WHITE PAPER

HOW TO GET THE MOST OUT OF YOUR CI/CD WORKFLOW USING AUTOMATED TESTING

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, the recommended technical approach to take, and suggests tools that enable teams to successfully adopt automated testing as part of a healthy continuous integration and delivery process. It also examines which tests to automate and which to continue to do manually.

SAUCELABS

TABLE OF CONTENTS

SAUCELABS

EXECUTIVE SUMMARY

In today's hyper-competitive cloud economy, it's important 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) 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 However, despite the acknowledged benefits of automated testing, the reality is that most organizations still use outdated manual testing processes.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 integrating automated testing, as well as the right test automation solution.

The right approach involves knowing which tests to automate, and which to continue manually. Having a testing framework enables the process to be scaled to larger projects and better 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 respectively. However, they can be resource-intensive to setup and maintain in-house. Thus, the ideal test automation tool should be based on Selenium and Appium, but avoid the pain of manual maintenance.

This paper is aimed at Test and QA Executives as well as Project Managers who are considering adopting automated testing, but are unsure how to get started. It will inform 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.

AUTOMATED TESTING AS PART OF THE BROADER CI/CD PIPELINE

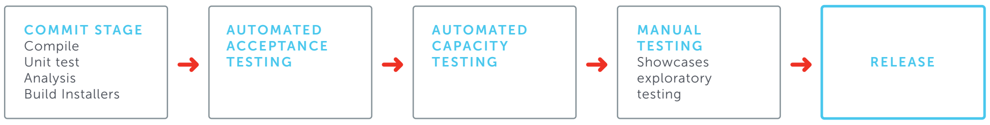
In their groundbreaking book, "Continuous Delivery", Jez Humble and David Farley begin by declaring that "The most important problem that we face as software professionals is this: If somebody thinks of a good idea, how do we deliver it to users as quickly as possible?"

Realizing that the traditional waterfall technique of software development is inadequate to meet this goal, organizations are slowly but surely adopting agile development techniques like continuous integration and continuous delivery.

- Definition of Continuous Integration (CI): "A development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early." - ThoughtWorks

- Definition of Continuous Delivery (CD): "The practice of releasing every good build to users" - Jez Humble

Humble and Farley define what a typical deployment pipeline looks like using the following illustration:



They describe this pipeline as "an automated implementation of your application's build, deploy, test, and release process." They go on to recommend that as much of this pipeline should be automated.

Commenting on this pipeline, renowned expert and ThoughtWorks Chief Scientist, Martin Fowler says, "A deployment pipeline breaks up your build into stages. Each stage provides increasing confidence, usually at the cost of extra time. Early stages can find most problems yielding faster feedback, while later stages provide slower and more thorough probing. Deployment pipelines are a central part of Continuous Delivery."

As seen from the above pipeline, automated testing is a vital component of the continuous delivery process.

THE KUBERNETES EFFECT

With the launch of Docker containers in 2014, the world of DevOps has never been the same. Shedding the heavy 'hypervisor tax', containers are able to bring agility and performance gains at every step of the CI/CD pipeline. As container usage grows explosively, Kubernetes has emerged as the de facto standard for orchestrating containers at scale.

Kubernetes puts even more focus on the CI/CD pipeline, looking beyond tools like Jenkins to deployment automation tools like Spinnaker and Helm. In doing so, Kubernetes goes the full length, enforcing automation not just to testing, but to deployments as well.

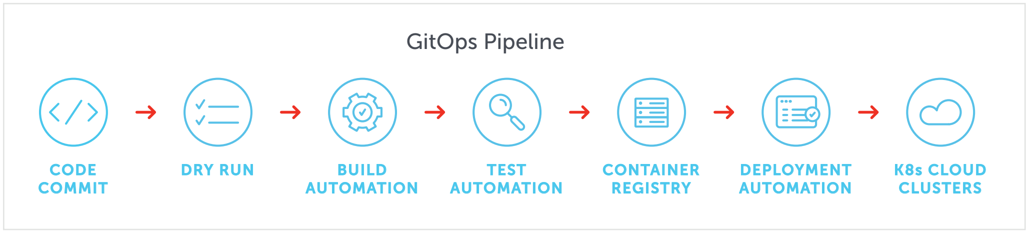
SAUCELABS

Learn more at saucelabs.com

Kubernetes made it possible to put together a fully-automated CI/CD pipeline that kicks off a chain reaction starting with a code commit and ending with a deployment, without being touched by another human along the entire process. It also enables the automated creation of environments, whether they be for development, staging, or production. To put it in a single word - 'GitOps' - That's what Kubernetes naturally leads us to.

GITOPS = FULLY-AUTOMATED CI/CD

'GitOps' is a term coined by Weaveworks, which they describe as 'Operations by pull request.' While the CI/CD pipeline called for end-to-end automation, in reality, most organizations were only able to perform build automation using tools like Jenkins. On the other hand, GitOps stresses on the importance of automating every part of the software delivery pipeline, not just builds. Weaveworks is careful to point out that GitOps is not entirely new, rather, it "builds and iterates on ideas drawn from DevOps." In other words, GitOps is what CI/CD has always strived for—end-to-end automation.

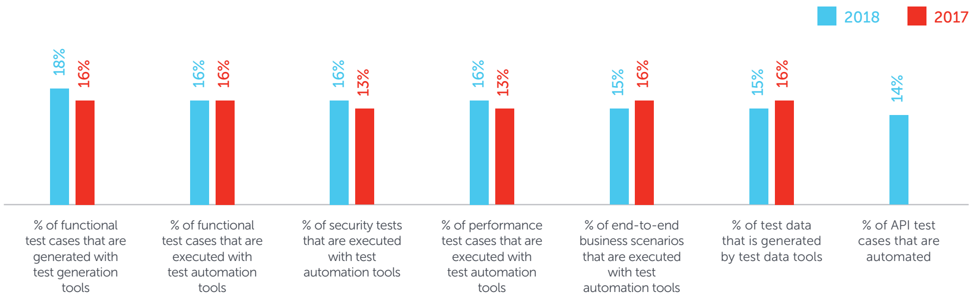


In GitOps, every code commit sets off a series of automated steps that result in a deployment. Explaining this process, Twain Taylor comments that every new piece of code goes through a 'dry run' before making it to a production environment. If the code fails the dry run, the deployment is not carried out. This dry run is an automated test that acts as the gatekeeper for every deployment. The dry run is an indispensable part of the GitOps pipeline and points to the fact that GitOps cannot be achieved without test automation.

REALITY CHECK - THE MAJORITY OF TESTING IS STILL MANUAL

Test automation is an integral part of the broader CI/CD methodology. But surprisingly, most organizations still do the bulk of their testing manually. According to the annual World Quality Report from Capgemini, "99% of respondents were using DevOps at least in some part of their business." While this is a promising sign, on the downside, "Organizations are not able to tap the full benefits [of DevOps]... due to low levels of test automation." The report goes on to say that the levels of basic test automation are very low - between 14%-18%.

INDICATIVE PROPORTION OF AUTOMATION OF EACH OF THE FOLLOWING ACTIVITIES



Source: Microfocus.com

Digging deeper into the issue, the survey found that 50% of respondents are unable to “apply test data at appropriate levels.” Further, 55% mentioned a “lack of end to end automation from build to deployment” as a key challenge. This is another reminder that test automation is an integral part of the CI/CD pipeline. The report concludes that “Test automation is the biggest bottleneck holding back the evolution of QA and testing today.”

WHAT IS TEST AUTOMATION?

Manual testing involves a human tester using a computer or mobile device, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Automated testing, on the other hand, uses tools to execute pre-scripted tests on a web or mobile app, and records the test data in detailed reports that can be reviewed by a human tester after the tests are run.

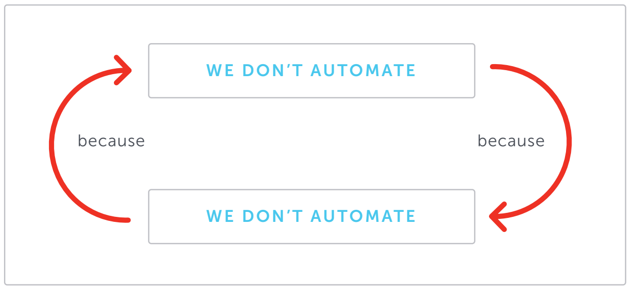
- Definition of test automation: It is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes. - Wikipedia

MANUAL TESTING VS AUTOMATED TESTING - WEIGHING THE BENEFITS

CRITERIA	MANUAL TEST	AUTOMATED TESTING
Faster time to market		✓
Ideal for cross-platform mobile tests		✓
Improves test coverage		✓
Accurate & error-free		✓
Performs complex tests		✓
Increases frequency of feedback		✓
Supports agile methodology		✓
Frees up testers for their best work		✓
Scales to larger projects		✓
Is repeatable		✓
Enables after-hours testing		✓
Improves documentation & traceability		✓
Cost effective		✓
Improves over time		✓
Ideal for exploratory tests	✓	
Ideal for one-off tests	✓	
Easy to adopt	✓	
Easy to maintain after adoption		✓

OBSTACLES TO ADOPTING AUTOMATED TESTING

In the landmark book, 'Implementing Automated Software Testing' Elfriede Dustin refers to 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:

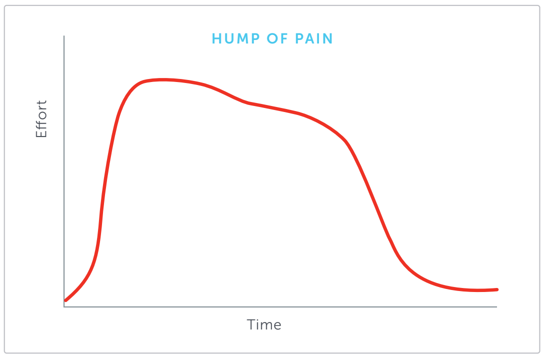


While it is necessary to spend adequate time testing an app to maintain quality, the goal of a testing project should be to continually reduce the time spent on manual testing and automate as much of the tests as possible.

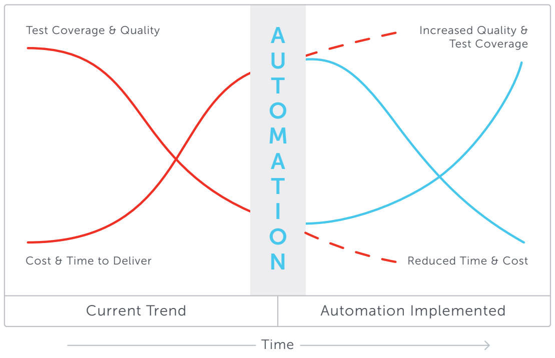
SAUCELABS

Learn more at saucelabs.com

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.

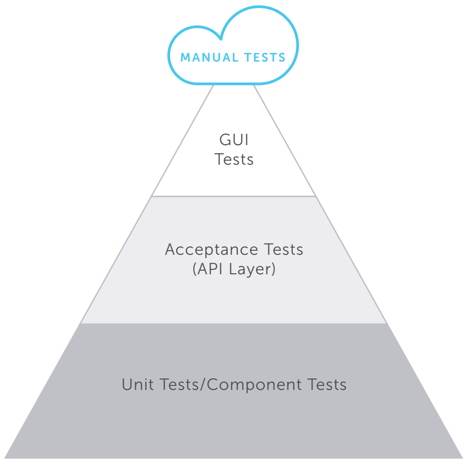


When adopting test automation, being prepared for the "hump of pain" makes all the difference. As Igor Khrol, a test automation specialist, says, "Select the right tool, the right framework, and the right technical approach."

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 and are the most labor-intensive are often the best candidates for automation.

[The Appendix includes a checklist with questions for deciding which tests to automate.]



The test automation pyramid developed by Mike Cohn suggests automating more low-level unit tests than high-level end-to-end tests.

This is a great way to start your journey in test automation - begin with the basic building blocks - unit tests.

SAUCELABS
Learn more at saucelabs.com

Unit and Component Testing

Unit tests are 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 comprised of 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 (around 80%) 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 a developer 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 or 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.

UI Testing

The goal of UI testing is to ensure the frontend 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. However, this is not possible in today's fragmented mobile ecosystem.

Cross-platform testing is a big challenge as the two major mobile platforms - iOS and Android - are poles apart in their approach to mobile app testing. A mature UI testing solution should then be able to transcend these barriers and enable true cross-platform testing across iOS and Android. This means a 'write once, run anywhere' approach with test scripts. Additionally, 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 is 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 platforms & devices 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.

WHICH TESTS TO CONTINUE MANUALLY

Usability Tests

Usability testing is done to discover how easy it is for users to accomplish their goals with your application. There are several different 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 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.

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, Open Source, or Commercial

Considering the varying needs of a test and QA team, it's most likely that you'll need a combination of tools to support your entire software testing lifecycle (STLC). That said, it helps to focus on one tool that meets most of your testing automation needs first, and then find 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 require expertise to build, and are hard to maintain. They also turn out to be expensive if you factor in all internal resources required.

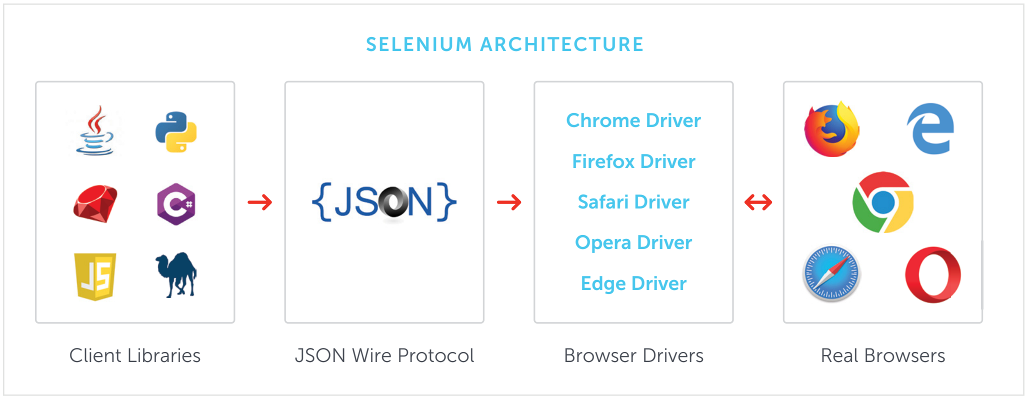
Open source tools have proven to be very capable of handling the most advanced automation projects, and are being preferred by more and more organizations. While they are easier to set up than building an in-house tool from scratch, they can become cumbersome to maintain. They require experts who specialize 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. It gives you the freedom to test your web app across browsers automatically, through test scripts. It is a technology that 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.

Investing in the WebDriver protocol means you are betting on a single, free and open protocol for testing that has become a defacto standard. This way, you won't lock yourself into a proprietary stack.

OPEN SOURCE TOOLS REQUIRE EXPERTISE TO RUN IN-HOUSE

While Selenium and Appium are powerful tools for browser-based test automation, and mobile test automation, these tools require a team with prior experience to be able to set up and maintain them.

Builds with Selenium tests can take longer to run, and need to be optimized regularly. It can 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. Also, scaling up a testing project can be complex and will require the Selenium Grid to be deployed, and tests to be rewritten to support multi-threading.

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. Because of this, it makes sense to evaluate a commercial option that leverages both Selenium and Appium.

THE IDEAL SOLUTION SHOULD COMBINE THE BEST OF BOTH WORLDS - SELENIUM & APPIUM

The ideal test automation solution would need to be cloud-based, and still deliver the high level of security that's required when testing apps behind firewalls. The solution you choose should start a pristine new VM or container for every browser instance to make sure your tests aren't polluted by data from previous activity.

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. Solutions that capture screenshots and videos of tests 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 tool would need to 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. The ideal test automation solution would be based on Selenium and Appium without the load of manual configuration. It would deliver pristine VMs for each browser instance, and 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.

ABOUT SAUCE LABS

Checklist for Deciding What to Automate

Use this checklist to assess which tests to automate, and which to continue manually. Tests with more yes' than no's are the best candidates for automation.

TEST AUTOMATION CRITERIA		
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 ensures the world's leading apps and websites work flawlessly on every browser, OS and device. Its award-winning Continuous Testing Cloud provides development and quality teams with instant access to the test coverage, scalability, and analytics they need to rapidly deliver a flawless digital experience. 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

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

SAUCE LABS EUROPE GMBH
Stralauer Allee 6
10245 Berlin DE

SAUCE LABS INC. - CANADA
128 West Pender Street, 8th Floor
Vancouver, BC V6B 1R8, Canada

SAUCE LABS - POLAND
Złota 59 St., 4th Fl.
00-120 Warsaw, Poland