**1)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.

2) Do I need Appium?

The answer to such a question is always: "It depends on what you need!". So the actual question becomes: "Which conditions make Appium suitable for me?". The most important assumption is that you are developing apps (pretty obvious I know). If you are developing an app for a specific platform (and have no intention of supporting others in future), Appium is not really required and this is basically the answer you are looking for. Appium becomes meaningful only when you have apps targeting more than one platform (Windows, Android or iOS to cite some). Appium becomes essential if you have a webview-based app (necessarily) targeting many platforms out there.

<http://akhilreddytechnologies.blogspot.in/2015/07/etl-testing-real-time-interview.html>

3) How difficult is it to set up a working environment?

The assumption is that Appium comes with a not-so-tiny documentation, so users are not really left alone. However it is not so straightforward to set up Appium to work on a Windows or Mac machine (did not try on Unix so far). In my experience, instead of installing the GUI-based application, it is much better to install the command-line application (which is released more often). Also beware [sudo], as Appium will surely bite you back late in time if you installed it as a [superuser] (this is probably the clearest point in the documentation)

4)What is Appium's strongest point?

Appium is based on [Selenium](https://code.google.com/p/selenium/) which is an HTTP protocol by Google designed to automate browsers. The idea is actually very nice as automating an app (especially a webview-based one) is not so different (in terms of required APIs) from automating a browser. Appium is also designed to encourage a 2-tier architecture: a machine runs the test written in one language ([csharp], [ruby], [javascript] are only a few among the many supported ones) and another one (the test server) actually executes it. Furthermore the WebDriver protocol targets scalability (because based on [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)), this makes Appium very scalable as well; remember that you will need to **write your test once,** Appium will be in charge of executing it on more platforms.

5) 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**is probably one of the most undeniable. My test suites need to run many tests every day and those tests must be stable (it means they should fail only when the product, or the test has a defect). However it took me much time to build layers on top of Appium to make my tests stable. This is especially true when it comes to synchronizing your tests with the automation: your test issues a command which translates into an automation command by Appium causing an interaction on the device; a command returns when after it has been issued, not when the interaction is actually over! For this reason you will probably find yourself adding delays to your tests (if you are doing things in the wrong/most-straightforward way) or developing synchronization layers on top of your test APIs (the good approach).

**6)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

**7)Why do the Appium clients exist?**

We have the Appium clients for 3 reasons:

1) There wasn't time to go through a full commit and release cycle for Selenium once we'd set a release date for 1.0

2) Some of the things that Appium does, and which its users really find useful, are never going to be an official part of the new mobile spec. We want a way to make these extensions available

3) There are some behaviors whose state is as yet unknown. They might make it into the spec and get deleted from the clients, or they might be in category #2

Ultimately, the only reason for the clients will be #2. And even that is actually evidence that we are conforming to the WebDriver spec (by implementing the extension strategy it recommends) rather than departing from it. The Appium clients are the easiest and cleanest way to use Appium.

**8) Explain what is Appium?**

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

**9)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.

10)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 server, a test which needs to be interfaced with Appium can simply use HTTP libraries to create HTTP sessions. You just need to know the Selenium protocol in order to compose the right commands and that's it!

However, as you can imagine, there are already some libraries doing this for the most common languages and development frameworks out there: C#, [dotnet], [java], Ruby, [python] and Javascript are just few examples; and they all are open source projects.

11) What type of tests are suitable for Appium?

When it comes to testing, especially webview-based apps, there are a lot of scenarios that can be tested also depending on the feature coverage you want to ensure. Appium is pretty handy for testing scenarios that users will go through when using your app. 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; do not misunderstand me on this: I am not saying it is impossible to do, just not so easy as you might think!

Another little nightmare with Appium is **exchanging data.** When your test needs to exchange data with your app (especially in the incoming direction), you will need to play some tricks. So always consider that sending and receiving information is not that straightforward. It is not Appium's fault, the WebDriver specification was designed for automating stuff, not exchanging data!

12) Can Appium be used for all my tests?

This is an implied question in this question. The answer is **No** (in general). As I said before Appium is not suitable for all types of tests you might want to write (this depends on the functionalities you need to cover). 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 for example. Remember that you can always get to do something no matter how hard it is, so you can test all your difficult scenarios using Appium, but always keep in mind one question: is it worth the time and the pain? Having Appium testing some scenarios leaving a few tests to other approaches is fine too! World is not black and white!

**13) 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

**14) 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

15)What is Appium's most considerable limitation?

Hand down my chin starting thinking and mumbling. If I had to provide one single thing you should be aware of about Appium before starting using it, it would surely be: **multiple session handling.**Since Appium is a server, it serves HTTP requests; you might have two different computers running a test each against the same Appium server: what happens? As for now, Appium does not support this scenario and the second test will be aborted. This is a considerable limitation, because no queuing system comes with Appium. If you need to support multiple sessions, you will need to implement this feature by yourself.

16)How active is Appium?

Appium is available on [GitHub](https://github.com/" \t "_blank) and there you can find all you need. The Appium team is responsible for developing many different subsystems revolving around Appium (like APIs for different languages), thus I can tell you that this product is alive and very active. The team is also pretty well responsive and once you open an issue you will find a reply after no more than 36 hours (this ETA comes by my personal experience). The community around Appium is also pretty large and growing every month.

17)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. In the case of Android for example, Appium will forward incoming commands to the [chromedriver] (90% of cases, Appium will not even change commands while routing them), this happens because ChromeDriver supports WebDriver and Selenium. For this reason Appium will not allocate much memory itself, you will see a lot of memory being allocated by other processes like [adb], ChromeDriver or the iOS automation toolkit (called by Appium while testing and automating).

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

This is a tough question because both options offer different levels of testability and flexibility when testing. There are also many problems associated with each. So my answer will be again: "It depends on your needs!".

Running test on a device is, always in 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. However devices must be connected to the Appium server via USB at least, and this is not always a very nice thing. ADB has a known issue for which a device disconnects after a while (even though it remained plugged all the time): because of this your tests might fail after a while and Appium will report that a device could not be found! I had to write a component which resets ADB after some time so that devices will not disconnect.

19)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.

20)What platforms are supported?

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

21) 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.

22**) 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

**23)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.

**24)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.

**25)What type of keyboard and buttons do the Android emulators have?**

Android Emulators have software buttons and a hardware keyboard. In a regular Android emulator the device buttons are software buttons displayed on the right size of the emulator. For the Android emulators with different skins (e.g Google Nexus 7 HD, LG Nexus 4, Samsung Galaxy Nexus, Samsung Galaxy S3, etc) the device buttons are also software buttons that are overplayed on top of the skin. For instance, if you hover the mouse around the edges of any of our Android emulators with an specified skin, a hover icon will appear and you should be able to find whatever buttons actually exist on the device that the skinned emulator is trying to emulate (e.g power button along the top, volume buttons along the edge, back/home buttons right below the screen, etc).

**26) Explain how to find DOM element or xPath in a mobile application?**

To find the DOM element use “UIAutomateviewer” to find DOM element for Android application.

**27) 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

**28) What language does Appium support?**

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

**29) 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

30) I already have platform-specific tests for my app, what should I do to migrate to Appium?

Unfortunately there is not a magic formula to translate your tests into Selenium tests. If you developed a test framework on different layers and observed good programming principles, you should be able to act on some components in your tests in order to migrate your suites to Appium. Your current tests are going to be easy to migrate if they are already using an automation framework or something close to a command-based interaction. Truth being told, you will probably need to write your tests from the beginning, what you can do is actually reusing your existing components.

31) 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.

32) Any tips or tricks to speed up my test writing activity or my migration process?

Here is **one piece of advice.** Since your tests will mostly consist in automation tasks (if this condition is not met, you might want to reconsider using Appium), make interactions reusable! Do not write the same sub-scenarios twice in your tests, make a diagram of what your scenarios are and split them in sub activities; you will get a graph where some nodes are reachable from more than one node. So make those tasks parametric and call them in your tests! This will make your test writing experience better even when you need to migrate from existing tests (hopefully you already did this activity for your existing suites).

33) 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 and.NET Unit Test Framework 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!

34)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.

35)Is it Returning the values?

However your Javascript interaction can get more advanced as your script can return a value which will be delivered to your test when the HTTP response is sent back by Appium once your Javascript has finished running. However this scenario comes with a limitation: your Javascript can send back only primitive types (integers, strings), not complex objects. The limitation can be overtaken by passing objects as JSON strings or by modifying Appium's or Selenium's code to support specific objects.

36)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.

37)What data exchange is?

When I say "data exchange" I am not referring to scenarios like getting or setting the value of a textbox. I am also not referring to getting or setting the value of an element's attribute. All these things are easy to achieve in Appium as Selenium provides commands just for those. By "data exchange" I mean exchanging information hosted by complex objects stored in different parts of your webview-based app like the window object. Consider when you dispatch and capture events, your app can possibly do many things and the ways data flows can be handled are many. Some objects might also have a state and the state machine behind some scenarios in your app can be large and articulated. For all these reasons you might experience problems when testing.

38)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 learned about this strategy from one of my colleagues *Lukasz* and such a technique can be really powerful. Enable your testability layers when testing in order to make data exchange easy.

39)Is it Exchanging data through Javascript?

Selenium provides commands do execute Javascript on the app, it is also possible to execute functions and have them return data (only basic types). If you exchange JSON strings it should be fine as JSON.stringify(str) will turn your[JSON](http://json.org/) string into an object on the app side, while on the test side (depending on the language you are using), you can rely on hundreds of libraries to parse the string you receive.

40) What are the most difficult scenarios to test with Appium?

As I already mentioned in [Question 6](http://blogs.technet.com/b/antino/archive/2014/11/30/q-amp-a-on-appium-should-i-use-it.aspx#q6) and [Question 16](http://blogs.technet.com/b/antino/archive/2014/11/30/q-amp-a-on-appium-should-i-use-it.aspx#q16), Appium is not suitable for all types of tests. There is a particular scenario that will make your tests more difficult to write: **data exchange**. I already said it but I will repeat the same thing because it is very important: Appium and WebDriver are designed to automate stuff... not to exchange data with them. So what if we need to exchange information with our app during tests? Should we give up on Appium and write our tests manually for each platform? I am not saying this, but there are cases where you should consider this option (not nice I know, but if the effort of writing tests for Appium is higher than the benefits, than just throw Appium away).

Appium is very nice because it will let you write tests once for all platofrms instead of writing as many tests as the numbers of platforms you need to support. So if you need to exchange data with your app while testing it and this data flow is the same for all platforms, then you should probably keep on using Appium and find a way to write a layer on top of it to handle data. Depending on your needs this might take time, but, in my experience, it is really worth it.

41) I don't want to set up a whole infrastructure for my tests and I don't want to spend money on HW. Can Appium help me?

If you think about it, what really is required from you is writing tests. Then the fact that you must deploy an Appium server somewhere is something more. If you want to skip this part, you can rely on some web services that already deployed a whole architecture of Appium servers for your tests. Most of them are online labs and they support Selenium and Appium

42) 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/" \t "_blank) 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, however there are some key points where setting a breakpoint is always worth: the *proxy* component is worth a mention. In appium/lib/server/proxy.js you can set a breakpoint in function doProxy(req,res), that will be hit everytime commands are sent to platform-specific components to be translated into automation commands.

43) 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.** Why am I being so shy about it? Because anything can happen and it actually happened to me!

Cordova and Appium are two different projects that are growing up separately and independently, of course a mutual acknowledgement is present, but both teams do not really talk to each other when pushing features. So problems can occur (I am currently dealing with a problem concerning Cordova's new version which is causing my tests to fail).

**44) 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.

**45) What are the basic commands that I can use in the [Selenium](https://code.google.com/p/selenium/" \t "_blank)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. Be careful about this functionality, I experienced some problems.

**46) 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 fail.

**47) Mention what are the basic requirement for writing Appium tests?**

For writing Appium tests you require,

* **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.

**48)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 "AndroidDriver Webview 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").

**49)How can I run iOS tests without Appium?**

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.

**50)What mobile web browsers can I automate in the Android emulator?**

Currently the only browser that can be automated in our Android emulators is the stock browser (i.e Browser). The Android stock browser is an Android flavor of 'chromium' which presumably implies that its behavior is closer to that of Google Chrome.