SAUCELABS

WHITE PAPER

Optimizing CI/CD for Continuous Testing - Codebase Improvements

CONTINUOUS TESTING STARTS WITH TESTABLE CODE

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 final 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 dramatically enhance 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 final in a series discussing the approaches, requirements and processes to consider when implementing continuous testing in a CI/CD workflow, and will focus on improving your codebase.

LEVERAGING APIS FOR TESTABILITY

Public APIs are a crucial piece of any well-architected system, regardless of whether it's a monolithic system or one composed of numerous service-based components. These APIs provide an effective way to dramatically improve automated testing of a system as it's moving through a delivery pipeline.

A previous technical paper in this series mentioned dual-purpose features that provide test data or feature switching. Automating calls to these features is best done through an API, versus re-writing configuration files or injecting changes into a database. Using APIs ensures appropriate business rules for the general feature are followed.

Such an approach is critical when setting up an environment for automated and/or exploratory testing. This might include such steps as validating that catalog items are in stock and available when pulling SKU data from a retail data warehouse. It might ensure that only active customers are exported from a database. It could also appropriate pre-requisite steps when creating unique data for automated testing.

Imagine a test that checks if a customer can search for a particular item and place it in a shopping cart. All good automated tests avoid sharing state and data between themselves due to the extreme fragility of such an approach. This requires all data for a test to be randomly or uniquely generated. Pseudo code for setting up prerequisites might look similar to this:

```
Create_test_customer
        Randomly generate customer name, address, etc.
        Call system APIs to create a customer with random data
        Create_test_store_item
Generate an item with random name, description, etc.
        Call system APIs to create a store item with random data
        Create_test_customer_cart(test_customer)
Call system APIs to create a cart for test customer from above
```

Each pseudo method uses something like the Faker library to randomly generate appropriate data and in turn calls the true system APIs to create real objects with the randomly generated data. Again, the system APIs do all the proper validation (is the new customer's phone correct? Is the store item's price correct? etc.), relieving the team of having to rewrite and possibly inject bugs in their own prerequisite or validation code.

MANAGING CODE DEPENDENCIES WITH SOFTWARE CRAFTSMANSHIP

Perhaps the most fundamental concept for fast-moving, CI/CD environments is managing dependencies at the lowest level of the system. Using sound Software Craftsmanship principles ensures external services and dependencies can be properly mocked or substituted in various environments through the delivery pipeline.

Software Craftsmanship is a complex, varied school of practice for software construction. While there are many tenants to it, including a Manifesto, one of its basic principles is ensuring software is flexible and adaptable. Part of that concept is handled by ensuring dependency management is carefully thought out and implemented. Using one form or another of dependency injection means no component is responsible for creating dependencies it relies on. Instead, those components have their dependencies injected or passed into them.

Injecting dependencies on external services is one way teams can stand up systems in lower environments without being reliant on those external

systems. As an example, imagine a payroll system. Editing an employee's hourly rates or annual salary should require a security check to ensure the user doing the editing is indeed authorized to do it. That security check likely relies on some system outside the payroll system—a larger human resources system, for example.

With a properly architected system, a test of the employee wage edit feature could simply swap out a call to the "real" security system for a fake call that simply approves a test user for the edit. This cuts the dependency on an external system, ensuring tests could run in lower environments, or within unit tests themselves.

CONCLUSION

Our previous technical papers in this series have focused on a number of factors that contribute to the success of continuous testing in CI/CD, such as environment and feature management, as well as adapting processes. This last topic really gets to the nuts and bolts of what makes a CI/CD pipeline run, which is the code that developers create. Codebase improvements are a crucial element because it requires developers to consider quality and testability before they write their first line of code. From a theory standpoint, it's important to understand that everyone in the organization, not just developers, is committed to the ideals of Software Craftsmanship. This, along with tactical strategies such as leveraging APIs to improve the testability of code, not only make the QA teams' lives easier but also development, Ops and across your organization.

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