Introduction to AWS Device Farm in 10 min

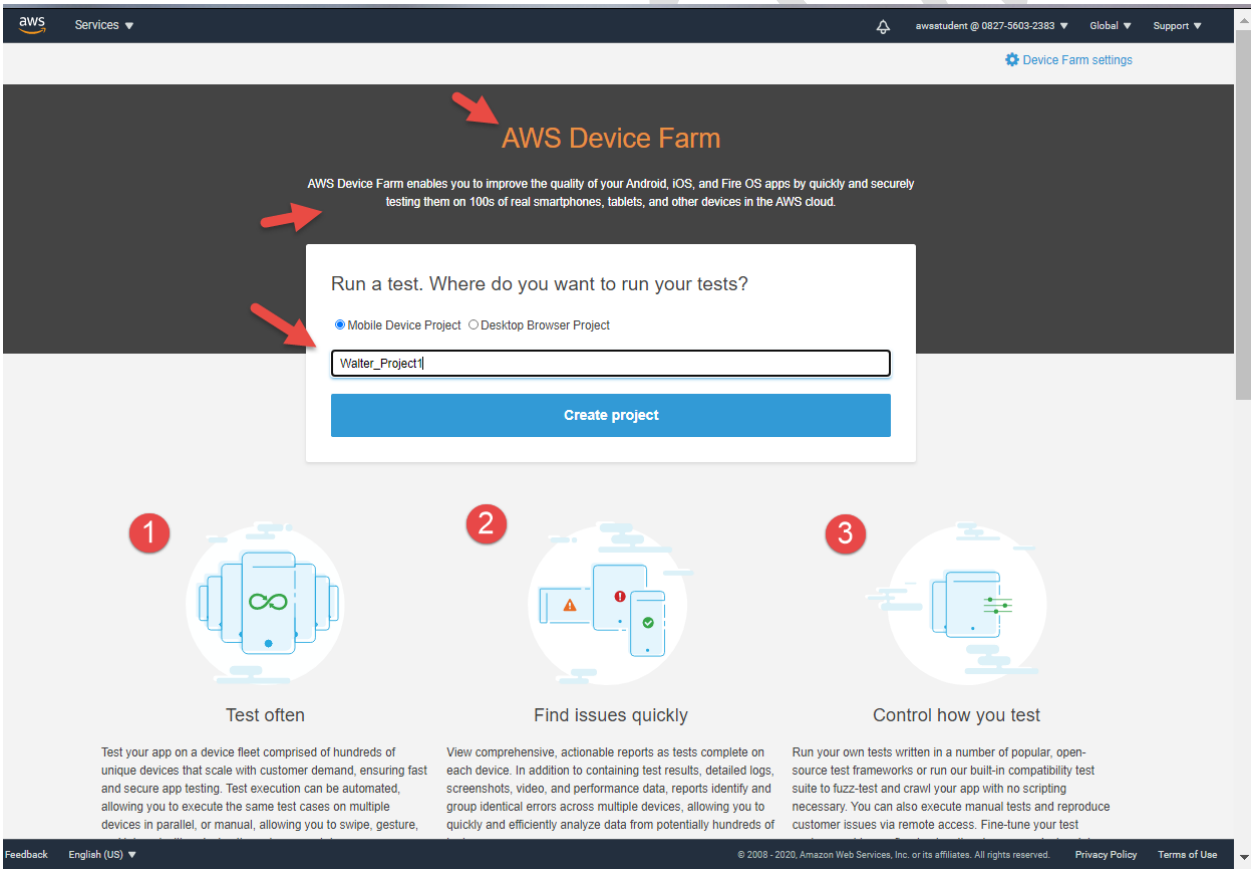
What Is AWS Device Farm?

PDF

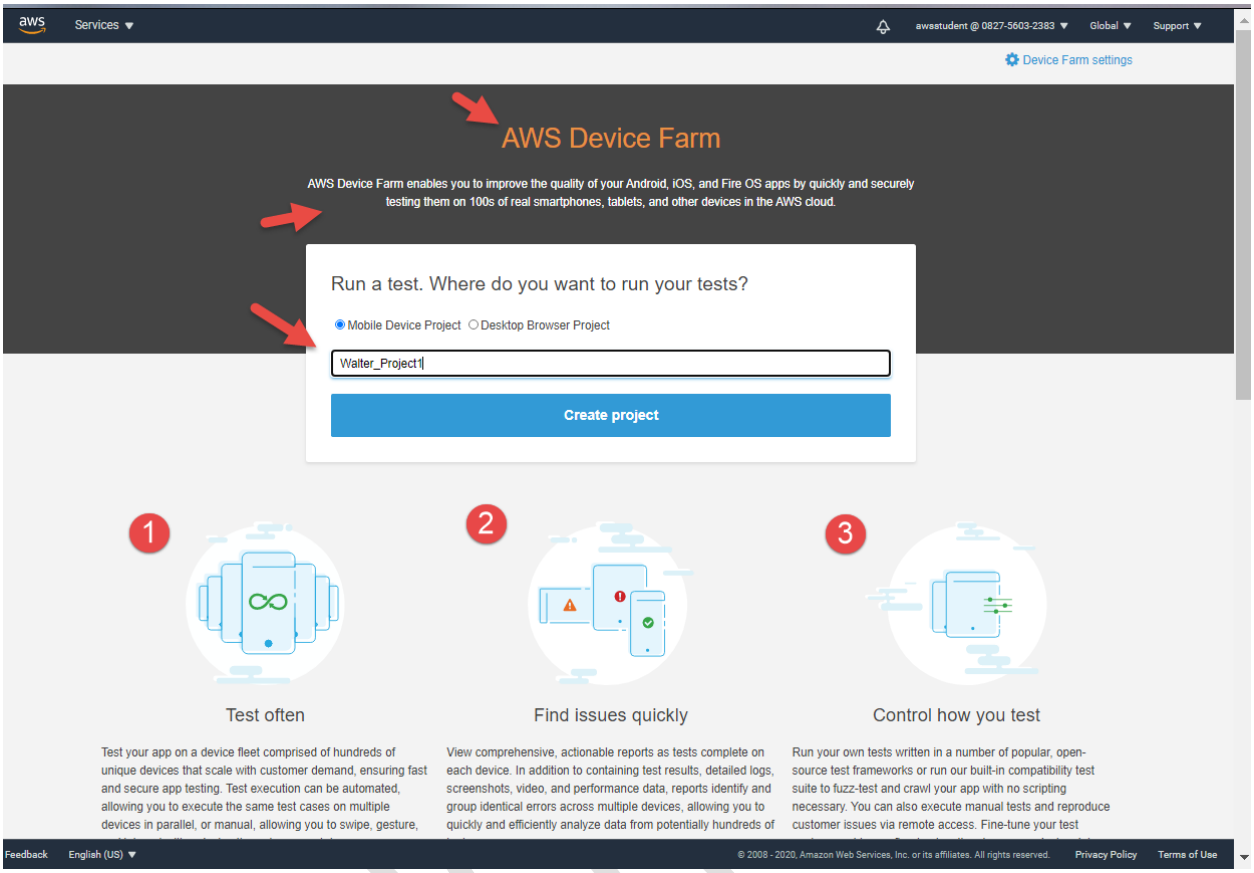
Device Farm is an app testing service that you can use to test and interact with your Android, iOS, and web apps on real, physical phones and tablets that are hosted by Amazon Web Services (AWS).

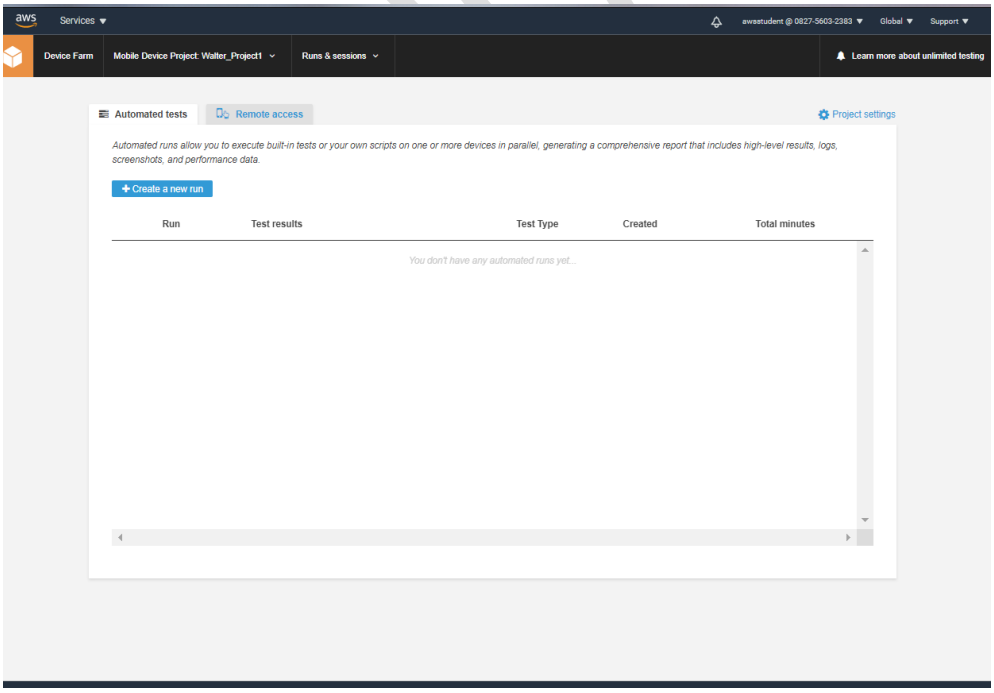
There are two main ways to use Device Farm:

- Automated testing of apps using a variety of testing frameworks.
- Remote access of devices onto which you can load, run, and interact with apps in real time.

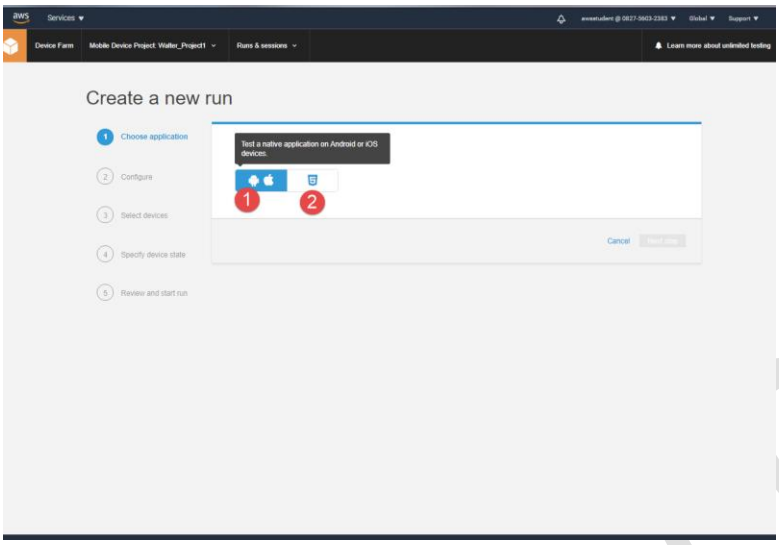


Step1: create a project and new test run:





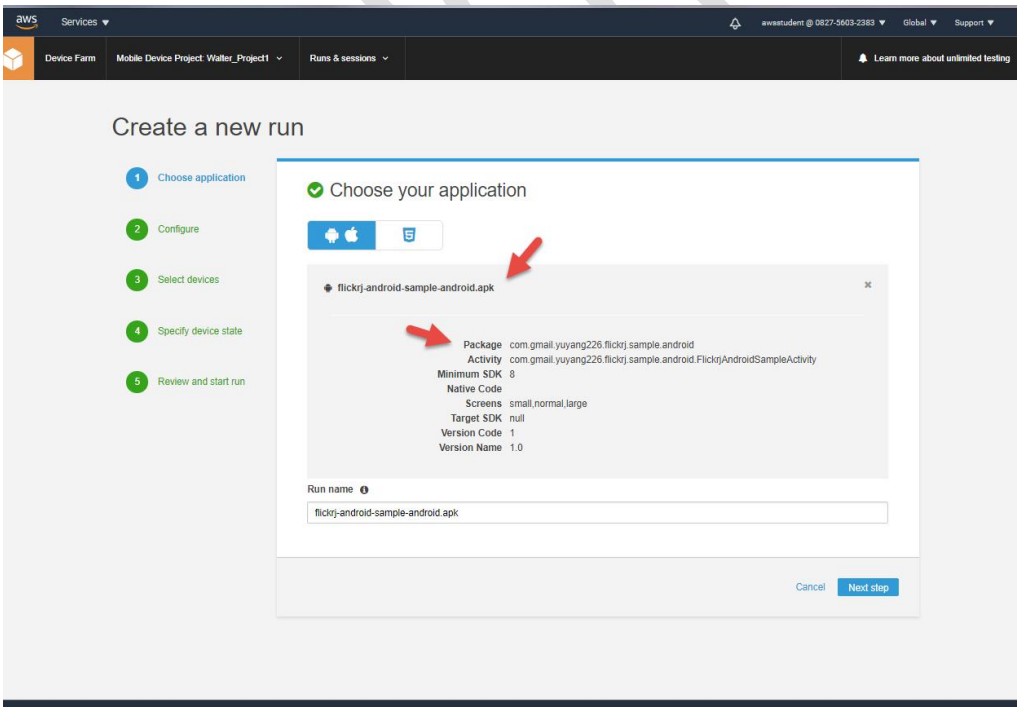
Select Android/Apple App or Mobile web app:



Configure the new run:

Upload your apk or sample app from :

[sample test app at https://code.google.com/archive/p/flickrj-android/downloads](https://code.google.com/archive/p/flickrj-android/downloads)

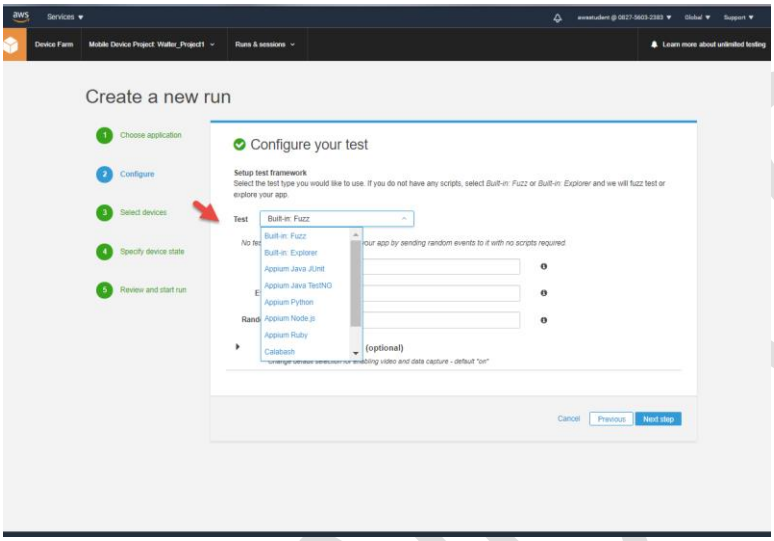


Select the test suites

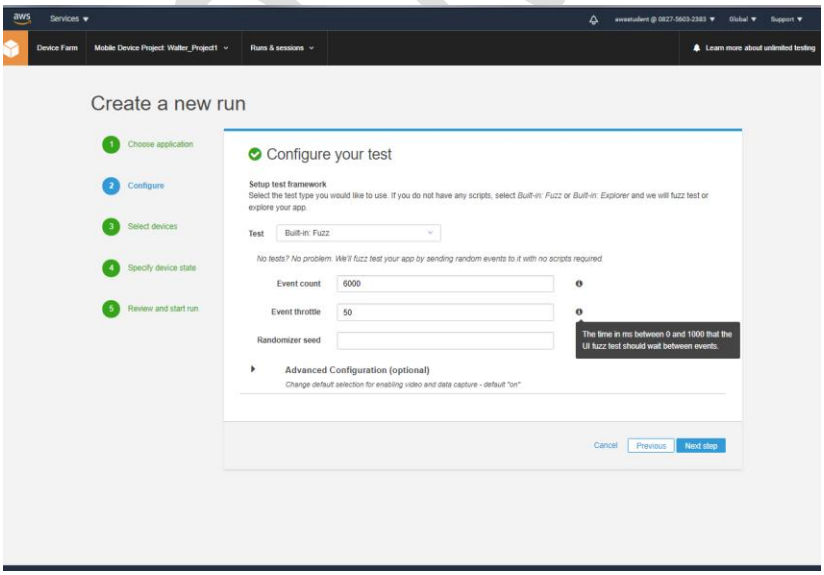
Many ready to use tests from the list:

For iOS:

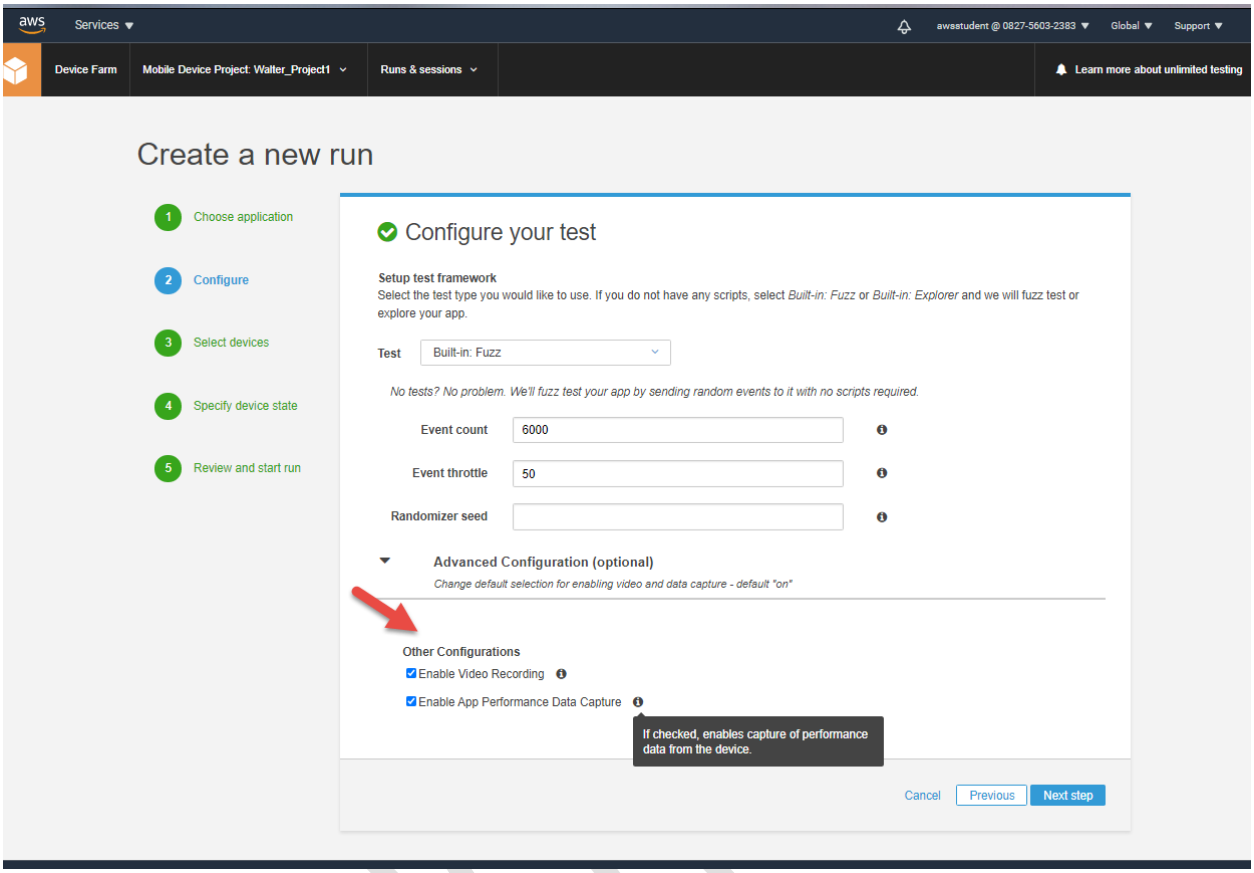
- Appium Java JUnit
- Appium Java TestNG
- Calabash
- UI Automation
- XCTest (including KIF)



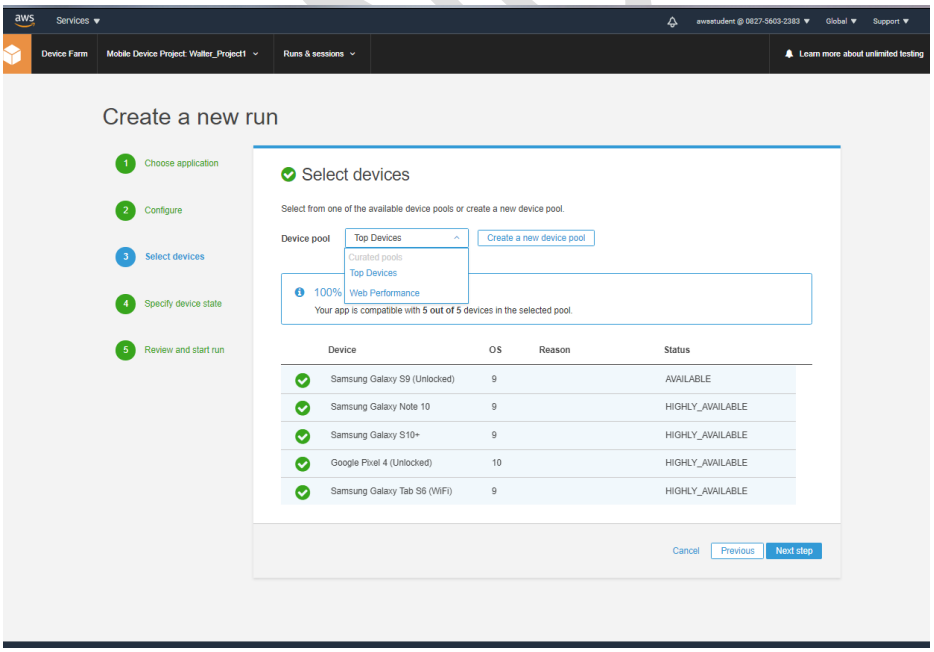
This hands on lab will demonstrate this feature. For more information on the tests implemented in Fuzz, see Built-in: Fuzz (Android and iOS).



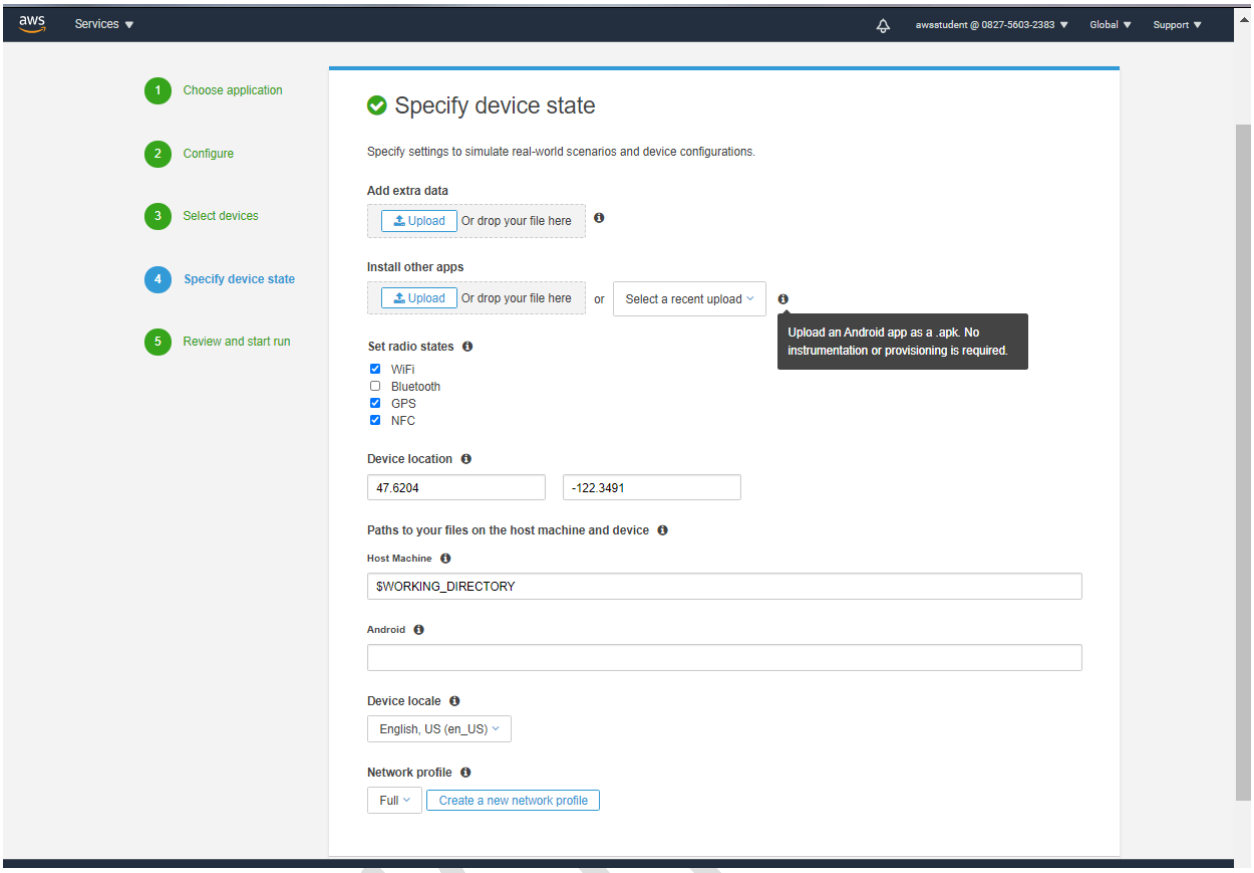
You can enable Video Recording and App Perf data capture:



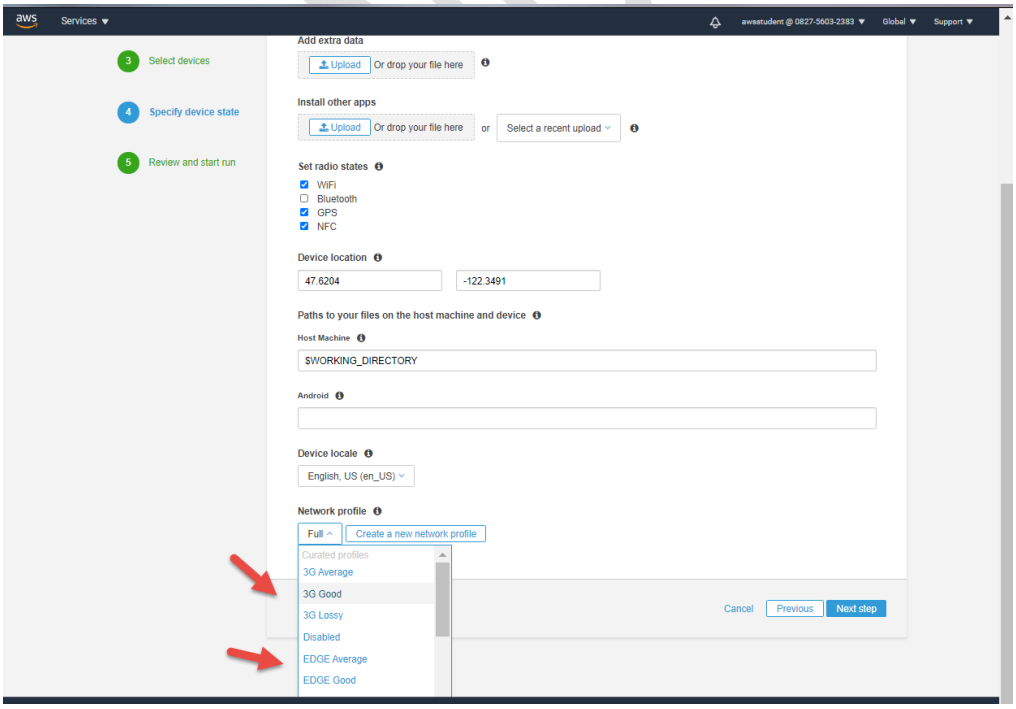
Select Devices



Specify Device state

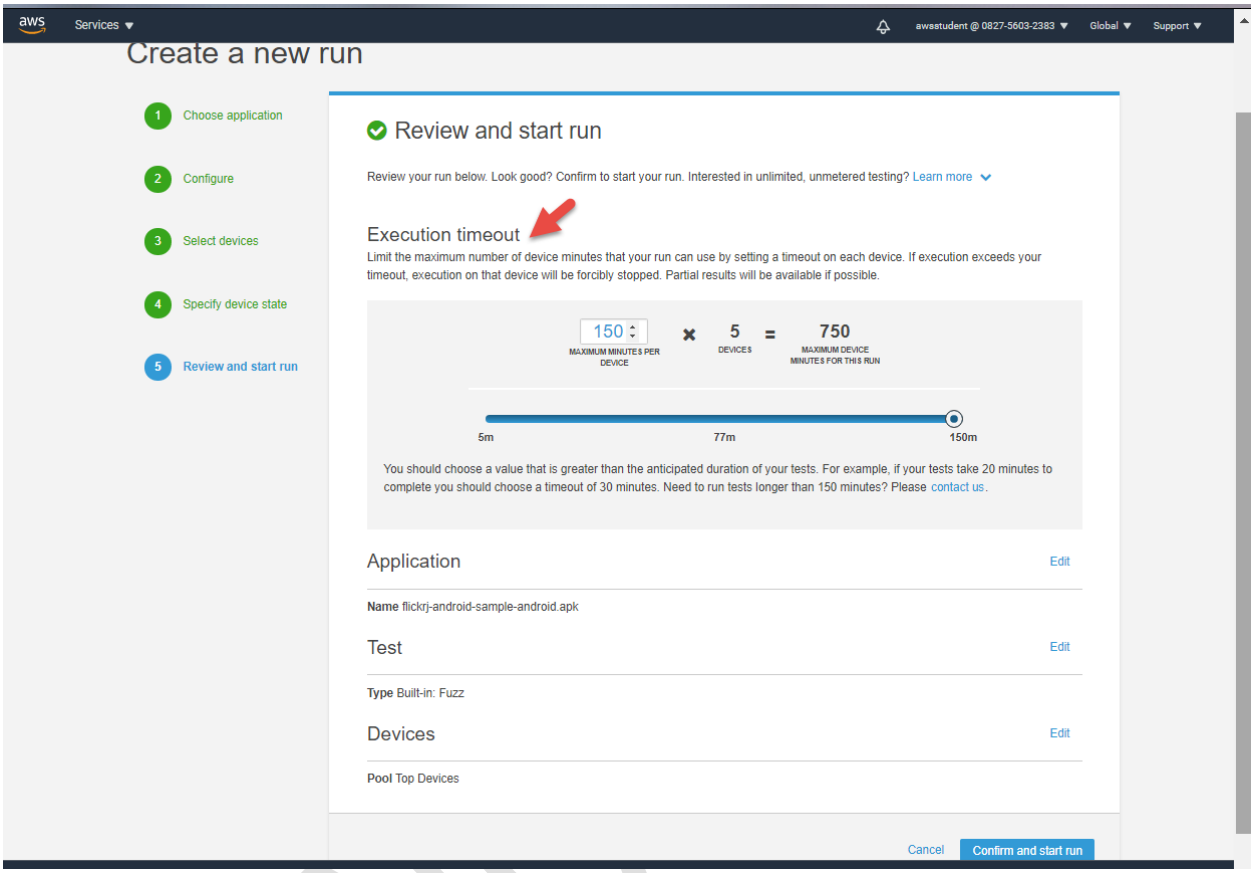


You can select Network speed profile:

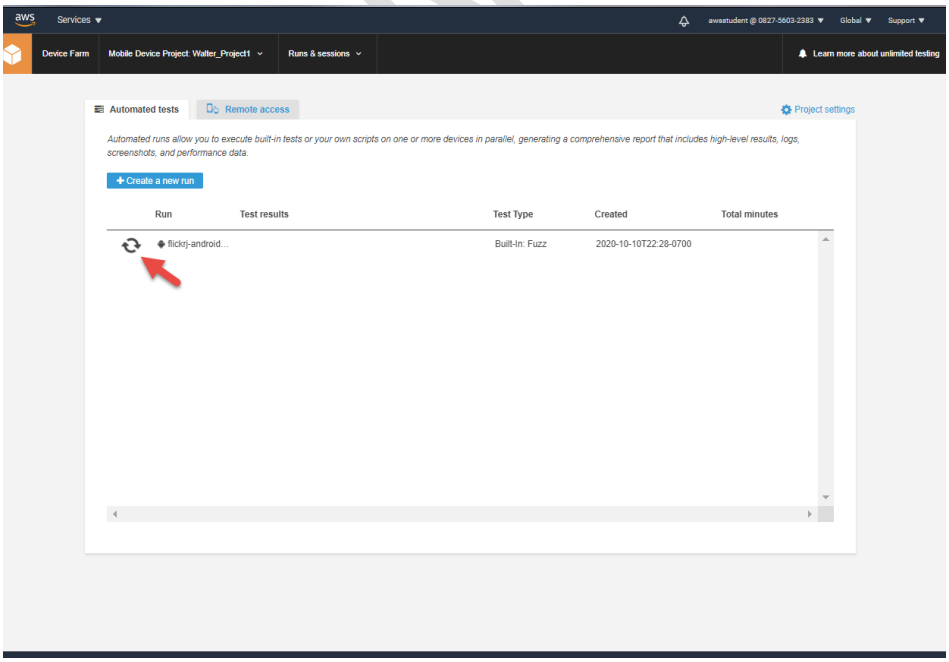


Also, can select Device Languages/Locale, e.g. Arabic, Spanish and Chinese, etc.

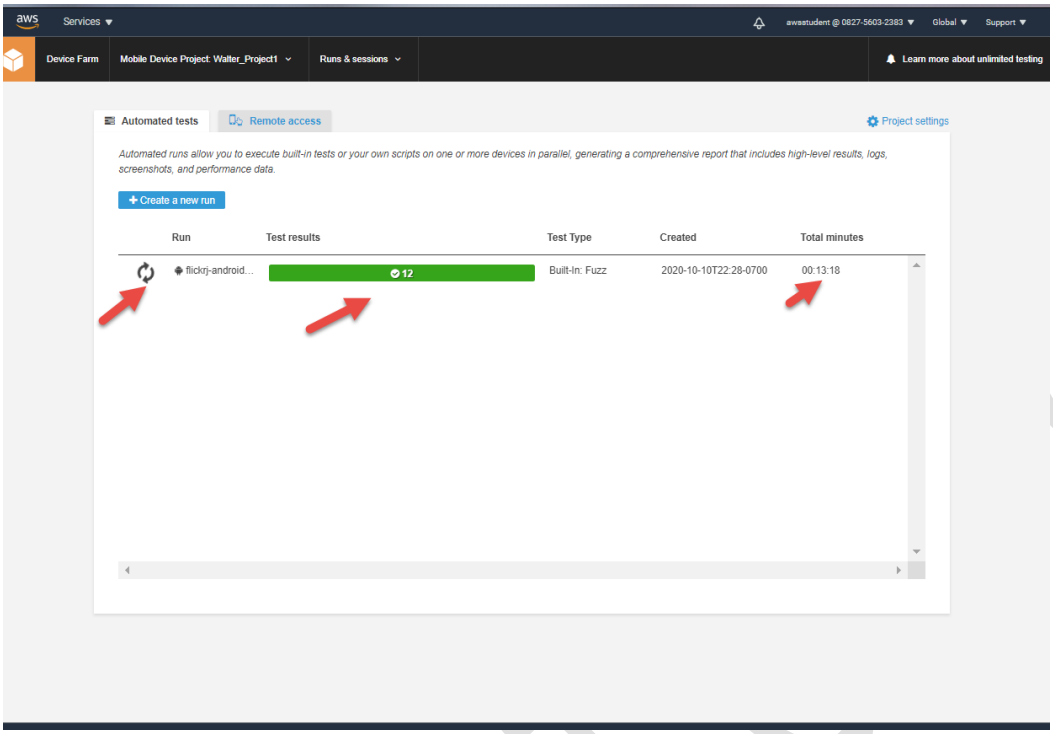
Review and start Run:



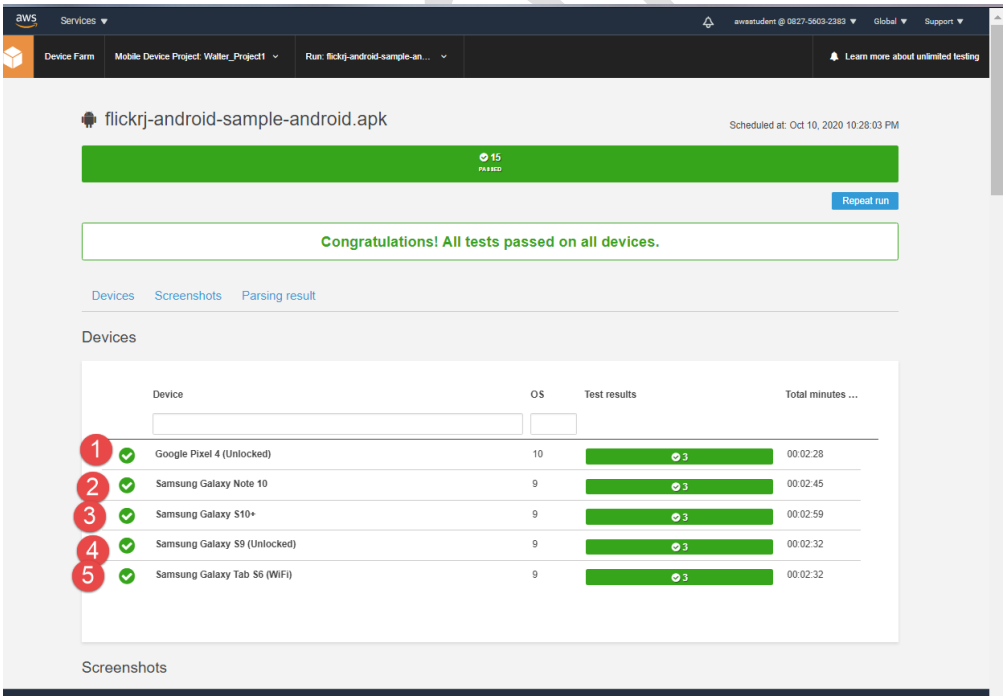
Then see the tests started:



12 out of 15 tests done:

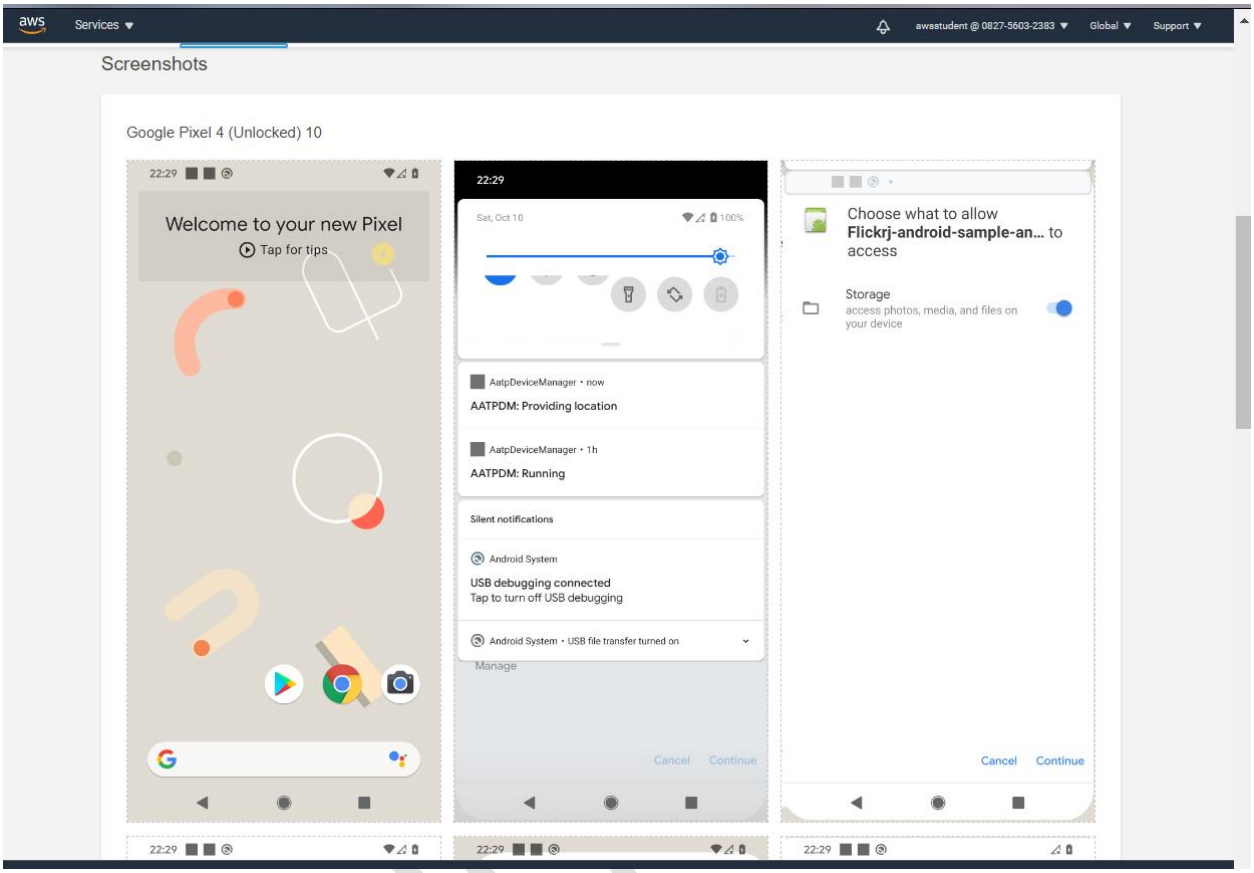


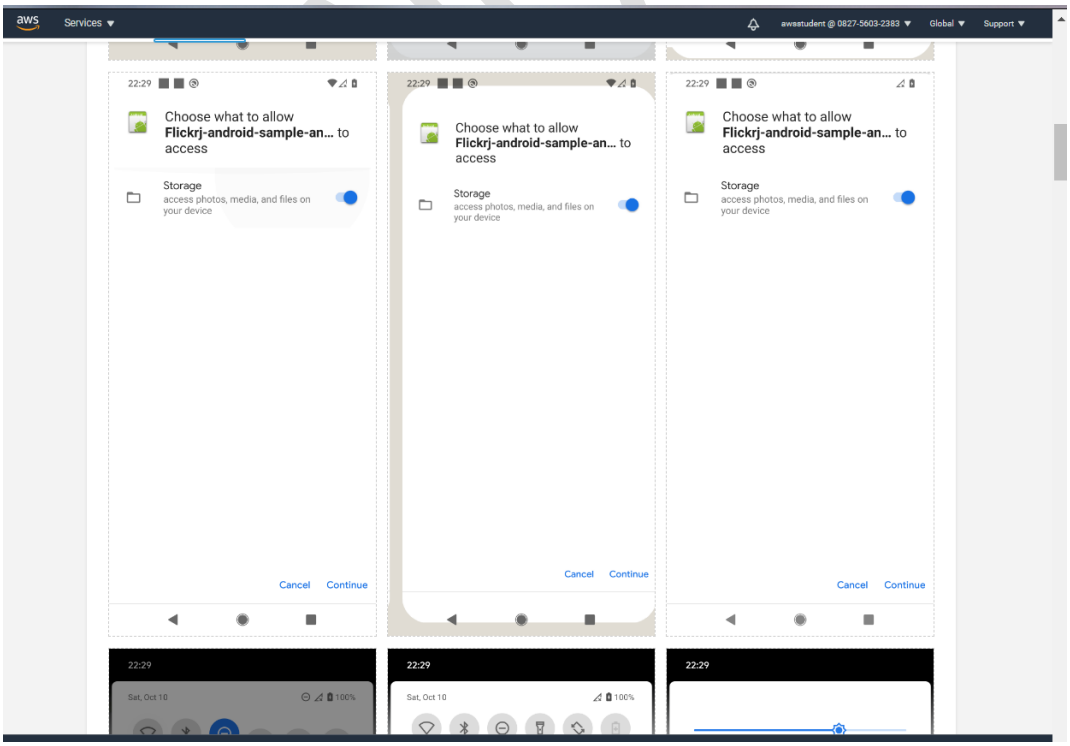
Finally all 15 tests done and about 2:45 min each:



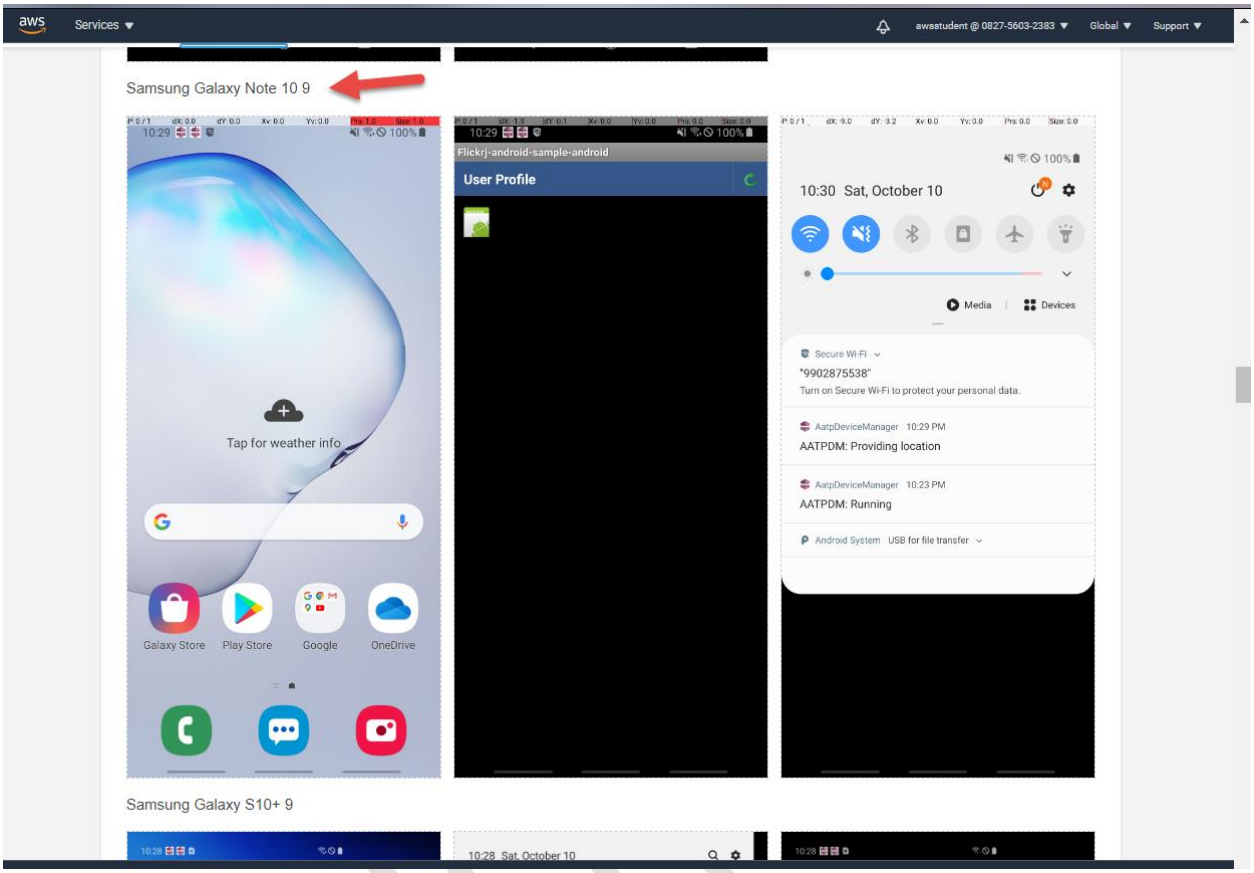
Each to review results with Screenshots:

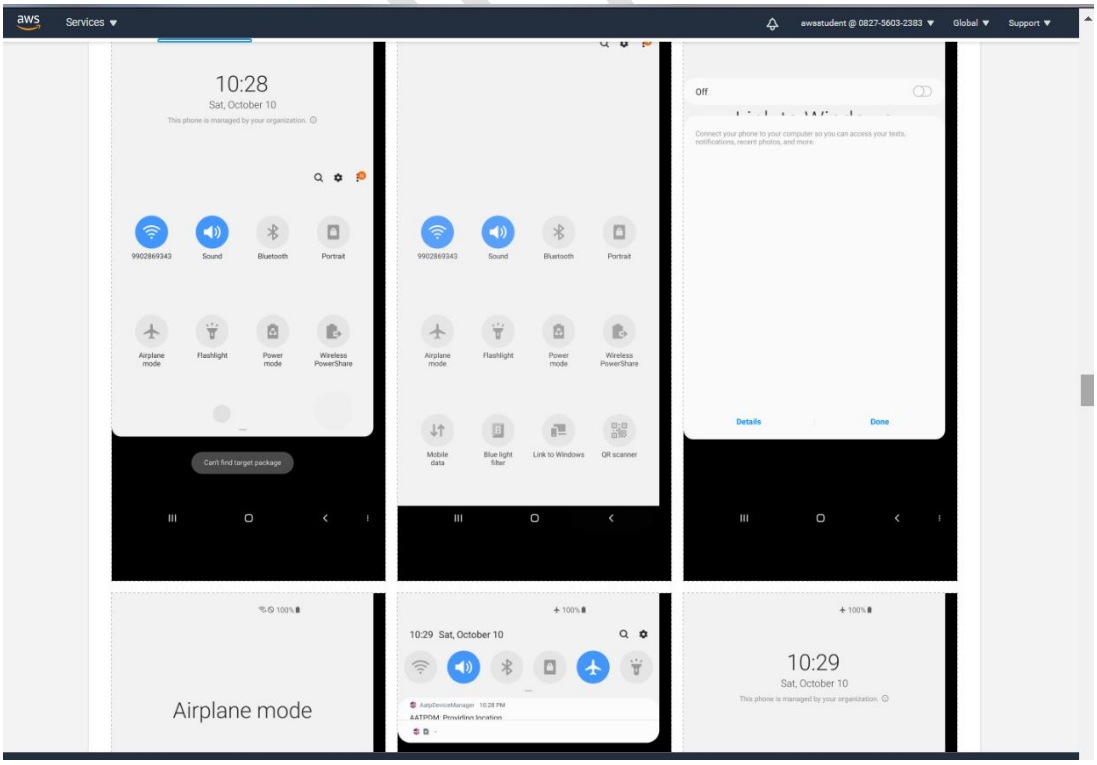
a/ first with Pixel 4 device



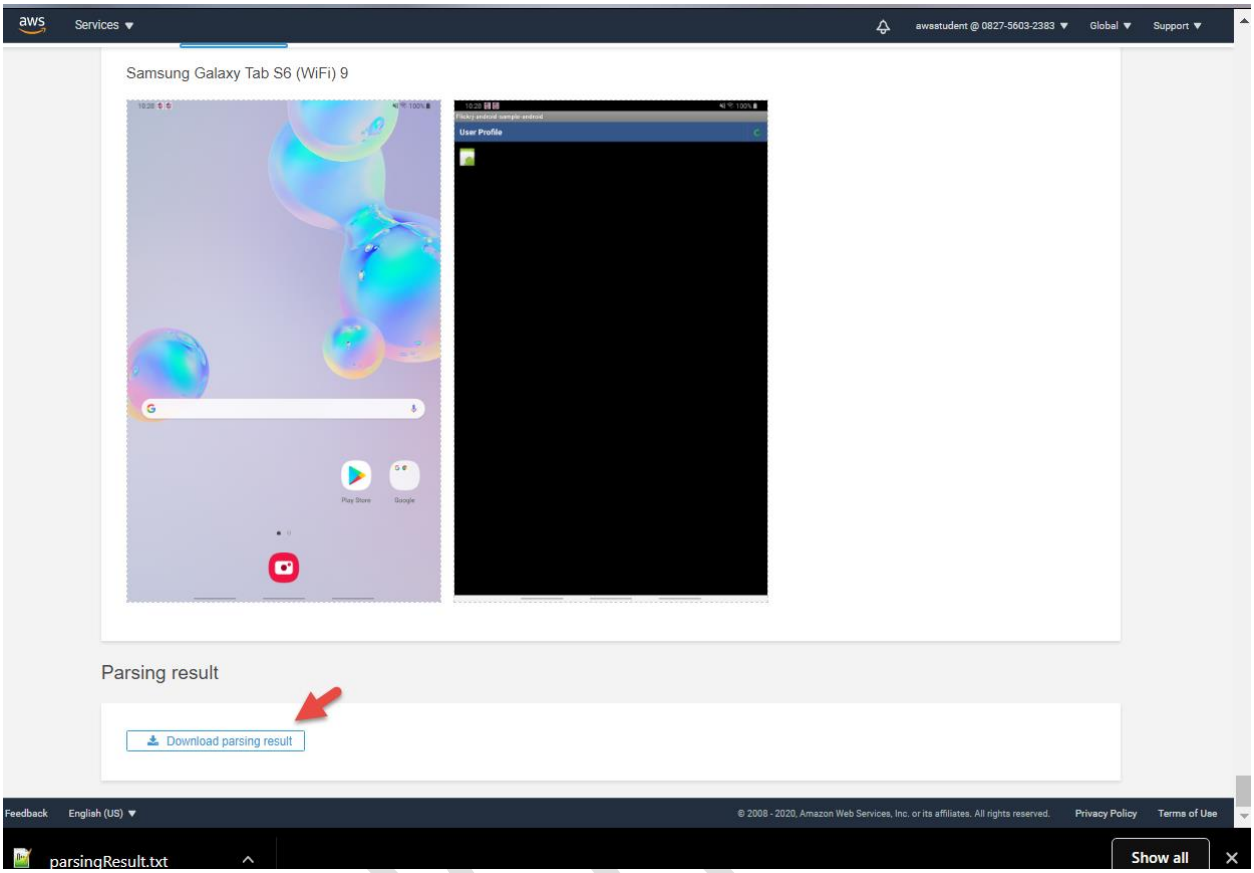


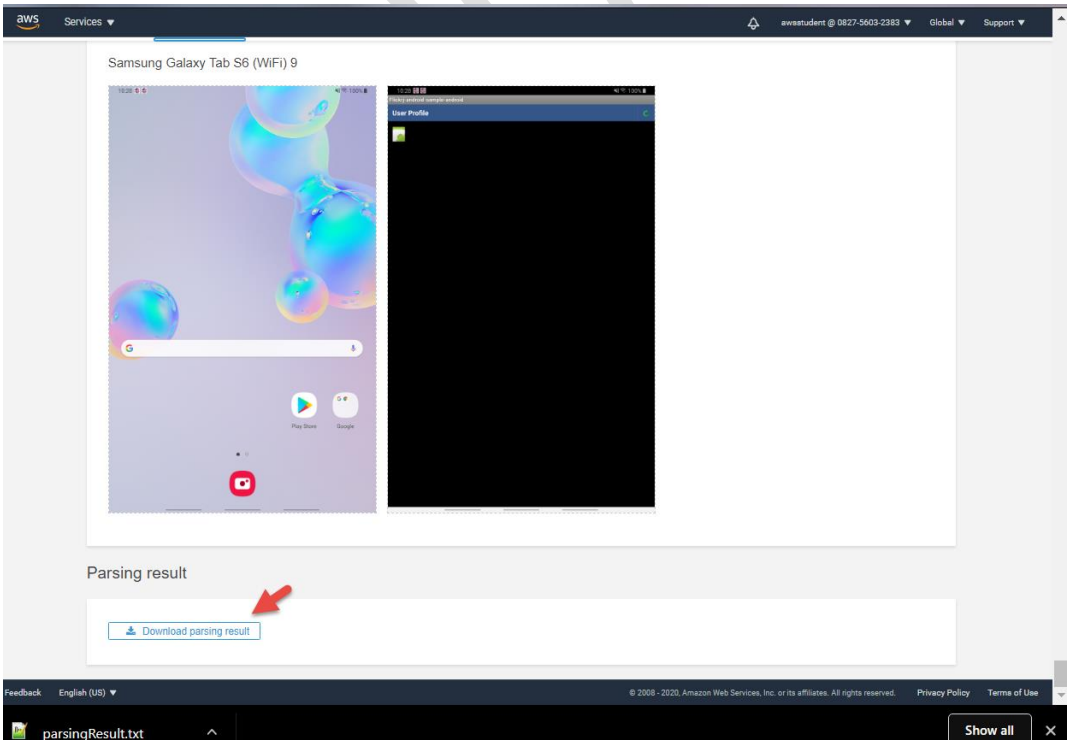
b/ second with Samsung Galaxy Note 10 device





c/ third with Samsung Galaxy Tab S6 (Wifi) device



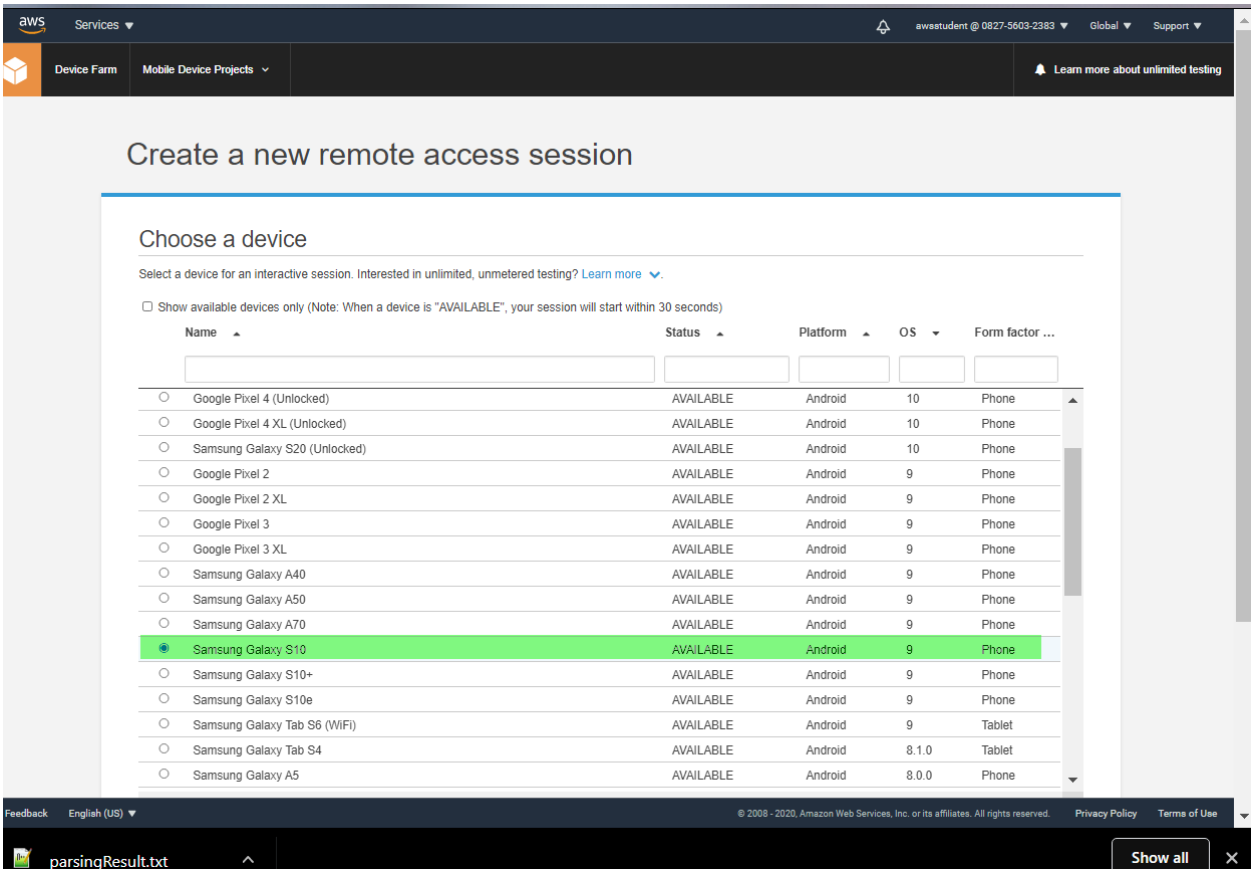


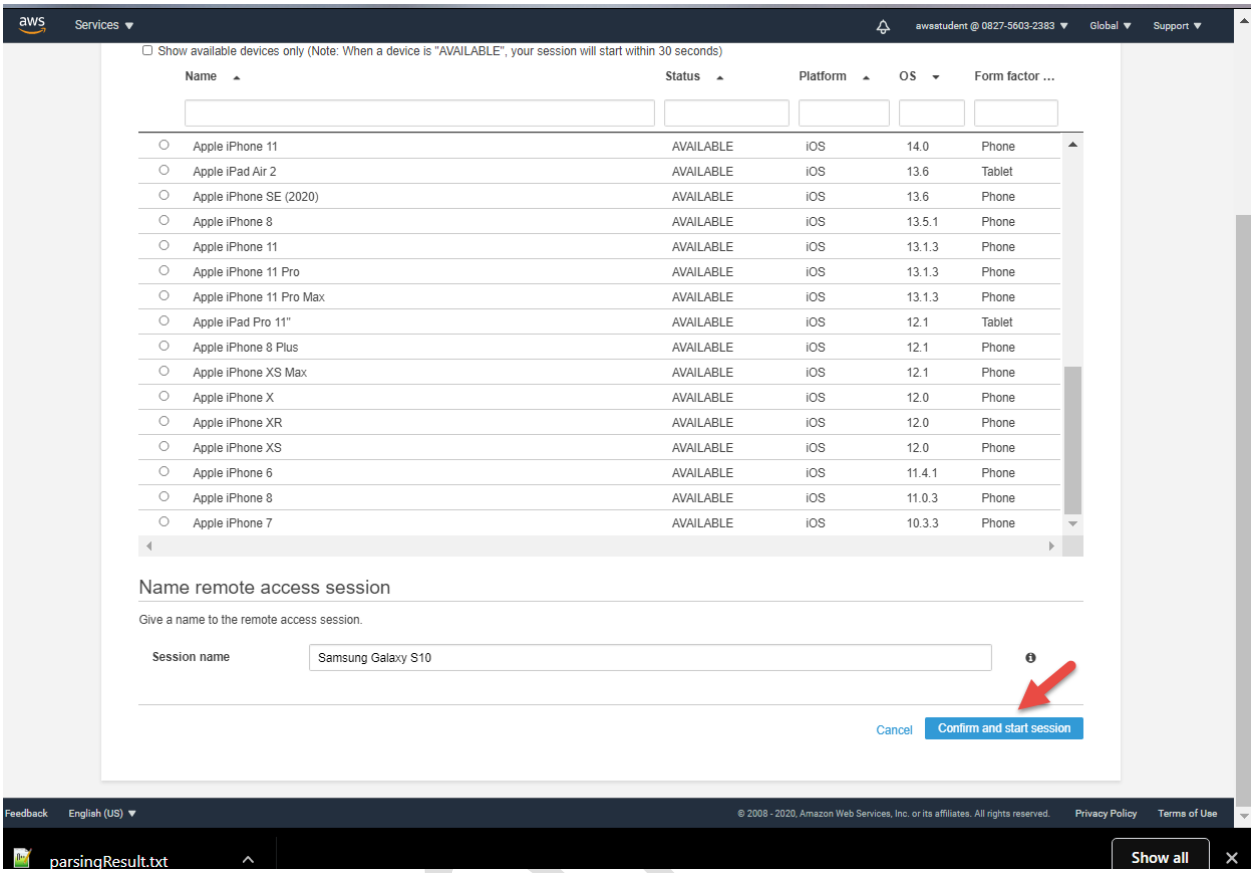
Easy to create a new remote access session

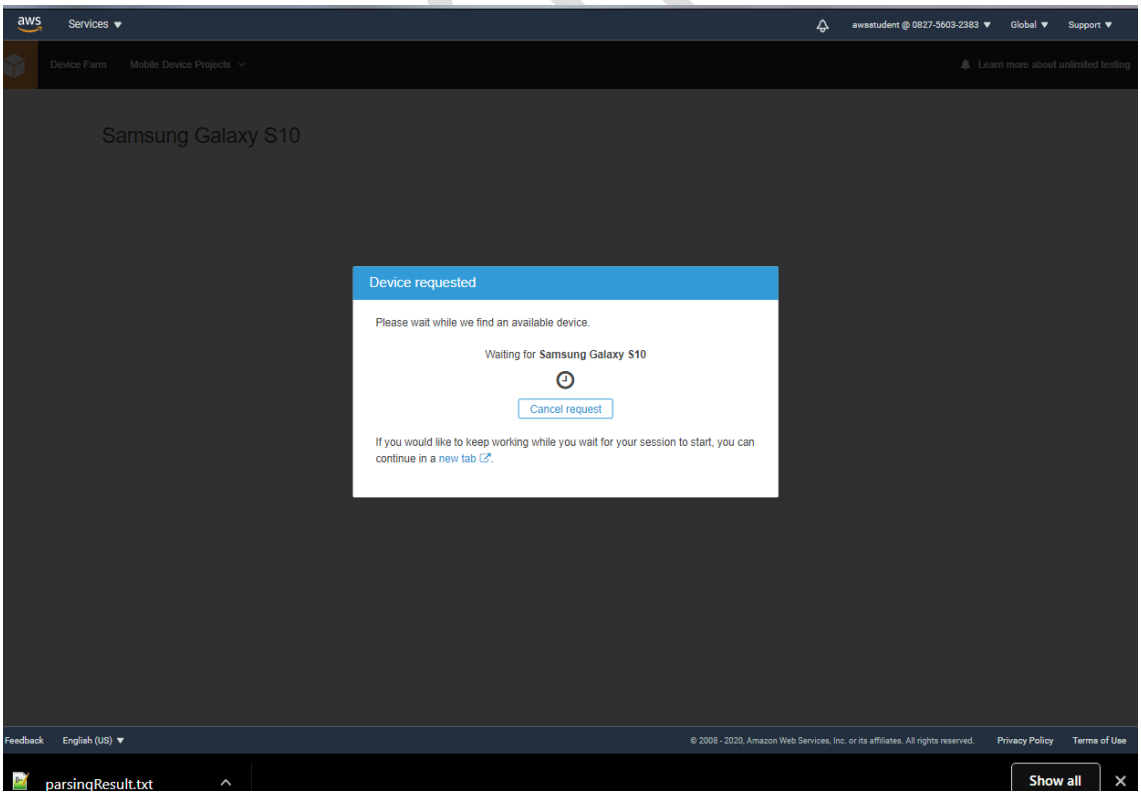
https://docs.aws.amazon.com/devicefarm/latest/developerguide/welcome.html

"Remote access allows you to swipe, gesture, and interact with a device through your web browser in real time. There are a number of situations where real-time interaction with a device is useful. For example, customer service representatives can guide customers through the use or setup of their device. They can also walk customers through the use of apps running on a specific device. You can install apps on a device running in a remote access session and then reproduce customer problems or reported bugs.

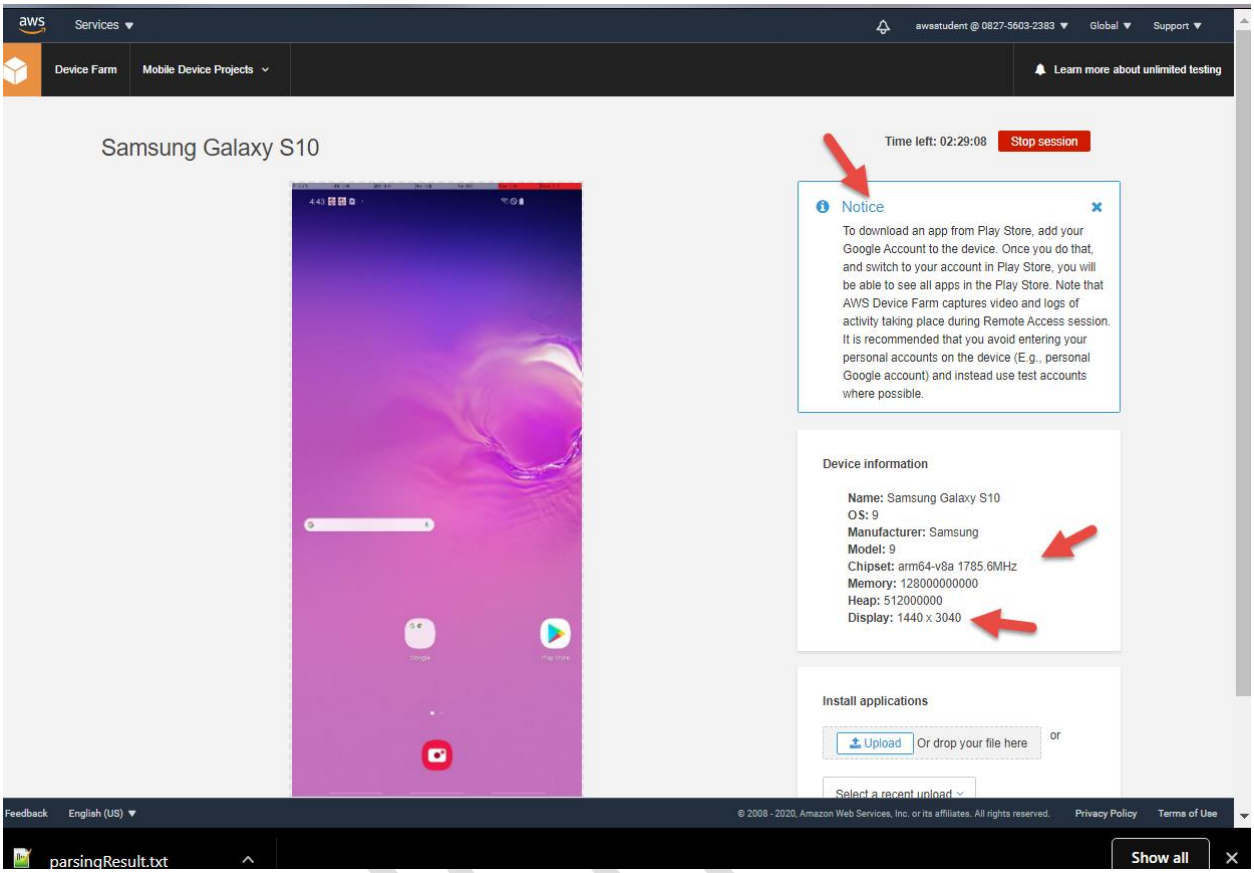
During a remote access session, Device Farm collects details about actions that take place as you interact with the device. Logs with these details and a video capture of the session are produced at the end of the session."



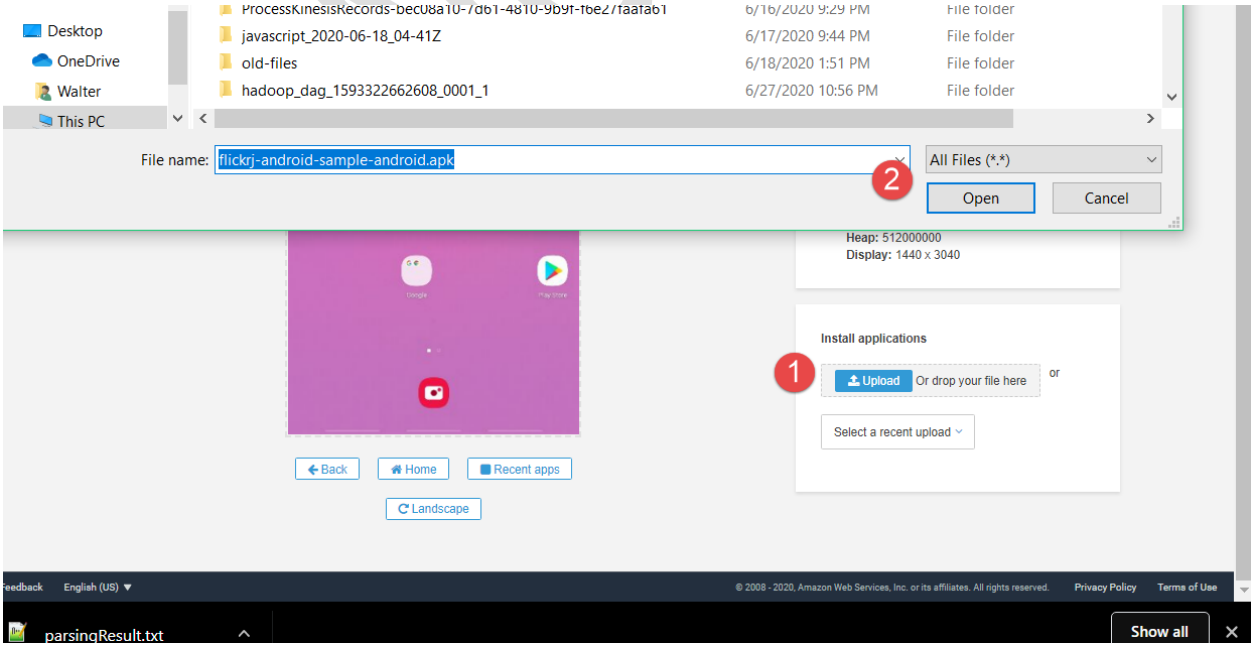


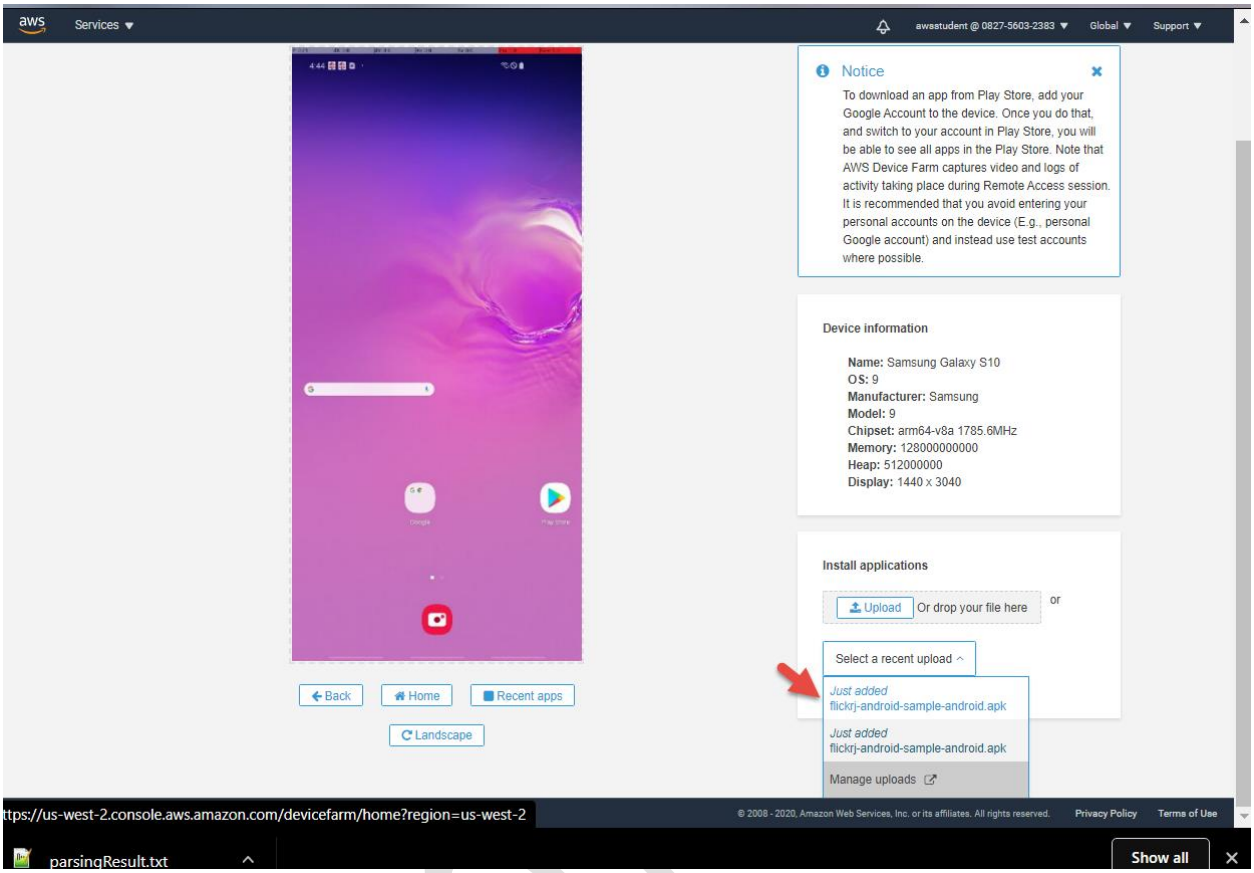


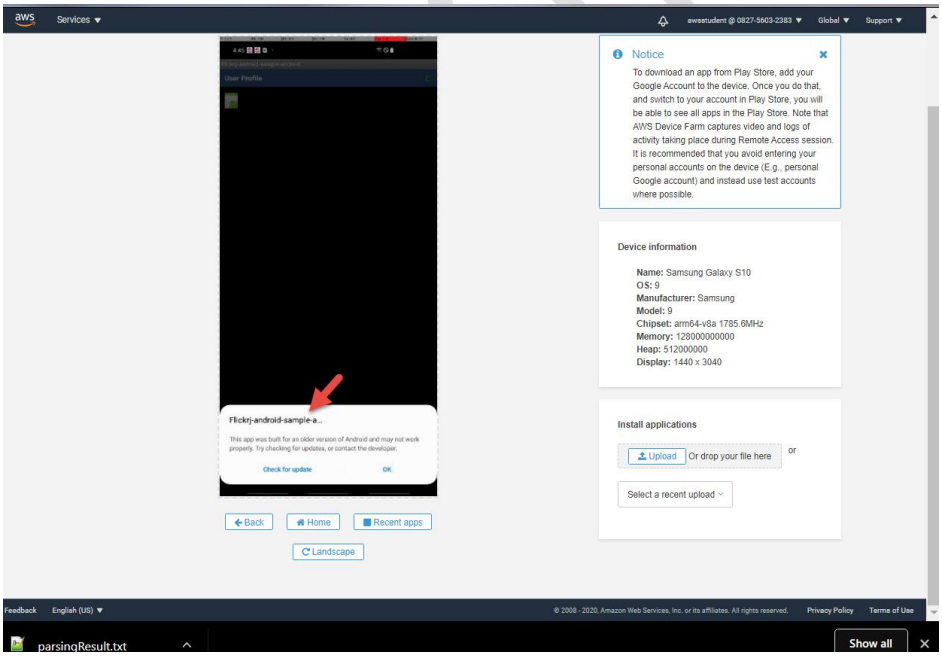
Then get the same Samsung Galaxy S10 device which can do interactively test more.



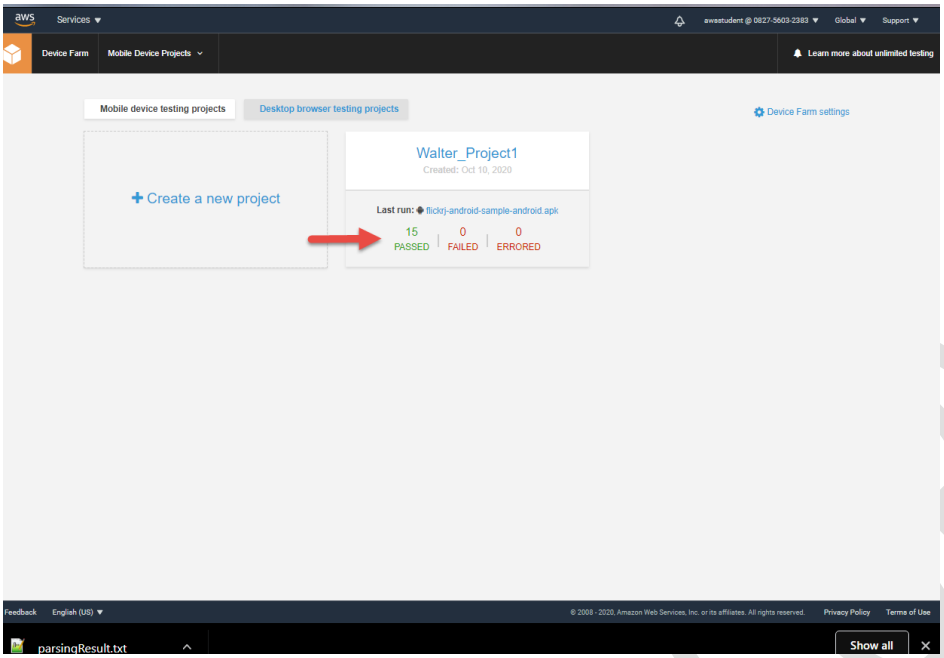
Easy to upload your APK file to test:



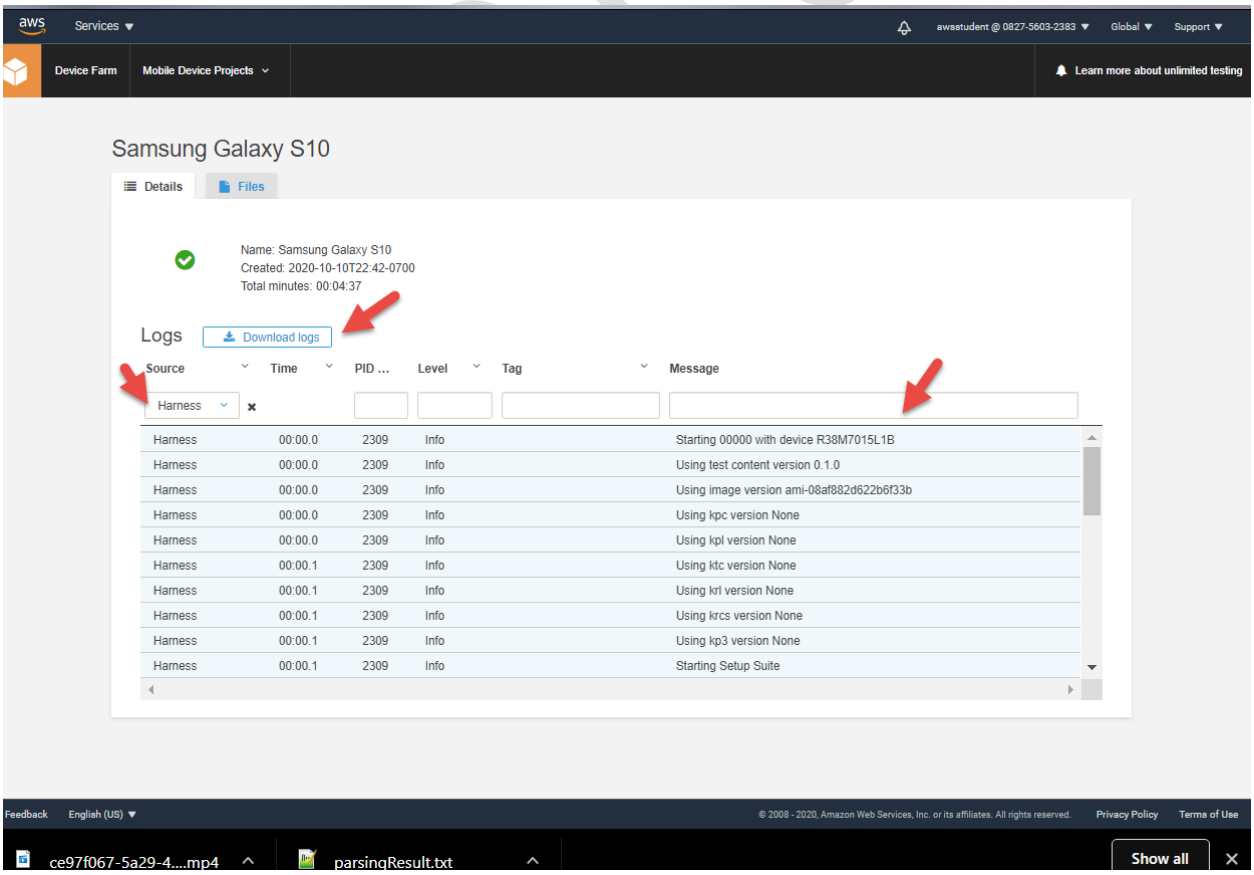




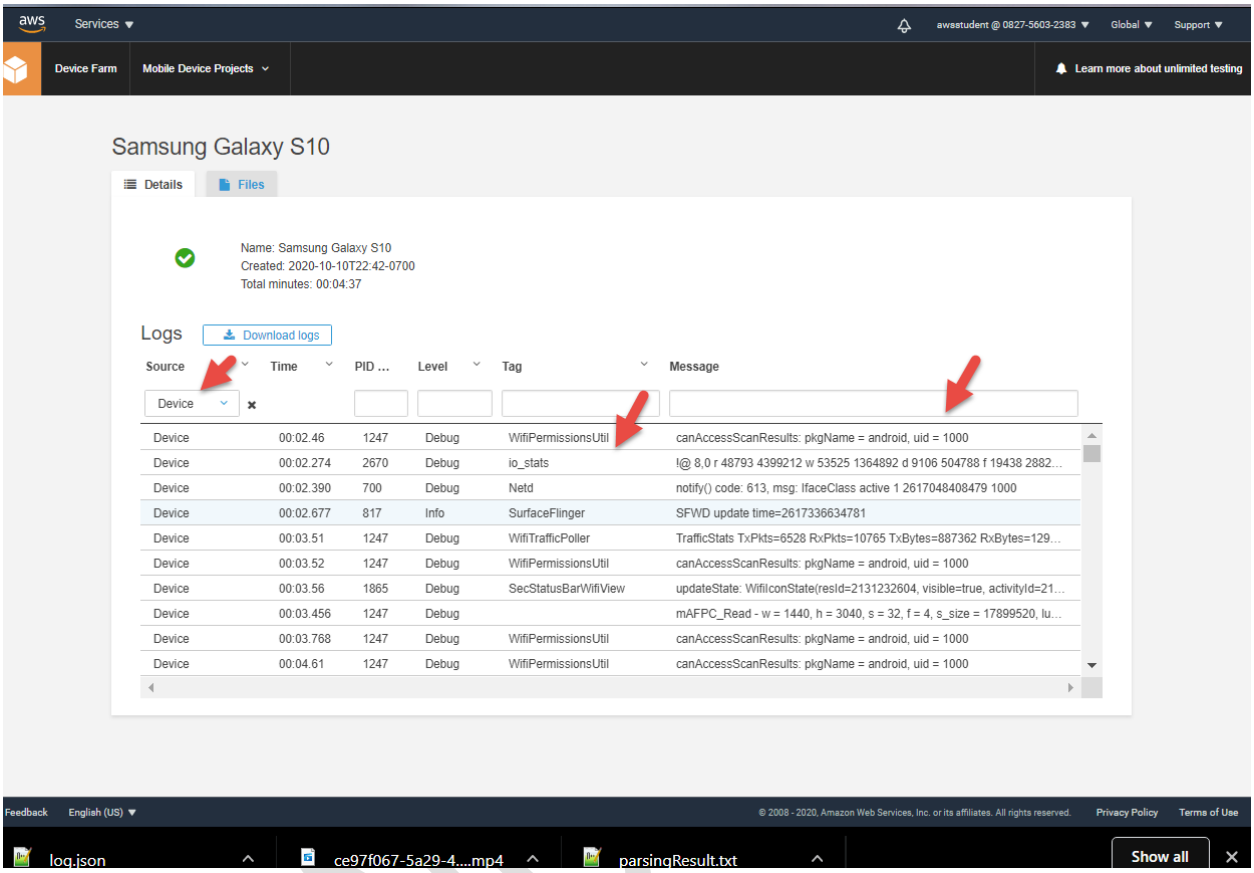
Check results



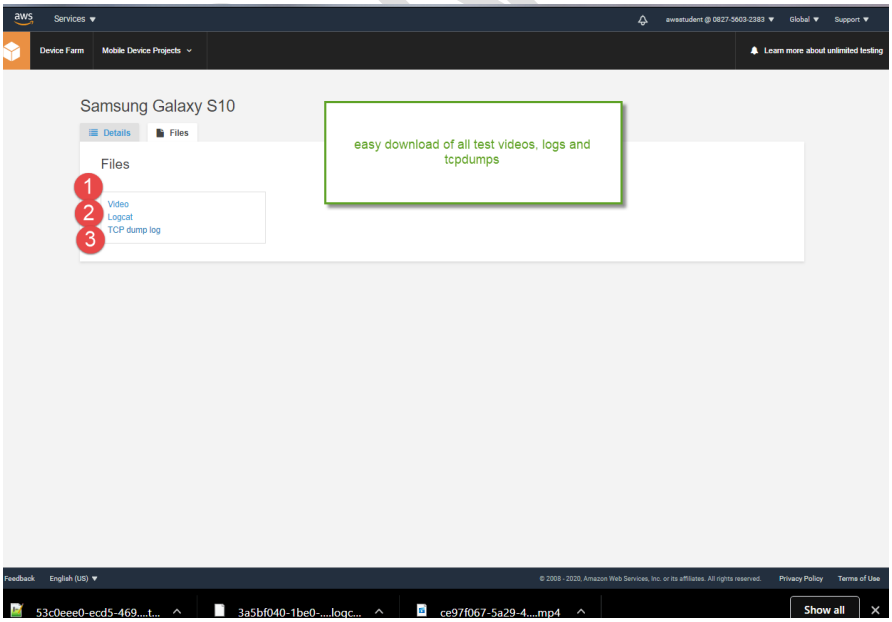
Detail logs – a/ Harness



Detail logs – b/ Device

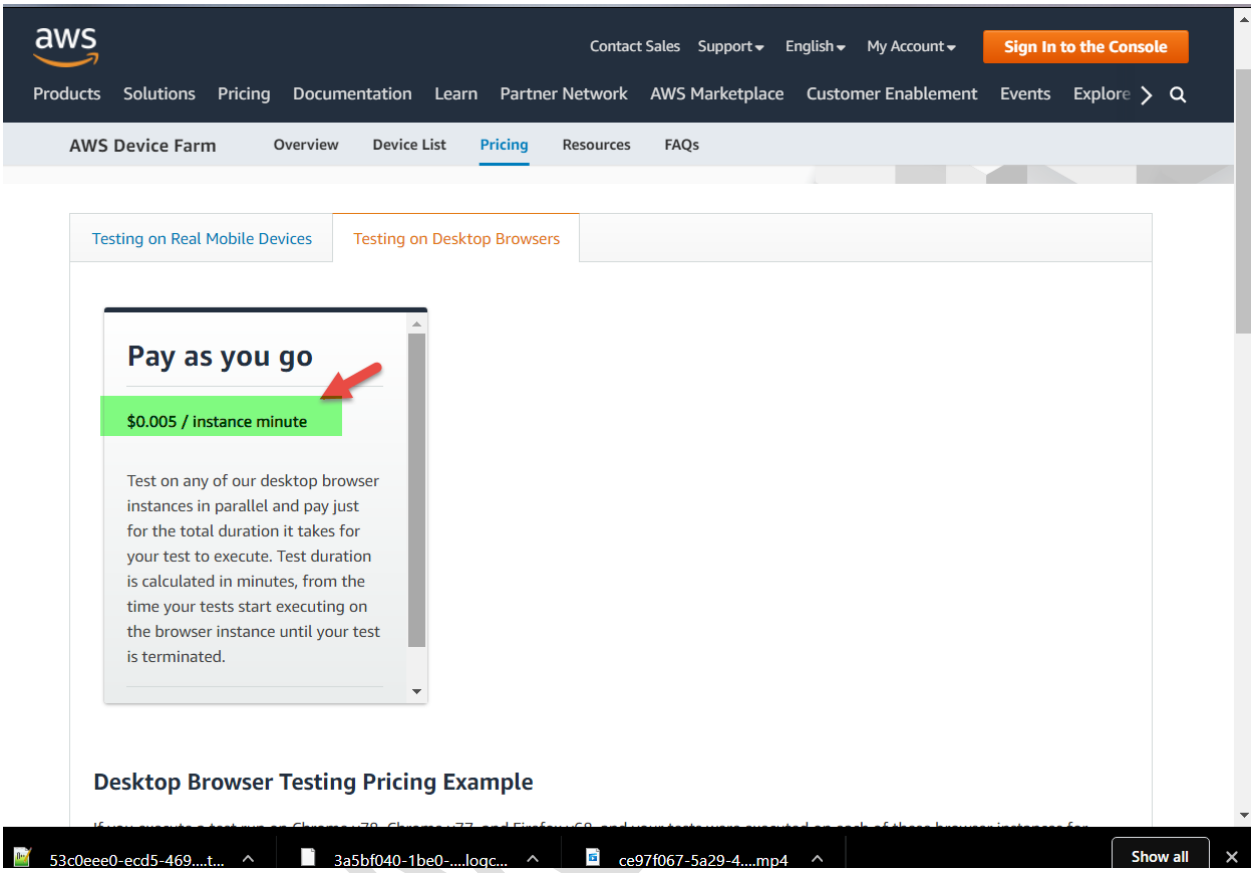


Easy download of all test videos, logs and tcpdumps:

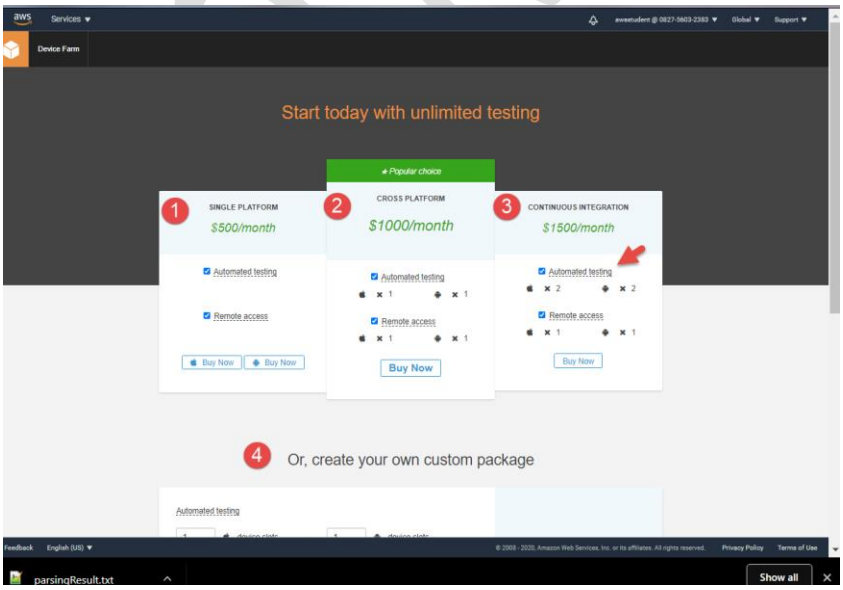


Pricing

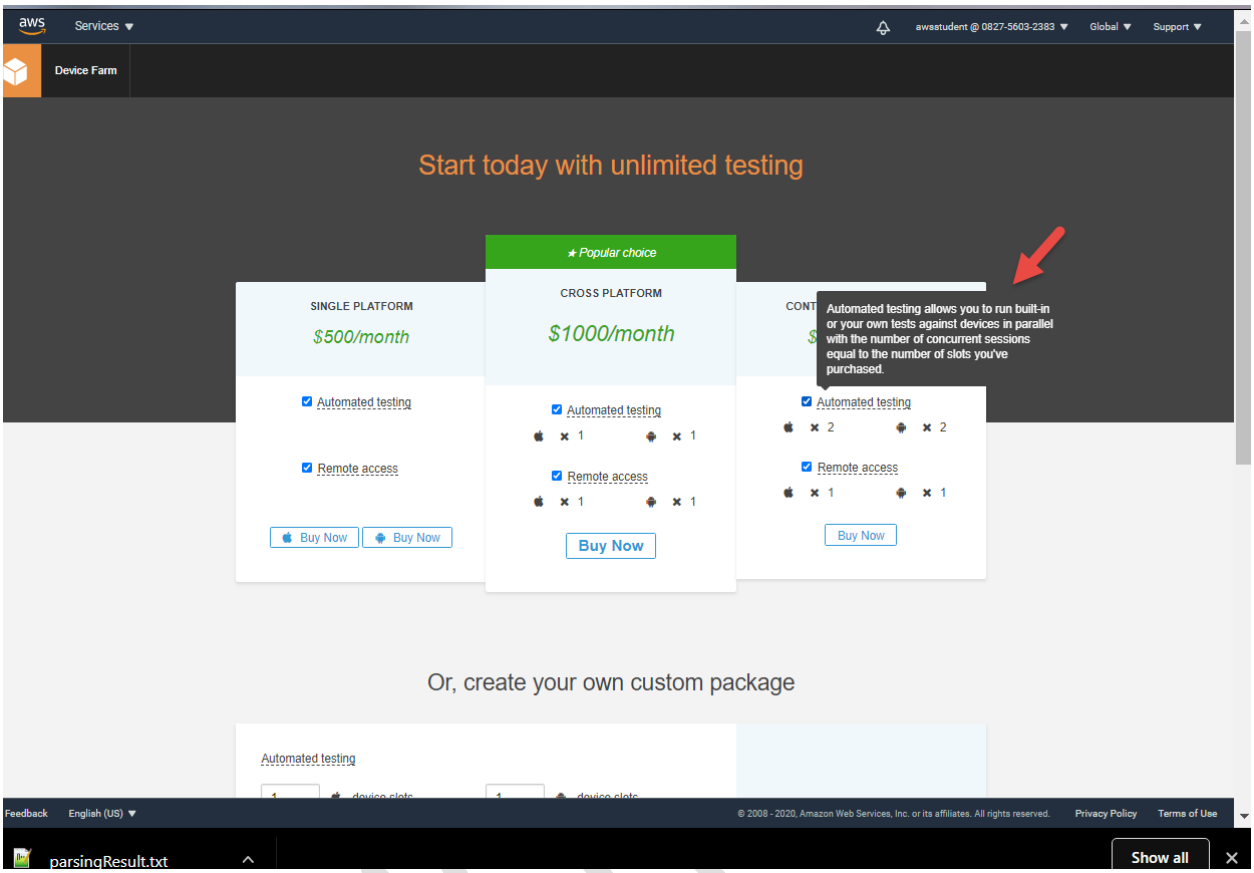
1/ Pay as you go:



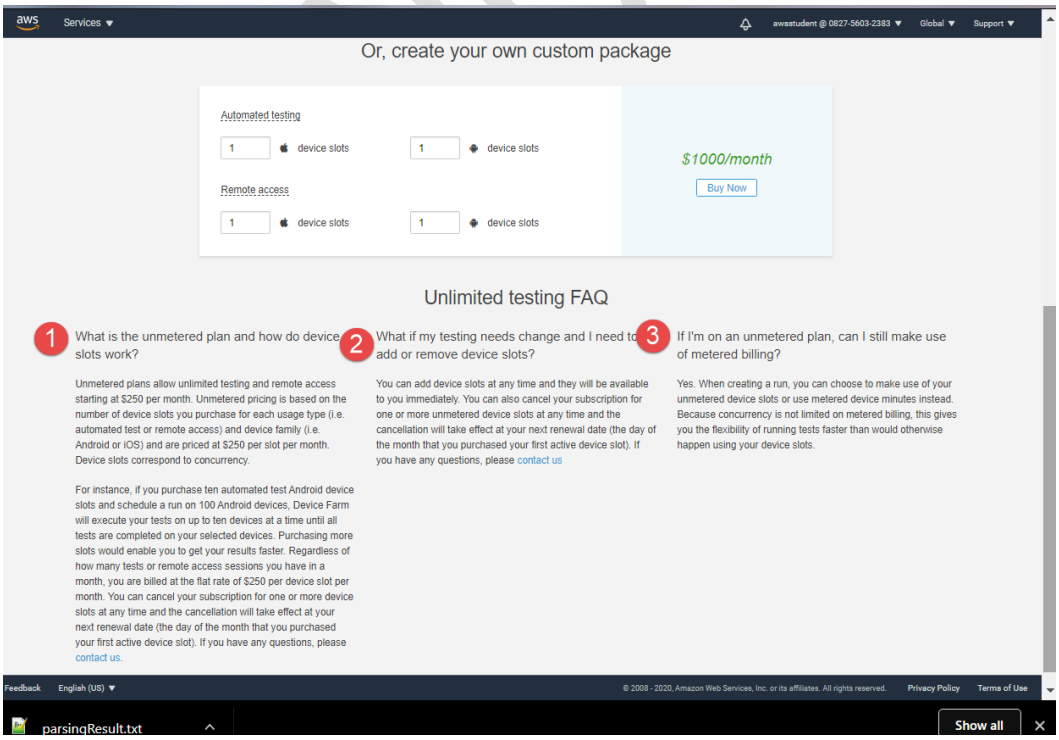
2/ unlimited pricing:

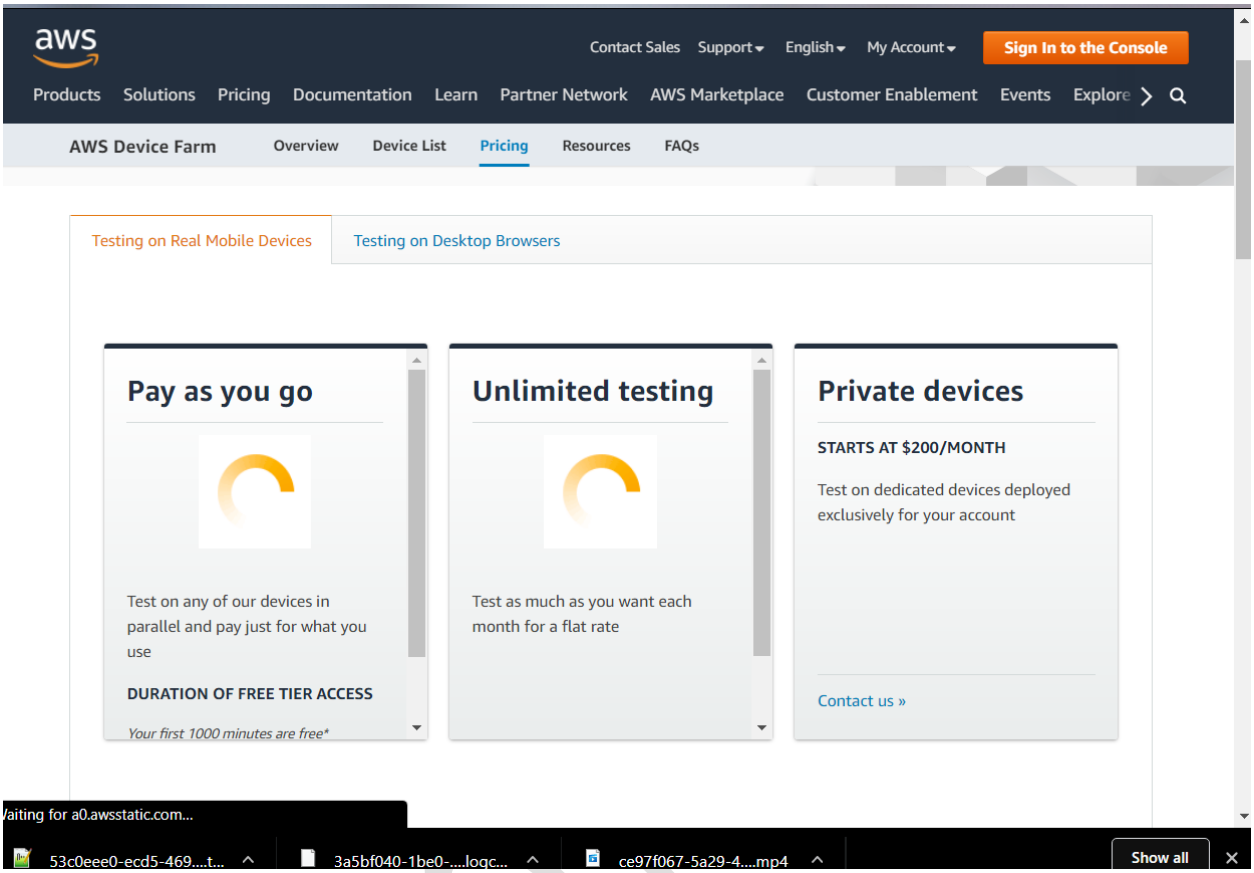


Allow Automated testings with CI:

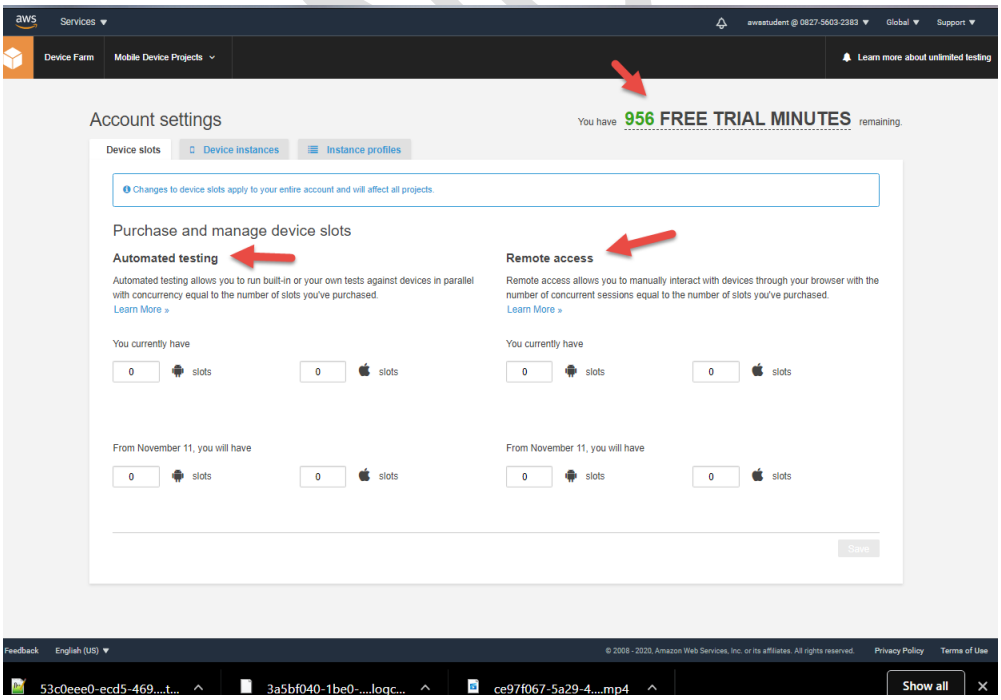


Create your own package:

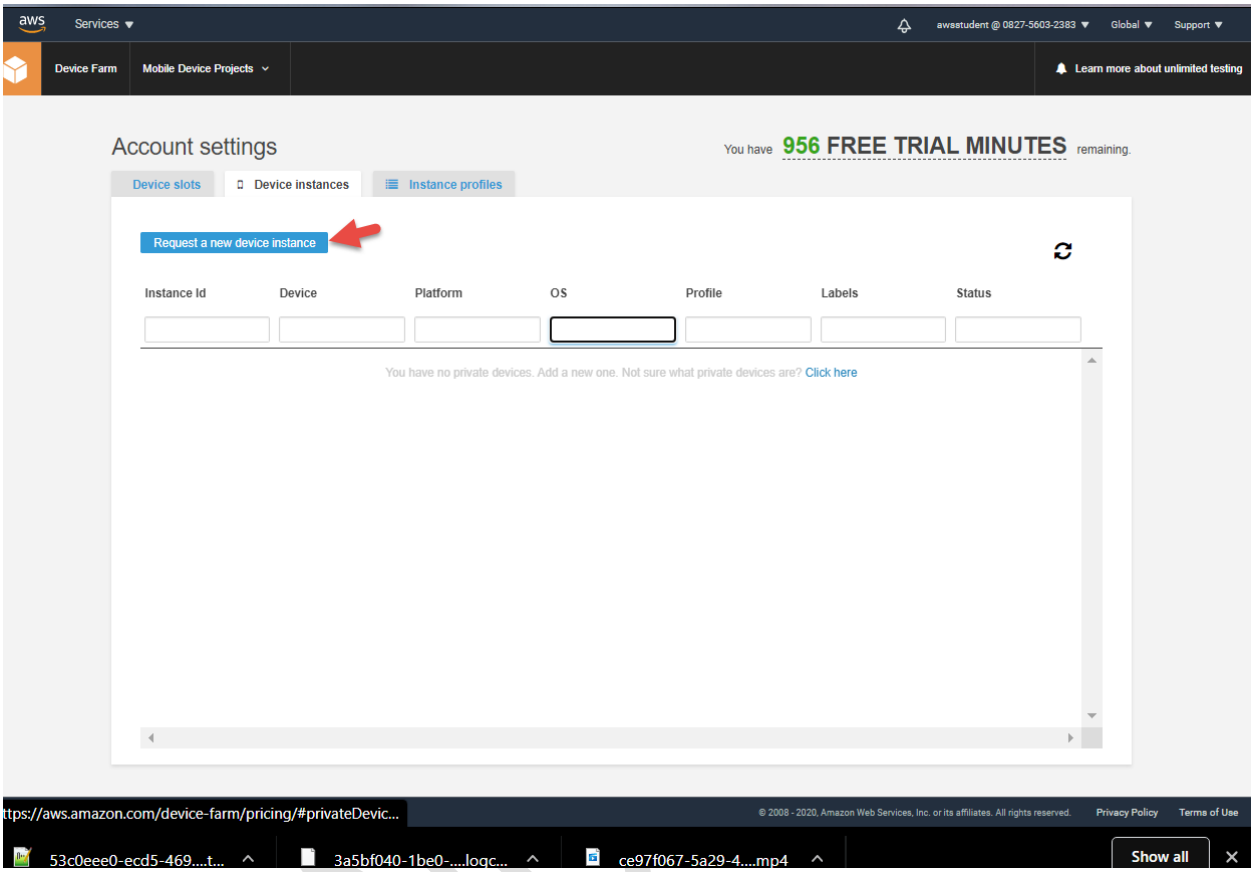




My account setting to show Free trial minutes:



Can request your own device instances:



Docs, Videos, Demos:

Docs:

https://docs.aws.amazon.com/devicefarm/latest/developerguide/welcome.html

Videos:

Getting Started with AWS Device Farm

https://youtu.be/ZWAUAerFB6s

Git :

https://github.com/aws-samples/aws-device-farm-sample-app-for-android

https://github.com/aws-samples/aws-device-farm-sample-app-for-ios

sample test app at https://code.google.com/archive/p/flickrj-android/downloads

Built-in Fuzz Test

https://docs.aws.amazon.com/devicefarm/latest/developerguide/test-types-built-in-fuzz.html

Lab:

https://www.qwiklabs.com/focuses/10377

Sample test suites:

```
[
  {
    "name": "Setup Suite",
    "tests": [
      {
        "name": "Setup Test"
      }
    ]
  },
  {
    "name": "Built-in Fuzz Suite",
    "tests": [
      {
        "name": "Built-in Fuzz Test"
      }
    ]
  },
  {
    "name": "Teardown Suite",
    "tests": [
      {
        "name": "Teardown Test"
      }
    ]
  }
]
```

Sample logs:

1/ harness logs

D:\Downloads>more log.json

[{"attachment_id": null, "parent_id": null, "timestamp": "2020-10-11T05:42:46.570250Z", "level": "Info", "pid": 2309, "data": "Starting 00000 with device R38M7015L1B", "source": "Harness", "tag": null, "subtype": "Text", "tid": 2314, "type": "Message", "id": 3}, {"attachment_id": null, "parent_id": null, "timestamp": "2020-10-11T05:42:46.570456Z", "level": "Info", "pid": 2309, "data": "Using test content version 0.1.0", "source": "Harness", "tag": null, "subtype": "Text", "tid": 2314, "type": "Message", "id": 4}, {"attachment_id": null, "parent_id": null, "timestamp": "2020-10-11T05:42:46.570594Z", "level": "Info", "pid": 2309, "data": "Using image version ami-08af882d622b6f33b", "source": "Harness", "tag": null, "subtype": "Text", "tid": 2314, "type": "Message", "id": 5}, {"attachment_id": null, "parent_id": null, "timestamp": "2020-10-11T05:42:46.570765Z", "level": "Info", "pid": 2309, "data": "Using kpc version None", "source": "Harness", "tag": null, "subtype": "Text", "tid": 2314, "type": "Message", "id": 6}, {"attachment_id": null, "parent_id": null, "timestamp": "2020-10-11T05:42:46.570894Z", "level": "Info", "pid": 2309, "data": "Using kpl version None", "source": "Harness", "tag": null, "subtype": "Text", "tid": 2314, "type": "Message", "id": 7},…

2/ logcat:

--------- beginning of main

10-11 16:43:04.880 1247 1653 D WifiPermissionsUtil: canAccessScanResults: pkgName = android, uid = 1000

10-11 16:43:05.412 27141 27155 I android.dqagen: Waiting for a blocking GC ProfileSaver

10-11 16:43:05.420 27141 27155 I android.dqagen: WaitForGcToComplete blocked ProfileSaver on ClassLinker for 7.703ms

10-11 16:43:05.767 24467 27193 D Aatp : /default-rotation

10-11 16:43:05.774 24467 27193 D Aatp : sha1: afbe92eb910a5b8c2b798f35cbfa6dbc1f2580ce

10-11 16:43:05.783 24467 24467 D AatpDeviceManager: taskStatus update (afbe92eb910a5b8c2b798f35cbfa6dbc1f2580ce: {"state":1})

10-11 16:43:05.788 24467 24467 D AatpDeviceManager: taskStatus update (afbe92eb910a5b8c2b798f35cbfa6dbc1f2580ce: {"result":{"rotation":0},"state":2})

10-11 16:43:05.793 24467 27193 D Aatp : /task/afbe92eb910a5b8c2b798f35cbfa6dbc1f2580ce

10-11 16:43:05.884 1247 1653 D WifiPermissionsUtil: canAccessScanResults: pkgName = android, uid = 1000

10-11 16:43:05.929 24467 27195 D Aatp : /task/afbe92eb910a5b8c2b798f35cbfa6dbc1f2580ce/status

10-11 16:43:06.263 24467 27200 D Aatp : /default-rotation

10-11 16:43:06.269 24467 27200 D Aatp : sha1: 60b7b0b120c87106df913cee5f41cec9b55d3242

10-11 16:43:06.279 24467 24467 D AatpDeviceManager: taskStatus update (60b7b0b120c87106df913cee5f41cec9b55d3242: {"state":1})

10-11 16:43:06.281 24467 24467 D AatpDeviceManager: taskStatus update (60b7b0b120c87106df913cee5f41cec9b55d3242: {"result":{"rotation":0},"state":2})

10-11 16:43:06.291 24467 27200 D Aatp : /task/60b7b0b120c87106df913cee5f41cec9b55d3242

10-11 16:43:06.425 24467 27202 D Aatp : /task/60b7b0b120c87106df913cee5f41cec9b55d3242/status

10-11 16:43:06.524 1247 1654 D WifiStateMachine: enter getWifiLinkLayerStats

10-11 16:43:06.525 1247 1654 I WifiVendorHal: getWifiLinkLayerStats(l.2973) before calling iface.getLinkLayerStats

10-11 16:43:06.561 1247 1654 I WifiVendorHal: getWifiLinkLayerStats(l.2973) after calling iface.getLinkLayerStats

10-11 16:43:06.580 27203 27203 D AndroidRuntime: >>>>>> START com.android.internal.os.RuntimeInit uid 2000 <<<<<<

3/ tcpdumps:

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes

05:44:42.277695 IP ip-192-168-1-100.us-west-2.compute.internal.53088 > ec2-52-45-155-205.compute-1.amazonaws.com.https: Flags [S], seq 2127107027, win 65535, options [mss 1460,sackOK,TS val 377166308 ecr 0,nop,wscale 8], length 0

05:44:42.366521 IP ec2-52-45-155-205.compute-1.amazonaws.com.https > ip-192-168-1-100.us-west-2.compute.internal.53088: Flags [S.], seq 1687700965, ack 2127107028, win 26847, options [mss 1360,sackOK,TS val 1768108349 ecr 377166308,nop,wscale 8], length 0

05:44:42.369072 IP ip-192-168-1-100.us-west-2.compute.internal.53088 > ec2-52-45-155-205.compute-1.amazonaws.com.https: Flags [.], ack 1, win 343, options [nop,nop,TS val 377166399 ecr 1768108349], length 0

05:44:42.372104 IP ip-192-168-1-100.us-west-2.compute.internal.53088 > ec2-52-45-155-205.compute-1.amazonaws.com.https: Flags [P.], seq 1:187, ack 1, win 343, options [nop,nop,TS val 377166402 ecr 1768108349], length 186

05:44:42.424318 IP ip-192-168-1-100.us-west-2.compute.internal.57496 > sea15s11-in-f3.1e100.net.https: Flags [S], seq 972138972, win 65535, options [mss 1460,sackOK,TS val 1559468470 ecr 0,nop,wscale 8], length 0

05:44:42.439219 IP sea15s11-in-f3.1e100.net.https > ip-192-168-1-100.us-west-2.compute.internal.57496: Flags [S.], seq 2618646787, ack 972138973, win 60192, options [mss 1360,sackOK,TS val 328146633 ecr 1559468470,nop,wscale 8], length 0

05:44:42.441664 IP ip-192-168-1-100.us-west-2.compute.internal.57496 > sea15s11-in-f3.1e100.net.https: Flags [.], ack 1, win 343, options [nop,nop,TS val 1559468487 ecr 328146633], length 0

05:44:42.444427 IP ip-192-168-1-100.us-west-2.compute.internal.57496 > sea15s11-in-f3.1e100.net.https: Flags [P.], seq 1:518, ack 1, win 343, options [nop,nop,TS val 1559468490 ecr 328146633], length 517

05:44:42.459166 IP sea15s11-in-f3.1e100.net.https > ip-192-168-1-100.us-west-2.compute.internal.57496: Flags [.], ack 518, win 240, options [nop,nop,TS val 328146654 ecr 1559468490], length 0

05:44:42.460771 IP ec2-52-45-155-205.compute-1.amazonaws.com.https >

ip-192-168-1-100.us-west-2.compute.internal.53088: Flags [.], ack 187, win 110, options [nop,nop,TS val 1768108443 ecr 377166402], length 0

05:44:42.461079 IP ec2-52-45-155-205.compute-1.amazonaws.com.https > ip-192-168-1-100.us-west-2.compute.internal.53088: Flags [.], seq 1:1349, ack 187, win 110, options [nop,nop,TS val 1768108443 ecr 377166402], length 1348