WHITE PAPER

REAL MOBILE DEVICES FOR CONTINUOUS TESTING

Testing mobile applications requires that the developers test under conditions that are as close to the real world as possible. This means choosing the right selection of real devices. Fortunately the creation of real device lab services has made procuring real devices easier than ever to ensure app quality on multiple device platforms.

This whitepaper will explore the advantages of using real devices for manual and automated testing.

SAUCELABS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Manual and automated testing of mobile applications can take place on both real and virtual devices. Real devices are physical phones or tablets, identical in hardware and software to the devices users have in their pockets. Leveraging real devices for testing unlocks a number of advantages for the tester, primarily around fidelity—real devices provide the highest-fidelity environment for catching or reproducing quality issues. A number of features (cell network change handling, SMS, phone calls, etc...) can only be adequately tested on real devices. Using a test framework like Appium makes it dead simple to switch between real and virtual devices as necessary, simply by modifying a bit of information in the test preamble. Once you have a number of Appium tests, it makes sense to consider running them in the cloud, to unlock greater speed and device coverage for your build, and to eliminate the various hassles that come with managing your own real devices. Real device testing is an essential aspect of Continuous Testing for mobile, and every step should be taken to make sure that choosing real devices for testing speeds up your release cycle, rather than bogging it down with additional complexity and expense.

REAL AND VIRTUAL DEVICES

When embarking on a QA effort for a mobile app, it's essential to understand the types of devices available for building out your test strategy. In general there are three ways to test mobile apps:

1. **On real devices.** By "real devices" we mean physical phones and tablets purchased from a vendor with an operating system like Android or iOS installed. The device may or may not be modified to support testing different scenarios (for example, rooting an Android device to unlock different layers of device automation), but in general a real device is exactly the same as what some segment of your users are actually using your app with.

2. **On virtual devices.** By "virtual devices" we mean software programs that approximate, to some degree, the workings of a real device. There are two mutually exclusive strategies employed by makers of virtual devices, namely simulation and emulation. Simulators essentially implement a version of the mobile OS which is tied to the hardware and system calls of the host machine (real or virtual). In other words, an iOS simulator runs a version of iOS which is still at the end of the day running on the kernel of whichever macOS machine has the simulator open. Emulators, on the

other hand, constitute a virtual representation of the entire mobile device, down to the low-level system calls. Emulators are therefore a kind of virtual machine, whereas simulators are not. Both kinds of virtual devices can be run on developers' local machines to facilitate app development or testing. Because virtual devices are simply software programs, there is a substantial difference in the ease of acquiring and scaling virtual devices, as compared with real devices (which must be purchased, physically received, and physically incorporated into a test rig).

3. **Using a desktop browser's dev tools.** For mobile web apps or hybrid apps only (not native apps), it is often convenient to design and manually test an application in a web browser, using a developer tool like Chrome's Device Mode. This mode displays the web application as if it were being rendered on a mobile device, by adjusting window size, interpretation of CSS, and how events are handled. Neither simulation nor emulation of the device is happening, and this test mode is primarily useful for quickly iterating on visual changes during development, or for manual testing of responsive design issues.

This whitepaper focuses on the use of real devices, and the subsequent sections will explore the benefits of using real devices for testing, how to take advantage of cloud-based real devices, and the place of real devices in the overall Continuous Testing (CT) paradigm.

ADVANTAGES OF REAL DEVICES FOR TESTING

Fidelity. The one word that summarizes all the advantages of using real devices in your CT system is "fidelity". There are higher- and lower-fidelity tests, and higher- and lower-fidelity testing tools. Functional testing (defined as black-box testing which manipulates the UI of an application the same way a user would) is the highest-fidelity kind of testing there is. Compared to, say, unit tests, functional tests most approximate real-world usage. In fact, functional tests are basically automated versions of what a human tester would do when running through a test scenario for a given app.

Similarly, real devices are the highest-fidelity kind of test device. Your users are using real devices, and hopefully the same ones that you have available for use in testing (whether manual or automated). Emulators are a significant notch below real devices in terms of fidelity, and simulators a bit further down yet. Most companies engage in functional testing not for the purpose of proving that code is covered by tests, but for ensuring that important user

flows are tested in as high-fidelity an environment as possible, utilizing the whole app stack from UI components down to backend service integration.

From the perspective of manual testing, this is all we need to say. Nothing beats reproducing user-reported bugs on the exact devices which the users experienced problems with. Likewise, there is no reason (apart from cost or difficulty of procurement) to use virtual devices instead of real devices when it comes to manual testing. With the advent of cloud-based real device testing, cost is minimized and the difficulty of procurement erased. We will discuss this more below.

Speed. From the perspective of automated testing, fidelity is obviously a core consideration for all the same reasons as manual testing. Another critical aspect of choosing real devices for automation is speed. Because virtual devices are either completely virtualized or running as one process among many on a host system, they can be relatively slow. Real devices have dedicated hardware which has been tuned by the manufacturer precisely for the purpose of running apps like yours as fast as possible. It is usually the case that both app performance and automation software performance are improved when running on real devices. Incorporating real devices into your build as much as possible therefore means your tests are likely to run faster than on virtual devices.

Performance Testing. Because real devices have their own CPUs, GPUs, and other hardware components, real devices are to be preferred when doing any kind of performance testing. Virtual devices can give at best a relative view of app performance, because they are running virtualized system instructions, or on a host OS. Even then, relative reports need to be scrutinized closely, because system events external to the virtual device can affect its operation.

Performance data captured from real devices is relevant even in small quantities, because it reflects the experience of your users.

Addressing Device Fragmentation. Finally, leveraging real devices is the only way to address device fragmentation among your user base. Especially on Android, the number of distinct mobile devices used by your customers can easily be in the dozens. Testing via virtual devices will never uncover device-specific bugs before your users report them. It's therefore a good idea to have a solid understanding of which devices are popular among your users, and to proactively test with them in advance of a release.

In addition, we can discuss specific automation capabilities available only to real devices. For example, using Appium as the reference framework, real devices are essential for the following test requirements:

1. **Testing Real Networks.** If you want to get feedback on your app performance in real-world conditions, over actual 3G or LTE networks, you will need a real device.

2. **Testing Network Change Events.** If you want to test how your app behaves when network conditions change (for example, moving from cellular network to WiFi), you will need a real device.

3. **Testing SMS, Phone Calls, and Push Notifications.** Android emulators support basic synthesizing of SMS and Phone events, however iOS simulators do not. Nor do iOS simulators support testing of Push Notifications. If you want to test these how your app sends and receives SMS, phone calls, or push notifications, you're best off using a real device.

4. **Testing Audio Input / Output.** If your test case requires injecting audio input, you need to use a real device connected to a microphone. Generating the sound itself is outside the scope of automated test frameworks, but regardless, you will need a real device to pick up any sound. Likewise, if your app produces sound, the easiest way to capture that is via a recording system connected to a real device.

REAL DEVICES AND APPIUM

Appium is a mobile automation tool that supports many platforms (including iOS and Android), app modes (native, hybrid, web), and device modes (both real and virtual devices). The way to use Appium to target a real device instead of a virtual device is via the Desired Capabilities that are used to initiate an Appium session. Each language-specific Appium client builds Desired Capabilities in its own way. For example, here is a set of basic Desired Capabilities used to automate an iOS app, from within the Appium Java client:

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability("platformName", "iOS");
capabilities.setCapability("deviceName", "iPhone 7");
capabilities.setCapability("platformVersion", "11.2");
capabilities.setCapability("app", "/path/to/local.app");
```

The the `DesiredCapabilities` object is used to create an instance of `AppiumDriver` which contains all the automation API methods on it. In the

minimal case, two updates to this set of Desired Capabilities are required in order to successfully target a real device:

1. A new capability must be added, "`udid`", with a value of "`auto`" (if only one device is connected), or with the actual device ID of a connected device.

2. The value of the "`app`" capability must be updated to be the location of a signed `.ipa` file (for iOS), or `.apk` file (for Android).

For iOS apps in particular, app signing and device provisioning can be a challenge. Because of the security model adopted by Apple, only signed apps can run on real devices. And for any app not signed by the official App Store, it is restricted to running on devices which have been listed in a provisioning profile tied to your application. This is a hurdle in and of itself, and has nothing to do with Appium. Users running iOS tests on real devices must have the appropriate signing certificates and provisioning profiles downloaded on their system through the Xcode certificate management interface. Once this is done, the steps to follow are:

1. Ensure your provisioning profile allows for a wildcard app ID, something like "`com.company.*`". This will enable the provisioning profile to validate the install of your own app, as well as a rebuilt version of WebDriverAgent which has been ported to this namespace, for example as "`com.company.webdriveragent`"

2. Find your team's Apple Team ID from your Apple Developer Portal account.

3. Find the name of your signing identity on your local machine (usually it is simply "iPhone Developer".

4. Set three new capabilities in your Appium session initialization: first "`xcodeOrgId`", with the value of your team ID. Then "`xcodeSigningId`" with the name of your signing identifier. Finally, "`updatedWDABundleId`", with a valid iOS bundle ID that will work with your wildcard provisioning profile (e.g., "`com.company.webdriveragent`").

With these new capabilities in place, Appium will be able to sign, install, and run the automation libraries on the phone, and you will be off and running. On Android, there is thankfully no requirement for devices to run only signed apps, or for apps in development to be restricted to only a set of pre-provisioned devices. In fact, on Android, starting sessions on real devices is no different at all than starting sessions on an emulator.

REAL DEVICES AND ESPRESSO, XCUITEST

When it comes to mobile automation, Appium is not the only game in town. All of the advantages listed above for real device-based automation also apply to the other popular automation frameworks, Espresso and XCUITest. Espresso is maintained by Google as their recommended tool for Android UI automation. It is integrated tightly with the app and with the Android Studio development environment, and many dev teams prefer Espresso for this reason, not to mention the view synchronization features which help to reduce test instability. XCUITest is released by Apple for the purpose of iOS testing, designed as a set of additional libraries for use with the XCTest framework. Like Espresso, XCUITest-based scenarios must be written in a particular set of languages (Objective-C or Swift, whereas for Espresso it is Java or Kotlin), and has first-class integrations with the Xcode IDE.

The requirements for running tests on real devices using Espresso or XCUITest are basically the same as for Appium, with the consequence that getting going with Espresso on real devices is marginally easier than with XCUITest (because of Apple's code-signing process). However, when staying within Xcode to write tests, Apple's code signing helper UI is available.

REAL DEVICES IN A CLOUD ENVIRONMENT

One of the big downsides of real device testing is the expense and manual labor associated with procuring and maintaining a grid of devices available for testing, especially with any kind of uptime guarantee. Real devices run out of power, wind up on odd states, lock their screens, and in general are less uniform than virtual devices. New devices come on the market and the devices required for testing change. Old devices need to be repurposed or recycled, and new devices procured (sometimes with great difficulty if the device is popular). It is a serious investment of time, money, and expertise to maintain an in-house real device grid. For this reason alone, many companies choose to satisfy their real device requirement by moving to the cloud.

With a cloud service like the Sauce Labs Real Device Cloud (RDC), you have access to real devices for manual or automated testing (via Appium, Espresso, and XCUITest) without having to procure or maintain those devices yourself. A huge variety of device models are hosted in secure datacenters around the world, available in an on-demand fashion over secure network connections that work even inside corporate firewalls. Private, customer-specific devices are also available. There are 5 main benefits of cloud-based real device testing:

1. **No setup or maintenance.** In-house real device grids are a major investment, sometimes with little return. The Sauce Labs RDC maintains everything, so you don't have to, and comes with uptime guarantees. You get the value of real device testing without the effort of building it yourself.

2. **Access to many device models.** Your test requirements change over time, and buying and recycling devices as you need them is not a good use of resources. The Sauce Labs RDC has an array of both currently popular and more rare devices so that, even if you have your own device grid, you may be able to find something in the cloud which you don't have access to locally.

3. **Forget signing woes.** We described above some of the complications of real device testing due to Apple's app security model. With the cloud, you don't worry about it; you simply upload your signed application, specify the device you want via the Desired Capabilities, and the rest is taken care of. You get to focus on your automation rather than asking your IT department to navigate the jungle of certificates and provisioning profiles.

4. **Speed.** The Sauce Labs RDC is purpose-built to scale to meet the testing needs of many customers. With the number of devices available, you will be able to run many more tests at once than on your local setup. The key to a fast release cycle is a fast build and verification process, and the key to a fast build is running tests in parallel. It's only with a cloud service that you'll be able to achieve a high enough parallelism to get your build time low and to make your developers happy with the feedback loop.

5. **Test Data and Analytics.** Getting your tests run is only one part of your job. The other part is debugging test failures and tracking the health of your build over time. Without easy access to device videos and screenshots and automation logs, you'd be unable to do your job effectively. The Sauce Labs RDC provides all these features as well as high-level insights into test performance over time or build-level statistics.

If you have Appium tests running locally, migrating them to the Sauce Labs RDC is as easy as uploading your app in the web interface (or using a REST API), adding your user token in the Desired Capabilities, and resetting certain other capabilities based on the names of the devices you want to run your test against. Similarly, moving Espresso or XCUITest test suites to the cloud is dead simple. Check out the RDC docs for more details on how to start leveraging Sauce Labs RDC with your mobile tests.

REAL DEVICES AND CONTINUOUS TESTING

Continuous Testing is all about integrating testing at every part of your release cycle. Real devices are a crucial step in this process because they represent the highest-fidelity kind of testing available. In an ideal world, all testing would be done on real devices, many times during the development of a feature as it moves toward production. In the real world, a mix of virtual and real devices often makes sense given various constraints. Even so, the more real devices can be integrated into the test process, the more confidence you will have in the real-world performance of your app. The advent of cloud-based real device testing makes this much more achievable than in the past. It's possible to have a set of smoke tests run on even one real device on every build, simply by adjusting a few Appium capabilities and running those tests over the network on the Sauce Labs RDC.

There are plenty of challenges to CT for mobile, from forced delays due to gatekeeping of releases by Apple or Google to the incredible fragmentation of devices which testers must face. Managing a grid of real devices is another big challenge, but it's one that's easily outsourced to the cloud. CT as a paradigm makes big demands on the speed of testing. Integrating testing at every point of the development cycle only works if testing is transparent, fast, and reliable at finding issues (no false positives). To achieve this speed, we need to employ every available tool which can shorten the length of a build or free testers to develop competencies more closely related to the automation of their particular apps. Real device grid maintenance is assuredly a competency which requires a high degree of effort and expertise, but it is unrelated to the competency of writing robust tests for your application. Cloud-based real device solutions like Sauce Labs enable you to offload that competency to allow your team to focus on tightening the feedback loops which are essential to a rapid release cycle.

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 deliver a flawless digital experience. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP and Adams Street Partners. For more information, please visit saucelabs.com.

SAUCELABS

SAUCE LABS INC. - HQ
116 NEW MONTGOMERY STREET, 3RD FL
SAN FRANCISCO, CA 94105 USA

SAUCE LABS EUROPE GMBH
NEUENDORFSTR. 18B
16761 HENNIGSDORF GERMANY

SAUCE LABS INC. - CANADA
134 ABBOTT ST #501
VANCOUVER, BC V6B 2K4 CANADA