Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2021, 8(10):162-167

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Continuous Integration and Deployment (CI/CD) Common Issues: Focusing on Testing for Streamlined Development

Chakradhar Avinash Devarapalli

Software Developer
Email: avinashd7[at]gmail.com

Abstract Continuous integration and deployment (CI/CD) is a major part of today's development procedures. The idea is to keep the process as smooth and simple as possible, both during the development and with the testing and launch of the backend applications. As the world focuses on better designs and improved experiences for users, one thing that stays constant is the need to get software out there quickly and reliably. That's where CI/CD for frontend development steps in. This paper looks closely at the concept of CI/CD and aims to understand why CI/CD matters for frontend development. It also considers the crucial problem of dependency management with CI/CD and proposes a solution on how to make it work where the backend does not depend entirely on the front end.

Keywords *Continuous Integration, Continuous Deployment, Frontend Development, Automation, Testing, Deployment.*

1. Introduction

Continuous integration and continuous deployment (CI/CD) have gained popularity lately, thanks to their many benefits. They help reduce time to market and boost software quality. Instead of waiting until the end of the development cycle, teams integrate small changes into the code more frequently with continuous integration. This early integration catches bugs sooner, making development faster and smoother. Continuous delivery and deployment allow teams to release software whenever they want, leading to rapid iterations and better responsiveness to market needs.

However, implementing CI/CD for frontend development isn't a walk in the park for everyone. Frontend development poses unique challenges in automating testing and deployment [3]. Since frontend applications are highly user-facing, testing needs to be meticulous to ensure a seamless user experience across various devices and browsers.

To tackle these challenges, a comprehensive CI/CD pipeline tailored for frontend development is crucial. This pipeline should include automated testing frameworks covering unit testing, integration testing, and end-to-end testing. In this paper, we'll propose an automated CI/CD pipeline for frontend development that integrates these testing frameworks and popular tools like Jenkins, GitHub Actions, and Docker [4].

2. Literature Review

Continuous Integration and Continuous Deployment (CI/CD) in the area of frontend development has been a topic of literary discussion for years. It outlines the problem and benefits in this field. Multiple materials point out that CI/CD became the driver of software development in the modern era many years ago. For frontend testing, a major element of CI/CD, is discussed in literature, giving rationale in providing the utmost convenience to the customers as they use different platforms.

A number of perspectives discuss the complexities of frontend development within the CI/CD framework. Authors underscore the need for testing procedures to address the unique challenges posed by frontend applications. The intricate packaging and bundling processes inherent in frontend development are also examined, emphasizing the need for streamlined workflows to optimize application performance.

A common theme across the literature is the identification of common issues encountered during CI/CD implementation. From performance concerns to interdependency issues between frontend and backend systems, authors delve into the intricacies of maintaining stability and scalability in CI/CD pipelines. Memory optimization emerges as a recurring challenge, with implications for both performance and energy efficiency.

3. Current Landscape

The current CI/CD methodologies mainly revolve around integrating changes into a shared repository frequently. While this strategy helps catch bugs early, it tends to overlook the specific hurdles faced in frontend development.

3.1 Testing & Deployment

In traditional CI/CD pipelines, testing and deployment are automated. However, frontend applications demand thorough testing across different devices and browsers for a smooth user experience. Automating this level of testing is tricky, leading to potential bugs slipping through.

These glitches may not be readily diagnosable either, and due to the interdependent nature of the backend on the front end, leading to the issue exacerbating over the course of a few days or months [5].

In larger organizations, where there are multiple development teams working on changes to the respective code repositories, even the smallest bugs can lead to widespread issues once the code is compiled, built, tested, delivered, or deployed.

By that time, the alpha and beta builds may have been finalized, and the front end may already be under heavy use. This means that scrutinizing the code will be done in a hurry to fix the bug, all while customer experience suffers. Over time, the effects may be visible as poor software speed, poor responsiveness, instability, and issues with the scalability of the site. The project then needs to be redistributed again, breaking down different elements to find the underlying cause.

3.2 Performance Issues

Performance glitches are a common hiccup in CI/CD setups. They manifest as longer page loading times or server response failures. Diagnosing such issues isn't always straightforward and can snowball into software speed, responsiveness, stability, and scalability woes.

3.3 Memory Optimization

Memory optimization poses another puzzle in CI/CD. When resources fail to efficiently ferry necessary data to their respective nodes, the application's operational costs may skyrocket. This problem is especially evident in applications dealing with large data volumes.

A study conducted by 451 Research showed that more than 60% of developers have reported that they don’t have an automated integrated security tool safeguarding their data, hence posing a major issue for CI/CD tool integration [6].

3.4 Security Testing

Security testing in CI/CD pipelines often impacts performance, creating a tug-of-war where security compromises performance, leading to more compromises in security, and so on. This cycle can flood developers with false positives, making it challenging to pinpoint real threats.

3.5 Developer Resistance

Developer resistance can impede effective CI/CD implementation. Developers might hesitate to dig into the code unless prodded, seeing it as a time-consuming task. This reluctance may result in non-compliance with security standards, raising legal stakes in case of breaches.

3.6 The Constant Performance-Security Issue Loop

Performance issues stemming from a not-so-perfect CI/CD implementation can cast a shadow over the development process's efficiency and the end product's quality. Consider a slow page loading and server responses moving at a pace that makes customers wait. Not only will this be a means for poor customer experience, but it also gives competitors a chance to introduce themselves as a solution for this issue.

The average attention span of humans is slowly decreasing in today's fast-paced world. The human brain loses focus every 8 seconds, until the 8-minute mark when customers become outright disinterested [7]. The slower the response of a website, the higher the chances of them turning away.

4. Proposed Solution

The solution proposed by this paper involves integrating several testing mechanisms across different steps across the development phase. It focuses on:

1. Realtime testing in and tool configuration in the pre-commitment phase
2. Onboarding procedures and incremental tests on the commitment phase
3. Implementing SCA and SAST tests on a deeper level on the build phase
4. Dedicated IAST, DAST, fuzz testing, and hardening checks during the testing phase
5. False positive mitigation strategies with the testing across all phases
6. Developer support initiatives during and after deployment, and
7. Continuous monitoring as part of the CD pipeline.

In addressing the manifold challenges posed by CI/CD implementation, a constant-testing approach is paramount. While many application security tools offer a command line interface (CLI) for integration into the CI/CD pipeline, a mere CLI presence falls short of ensuring pipeline compatibility. Figure 1 shows the different phases in development along with the tests developers should implement at each stage.

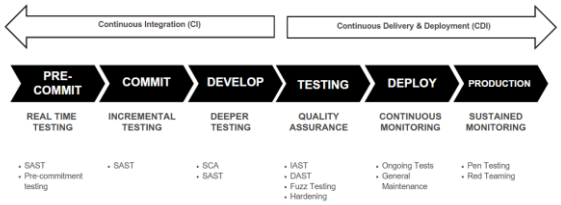

Figure 1: Different Tests at Different Phases of CI/CD Front End Development

First, it is important to ascertain that all Application Security Testing (AST) tools can communicate effectively with a centralized metrics dashboard. This must be done at the pre-commitment phase [5]. Failure to establish this communication may necessitate the development of numerous plugins or bespoke code to synchronize updates across a common defect tracking system, potentially disrupting the pipeline's integrity [3] [6] [7].

When resorting to custom code, abstraction is critical. Abstracting functionalities allows subsequent tool replacements, integrations, and tests to incur minimal disruptions to the existing pipeline structure. It also reduces the effort(s) involved with testing and making amendments once the final product is launched.

The resilience of pipeline scripts to external tool alterations is another critical aspect to consider here [8]. Ensuring that modifications to SAST tests at the pre-commitment and commitment stages is necessary to ensure that they do not disrupt the pipeline scripts later on, while maintaining operational stability. It also effectively minimizes the need for extensive script overhauls later on as well as during development. Furthermore, it eliminates the risks of clashes when different development teams are working on the same project.

Reflecting on the evolution of service provision, the transition from virtual machines (VMs) to containers and cloud infrastructure has reshaped deployment strategies [9] [10].

5. Academic Review of Perceived Challenges

Table 1: Table of Studied Literature Regarding Challenges

Name	Title	Challenge Discussed
N. Choudhary	Top 10 CI/CD Pipeline Implementation Challenges and Solutions	Various challenges in implementing CI/CD pipelines
M. Rao	Common Security Challenges in CI/CD Workflows	Security challenges in CI/CD workflows
R. Warren	How to Add Application Security Tests to Your CI/CD Pipeline	Integration of application security tests in CI/CD pipelines
O. Cobles	How to Solve For: Migrating Virtual Machines to Containers	Strategies for migrating from VMs to containers

6. Potential Use Cases

In the early 2000s, AST tools predominantly operated within VMs, a practice facilitated by the lengthy release cycles characteristic of the era. However, as technology advanced, VMs yielded to the ascendancy of containers and cloud services, leading to an era of accelerated deployment cycles and heightened demand for lightweight, agile solutions [11].

The transition to containers has encountered friction when it comes to CI/CD tools as well as SAST security test compatibility. Challenges such as resource-intensive memory requirements and processor dependencies have hindered the seamless integration of SAST tools into containerized environments.

Despite the inherent advantages of Docker containers in terms of efficiency and scalability, the divergence in tool requirements poses a significant impediment to their effective deployment.

The integration of CI/CD into Agile and DevOps methodologies revolutionizes the speed and efficiency of software delivery. By automating the build, test, and deployment phases, CI/CD enables organizations to adopt a more iterative approach, significantly reducing the time to market for new features and updates.

Furthermore, as organizations shift towards microservices architecture to enhance scalability and flexibility, CI/CD emerges as a pivotal tool. Supporting the development and deployment of independent microservices, CI/CD pipelines facilitate easier updates and maintenance, reduce the risk of system-wide failures, and enable more granular scalability.

Finally, CI/CD pipelines are also instrumental in accelerating cloud adoption and optimizing cloud-native application development. They compile containerization and orchestration tools like Kubernetes, CI/CD, which enables organizations to deploy and manage applications more efficiently in the cloud.

7. Methodology

In mitigating the challenges encountered within CI/CD environments, adopting proactive measures is crucial, as Figure 1 shows. The security and code tests must begin from the pre-commitment phase, especially in cases where there is more than one team working on it.

7.1 Select Container-Compatible AST Tools

Prioritize tests that seamlessly integrate with containerized environments and cloud platforms, such as SAST, SCA, IAST, DAST, and more, as relevant. These tools should refrain from storing data within containers, ensuring easy data retrieval from shared locations post-analysis. Additionally, opt for lightweight AST tool images to facilitate effortless scalability without imposing undue resource burdens.

The GitHub repository python-compiler-tools/ast-compat provides a Python AST library with cross-version compatibility. Red Hat Enterprise Linux 9 provides command-line tools for working with container images, which can be used to manage pods and container images. Figure 2 shows the usage for code.

```
import ast_compat as astc
from ast_compat import get_constant

assert get_constant(astc.Constant((1, 2))) == (1, 2)

empty_args = astc.arguments() # work for all of Python 3.5-3.11
```

7.2 Flexible Licensing Management

Choose AST tools that offer flexibility in license management (for example, FlexLM [12]), enabling easy updates without necessitating hard-coded licenses within image configurations.

7.3 IP Address Independence

Mitigate reliance on container IP addresses within AST tools, as container stoppage and restarts may trigger IP address fluctuations. Implement robust solutions that decouple tool functionalities from container-specific identifiers to maintain operational stability.

7.4 Streamlined Report Formatting

Harmonize disparate report formats across AST tools to facilitate seamless integration with common pipeline activities. Prioritize tools that support varied report formats, thereby streamlining plugin customization for defect tracking and metrics dashboard updates.

7.5 Optimize Security Testing Workflows

Recognize the inherent time constraints associated with AST tool execution and tailor testing workflows accordingly. Embrace strategies to streamline testing processes and minimize workflow disruptions:

1. Implement incremental analysis support
2. Leverage granular scan controls
3. Evaluate AST tool performance metrics, such as scan times and findings. [13]

7.6 Tool Configuration for CI/CD Tool Resolution

Two key strategies to optimize tool configuration and address specific issues include:

1. **Technology, Language, and Framework Proficiency**: Equip your team with comprehensive knowledge of the technology, language, and framework in use.
2. **Incremental SAST Integration**: For frequent production deployments, incorporate incremental SAST runs within the pipeline as an inline activity. Conduct more extensive SAST operations as out-of-band activities or asynchronously within the pipeline.

A common bi-issue highlighted in the problem section includes the false positives in security findings that arise due to the constant security-performance loop scenario. It is important to implement effective measures to mitigate false positives. Developers should keep in mind that due to the extensive testing nature of this solution the false positive mitigation proposal is applicable in every step of the development cycle.

Before automating tools in the pipeline, conduct thorough onboarding for each application.

Next, it is important to enhance false positive identification by looking at the context of the application. Developers must acknowledge that not all SAST engines exhibit the same accuracy. Semantic analyzers may produce more false positives, while dataflow engines tend to be more precise.

Any security issues that may have taken a while to be identified and dealt with may lead to certain compliance requirements not being met. Often, this includes PCI or HIPAA compliance issues, especially for companies that handle PII (Personally Identifiable Information).

8. Conclusion

When implementing Continuous Integration and Continuous Deployment (CI/CD) for frontend development, there is extensive potential and countless possibilities, considering its future-integrative nature. However, there are also several challenges that it presents. Due to the sheer number of benefits it offers, it has slowly become the backbone of modern software development – especially for front-ends and backends dependent on them.

The proposed solution outlined in this paper advocates for a comprehensive CI/CD pipeline tailored specifically for frontend development. It proposes integrating automated testing frameworks, leveraging container-compatible AST tools, and implementing flexible licensing management. The idea is to help organizations and developers address the unique challenges posed by frontend development in the CI/CD environment – on an individual and organizational level.

Furthermore, the emphasis on incremental SAST integration, false positive mitigation strategies, and compliance differentiation underscores the importance of meticulous tool configuration and adherence to best practices.

References

[1]. P. Singh, "Docker for Front-End Developers," Medium, 13 08 2019. [Online]. Available: https://betterprogramming.pub/docker-for-front-end-developers-c758a44e622f.

[2]. C. Roso, "Deploying frontend applications—the fun way," DEV, 21 05 2020. [Online]. Available: https://dev.to/caroso1222/deploying-frontend-applications-the-fun-way-1fpf.

[3]. Andrew Mulholland, Github, "CI/CD: The what, why, and how," Github, [Online]. Available: https://resources.github.com/ci-cd/.

[4]. M. Wittpohl, "Automating the Build- And Deployment-Process," [Online]. Available: https://milanwittpohl.com/projects/tutorials/full-stack-web-app/automating-using-gitlab-ci-cd.

[5]. N. Choudhary, "Top 10 CI/CD Pipeline Implementation Challenges and Solutions," LambdaTest, 24 11 2020. [Online]. Available: https://www.lambdatest.com/blog/cicd-pipeline-challenges/.

[6]. You Now Have a Shorter Attention Span Than a Goldfish" Kevin McSpadden, 14 05 2015. [Online]. Available: https://time.com/3858309/attention-spans-goldfish/

[7]. M. Rao, "Common Security Challenges in CI/CD Workflows," Synopsys, 10 07 2018. [Online]. Available: https://www.synopsys.com/blogs/software-security/security-challenges-cicd-workflows.html.

[8]. R. Warren, "How to Add Application Security Tests to Your CI/CD Pipeline," StackHawk, 16 11 2020. [Online]. Available: https://www.stackhawk.com/blog/how-to-automate-appsec-testing-in-cicd/ .

[9]. Heike, "Why should I use Containers in CI/CD?," Deploybot, 25 06 2021. [Online]. Available: https://deploybot.com/blog/why-should-i-use-containers-in-ci-cd.

[10]. O. Cobles, "How to Solve For: Migrating Virtual Machines to Containers," Equinix, 03 08 2021. [Online]. Available: https://blog.equinix.com/blog/2021/08/30/how-to-solve-for-migrating-virtual-machines-to-containers/

[11]. Stack.io, "The complete guide to containers vs VMs for DevOps," Stack.io, 16 12 2019. [Online]. Available: https://www.stack.io/blog/containers-vs-vms.

[12]. OpenLM, "What is FLEXlm? What is FlexNet?," OpenLM, [Online]. Available: https://www.openlm.com/what-is-flexlm-what-is-flexnet/.

[13]. O. Peterson, "How to Incorporate Security Best Practices Into Your Workflow," Process.st, 02 04 2020. [Online]. Available: https://www.process.st/security-best-practices/.