klocwork

CI/CD Best Practices

Learn How to Optimize CI/CD Pipelines

Introduction

Continuous Integration (CI) and Continuous Deployment (CD) are popular software development practices for automation and shortening feedback times. However, setup improperly, your CI/CD pipelines could instead cause delays in development.

For that reason, review these CI/CD best practices to ensure that your pipelines are effective and efficient.

What Is Continuous Integration?

The processes used by your team have a direct impact on the efficiency of your software development workflow. For that reason, it is important for your team to adopt processes that streamline your software development — like Continuous Integration.

Continuous Integration (CI) is the practice of automating the build and testing of code every time a change is made — and committing that code back to a central repository.

One of the fundamental cornerstones of Continuous Integration is that it encourages breaking up development tasks into small bite-sized pieces that can be performed frequently by every developer on the team.

Each new code commit triggers a consistent, automated build and test process — often called a "pipeline" — to report any defects found during compilation or testing as quickly as possible.

Continuous Integration is one of the key components of DevOps Automation.

What are the Benefits of Continuous Integration?

By implementing Continuous Integration, software development teams benefit from:

EASIER BUG FIXES

Identifying issues sooner makes it easier for developers to fix errors, vulnerabilities, and defects in the code. What's more, this helps to ensure that an issue will be fixed correctly, resulting in a build that's issue free and working as quickly as possible.

REDUCED PROJECT RISK

Encouraging small, modular changes to the code enables new functionality to be backed out of a release more quickly, or even prevented from entering the main code stream altogether. This minimizes the impact on other developers.

IMPROVED SOFTWARE QUALITY

Maximizing the value of CI means detecting as many issues as possible in each integration build, through automation. This increases the breadth, depth, and repeatability of the tests while avoiding manual testing.

HIGHER PRODUCTIVITY

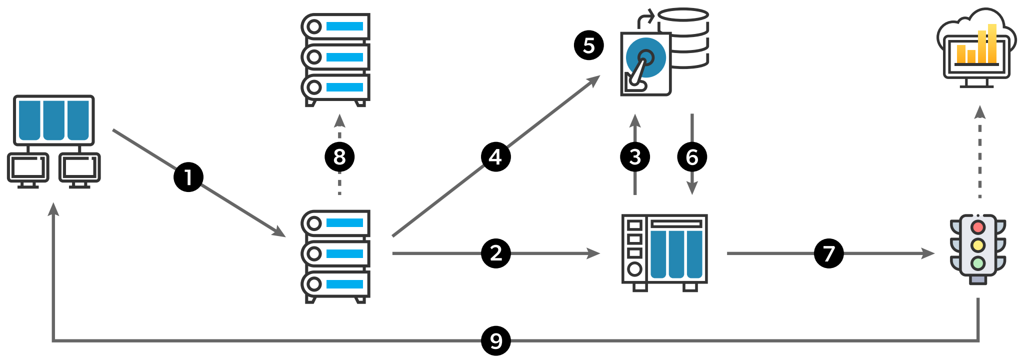
Automating these tasks frees up developers to focus on higher-value feature development.

WHAT IS THE DIFFERENCE BETWEEN CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY?

Continuous Integration (CI) and Continuous Delivery (CD) are both software development practices.

CI is used during the build and test phase. CD is used once changes are committed. The ultimate aim of CD is to always have validated and verified code in the code repository — or version control system — ready for release.

9 STEPS OF CONTINUOUS INTEGRATION



1. Developers check code into the version control system's staging repository.
2. The Version Control System (VCS) or code repository notifies the CI server that a commit has occurred. Or, the CI server polls the repository periodically looking for commits.
3. The CI server starts the build process on a build server.
4. The code containing the latest commit — ideally just the minimum file set — is checked out of the repository into a local workspace on the build server.
5. The changed code is built, analyzed, and tested.
6. Important results are reported back to the CI server, along with any important details and files that need to be retained.
7. The CI server sets the final — Pass / Fail — result of the build.
8. If the build met the success criteria, then the committed change may proceed through the development cycle - transferred to the real repository or merged to the main development stream. If the build failed, the committed changes are blocked from proceeding until those issues are resolved.
9. The CI server notifies any parties who have registered interest in the build. They can then log into the CI server to view the status plus any additional information.

WHY CONTINUOUS INTEGRATION IS IMPORTANT IN SOFTWARE DEVELOPMENT?

Continuous Integration accelerates software development to avoid these common pitfalls of development:

- Frequent code integration helps to eliminate code conflicts and code incompatibility.
- Developers are encouraged to have the most up-to-date repository code when working.
- Reduces the refactoring complexity.
- Quality gates ensure only clean, working and tested code makes it to the repository.
- Reduces repository commit bottlenecks.

With a CI pipeline, every change is integrated, tested and verified which brings the commit closer to being a viable release candidate.

HOW STATIC ANALYSIS EXTENDS CONTINUOUS INTEGRATION

Static analysis is a natural addition to any CI development process and done correctly, adds the possibility for almost immediate feedback of new coding issues, specific to the branch or commit containing them. This provides the opportunity for quality gates to prevent those issues from ever entering the main codestream and needing to be resolved later — improving development efficiency.

Static analysis complements other verification and validation techniques, such as dynamic testing, because:

- Static analysis provides coverage of all possible execution paths, whereas for dynamic analysis error or fault conditions of the code that are typically very difficult, or even impossible, to induce at runtime.
- Static analysis is very cost-efficient in terms of detecting bugs earlier in the lifecycle — and it requires much less time to run.
- Issues detected with static analysis, prior to dynamic tests being written also saves downstream costs of reworking those dynamic tests, once the issues have been resolved — most dynamic tests are quite dependent on the code itself and so a change has an impact also on the tests.

The ideal static analysis engine for CI processes only modified code and affected execution paths, rather than processing the entire codebase all the time, and reports the impact of those changes. Since static code analysis operates on source code and doesn't have to actually execute the code, it can perform a complete analysis of the submitted code changes in a specific time frame. In addition, static code analysis doesn't require specific test cases to be written.

To be effective in complementing CI, a static code analysis tool must be fast, scalable, and automated.

Learn More About: What Is Static Code Analysis?

EXTEND CI WITH KLOCWORK

Learn more about how to apply static code analysis to Continuous Integration

EXPLORE KLOCWORK

perforce.com/resources/kw/bringing-static-code-analysis-continuous-integration

What You Need to Know About Continuous Integration

Continuous Integration (CI) helps to automate the integration of code changes from multiple contributors into a single software project. However, a CI process is only effective if it has been properly implemented. For that reason, we provide guidance on how to properly implement your own CI process.

HOW DOES CONTINUOUS INTEGRATION WORK?

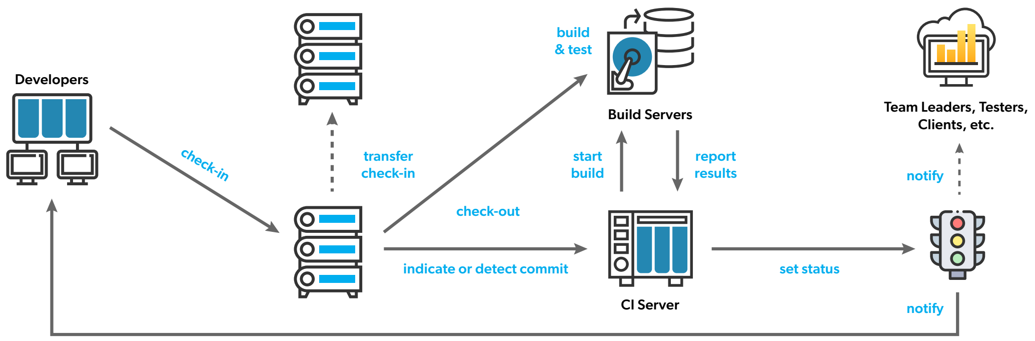
Continuous Integration streamlines the software development process, ensuring that there are no disconnected segments. It accomplishes this by including all the stages — integration, testing, verification, and deployment — into each segment of development.

IMPLEMENT QUALITY GATES

Quality gates ensure that only clean, working and tested code makes it into the repository. So, when a developer checks some code into the repository, it goes into a staging repository or some form of locked state, until the verification and validation requirements are met.

If the code passes all the tests, then the commit is transferred to the real repository, unlocked for pull requests, or merged to the main branch as appropriate for the system used.

If the code doesn't pass all the tests, then the commit remains locked, the developer is notified, and they can take the necessary corrective action before resubmitting the changes for a further check.



USING NIGHTLY BUILDS

Many organizations use nightly builds for CI. Nightly builds are when the codebase is checked out of the repository at the end of the working day, built, and tested. And the results from this build are examined the next morning and the full set of issues are then shared with the development team.

However, there are some problems with using nightly builds. The main problem is that there is a time lag between checking code in and seeing the results.

For example: A developer may check in some code in the morning that will break the build, but is unaware of that until the next day. By that time, the developer may have moved on to a different task and no longer remembers what the original problem could be.

This leads to more time spent trying to understand the problem, which increases the risk of fixing the problem incorrectly or breaking something else.

WHAT ARE CONTINUOUS INTEGRATION SYSTEMS?

Automated testing is key to Continuous Integration. Once the test phases have been automated, the CI system can run the appropriate tests and take action on the results. That is why a CI system can be thought of as a sophisticated program that schedules, launches compilations, packages, and tests processes.

The five core requirements of a CI system are the ability to automatically:

1. Detect code that has been committed either by monitoring the repository for commits or accepting some external stimulus — usually from the source code repository.
2. Checkout code from the repository onto a machine equipped with all the necessary build and testing tools available.
3. Build the code.
4. Run all the tests on the resulting executable(s).
5. Report the results of the build and test to the appropriate team member(s).

HOME-GROWN CI SYSTEMS

Continuous Integration automation can be accomplished with some scripting and command line tools. And, many companies have traditionally started off with this type of setup.

However, administering these scripts can often become a full-time responsibility that draws developers away from their core task of writing code. What's more, this can also complicate the readability and understanding of the development processes.

CI platforms, such as Jenkins, provide a centralized and standardized base from which to work. And the output of these projects and builds are then available to the whole team, and not just the developers.

Even higher-level management would probably like to know if the number of tests passed today was greater than the number of tests passed last week.

CI USING JENKINS

Jenkins is a popular CI system. It is customizable (with countless plugins) and scalable. That's why Perforce tools come with Jenkins plug-ins for version control and static code analysis.

HOW STATIC ANALYSIS HELPS WITH CONTINUOUS INTEGRATION

Klocwork is the ideal static analyzer to support Continuous Integration, and its unique Differential Analysis technology provides the fastest analysis results for CI pipelines. What's more, by using Klocwork, you are able to:

- Ensure complex software is safe, secure, and reliable.
- Reduce the cost of finding and fixing defects earlier in development.
- Prove compliance by enforcing software coding standards.
- Improve developer productivity, testing efforts, and velocity of software delivery.
- Report on quality over time and across product versions.

Klocwork integrates with build systems and CI environments and its unique Differential Analysis technology provides the fastest analysis results for CI pipelines. Learn more about how Klocwork can help. Extend CI with Klocwork >>

How to Pair Static Analysis With CI/CD Pipelines

As more software is installed into devices across all industries, it has become essential that the embedded code is safe and secure, reliable, and high quality. At the same time, competitive pressures often mean tighter project schedules.

Ensuring that the embedded code meets these standards and is delivered in a timely manner can be a daunting and time-consuming challenge. For that reason, many teams have adopted CI/CD pipelines as a component of a more efficient software development process.

WHAT ARE CI/CD PIPELINES?

Continuous integration and continuous delivery (CI/CD) pipelines are software engineering approaches that are a part of the larger software delivery pipeline. Put simply, continuous integration is the practice of merging each developer's working copies of code together in a shared mainline several times throughout the day. While continuous delivery refers to the regular, frequent delivery of software functionalities.

CI/CD pipelines form the backbone of DevOps automation. Additionally, quality assurance and security checking can easily be integrated into the CI/CD process. The goal is for developers to receive immediate feedback on any issues found within their most recent code revisions. Fixes can be made at the earliest opportunity (and lowest cost). This all helps ensure delivery of a high quality, reliable, and competitive software product on time.

WHY STATIC ANALYSIS IS NECESSARY FOR CI/CD PIPELINES

Static analysis inspects your source code to identify defects, vulnerabilities, and compliance issues as you code — without having to run the program. This makes static analysis an essential component of a CI/CD pipeline, as it helps with:

- Detection of common security vulnerabilities, including those highlighted by security coding standards such as CERT and CWE, DISA STIG, and OWASP.
- Early detection of potential runtime errors. These include memory leaks, concurrency violations, or uninitialized data — all of which can cause system failures.
- Compliance with safety-related coding standards, such as MISRA C/C++ and AUTOSAR.
- Enforcement of company or project-wide coding guidelines or naming conventions, and maintainability requirements.

WHAT IS DIFFERENTIAL STATIC ANALYSIS?

Most code edits only change a tiny fraction of the total amount of code in a project, but minor changes can still have a large impact on the overall system.

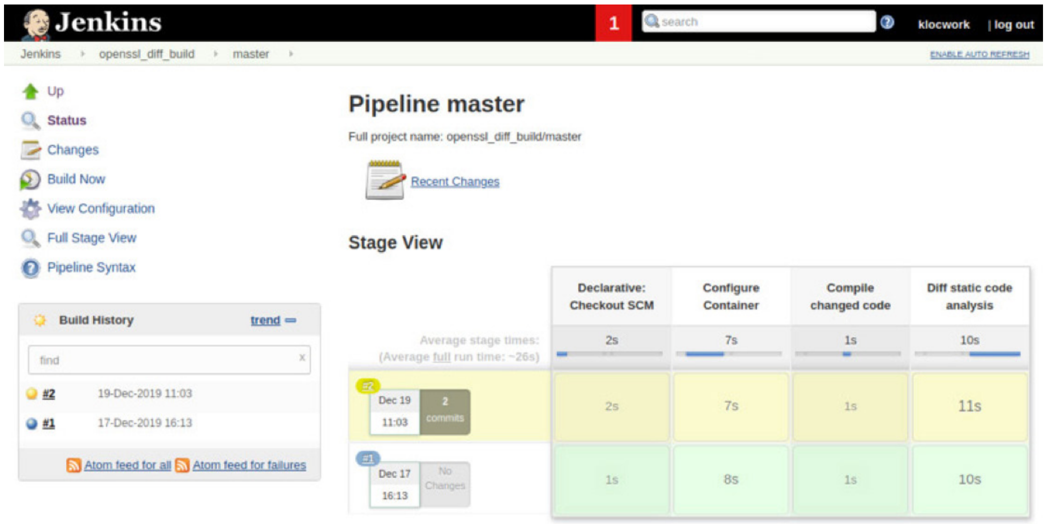
A single developer's local analysis of a changed source file may not flag any issues; however, the changes may still lead to issues that can only be detected through complete, system-wide analysis. With a traditional static analyzer, the only way to find these issues is to perform an analysis of the entire, merged codebase. The time to complete this analysis will grow in proportion to the size and complexity of the project. This means that as the project grows, the time taken to feed issues back to developers will increase — making it harder to achieve the CI/CD pipeline goal.

KLOCWORK IS THE ONLY STATIC ANALYZER THAT SOLVES THIS PROBLEM

Klocwork maintains system-wide knowledge of the code in a centralized server. This means it only needs to analyze the small part of the code that has changed in order to work out if there are any resulting system-wide issues.

This means that Klocwork can analyze thousands of source files and tens of millions of lines of code in a matter of seconds — not hours. What's more, differential static analysis provides developers with the shortest possible analysis time and provides an impact analysis of the changes — no matter how large the codebase.

For that reason, adding static analysis to every CI/CD pipeline is practical, efficient, and helps to ensure that there is no need to trade feedback times for quality and security.



Pipeline master

Full project name: openssl_diff_build/master

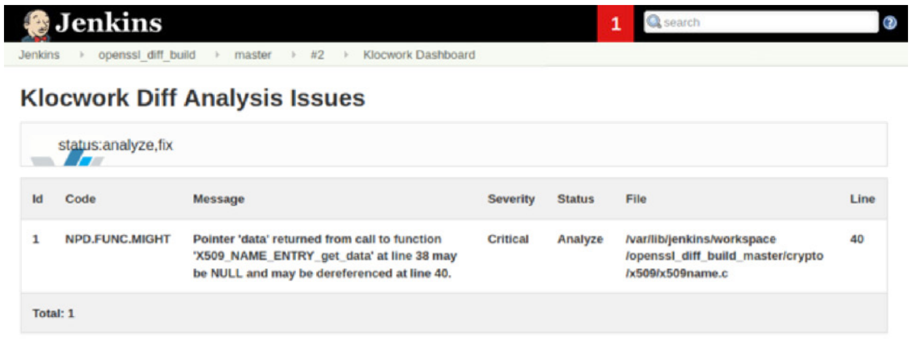
Stage View

	Declarative: Checkout SCM	Configure Container	Compile changed code	Diff static code analysis
Average stage times: (Average full run time: ~26s)	2s	7s	1s	10s
#2 Dec 19 11:03 2 commits	2s	7s	1s	11s
#1 Dec 17 16:13 No Changes	1s	8s	1s	10s

#2	19-Dec-2019 11:03
#1	17-Dec-2019 16:13

Klocwork diff analysis included in the CI-commit pipeline.

HOW CI/CD PIPELINES SAVE CLOUD COMPUTING COSTS


Jenkins
1
search
Jenkins › openssl_diff_build › master › #2 › Klocwork Dashboard

Klocwork Diff Analysis Issues

status:analyze,fix

Id	Code	Message	Severity	Status	File	Line
1	NPD.FUNC.MIGHT	Pointer 'data' returned from call to function 'X509_NAME_ENTRY_get_data' at line 38 may be NULL and may be dereferenced at line 40.	Critical	Analyze	/var/lib/jenkins/workspace /openssl_diff_build_master/crypto /x509/x509name.c	40

Total: 1

Klocwork diff analysis results for new defects.

In general, the cost of a DevOps pipeline grows in proportion to its execution times. Klocwork's differential analysis dramatically reduces execution times and therefore also reduces cloud computing costs.

Klocwork's differential analysis applies even if you are using an internal cloud computing resource, such as OpenStack. When deploying static code analysis in your CI/CD pipeline, Klocwork's Differential Analysis provides results fast.

CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY WITH STATIC ANALYSIS

Klocwork is the ideal static analyzer for CI/CD pipelines, and its unique Differential Analysis technology provides the fastest analysis results for DevOps pipelines. What's more, by using Klocwork, you are able to:

- Ensure complex software is safe, secure, and reliable.
- Reduce the cost of finding and fixing defects earlier in development.
- Prove compliance by enforcing software coding standards.
- Improve developer productivity, testing efforts, and velocity of software delivery.
- Report on quality over time and across product versions.

Learn more about how to add static code analysis to CI/CD Pipelines

OPTIMIZE YOUR CI/CD PIPELINE

perforce.com/webinars/kw/add-static-code-analysis-to-ci-cd-pipelines