SAUCELABS

WHITE PAPER

Optimizing CI/CD for Continuous Testing - Adapting Processes

As more organizations embrace Continuous Integration (CI) and Continuous Delivery (CD) as a mechanism to release apps faster, many find that there are a number of options to consider when making this transformational shift. However, while there is significant thought put into how development practices will change, very few teams consider how CI/CD will change the way they test the code that they create.

This technical paper is the third in a series outlining various topics development organizations of all sizes should consider when optimizing their processes for CI/CD, and how they relate specifically to testing. This critical piece of your engineering strategy can influence not only the quality of your applications, but also how quickly you can deliver them to your users. For many teams, these considerations can effectively make or break your CI/CD initiatives.

TABLE OF CONTENTS

INTRO

Moving to an effective and efficient CI/CD pipeline requires significant effort from organizations. It involves process and policy changes across every team. The payoff can be extraordinary: continual improvement as the organization moves to consistently deliver high quality digital experiences to their customers at speed. However, as your release velocity begins to increase, how can you still ensure your apps are well-tested?

Adding continuous testing best practices from the outset can enable the transformation to CI/CD by allowing code to move through an accelerated pipeline without testing becoming a bottleneck. To achieve this requires an entirely new set of features -- testability features -- built into the architecture itself, along with other changes to the way software is built. Without these changes, organizations typically struggle to see the benefits that CI/CD promise. This technical paper is the third in a series discussing the approaches, requirements and processes to consider when implementing continuous testing in a CI/CD workflow, and will focus on adapting processes.

ADAPTING PROCESSES

Moving to a pipeline that constantly delivers high-quality, well-tested software is far more a process change than a technical one. Of course significant changes will be required to an organization's technical practices; however, organizations will need to adjust their delivery processes. This can often be more challenging than technical changes, as it requires non-technical roles such as stakeholders, product owners, and users to adjust their work habits and mindsets. Getting past the "I just want to do my job!" mindset to "How can I improve my job?" isn't always easy, but there are a number of ways to help ease this transition as team members adapt how they're doing their work in order to successfully move to a CI/CD model.

MOVING TO SMALLER BRANCHES

Branching strategy is often as hotly debated as tabs versus spaces or Emacs versus Vim. The branching strategy a team chooses will have a significant impact on the team's ability to work as smoothly as possible in a CI/CD environment.

Small, short-lived branches (feature or smaller-sized) allow teams or even small groups (pairs!) of team members to do their own work isolated from others' churn in the codebase. Development and testing can be accomplished

without interruption from other work in the database. Small branches aren't something new: Martin Fowler wrote about them back in 2009 based on years of his experience. Dependency management across branches is still an issue, but it's lessened by good communication around APIs and automated tests to guard against regressions.

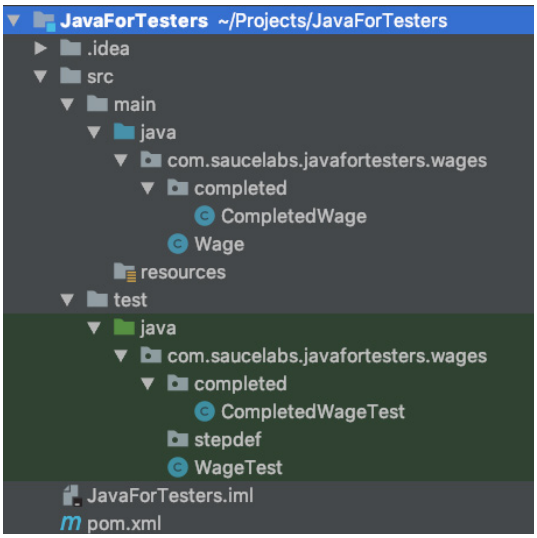
Normally each branch will have its own build job in the pipeline. Each branch has a job to monitor that branch in source control, then pull, build, deploy, and test as required. This requires more setup time up front; however, job/task templates make this much easier. (Every popular CI/CD toolset offers some form of job/task templating just for these situations.).

STORING TESTS IN SOURCE ALONGSIDE CODE

Keeping automated tests in sync with the application code they cover is critical to automated delivery pipelines. The entire team needs to have complete confidence in the checks guarding against regressions and confirming high-value business features.

The smoothest approach to handling this is simply keeping automated test code in the same repository as the system code. Having tests right alongside the system makes it simple to pull the current branch from source, build, deploy, and test. There's no mess about determining versions from other repositories, passing of messy variables, etc.

Organizing test and system code in a repository is something each project team needs to work out for their own requirements. Larger organizations might have a few guidelines, but teams will need to evaluate approaches that meet their environments and pipelines. Moreover, different toolsets prefer different organization of tests.



A common approach is to have unit tests very close in layout to the code they're testing, as shown in the following figure depicting a very simple Java project in IntelliJ. Source code is under /src/main with test code under /src/test.

Further organization of integration, functional, security, performance, and other test types generally are in separate projects. Again, approaches vary greatly across different organizations, teams, and toolsets.

CONCLUSION

While there are a number of technical considerations when building a well-tested CI/CD pipeline (see our previous technical papers on environment and feature management for more), no transformation is more difficult than process. This is because it is primarily a culture change, and requires buy in from all levels of the organization to take effect. However, when showing the benefits of these process changes -- the ability to work autonomously in smaller branches without fear of regressions, and full insight into quality with tests stored alongside source code -- the barrier becomes more surmountable. By framing these process shifts as beneficial for everyone's daily work, it becomes easier to get the required buy in across the organization, and sets up your CI/CD initiative for success.

Sauce Labs provides the world's most comprehensive Continuous Testing Cloud. Optimized for CI/CD with integrations to the industry's most popular tools, Sauce Labs is the perfect platform for all of your continuous testing requirements throughout your CI/CD pipeline. To learn more, take a look at this tech talk on integrating continuous testing into your CI/CD pipeline.

SAUCELABS

ABOUT SAUCE LABS

Sauce Labs is the leading provider of continuous testing solutions that deliver digital confidence. The Sauce Labs Continuous Testing Cloud delivers a 360-degree view of a customer's application experience, ensuring that web and mobile applications look, function, and perform exactly as they should on every browser, OS, and device, every single time. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP, Adams Street Partners and Riverwood Capital. For more information, please visit saucelabs.com.

saucelabs.com/signup/trial

FREE TRIAL

SAUCE LABS INC. - HQ 116 New Montgomery Street, 3rd Fl San Francisco, CA 94105 USA