everyone has to help.
7. **Continuous Improvement**: True north is perfection, small daily improvements are the steps towards that goal.

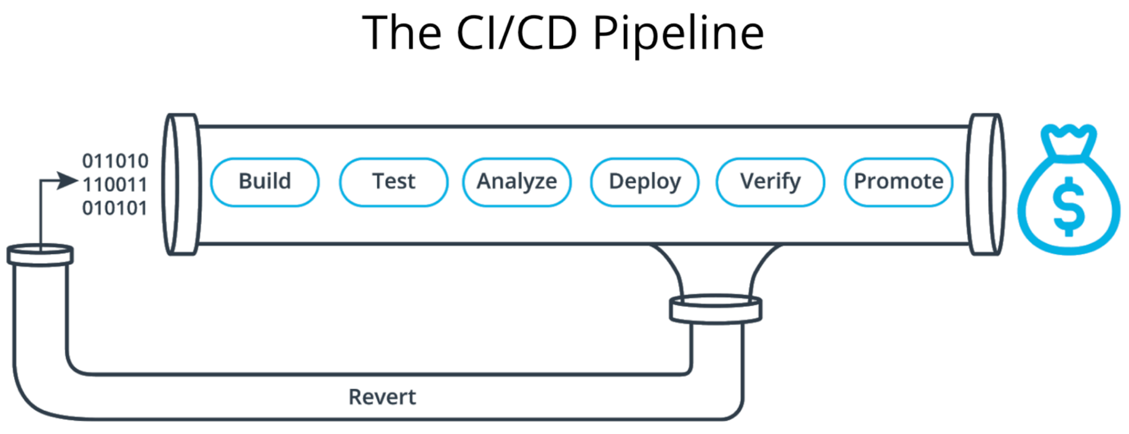
Source: https://www.amazon.com/dp/0321601912?tag=contindelive-20

Costs for adoption of CI/CD.

- 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

Process of Deploying to Production.





The Phases of CI/CD Pipeline

Steps to get closer to Continuous Delivery.

- Expect collaborative, comprehensive grooming of features that include team and stakeholders
- Ruthless slicing of features to smallest valuable increments
- Build team-wide, deep understanding of each feature's requirements and characteristics before coding starts
- Write comprehensive automated unit tests in front-end and back-end layers
- Shoot for high coverage from automated back-end integration tests
- Shoot for high feature critical-path coverage from end-to-end UI tests
- Include automated smoke tests that can be run on production-candidates
- Ensure all post-commit tasks and hand-offs must be automated in CI/CD
- Strive for quick, reliable rollback if smoke tests fail

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!

Backup

Deployment Strategies.

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 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.

Blue-Green Deployment.

Router Option	Description
Load Balancer	Instant switch for FE or BE, ideal router in most cases
CDN	Instant switch for front-end web apps.
DNS	A bit slow because of DNS propagation.

Step	Description
Integrate Code in a Build	Compile and create artifact
Run Tests	Run unit and/or integration tests
Ensure Infrastructure is Present	Create green infrastructure
Provision the Environment	Configure green instance, migrate DB, etc
Deploy Artifact	Copy artifact files to instance
Run Smoke Tests	Run a few tests that don't impact the prod server
Perform Rollback if Failure	Rollback here is more of a cleanup of green
Switch Router	Redirect traffic to new version
Run Sanity Test	Run a few tests that don't impact the prod server
Perform Rollback If Failure	Rollback here is switching the router back to blue and cleaning up green

Build Stages of CI/CD

Stage	**Description**
Build	Everything that has to do with making code executable in production (e.g. Compile). The goal is to produce an artifact.
Test	All automated tests that verify at the code level.
Analyze	Any static analysis on the code or checking of dependencies.
Deploy	Anything to do with creating server instances or copying pre-built application files to an instance.
Verify	Any tests that can be run against a running instance of the application, often against a pre-production instance.
Promote	Replacing the current production environment with the new version which was just built and deployed.
Revert	Rolling back or undoing changes in case any verification fails after deployment.

Key Terms

- **Pipeline**: A set of data processing elements connected in series, where the output of one element is the input of the next one.
- **Continuous Integration**: The practice of merging all developers' working copies to a shared mainline several times a day.
- **Continuous Delivery**: An engineering practice in which teams produce and release value in short cycles.
- **Continuous Deployment**: A software engineering approach in which the value is delivered frequently through automated deployments.
- **Infrastructure as Code**: The management of infrastructure using code.
- **Provisioning**: The process of setting up IT infrastructure.
- **Artifact**: A product of some process applied to the code repository.
- **DevOps**: A set of practices that works to automate and integrate the processes between software development and IT teams.
- **Testing**: A practice that seeks to ensure the quality of the software.