2022

CI/CD proposal

UDAPEOPLE
AHMED KHALLAF

Contents

what does ci/cd stand for?

CI stands for 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 goal of CI: a high quality, deployable artifact! Some common CI-related phases might include:

1. Compile
2. Unit Test
3. Static Analysis
4. Dependency vulnerability testing
5. Store artifact

CD stands for Continuous Delivery or 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:

1. Creating infrastructure
2. Provisioning servers
3. Copying files
4. Promoting to production
5. Smoke Testing (aka Verify)
6. Rollbacks

Why is CI/CD important?

CI/CD allows organizations to ship software quickly and efficiently. CI/CD facilitates an effective process for getting products to market faster than ever before, continuously delivering code into production, and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method.

Did you think CI/CD was going to solve all your woes and ask nothing in return?

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

Why Is Translation Important?

There are several "warning signs" that teams exhibit that suggest they would be good candidates for CI/CD or Continuous Delivery. If you identify with any of these items, you should consider CI/CD an essential piece of your development workflow.

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

Best Practices for CI/CD

1. Fail Fast
2. 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.
3. Measure Quality
4. 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
5. 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.
6. Maximum Automation
7. If it can be automated, automate it. This will only improve your process!
8. Config in Code
9. All configuration code must be in code and versioned alongside your production code. This includes the CI/CD configuration files!

Blue/Green Deployments

Blue/green deployments provide releases with near zero-downtime and rollback capabilities. The fundamental idea behind blue/green deployment is to shift traffic between two identical environments that are running different versions of your application. The blue environment represents the current application version serving production traffic. In parallel, the green environment is staged running a different version of your application. After the green environment is ready and tested, production traffic is redirected from blue to green. If any problems are identified, you can roll back by reverting traffic back to the blue environment.