



How to Get the Most Out of Your CI/CD Workflow Using Automated Testing

Introduction

This paper is aimed at Test and QA Executives as well as Project Managers who are considering adopting automated testing, but are unsure of how to get started. It highlights the benefits of automated testing within organizations that have already adopted DevOps practices like CI/CD. It recommends specific technical approaches to take to automated testing, and it highlights useful tools that enable teams to adopt automated testing successfully as part of a healthy continuous integration and delivery process. It also touches on how and when to incorporate manual testing into the pipeline.




Table of Contents

4	Executive Summary	8	Usability Tests
4	The Role of CI/CD in the Modern Organization	8	One-off or Infrequent Tests
4	Kubernetes and GitOps Mean Even More CI/CD	9	Selecting the Right Test Automation Solution
5	Reality Check: Most Tests are Still Manual	9	The Crucial Decision: In-House vs. Open Source vs. Commercial Testing Tools
5	Obstacles to Adopting Automated Testing	10	Selenium - The Leading Test Automation Tool for Web Apps
6	The Right Approach to Test Automation	10	Appium - The Leading Test Automation Tool for Mobile Apps
6	Unit Tests/Component Tests	10	The Challenge of Open Source Testing: Expertise
6	Headless Testing	11	The Best of Both Worlds: Open Source Testing in a Test Cloud
7	API/Contract/Web Services Testing	11	Conclusion
7	UI Testing	12	Appendix: Checklist for Deciding What to Automate
7	Regression Testing		
7	Functional Testing		
8	Mobile Testing		
8	When to Stick to Manual Tests		

Executive Summary

In today's hyper-competitive cloud economy, it's more important than ever to be first to market to gain a competitive edge. This makes organizations prefer modern software development techniques like continuous integration and continuous delivery (CI/CD), which enables faster innovation, over the traditional waterfall approach to building software.

Automated testing is an integral part of the continuous delivery pipeline. However, despite the acknowledged benefits of automated testing, the reality is that most organizations still rely heavily on outdated, manual testing processes. The initial effort required to set up a test automation process makes teams want to avoid the pain, and make do with manual testing. However, to benefit from CI/CD, it's important to use the right technical approach to automated testing, as well as the right test automation solution.

The right approach involves knowing which tests to automate, and which to continue manually. Test automation scales the process to larger projects, and allows the team to cope with changes along the way. When selecting the right test automation tool, organizations are faced with three options - build one in-house, leverage an open source tool, or buy a commercial tool. Among open source frameworks, Selenium and Appium have emerged as the ideal way to automate testing for web apps and mobile apps—they are official standards, supported by browser and device-makers. However, they can be resource-intensive to set up and maintain in-house. Thus, the ideal test automation tool should be based on Selenium and Appium, but avoid the pain of manual code maintenance.

This paper aims to help Test and QA Executives and Project Managers who are considering adopting automated testing, but are unsure how to get started. It informs organizations about the benefits of test automation, the right approach to take, and the recommended tools to successfully adopt test automation as part of your CI/CD pipeline.

The Role of CI/CD in the Modern Organization

More than a decade ago, Continuous Integration / Continuous Delivery (CI/CD) began gaining popularity among developers. CI/CD is a software development and deployment technique that focuses on making rapid, incremental changes to applications and deploying updated applications on a frequent basis. It's a dramatic change from the waterfall software development models, in which applications are updated only a few times a year, at best.

Today, CI/CD has become the de facto way to build software. According to the Continuous Delivery Foundation's 2021 report on the "[State of Continuous Delivery](#)," a solid majority of developers now report that they use CI/CD techniques to build software.

CI/CD owes its popularity, in part, to the fact that it allows developers to break work into smaller chunks – as opposed to implementing, testing and deploying a large set of new features at once. In addition, CI/CD leads to higher rates of user satisfaction because it enables more frequent enhancements to application functionality, and faster resolution of performance and security issues.

Kubernetes and GitOps Mean Even More CI/CD

The shift toward cloud-native application architectures, in which applications are deployed as a set of distributed microservices, has encouraged adoption of CI/CD even further. Kubernetes, the most

popular platform for running cloud-native apps, is now used by more than 80% of companies that have also embraced DevOps, [Red Hat found in a 2021 survey](#).

When deploying to Kubernetes and other cloud-native platforms, the ability to update each microservice in an application continuously using CI/CD is even more important than it is when dealing with conventional, monolithic applications.

Widespread adoption of GitOps has had a similar effect. GitOps is a term coined by Weaveworks, who define it as ‘Operations by pull request.’ GitOps stresses the importance of automating every part of the software delivery pipeline, not just builds; in this respect, it makes CI/CD as wide-reaching as possible.

Reality Check: Most Tests are Still Manual

Yet, despite the signal importance of CI/CD for modern businesses, there is one glaring gap in the way many organizations approach CI/CD: Testing. In many cases, companies are still running tests manually. Doing so hampers not just the testing process, but the entire CI/CD pipeline.

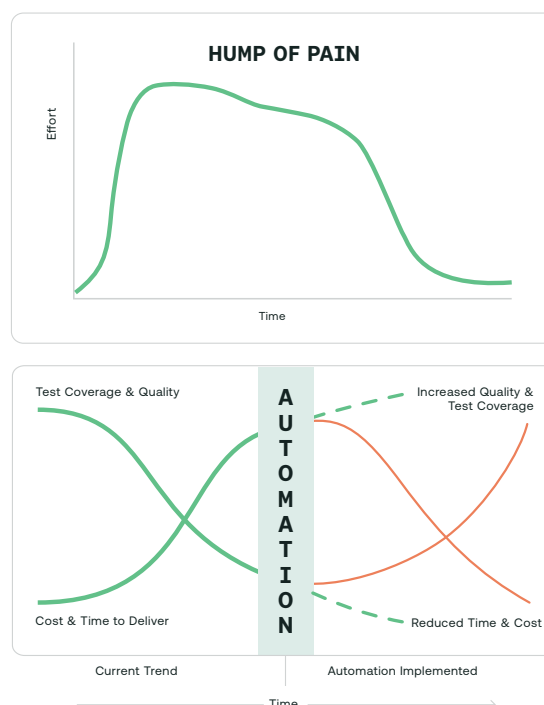
Indeed, test automation is an integral part of the broader CI/CD methodology. But surprisingly, most organizations still do the bulk of their testing manually – which is much slower, and requires more effort, than automated testing. According to data from [RWS and Simform](#), as of 2020, only 44% of businesses were using automated testing at all. And even among those companies, only about 50% of tests were automated, on average.

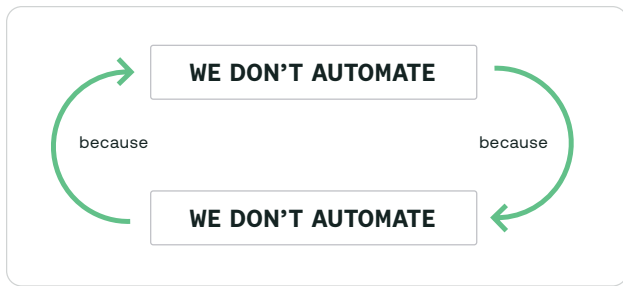
That’s a startling statistic, given that test automation is nothing new – the tools and processes that it requires have been widely available for well over a decade – and that automated tests deliver so many benefits for improving CI/CD.

So, let’s take a closer look at why test automation adoption remains relatively low, and what businesses can do to take better advantage of it.

Obstacles to Adopting Automated Testing

In the book *Implementing Automated Software Testing*, Elfriede Dustin cites an IDT survey in which 37% of respondents that were not users of test automation claimed a lack of time to be the main reason. Considering that one of the key benefits of test automation is quicker time to market, this leads to the common ‘chicken and egg’ paradox: While it is necessary to spend adequate time testing an app to maintain quality, the goal of a testing project should be to continually reduce the time spent on manual testing as much as possible. However, when they take their first steps towards this goal, QA teams find that there is normally a “hump of pain” associated with the change in workflow and mindset of the team members. This is when teams are most likely to give up on test automation.



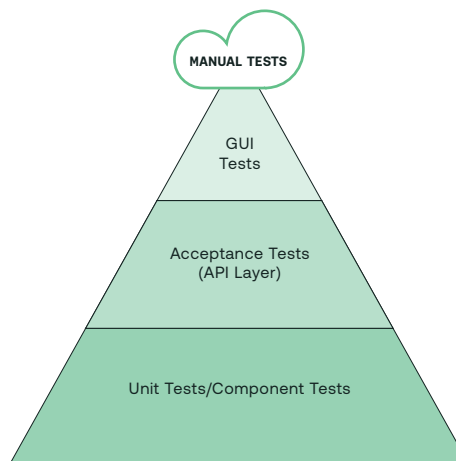


However, persisting with the test automation strategy will pay off. If done right, the rewards can be worth the temporary pain. Over time, the effectiveness of manual testing reduces, while that of automated testing only increases. When adopting test automation, being prepared for the “hump of pain” makes all the difference. As [Igor Khrol](#), a test automation specialist, says, “Select the right tool, the right framework, and the right technical approach.”

The Right Approach to Test Automation

The right technical approach involves knowing which tests to automate and which to continue manually. Tasks that are repeated most often, are the most labor-intensive—and are the highest priority—are the best candidates for automation.

The test automation pyramid developed by Mike Cohn, which suggests automating more low-level unit tests than high-level end-to-end tests, is a useful guide for determining which tests to automate. Let’s look in more detail at what it means.



Unit Tests/Component Tests

Unit and component testing is a way to test individual units of the source code to determine whether they are fit for use. They are a series of low-level tests that pinpoint exactly where to search to find bugs very early on in the cycle. The vast majority of commit tests should be unit tests.

Because of the need for quick feedback, unit tests should be very fast to execute and be run in memory. They should cover a large proportion of the codebase (80% is a good goal) to give a good level of confidence that when they pass, the application is fairly likely to be working. As Humble & Farley caution, “If your team is not automating unit tests, its chances of long-term success are slim. Make unit-level test automation and continuous integration your first priority.”

Headless Testing

Considering the thousands of commits per day in a large organization, it would take a lean and fast testing solution to execute the dry run tests we referred to earlier when talking about GitOps. These tests need to return results at near-real-time speeds. This calls for a stripped-down test framework that is powered by lightweight containers and sheds the weight of a full-blown browser.

Headless testing, or browserless testing, is just the solution for a frantic GitOps pipeline. Headless testing provides live feedback to developers while they are in the process of writing code. This is very

early in the software delivery pipeline. The result is that developers start to own the quality of the code they write. Bugs are caught early, saving the organization time and money. Lastly, this approach greatly improves the quality of code at every step of the pipeline.

API/Contract/Web Services Testing

API testing operates at the business logic layer of the software. Instead of using standard user inputs (keyboard) and outputs, in API testing, you use software to send calls to the API, get output, and note down the system's response. One challenge with API testing is to set up the test environment. The database and server should be configured as per the application requirements. Once the installation is done, the API function should be called to check whether that API is working.

Additionally, APIs are very inter-dependent among teams. It is vital that any change to the API not break "contract" (the set of endpoints as well as the expected request/response format) your API has published to other teams. Without contract testing, a microservice can go all the way through deployment before a team realizes that their API interactions are no longer compatible. This causes avoidable delays to the release process.

UI Testing

The goal of UI testing is to ensure the front-end interface of the application works as expected on all devices, browsers, and platforms. In the early days, UI testing used to be done by testers manually rendering the app on every possible combination of browsers and platforms.

This is no longer possible in today's fragmented ecosystem, particularly when it comes to mobile devices. In this case, cross-platform testing is a big challenge as the two major mobile platforms - iOS and Android - are poles apart in their approach to testing. A mature UI testing solution needs to transcend these barriers and account for the requirement to perform true cross-platform testing across iOS and Android. Additionally, these UI tests benefit greatly from testing solutions that are strongly visual in nature.

Regression Testing

Regression testing seeks to uncover new bugs or regressions after changes such as enhancements and configuration changes have been made to the application. Before a new version of the application is released, the old test cases are run against the new version to make sure that all the old capabilities still work.

With more frequent releases, teams typically tend to compromise on regression testing to save time, and in the process let bugs slip through the cracks. To keep up with faster development times, regression tests can be made faster using automation. Changes in the application may require the regression test scripts to be updated as well, which may require some manual intervention. However, automating regression tests saves time as teams don't need to run the same tests over and over again.

Functional Testing

Functional testing starts with a list of specifications or a design doc. Based on the specifications, various functionalities of the app are tested by feeding them input and examining the output against expected output. Functional testing doesn't consider the internal structure of the app being tested and is purely focused on the functionality of the app from an outcomes point of view.

For a web application, a functional test could be simply that a user manually navigates through the application to verify the application behaves as expected. But since automating a test is the best way to make sure it is run often, functional tests should also be automated whenever possible.

Mobile Testing

While much of what's been discussed so far applies to both web and mobile app testing, automating tests on mobile is more complex than for web apps. Here are some of the differences to keep in mind when automating mobile app tests:

- Cross-platform and device testing is even more complex with mobile. Testers carry multiple devices around and need to chase trends to test new and upcoming devices and OS releases. To add to this, mobile apps can be either native, HTML5, or hybrid, further increasing the complexity. Cross-platform and device compatibility is one of the main reasons mobile app testing requires automation rather than manual testing.
- Emulators may not give you the pixel-perfect resolution you might need for some testing, or allow you to see how your app functions with the quirks of real-life phone hardware. But they do allow you to do cost-efficient testing at scale.
- Testing on real mobile devices is indispensable, because of having to test each device's specific features like battery life, carrier networks, and OEM firmware. Additionally, there are numerous sensors like a touchscreen sensor, accelerometer, gyroscope, GPS, light sensor, fingerprint sensor, proximity sensor, and many more. While humans can test the performance of some of these aspects, only a machine can thoroughly test all variations accurately. In the past, automating tests on real mobile devices was difficult because of the lack of tooling. Today, however, there are powerful real mobile device cloud solutions that offer every possible combination of iOS and Android on a huge range of devices - all on demand without the hassle of buying and maintaining those devices in an expensive in-house device lab.

When to Stick to Manual Tests

While test automation is the way to go in most cases, there are exceptions. In some cases, it makes more sense to use manual tests.

Usability Tests

Usability testing helps you discover how easy it is for users to accomplish their goals with your application. There are several approaches to usability testing, from contextual inquiry to sitting users down in front of your application and filming them performing common tasks.

Usability testers gather metrics, noting how long it takes users to finish their tasks, watching out for people pressing the wrong buttons, noting how long it takes them to find the right text field, and getting them to record their level of satisfaction at the end. Usability, consistency of look and feel, and so on are difficult things to verify in automated tests and are ideal for manual testing.

One-off or Infrequent Tests

Tests that are run only one time or infrequently will not be high-payoff tests to automate. They are better done manually. If the one-off tests keep coming back after a point, it makes sense to consider automating them too. Thus, it makes sense to have a threshold for how much one-off testing you'd like to keep manual.

An example of a test that should be kept manual is an A/B test—if you're running an experiment where you want to test the traffic differences of an enhancement, you don't know until the experiment period is complete whether the "control" or the "enhanced" path will succeed. Automating the "enhanced" path too early might lead to wasted effort if the experiment isn't a success.

Selecting the Right Test Automation Solution

This is an important step that can have a big impact on the success of your automation efforts. Choosing the wrong tool can complicate the “hump of pain” problem, and increase the chances of giving up on test automation. Because of this, it’s important to select a tool that’s suitable for your needs.

If your app targets only a few platforms and browsers, a niche testing tool may just do the job, but most of today’s web and mobile apps are built to run on all possible combinations of operating systems and browsers, and you will most likely need a tool that covers all required platform and browser combinations. Additionally, the tool should be compatible with your existing technology stack.

For today’s web and mobile apps, cloud-based testing tools are gaining popularity because of their ease of installation, low maintenance, and secure handling of data.

The Crucial Decision: In-House vs. Open Source vs. Commercial Testing Tools

Considering the varying needs of Test and QA teams, it’s most likely that you’ll need a combination of tools to support your entire software testing lifecycle. Some will be custom testing solutions that your developers build in-house. Others will be open source solutions. Still others will come in the form of commercial, vendor-supplied tools.

It’s a best practice to focus on one tool that meets as many of your automation needs as possible, and use that as your primary testing solution. Then, find additional tools that perform specific tasks that the primary tool doesn’t perform.

When selecting the primary testing tool, the important decision is to choose between in-house, open source, and commercial tools. In-house tools give you the most control, but they require extensive development resources and expertise to build, and they are hard to maintain. They also can be quite expensive if you factor in all internal resources required.

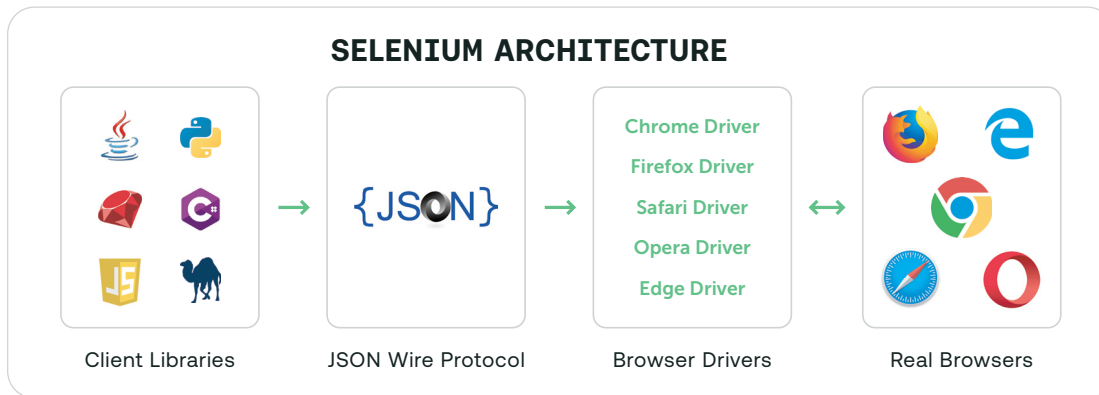
Open source tools have proven to be very capable of handling the most advanced automation projects, and are being preferred by more and more organizations. While they are easier to set up than building an in-house tool from scratch, they can become cumbersome to maintain. They require experts who specialize in open source tools. With no formal support available, the project can be delayed when complex issues arise.

Commercial tools are available in plenty, and they require careful examination before adoption. They are typically easier to set up and maintain, as the vendor would take on the load of keeping the tool up to date. Interestingly, many commercial solutions are based on open source tools. This enables them to follow industry-standard technologies, and prevent vendor lock-in. Commercial tools also come with support, which gives organizations that are new to test automation a great deal of confidence.

Considering the importance of open source tools in test automation, let’s look at two of the most popular open source test automation tools, and consider how they may fit into your testing workflow.

Selenium - The Leading Test Automation Tool for Web Apps

Selenium is the most widely used open source test automation tool, and it is the only official standard for test automation, supported by all browser vendors. Using the WebDriver protocol, you have the freedom to test your web app across browsers automatically, through test scripts. It allows testers to send commands to web browsers to make them perform tasks as though they were being used by a human. In this sense, it is like a robot for web browsing.



Jason Huggins, the creator of Selenium, [says on LinuxInsider](#) that “Selenium’s success came from its ability to test Firefox and IE on Windows or Mac or Linux and be able to drive it from Ruby or Python.” When thinking of automated testing for web apps, a Selenium-based solution is an ideal choice.

Appium - The Leading Test Automation Tool for Mobile Apps

Appium is the leading open source test automation framework for use with native, hybrid, and mobile web apps. It drives iOS and Android apps using the WebDriver protocol, the same API that controls the behavior of browsers with Selenium.

Appium allows automated tests to be written in any programming language. This is a big advantage over iOS’ XCTest framework, which lets you write tests using only Objective-C or Swift, or Android’s UI Automator, which supports Java or Kotlin. Under the hood, Appium still runs tests using XCTest and UI Automator, which are native frameworks. Thus, it doesn’t sacrifice on the experience of running tests. Appium opens up the possibility of true cross-platform native mobile automation and has become an indispensable tool for test automation.

Because Selenium and Appium both use the WebDriver protocol, these two open source tools in tandem provide you with a single, free testing protocol that has become a de facto standard. You won’t lock yourself into a proprietary stack. Nor will your engineers have to worry about mastering separate testing protocols, since both tools use WebDriver.

The Challenge of Open Source Testing: Expertise

While Selenium and Appium are powerful tools for browser-based and mobile test automation, the major caveat is that they both require a team with prior experience to be able to set up and maintain them.

There are a number of reasons why. One is that builds with Selenium tests can take longer to run, and need to be optimized regularly. It can also be hard to cover all the relevant browsers and platforms which will need to be manually installed, and configured frequently. This adds manual effort and defeats the purpose of test automation. In addition, scaling up a testing project can be complex and will require the Selenium Grid to be deployed, and continually maintained whenever Chrome or Firefox release new versions.

Thus, unless there's an in-house team with adequate experience in both frameworks, organizations looking to leverage Selenium and Appium on their own may be setting themselves up for failure.

The Best of Both Worlds: Open Source Testing in a Test Cloud

Fortunately, you can leverage open source test frameworks even if you don't have deep in-house expertise for them. By leveraging a cloud-based test service that is driven by Selenium and Appium, you get the best of both worlds: Open source flexibility on the one hand, and a secure, cloud-based, easy-to-use test platform on the other.

The best solutions in this category start a pristine new VM or container for every browser instance to make sure your tests aren't polluted by data from previous activity. However, most commercial testing solutions today run their VMs or containers on public cloud infrastructure. This results in false positives associated with browsers that don't close completely.

A better testing solution is a cloud that runs tests in dedicated VMs. You should also look for features like the ability to capture screenshots and videos of tests, which can make debugging a lot easier and would be preferred over solutions that don't offer these features. Other desirable features include enabling massive amounts of parallel tests, supporting tests written in any programming language, and integration with your CI server.

Finally, the best testing tools should be easy to configure, have a large existing user base, and be topped off with prompt customer support. A solution that meets these requirements would make the ideal test automation tool, resulting in a smoother transition from manual testing, a more effective CI/CD workflow, and eventually, higher-quality software that reaches the market much faster.

Conclusion

Organizations making the transition to CI/CD, or even GitOps, should simultaneously evolve their testing efforts from being predominantly manual to being increasingly automated. Despite the initial pains, the benefits of test automation make it worthwhile to invest in the right tool, the right methodology, and the right technical approach.

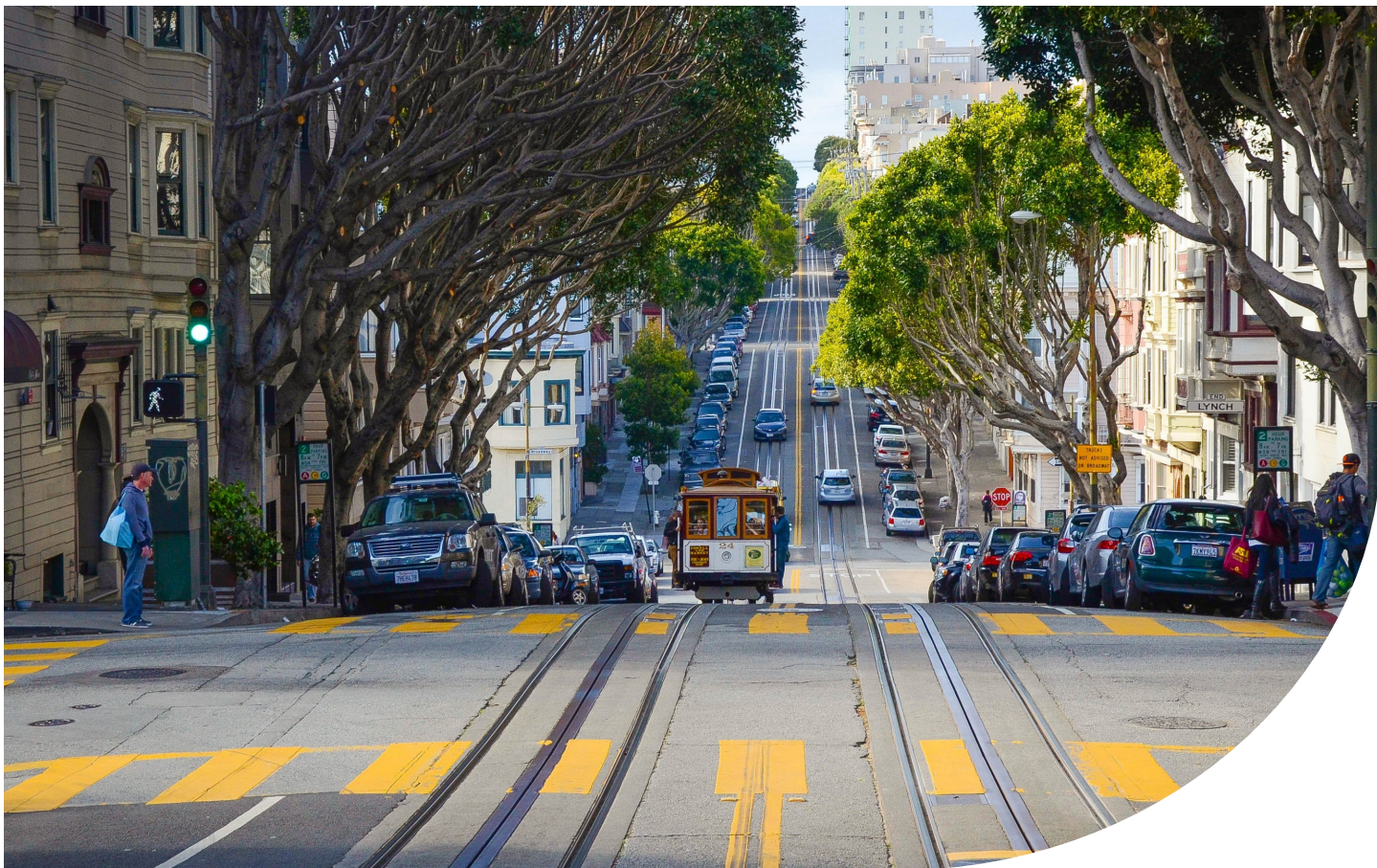
Selenium and Appium have emerged as the right tools for testing web and mobile apps, but they are tedious to maintain with in-house resources. For that reason, the ideal test automation is based on Selenium and Appium, but provides an easy-to-manage, cloud-based testing environment for executing Selenium and Appium tests in pristine VMs. It also enables a secure path to test apps behind a firewall, among other important features.

Organizations that invest in their test automation efforts are poised to get the most out of their CI/CD workflows. They will eventually be the ones to ship higher quality products faster, and in doing so, will put their competition to the test.

Appendix: Checklist for Deciding What to Automate

Use this checklist to assess which tests to automate, and which to continue manually. Tests with more yeses than nos are the best candidates for automation.

TEST AUTOMATION CRITERIA	Y	N
What risks does this test cover? Is the test critical to the user experience? If this test isn't run, and a problem is found, will it interrupt the user's ability to get their task done? Will it negatively affect revenue?		
Is the test executed more than once? Is the test run on a regular basis, i.e., often reused, such as part of regression or build testing? Does the test need to be run on multiple device and platform combinations?		
Does the test cover the most critical feature paths (often the most error-prone)?		
Is the test run on a dynamic, ever-changing application?		
Does the test require multiple data inputs to test the same feature? Will someone else in the organization need to run the same test to reproduce or verify an issue? Is the test very time-consuming?		
Is the test expected to return a response in real-time?		
Is it more effective to run the test in parallel with many other tests?		
Is the test prohibitively expensive to perform manually?		
Will you require high definition reporting for the test results?		



About Sauce Labs

Sauce Labs is the leading provider of continuous test and error reporting solutions that gives companies confidence to develop, deliver and update high quality software at speed. The Sauce Labs Continuous Testing Cloud identifies quality signals in development and production, accelerating the ability to release and update web and mobile applications that look, function and perform exactly as they should on every browser, operating system and device, every single time. Sauce Labs is a privately held company funded by TPG, Salesforce Ventures, IVP, Adams Street Partners, and Riverwood Capital.

For more information, please visit
→ saucelabs.com



saucelabs.com/sign-up

FREE TRIAL