**What language does Appium support?**

Appium support any language that support HTTP request like Java, JavaScript with Node.js, Python, Ruby, PHP, Perl, etc.

**What are the Advantages of using Appium?**

It allows you to write tests against multiple mobile platforms using the same API.

You can write and run your tests using any language or test framework.

It is an open-source tool that you can easily contribute to.

**Explain the design concept of Appium?**

Appium is an “HTTP Server” written using Node.js platform and drives iOS and Android session using Webdriver JSON wire protocol. Hence, before initializing the Appium Server, Node.js must be pre-installed on the system

When Appium is downloaded and installed, then a server is setup on our machine that exposes a REST API

It receives connection and command request from the client and execute that command on mobile devices (Android / iOS)

It responds back with HTTP responses. Again, to execute this request, it uses the mobile test automation frameworks to drive the user interface of the apps. Framework like

Apple Instruments for iOS (Instruments are available only in Xcode 3.0 or later with OS X v10.5 and later)

Google UIAutomator for Android API level 16 or higher

Selendroid for Android API level 15 or less

**What is Appium's strongest point?**

Appium is based on Selenium which is an HTTP protocol by Google designed to automate browsers. Appium is also designed to encourage a 2-tier architecture:  the WebDriver protocol targets scalability

**What is Appium's weakest point?**

Open source software is great, however it naturally comes with some downsides which nobody is to be blamed for: unreliability

Test suites are not stable.

**What is Appium's most considerable limitation?**

**Multiple session handling.**

**What is the Appium Philosophy?**

R1. Test the same app you submit to the marketplace

R2. Write your tests in any language, using any framework

R3. Use a standard automation specification and API

R4. Build a large and thriving open-source community effort

**Explain what is Appium?**

Appium is a freely distributed open source mobile application UI testing framework.

**What are main Advantages of using Appium on Sauce Labs?**

You save the time it takes to set up the Appium server locally.

You don't have to install/configure the mobile emulators/simulators in your local environment.

You don't have to make any modifications to the source code of your application.

You can start scaling your tests instantly

**Which language should I use to write my tests?**

This is probably the best thing about Appium: you can write your tests in **any language.** Since Appium is nothing more than an HTTP

**What type of tests are suitable for Appium?**

When it comes to testing, especially web view-based apps, there are a lot of scenarios that can be tested also depending on the feature coverage you want to ensure. But if you need to test more than UX simple interactions, then Appium will become a limitation. Think about features like keyboarding. It is not so easy when complex touch/keyboard mixed scenarios are involved, the probability of a false failure is high.

**Can Appium be used for all my tests?**

There are some scenarios that can be difficult to test and some of them are so platform specific that you will need to write some suites just for Android or iOS.

**List out the Appium abilities?**

Appium abilities are

Test Web

Provides cross-platform for Native and Hybrid mobile automation

Support JSON wire protocol

It does not require recompilation of App

Support automation test on physical device as well as similar or emulator both

It has no dependency on mobile device

**List out the pre-requisite to use APPIUM?**

Pre-requisite to use APPIUM is

ANDROID SDK

JDK

TestNG

Eclipse

Selenium Server JAR

Webdriver Language Binding Library

APPIUM for Windows

APK App Info On Google Play

Js

**How active is Appium?**

Appium is available on [GitHub](https://github.com/) and there you can find all you need. The Appium team is responsible for developing many different subsystems revolving around Appium. thus I can tell you that this product is alive and very active.

**What about performance?**

Appium is not a huge application and **requires very little memory.** Its architecture is actually pretty simple and light as Appium acts like a proxy between your test machine and each platform automation toolkit. Once up and running, Appium will listen to HTTP requests from your tests; when a new session is created, a component in Appium's [Node.js](http://nodejs.org/) code called \_proxy\_ will forward these Selenium commands to active platform drivers.

**Which approach is the best? Testing on real devices or simulators/emulators?**

Both options offer different levels of testability and flexibility when testing. There are also many problems associated with each. Running test on a device is my opinion, the best solution because it offers a testing environment completely aligned with the running environment: tests run on those devices where your apps will be used once published on stores.

**Tests on emulators or simulators?**

On the other hand emulators/simulators will never disconnect from Appium. They also offer nice options like the ability of choosing the orientation or other hardware-related configurations. However your tests will run much slower (sadly, my tests ran 3 times slower) and do expect some crazy behavior from the Android emulator which sometimes shuts down unexpectedly. Another problem is that emulators tend to allocate a lot of memory.

**What platforms are supported?**

Appium currently supports **Android** and **iOS**, no support for Windows unfortunately.

**List out the limitations of using Appium?**

Appium does not support testing of Android Version lower than 4.2

Limited support for hybrid app testing. E.g., not possible to test the switching action of application from the web app to native and vice-versa

No support to run Appium Inspector on Microsoft Windows

**Do I need a server machine to run tests on Appium?**

**No!** Appium promotes a 2-tier architecture where a test machine connects to a test server running Appium and automating the whole thing. However this configuration is not mandatory, you can have Appium running on the same machine where your test runs. Instead of connecting to a remote host, your test will connect to Appium using the loopback address.

**How can I test Android tablets?**

The best way to test on different Android emulators screen sizes is by using the different [Android Emulator Skins](https://docs.saucelabs.com/tutorials/appium/#android-emulator-skins) . For instance, if you use our [Platforms Configurator](https://docs.saucelabs.com/reference/platforms-configurator/#/) you'll see the available skins for the different Android versions (e.g Google Nexus 7 HD, LG Nexus 4, Samsung Galaxy Nexus, Samsung Galaxy S3, etc). Some of these skins are tablets, for example the Google Nexus 7C is a tablet which has a very large resolution and very high density.

**How can I run manual tests for my mobile native app or mobile hybrid app?**

Sauce Labs doesn't support manual tests for mobile native app or mobile hybrid app tests.

**What test frameworks are supported by Appium?**

Appium does not support test frameworks because there is no need to support them! You can use Appium with **all test frameworks** you want. [NUnit](http://www.nunit.org/) and[.NET Unit Test Framework](http://msdn.microsoft.com/en-us/library/dd264975.aspx) are just a few examples; you will write your tests using one of the drivers for Appium; thus your tests will interface with Appium just in terms of an external dependency. Use whatever test framework you want!

**How much time does it take to write a test in Appium?**

Of course it depends by the test. If your test simply runs a scenario, it will take as many commands as the number of interactions needed to be performed (thus very few lines). If you are trying to exchange data, then your test will take more time for sure and the test will also become difficult to read.

**Explain the design concept of Appium?**

Appium is an “HTTP Server” written using Node.js platform and drives iOS and Android session using Webdriver JSON wire protocol. Hence, before initializing the Appium Server, Node.js must be pre-installed on the system

When Appium is downloaded and installed, then a server is setup on our machine that exposes a REST API

It receives connection and command request from the client and execute that command on mobile devices (Android / iOS)

It responds back with HTTP responses. Again, to execute this request, it uses the mobile test automation frameworks to drive the user interface of the apps. Framework like

Apple Instruments for iOS (Instruments are available only in Xcode 3.0 or later with OS X v10.5 and later)

Google UIAutomator for Android API level 16 or higher

Selendroid for Android API level 15 or less

**What language does Appium support?**

Appium support any language that support HTTP request like Java, JavaScript with Node.js, Python, Ruby, PHP, Perl, etc.

**Explain the pros and cons of Appium?**

Pros:

For programmer irrespective of the platform, he is automating ( Android or iOS) all the complexities will remain under single Appium server

It opens the door to cross-platform mobile testing which means the same test would work on multiple platforms

Appium does not require extra components in your App to make it automation friendly

It can automate Hybrid, Web and Native mobile applications

Cons:

Running scripts on multiple iOS simulators at the same time is possible with Appium

It uses UIAutomator for Android Automation which supports only Android SDK platform, API 16 or higher and to support the older API’s they have used another open source library called Selendroid

**Can I interact with my apps using Javascript while I am testing with Appium?**

**Yes!** Selenium has commands to execute Javascript instructions on your app from your tests. Basically you can send a JS script from your test to your app; when the commands runs on Appium, the server will send the script to your app wrapped into an anonymous function to be executed.

**How can I exchange data between my test and the app I am testing?**

Appium, actually the WebDriver specification, is not made for exchanging data with your app, it is made to automate it. For this reason, you will probably be surprised in finding data exchange not so easy. Actually it is not impossible to exchange data with your app , however it will require you to build more layers of testability.

**How can I exchange data between my test and the app I am testing?**

Appium, actually the WebDriver specification, is not made for exchanging data with your app, it is made to automate it. For this reason, you will probably be surprised in finding data exchange not so easy. Actually it is not impossible to exchange data with your app , however it will require you to build more layers of testability.

**What are Testability layers?**

In order to make things better, as a developer, what you can do is adding testability layers to your app. The logic behind this approach is simply having some test-related objects in your app which are activated only when your tests run.

**I need to debug Appium, is it difficult?**

**No really**! Appium is a [Node.js](http://nodejs.org/) application, so it is Javascript in the essence. The code is available on [GitHub](https://github.com/) and can be downloaded in few seconds as it is small and not so complex. Depending on what you have to debug, you will probably need to go deeper in your debugging experience

**I build my apps with**[**Cordova**](http://cordova.apache.org/)**, is it supported by Appium?**

[Cordova](http://cordova.apache.org/) is a very famous system that enables you to develop webview-based apps for all platforms in short time. Appium does not explicitely say that Cordova is supported, even though they do it implicitely as some examples using apps built with Cordova are provided on Appium's website. So the answer is that **Cordova should not be a problem.**

**Explain what is APPIUM INSPECTOR?**

Similar to Selenium IDE record and Playback tool, Appium has an “Inspector” to record and playback.  It records and plays native application behavior by inspecting DOM and generates the test scripts in any desired language.  However, Appium Inspector does not support Windows and use UIAutomator viewer in its option.

**I want to run my tests in a multithreaded environment, any problems with that?**

Yes! You need some special care when using Appium in a multithreaded environment. The problem does not really rely on the fact of using threads in your tests: you can use them but you must ensure that no more than one test runs at the same time against the same Appium server. As I mentioned, Appium does not support multiple sessions, and unless you implemented an additional layer on top of it to handle this case, some tests might

Driver Client: Appium drives mobile applications as though it were a user. Using a client library you write your Appium tests which wrap your test steps and sends to the Appium server over HTTP.

Appium Session: You have to first initialize a session, as such Appium test takes place in the session. Once the Automation is done for one session, it can be ended and wait for another session

Desired Capabilities: To initialize an Appium session you need to define certain parameters known as “desired capabilities” like PlatformName, PlatformVersion, Device Name and so on. It specifies the kind of automation one requires from the Appium server.

Driver Commands: You can write your test steps using a large and expressive vocabulary of commands

**How can I run Android tests without Appium?**

For older versions of Android Appium might not be supported. For instance, Appium is only supported in Android versions 4.4 or later for [Mobile Web Application](https://docs.saucelabs.com/tutorials/appium/#mobile-web-application) tests, and Android versions 2.3, 4.0 and later for [Mobile Native Application](https://docs.saucelabs.com/tutorials/appium/#mobile-native-application) and [Mobile Hybrid Application](https://docs.saucelabs.com/tutorials/appium/#mobile-hybrid-application) tests.

For those versions in which Appium is not supported you can request an emulator driven by Webdriver + Selendroid. All you need to do is use our [Platforms Configurator](https://docs.saucelabs.com/reference/platforms-configurator/#/) and select Selenium for the API instead of Appium.

In the Sauce Labs test you will notice that the top of the emulator says "AndroidDriverWebview App". In addition, you will notice that you will get a "Selenium Log" tab which has the output of the Selendroid driver.

With an emulator driven by Webdriver + Selendroid you will be able to test[Mobile Web Application](https://docs.saucelabs.com/tutorials/appium/#mobile-web-application) only. You should be able to select any Android emulator version from 4.0 to the latest version and any Android emulator skin (e.g "deviceName":"Samsung Galaxy Tab 3 Emulator").

For older versions of iOS Appium might not be supported. For instance, Appium is supported in iOS versions 6.1 and later. For earlier versions of iOS the tool or driver used to drive your mobile applications automated test is called iWebdriver.

To obtain a simulator driven by iWebdriver use our [Platforms Configurator](https://docs.saucelabs.com/reference/platforms-configurator/#/) and select Selenium for the API instead of Appium. With an emulator driven by iWebdriver you will be able to test [Mobile Web Application](https://docs.saucelabs.com/tutorials/appium/#mobile-web-application) only. In addition, in the Sauce Labs test you will notice a "Selenium Log" tab which has the output of iWebdriver.

**What are the basic commands that I can use in the**[**Selenium**](https://code.google.com/p/selenium/) **protocol?**

Google's [Selenium](https://code.google.com/p/selenium/) provides a collection of commands to automate your app. With those commands you can basically do the following:

Locate web elements in your webview-based app's pages by using their ids or class names.

Raise events on located elements like Click().

Type inside textboxes.

Get or set located element's attributes.

Execute some Javascript code.

Change the context in order to test the native part of your app, or the webview. If your app uses more webviews, you can switch the context to the webview you desire. If your webview has frames or iframes inside, you can change context to one of them.

Detect alert boxes and dismiss or accept them.