CI/CD

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

Business Benefits of CI/CD

- Faster product delivery = increasing productivity
- Automated Rollback = Quick rollback
- Automated Infrastructure Creation and Cleanup

- Catch compile Errors After Merge = Less developer time on issues from new developer code
- Catch unit test failures = Less bugs in production
- Detect Security Vulnerabilities = Prevent embarrassing security holes
- Efficient testing & monitoring = Predict application performance

CI/CD fundamentals

There are eight fundamental elements of CI/CD that help ensure maximum efficiency for your development lifecycle. They span development and deployment. Include these fundamentals in your pipeline to improve your DevOps workflow and software delivery

https://about.gitlab.com/topics/ci-cd/

A single source repository

Source code management (SCM) that houses all necessary files and scripts to create builds is critical. The repository should contain everything needed for the build. This includes source code, database structure, libraries, properties files, and version control. It should also contain test scripts and scripts to build applications.

Frequent check-ins to main branch

Integrate code in your trunk, mainline or master branch — i.e., trunk-based development — early and often. Avoid sub-branches and work with the main branch only. Use small segments of code and merge them into the branch as frequently as possible. Don't merge more than one change at a time.

Automated builds

Scripts should include everything you need to build from a single command. This includes web server files, database scripts, and application software. The CI processes should automatically package and compile the code into a usable application.

Self-testing builds
CI/CD requires continuous testing. Testing scripts should ensure that the failure of a test results in a failed build. Use static pre-build testing scripts to check code for integrity, quality, and security compliance. Only allow code that passes static tests into the build.

Frequent iterations
Multiple commits to the repository results in fewer places for conflicts to hide. Make small, frequent iterations rather than major changes. By doing this, it's possible to roll changes back easily if there's a problem or conflict.

Stable testing environments
Code should be tested in a cloned version of the production environment. You can't test new code in the live production version. Create a cloned environment that's as close as possible to the real environment. Use rigorous testing scripts to detect and identify bugs that slipped through the initial pre-build testing process.

Maximum visibility
Every developer should be able to access the latest executables and see any changes made to the repository. Information in the repository should be visible to all. Use version control to manage handoffs so developers know which is the latest version. Maximum visibility means everyone can monitor progress and identify potential concerns.

Predictable deployments anytime
Deployments should be so routine and low-risk that the team is comfortable doing them anytime. CI/CD testing and verification processes should be rigorous and reliable, giving the team confidence to deploy updates at any time. Frequent deployments incorporating limited changes also pose lower risks and can be easily rolled back.