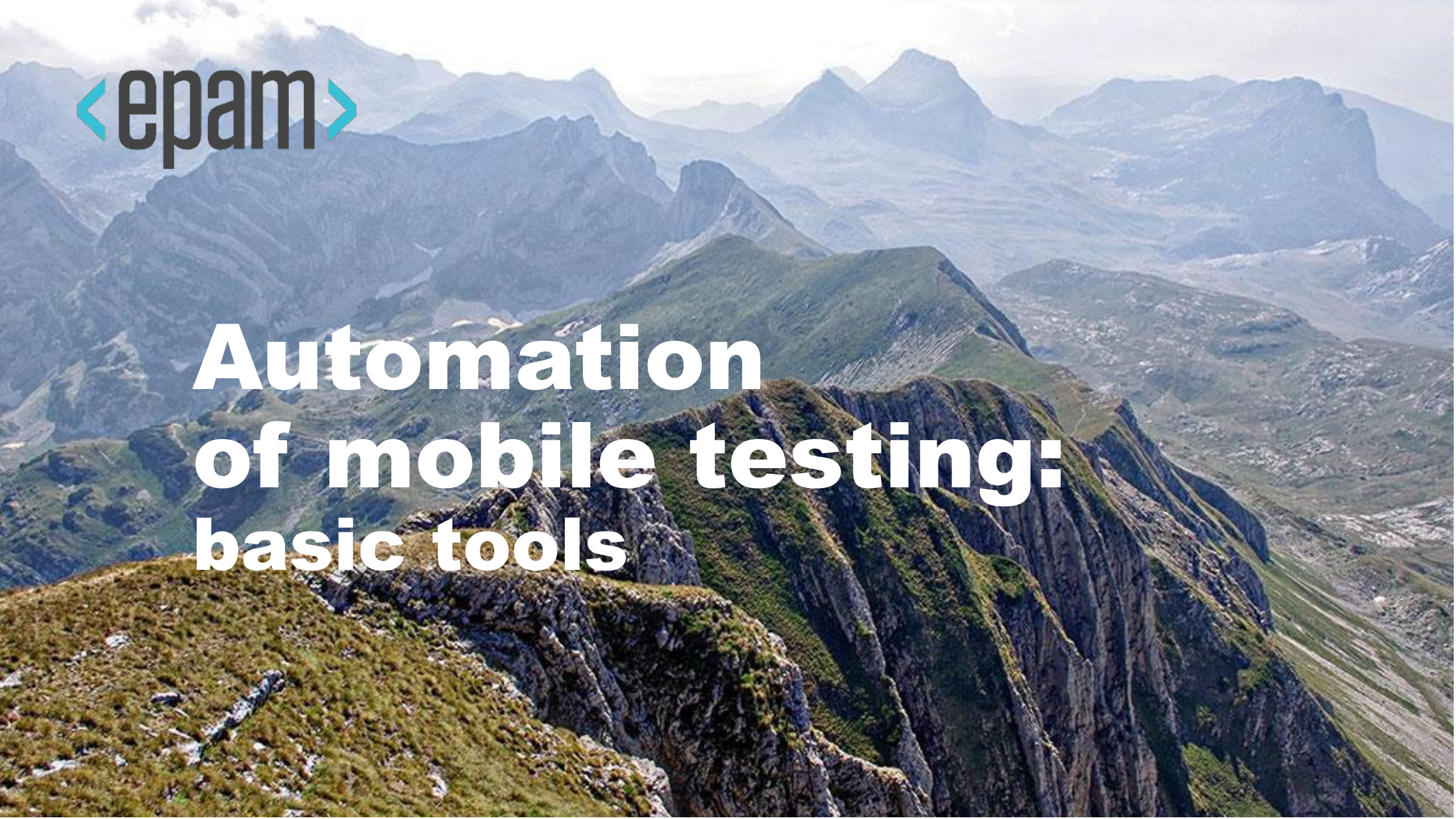

epam
Automation
of mobile testing:
basic tools

Plan

- Basic tools
  - Physical devices, emulators and cloud solutions
  - Appium
- How to write auto-tests
- Basics of Mobile Cloud Services
- Tips and tricks

Physical devices, emulators and cloud solutions

Device vs emulator

Physical device

Pros:

- **Expected user experience**

Contras:

- Expensive
- You need a lot of physical items
- Power and space consumption

Emulator

Pros:

- Cheap
- A lot of parameters can be adjusted: dimensions, RAM, disk space, set of sensors, ...
- No power and space consumption

Contras:

- **Not realistic behaviour**
- Computing consumption
- Performance issues
- Additional software

Device vs emulator: purposes

Emulator:

- Prototyping (GUI, layouts, ...)
- Early stage of auto-tests development

Device:

- Auto-tests finalising and debugging
- Auto-tests run

Mobile cloud services

Mobile cloud services (mobile farms) are the modern approach

They provide developers and testers with remote access to sets of physical devices *for fixed prices*

Remote access to set of emulators can be provided as well for less prices

Mobile platforms

- IOS
  - You have to be a registered Android developer
  - You have to use Apple/Mac environment (Xcode)
- Android
  - You can use free open-source tools on Win/Mac/Linux
  - Occupied most of mobile market at the moment

General environment settings for Android platform

The best way

You need JDK to work with Android development tools. Please use 8th release (9th has some problems yet.)

With the Java SDK ver.8, please install Android Studio **Bundle**.

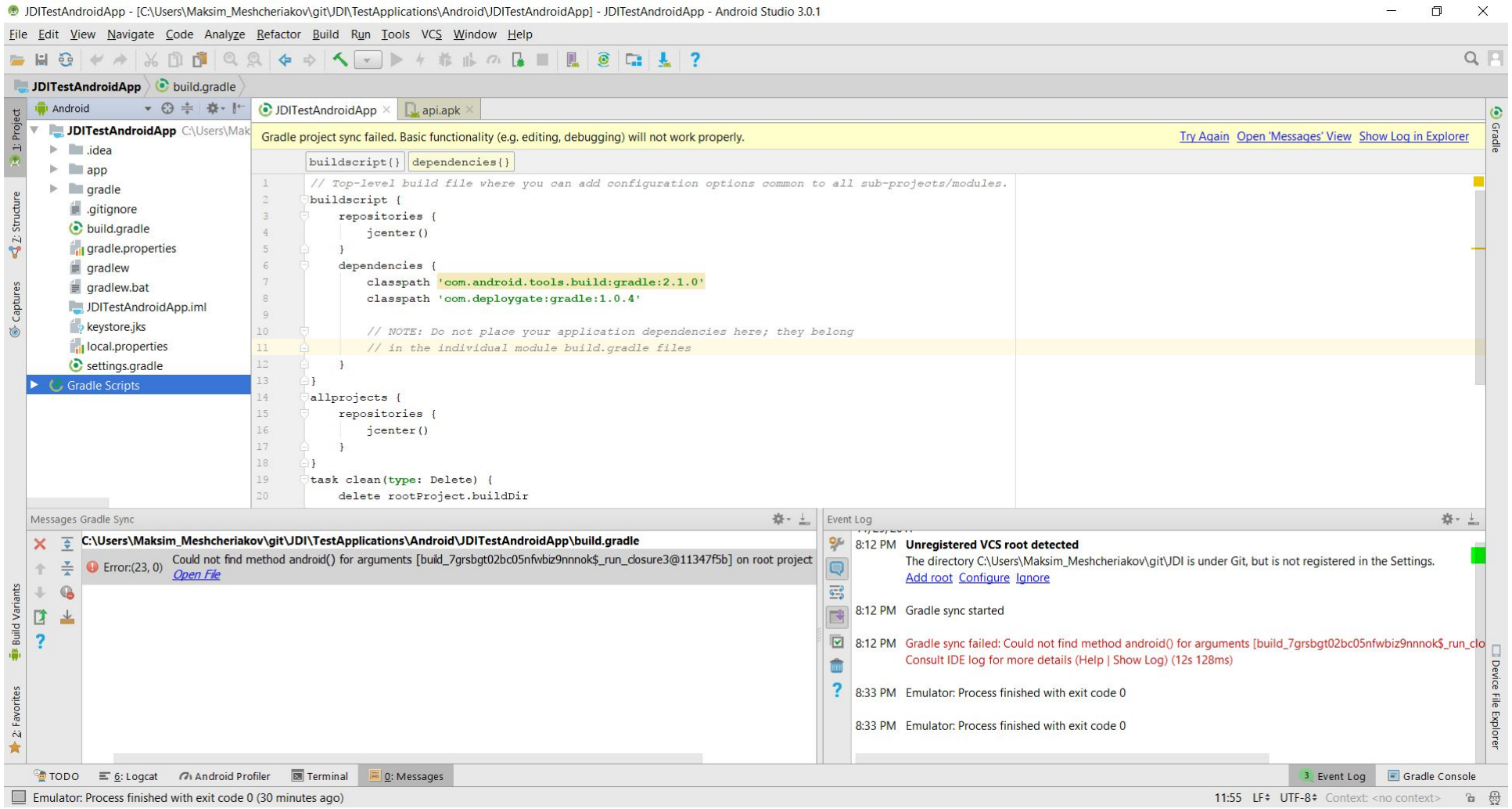
Bundle includes the complete set of all required tools, including Android SDK.

Otherwise, you will have to install and configure several packages by himself.

Android basic toolset

- Android Studio is the common toolset
- Android SDK (includes some CLI tools)
- Android Debug Bridge - ADB
- Android Virtual Device - AVD, and AVD Manager
- Android Device Monitor

Android Studio



Currently, Android Studio is used for most tasks to develop, debug and test Android applications

https://developer.android.com/studio/intro/index.html

Environment variables

- JAVA_HOME = Program Files\Java\jdkXX.YY (actual JDK location)
- ANDROID_HOME = ~\AppData\Local\Android\sdk (actual path to Android SDK)
- PATH = %PATH%, %ANDROID_HOME%\tools, %ANDROID_HOME%\platform-tools

<epam>

Android emulators setting up

Android Virtual Device

This is emulator of a certain Android device.
NOTE: AVD emulates mobile *hardware* (instead of iOS *simulator*)- first of all, ARM-based processor.

You can create a set of emulator that have different capabilities:

- Dimensions and form-factor
- Display parameters
- API level (Android version)
- RAM and disk space size
- Set of sensors

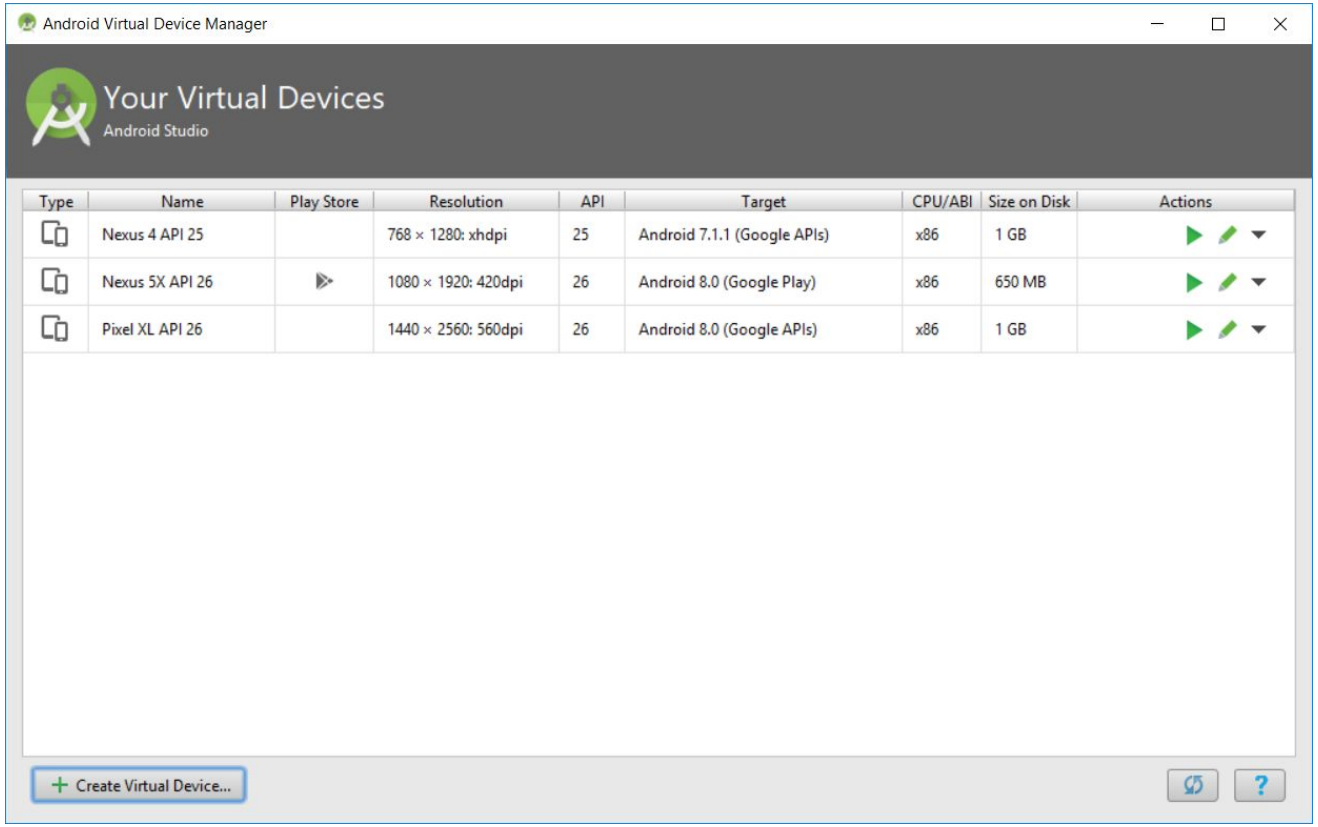
Consist of AVD

- **Hardware profile**: pre-sets of characteristics of a (real) devices. Some profiles include Play Store (indicated). Could be created and/or imported as well
- **System image**: set of software options - certain API version, set of applications
- **Storage area**: dedicated storage area on host computer. It stores the device user data (apps and settings), emulated SD card
- **Skin**: the appearance of a device. The AVD Manager provides some predefined skins. User can define his own skins or use 3d-party ones

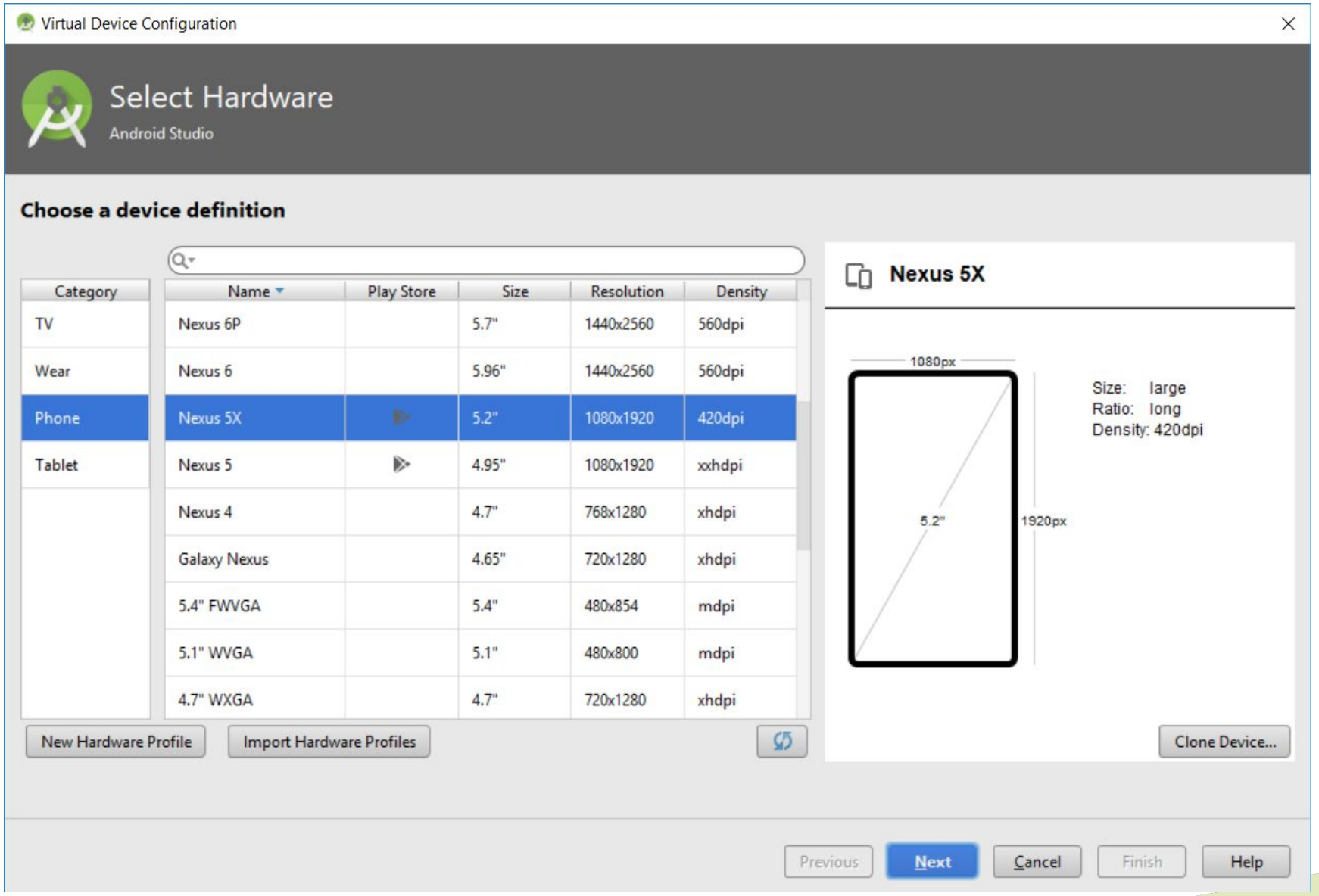
Access to AVD manager

To open the AVD Manager in Android Studio, do one of the following:

- Select Tools > Android > AVD Manager
- Click AVD Manager icon in the toolbar



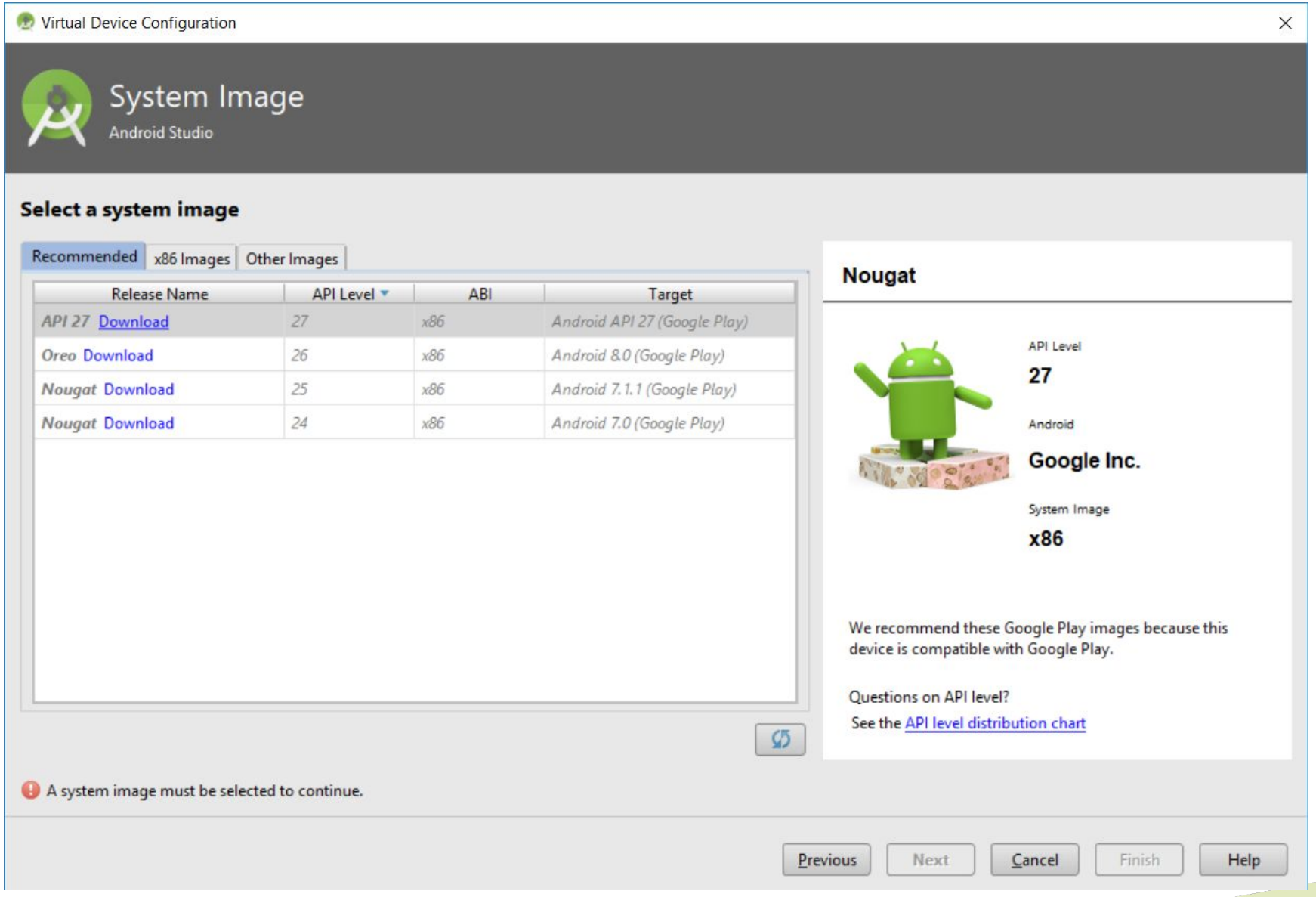
Start to create an AVD



Click Create Virtual Device at the bottom of the AVD Manager dialog

The Select Hardware page appears.

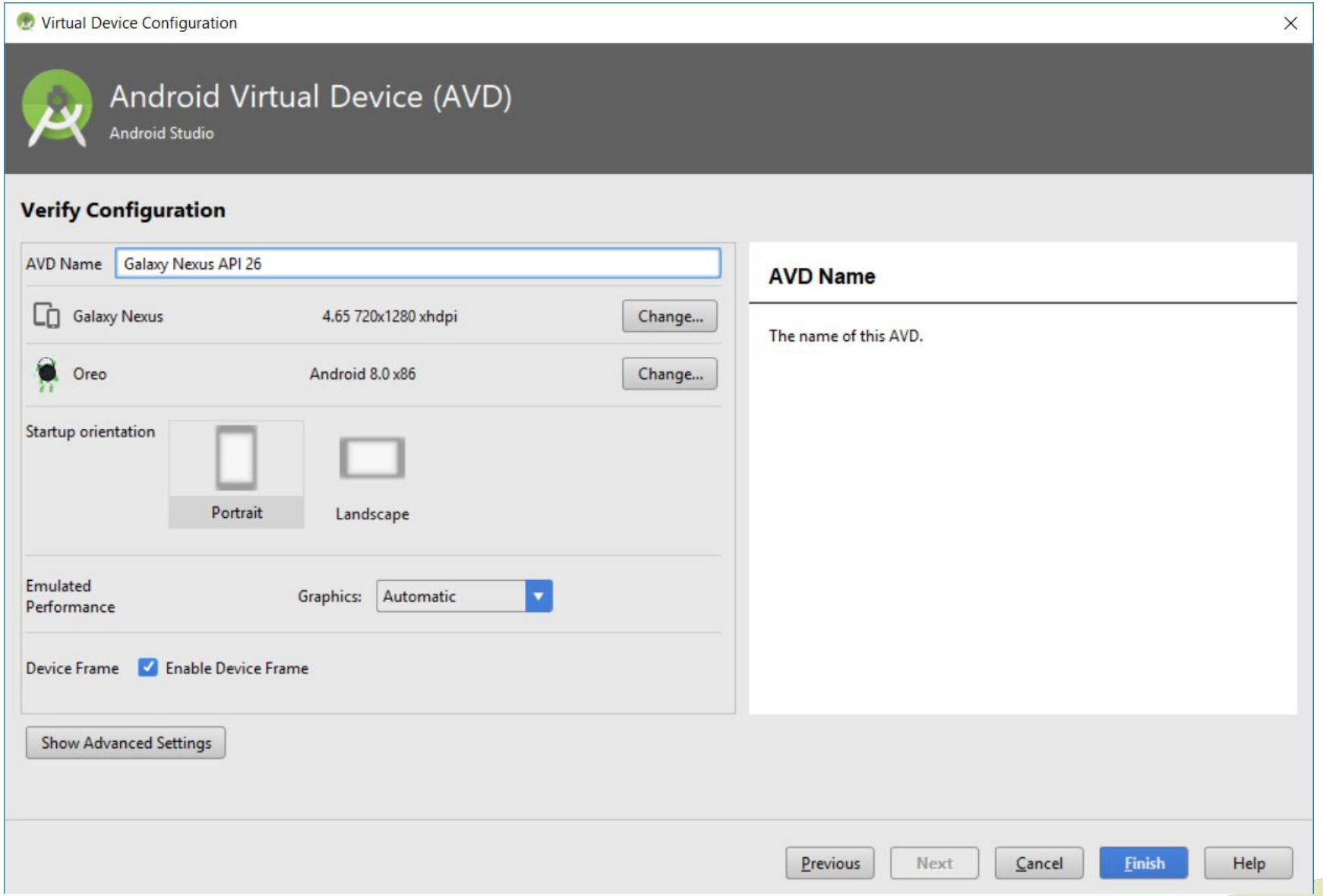
Select hardware profile



- Select a hardware profile
- Click Next

The System Image page appears.

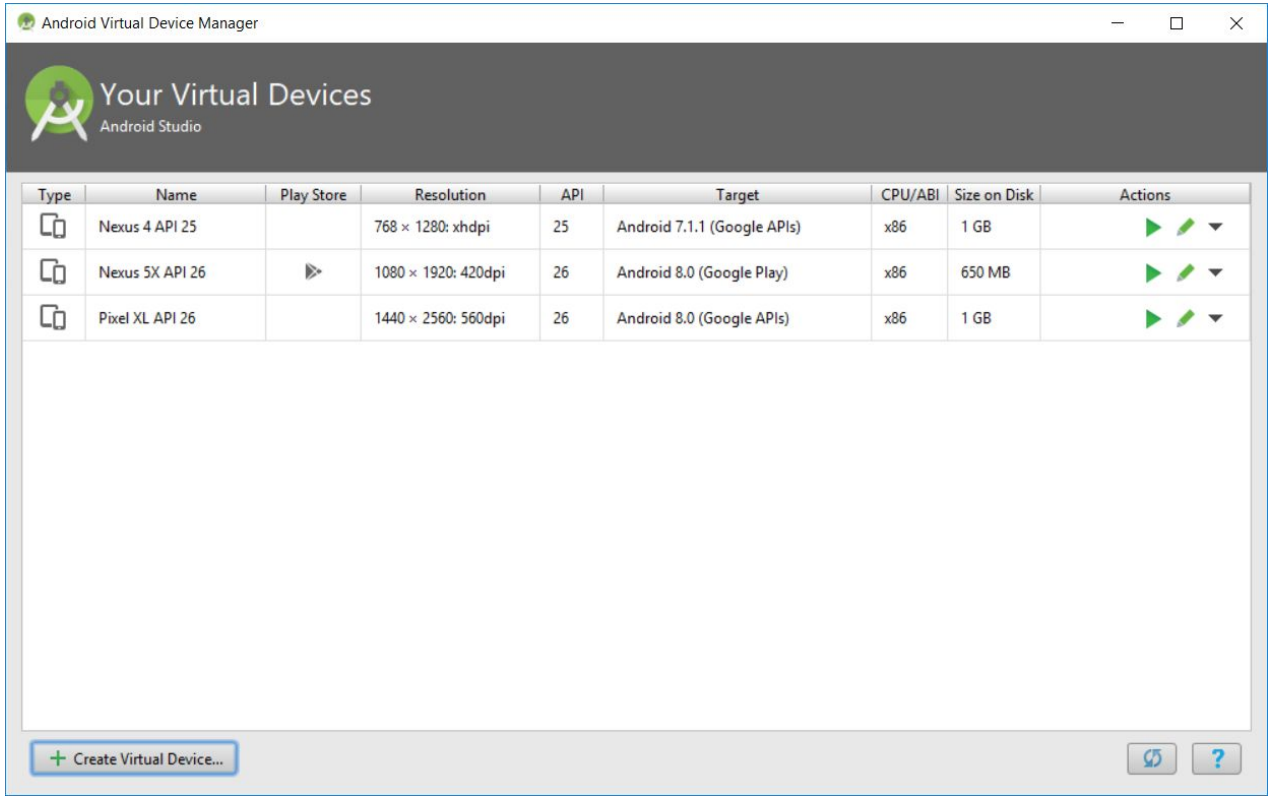
Verify AVD



Verify new AVD and accept (Finish) or adjust its parameters (Previous, Change, Show Advanced Settings)

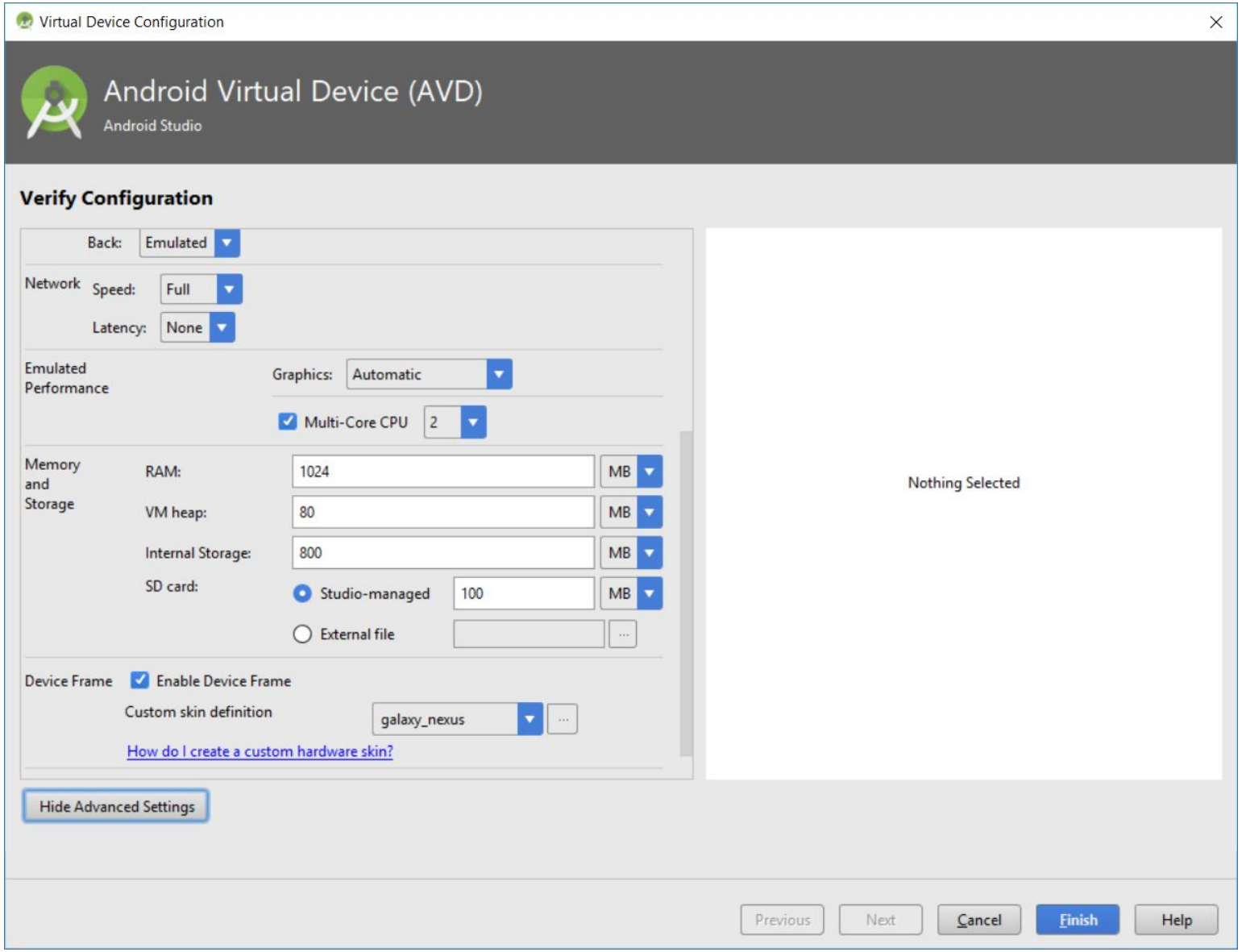
Editable AVD

- Each existing AVD parameter or feature can be changed and saved for future using
- New changes overwrite default ones of hardware profile and other AVD parts



- Use “Edit” icon of certain AVD to change required parameters

Advanced settings of AVD



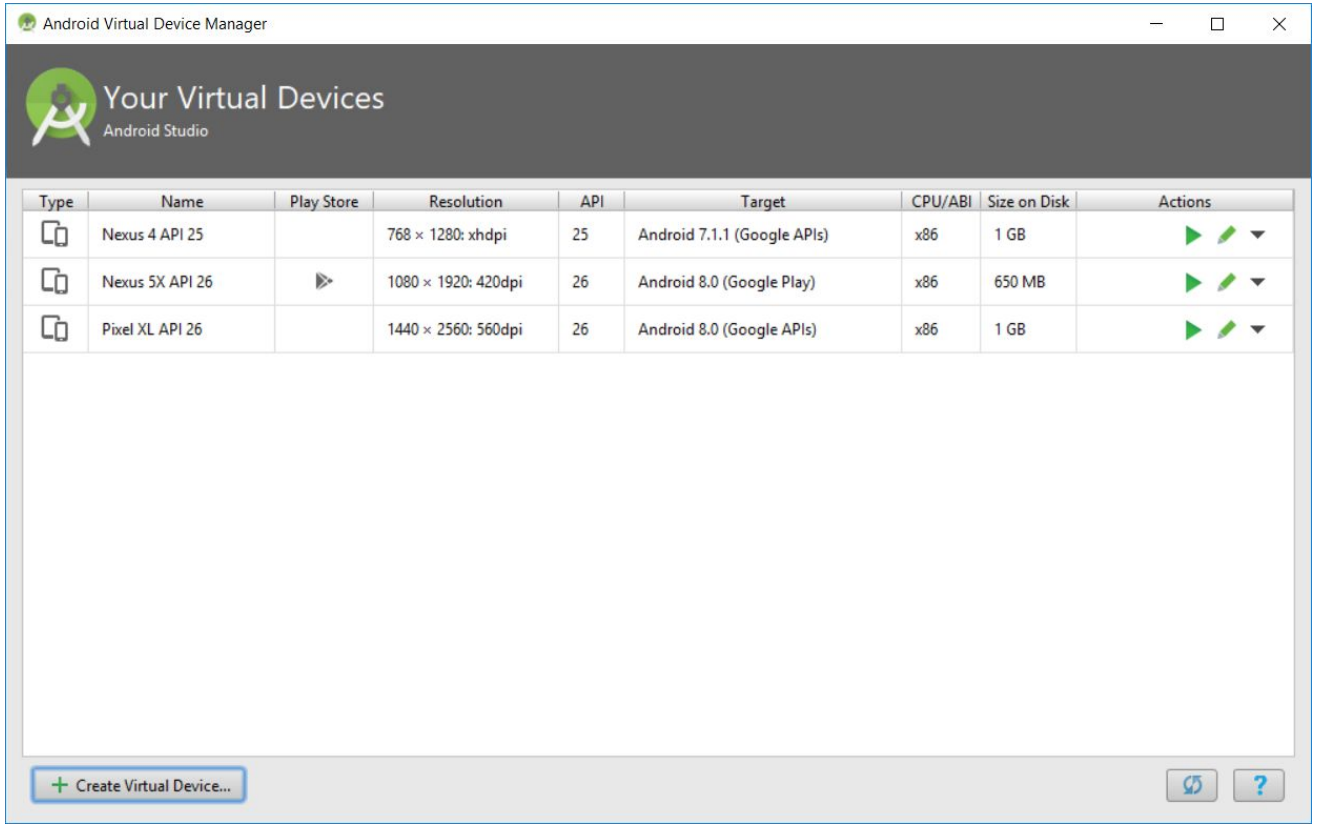
Click

“Show Advanced Settings“

button to get access to more editable settings.

Scroll down to see full list of ones.

Run, stop and wipe an AVD



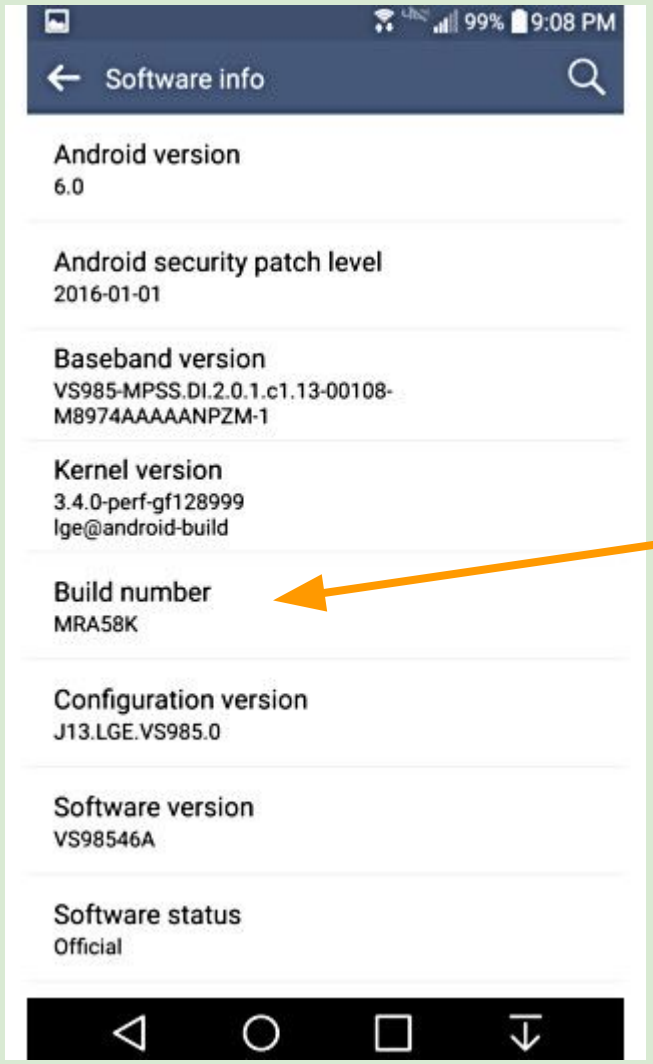
- Double-click the required AVD or click Launch ▶ **to run an emulator**
- Right-click an AVD and select Stop, or click Menu ▾ and select Stop **to stop a running emulator**
- Right-click an AVD and select Wipe Data, or click Menu ▾ and select Wipe Data **to clear the data for an emulator**, and return it to the same state as when it was first defined

Access to emulator via ADB

- Set up the environment for ADB as described before (if not yet)
- Run AVD instance from Android Studio AVD Manager

Android physical devices setting up

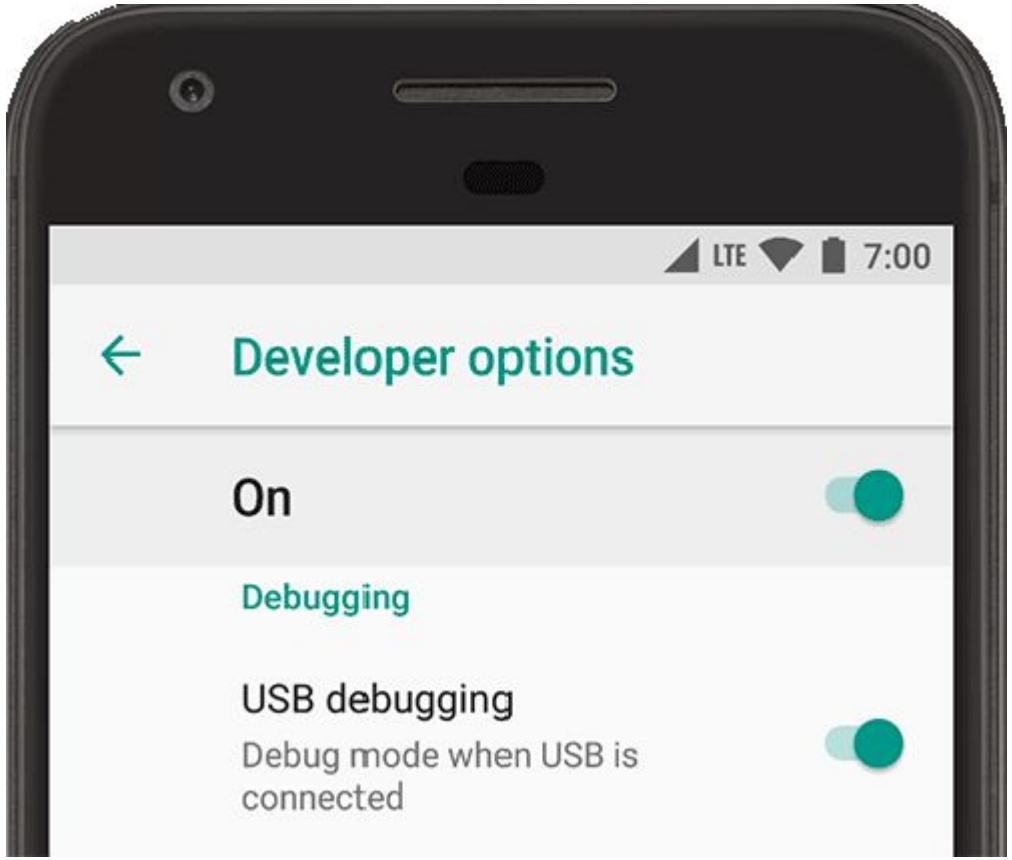
Enable Developer Options



If not yet
(starting from ver. 4.2):

- Settings > About device > Software Info
- Press Build Number 7 times

Enable options



- “Developer options” item appears
- Enable “On”
- Enable “USB debugging” (scroll down a little)
- Set “USB configuration” to MTP
- Full options guide

Check connection

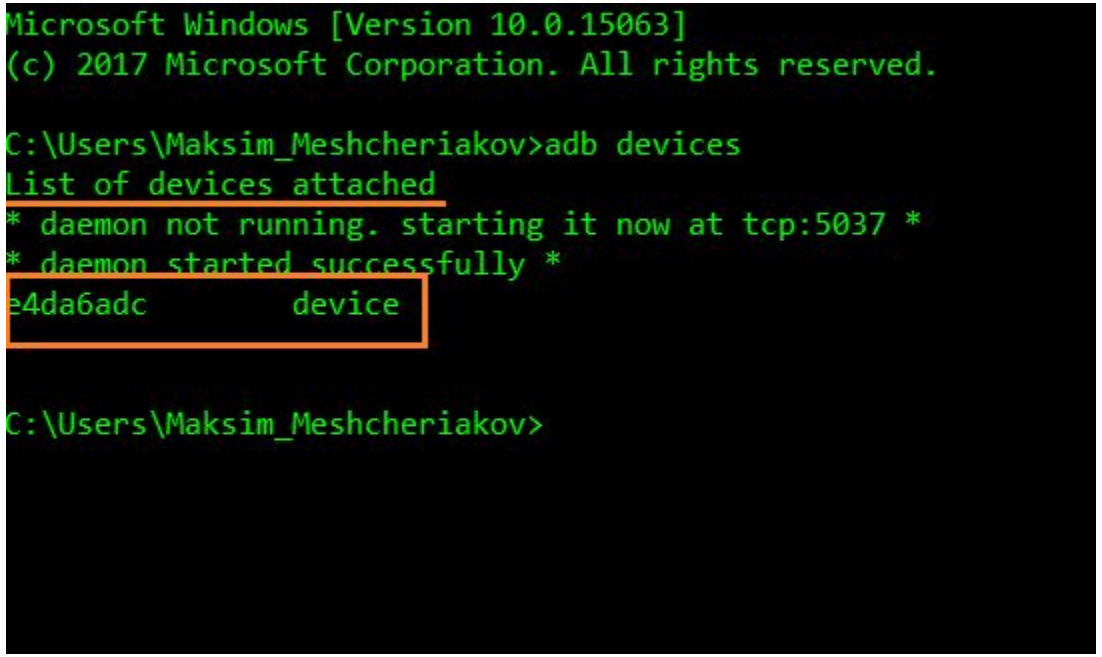
Android Debug Bridge

Use ADB to get access to Android device under testing via USB or WiFi (TCP/IP)

- Install and delete applications
- Add and remove files
- Get logs and dumps
- Get information about state of device and processes

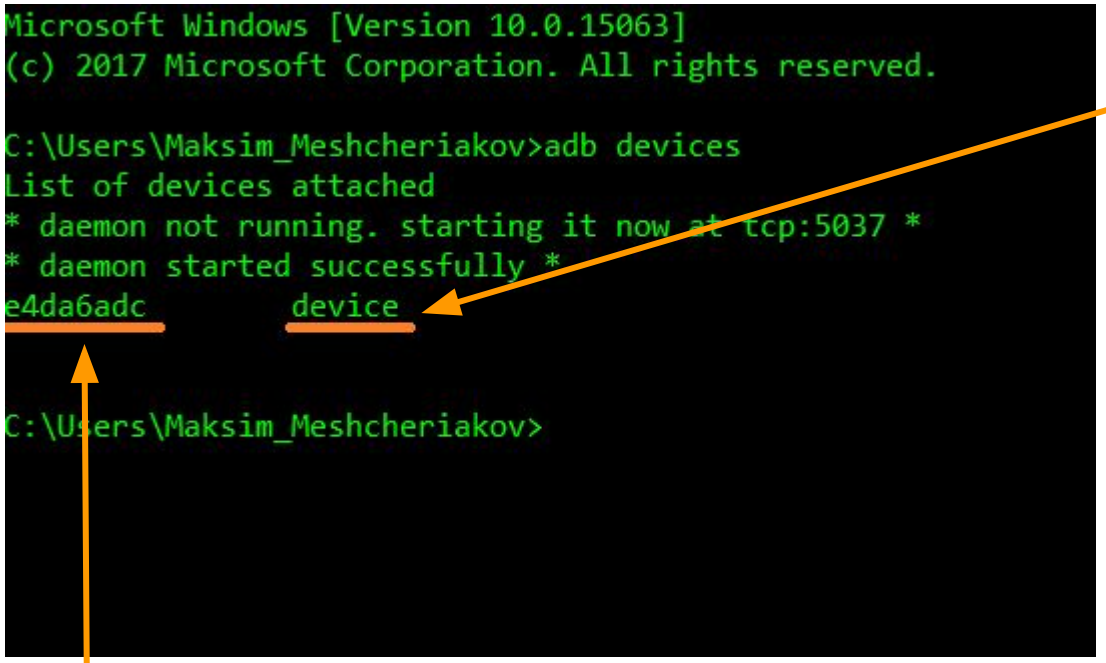
Access via USB

If you have properly configured environment:



- Connect Android device to computer by USB
- Open command-line terminal
- Use adb devices command to verify connection

Device identification by ADB

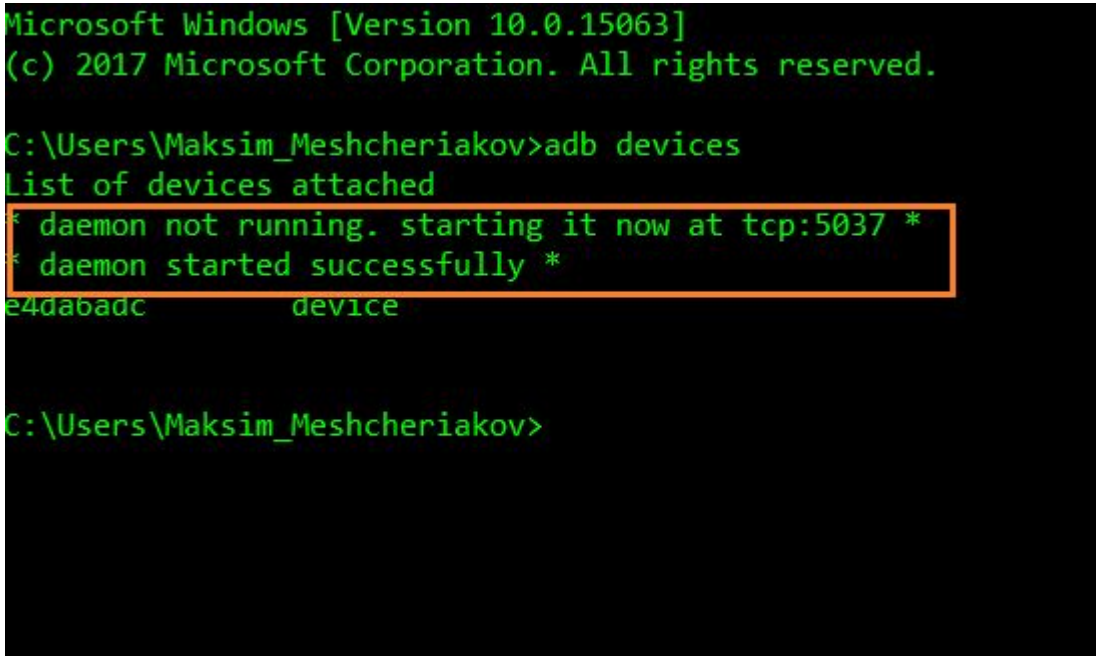


Device status:

- **Device** - device connected
- **Offline** - device is not connected

Serial number: A string created by adb to uniquely identify the device

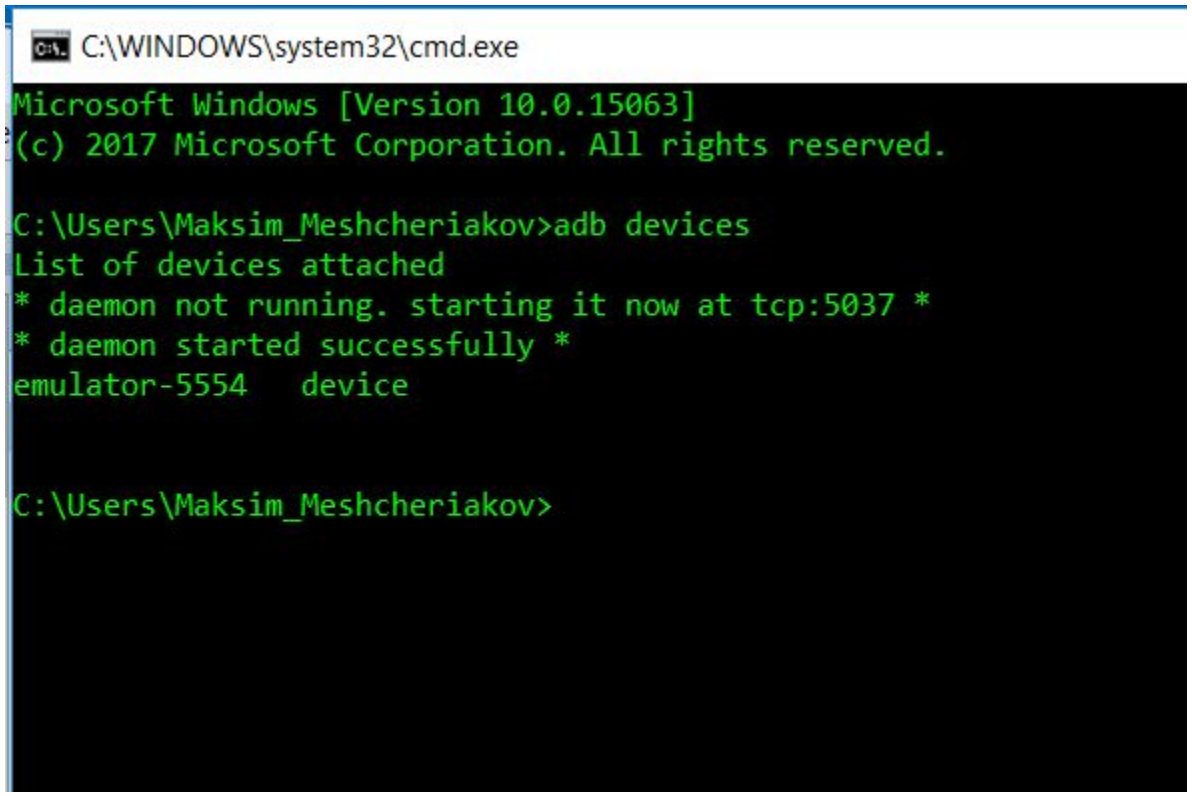
ADB is a client-server system



- Daemon **adbd** on the device
- Command-line client

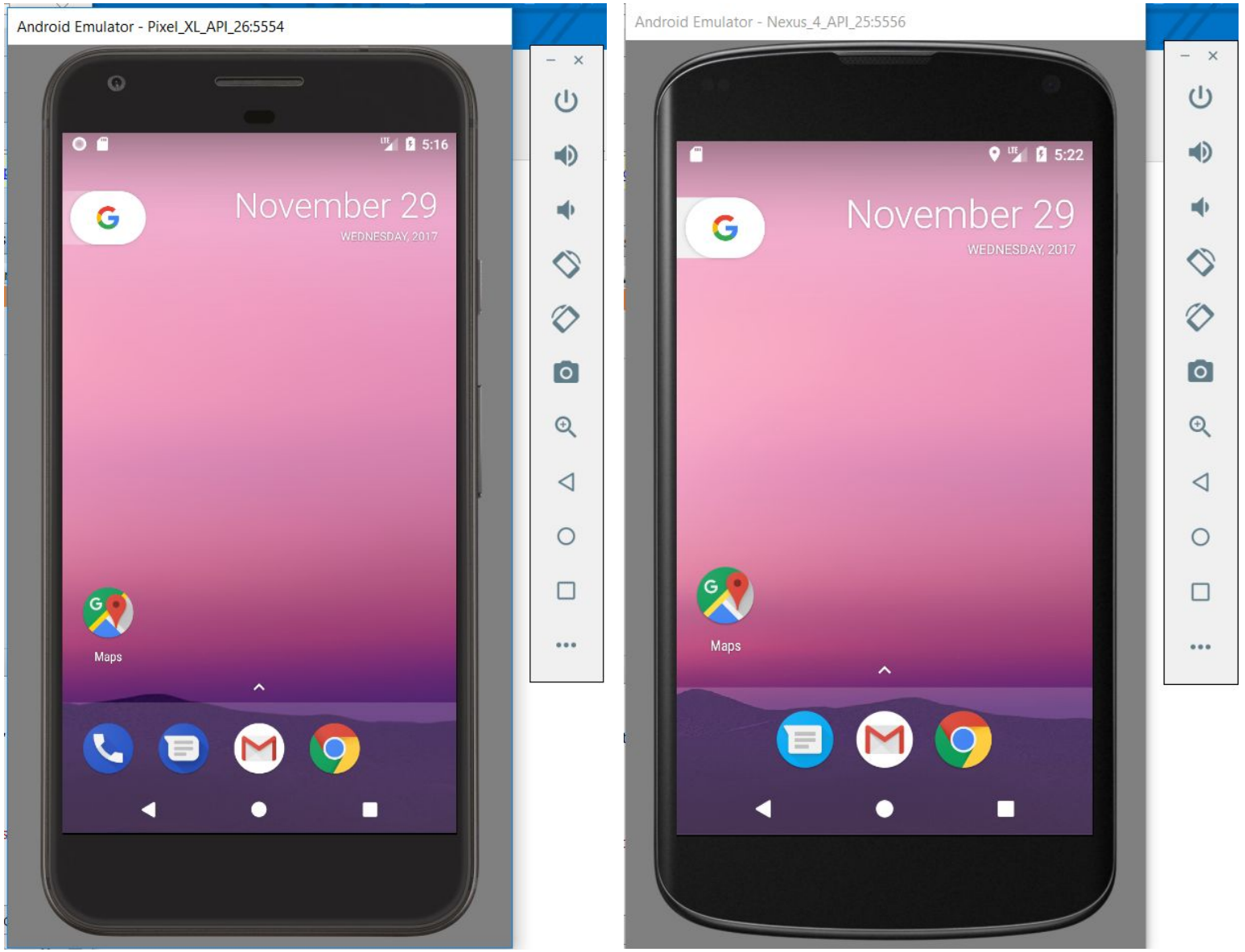
- **adb kill-server**: for re-initialization of adb if something goes wrong
- **adb start-server**: the adb server start automatically on typing of some adb command

AVD listed by ADB



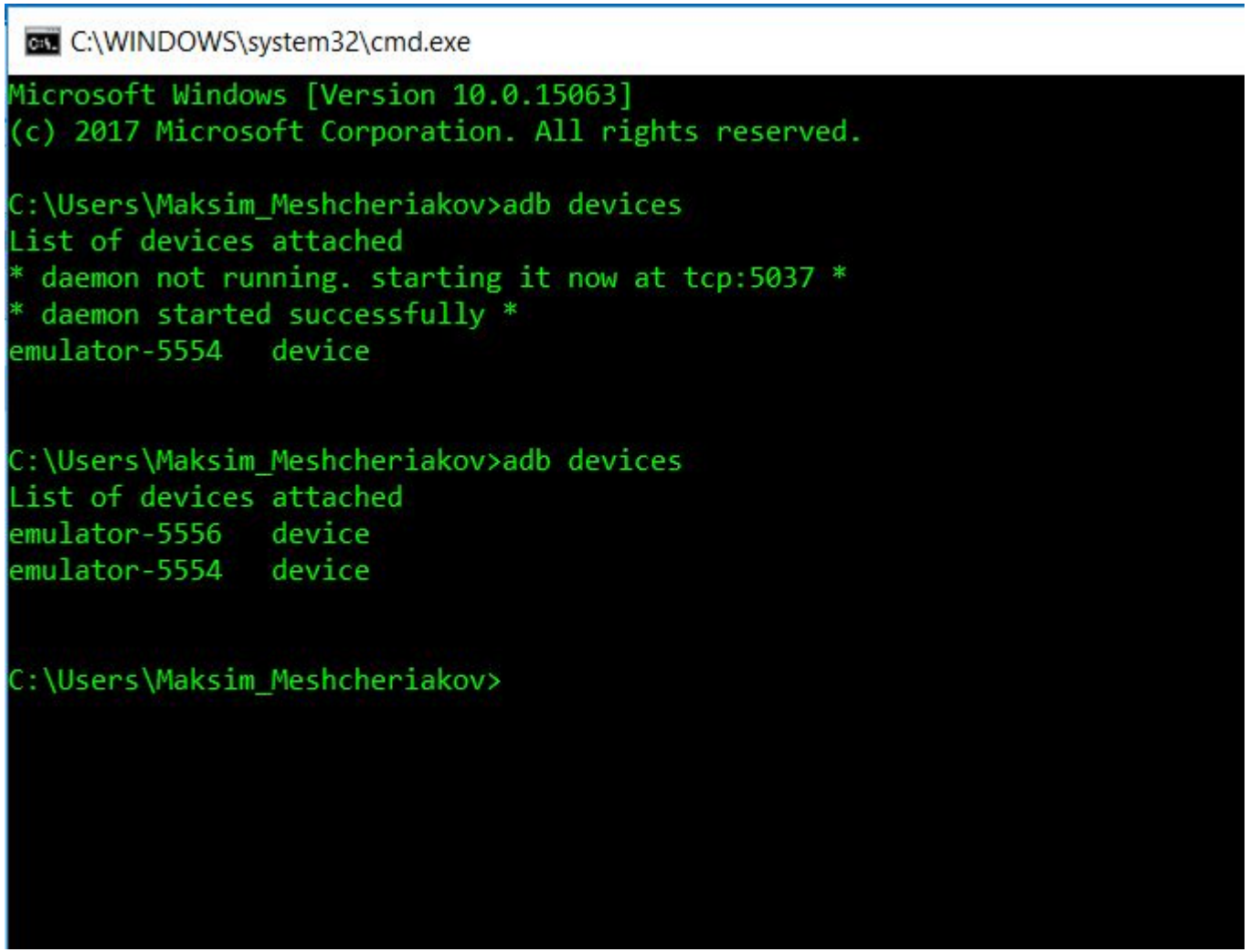
Type “adb devices” in console

Get another one AVD



Run another one AVD instance from Android Studio AVD Manager

2 AVDs listed by ADB



Type “adb devices” in console once again

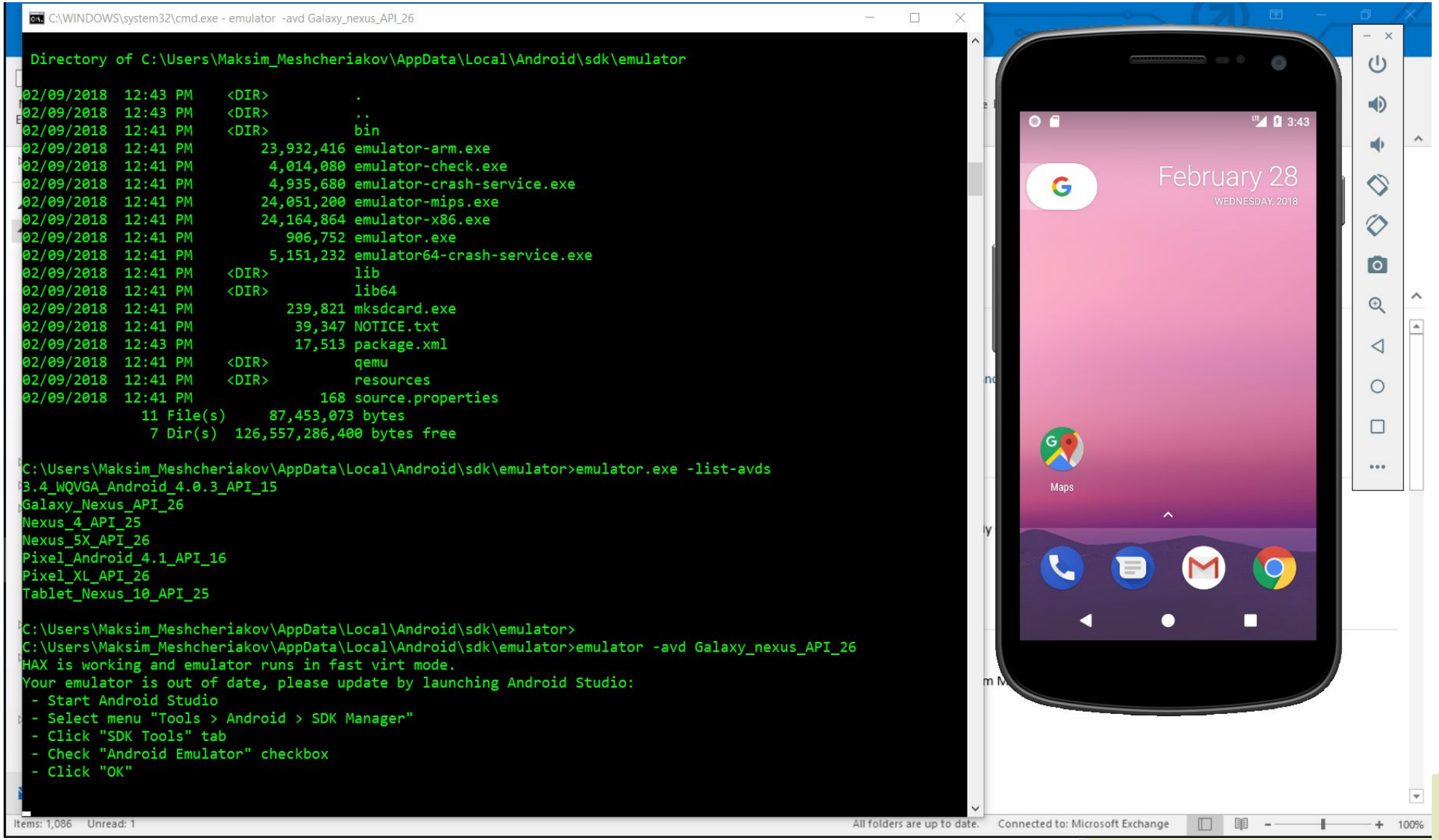
2 emulators and real device

```
C:\Users\Maksim_Meshcheriakov>
C:\Users\Maksim_Meshcheriakov>adb devices
List of devices attached
emulator-5556   device
emulator-5554   device
e4da6adc        device


C:\Users\Maksim_Meshcheriakov>
```

- Connect physical device to USB (do not stop running AVDs)
- Type “adb devices” in console once again

Run AVD via CLI



CLI for AVD

You can use CLI AVD tools as well:

- tools/bin/avdmanager.bat
  to create and maintain AVD instances
- ANDROID_HOME/emulator
  to run certain AVD instance
  https://developer.android.com/studio/run/emulator-commandline.html

Install and remove application

```
$ adb install path_to_apk
```

```
$ adb uninstall package
```

In case of uninstallation you have to use Java package name instead of .apk filename.

```
$ adb shell pm list packages -f
```

```
C:\Users\Maksim_Meshcheriakov>adb shell pm list packages -f
package:/system/app/FilterProvider/FilterProvider.apk=com.samsung.android.provider.filterprovider
package:/data/app/com.skype.raider-1/base.apk=com.skype.raider
package:/data/app/com.samsung.android.gearoplugin-1/base.apk=com.samsung.android.gearoplugin
package:/system/app/RootPA/RootPA.apk=com.gd.mobicore.pa
package:/system/app/GalaxyAppsWidget_Phone_EssentialsOnly/GalaxyAppsWidget_Phone_EssentialsOnly.apk=com.sec.android.widgetap
p.samsungapps
package:/data/app/com.google.android.youtube-1/base.apk=com.google.android.youtube
package:/system/priv-app/SFinder_v4/SFinder_v4.apk=com.samsung.android.app.galaxyfinder
```

Copy files to/from device

- Push a file to device
  ```
  $ adb push path2local_file path2remote_file
  ```
- Pull a file from device
  ```
  $ adb pull path2remote_file path2local_file
  ```
- Example:
  ```
  $ adb push foo.txt /sdcard/foo.txt
  ```

Get logs with Logcat

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the ‘Log’ class.

```
$ adb logcat
```

```
$ adb logcat --help
```

or

```
$ adb shell
```

```
> logcat
```

Logcat options

- Default output is ‘stdout’, but you can write output down to required file with -f <*filename*> option
- output filtering:
  Verbose (lowest) / Debug / Info / Warning / Error / Fatal / Silent (highest))
- output formatting with -v <*format*> option

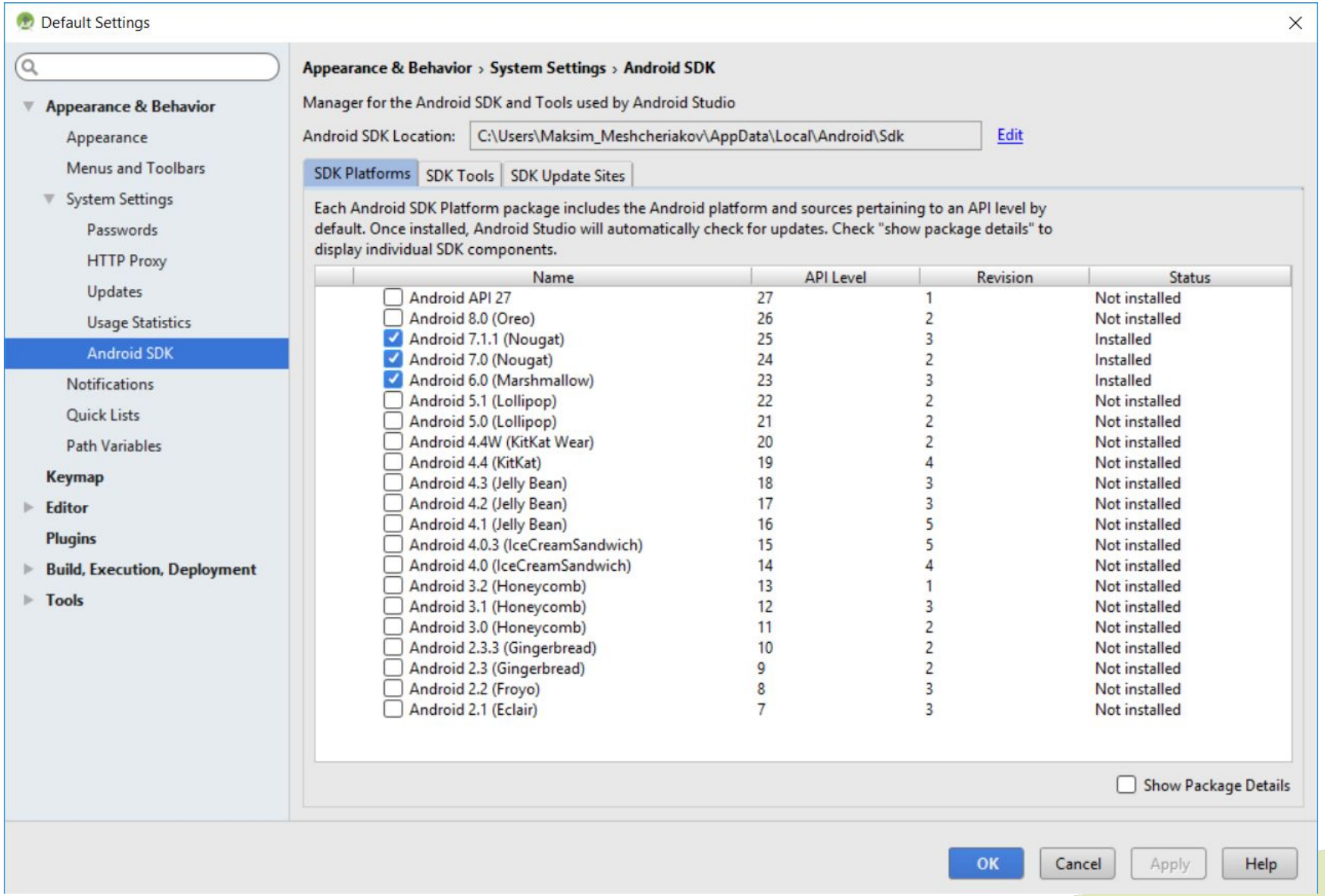
The full syntax description:
https://developer.android.com/studio/command-line/logcat.html$Syntax

Command line tools info

The root of information about Android-related command line tools:

https://developer.android.com/studio/command-line/index.html

Update platforms



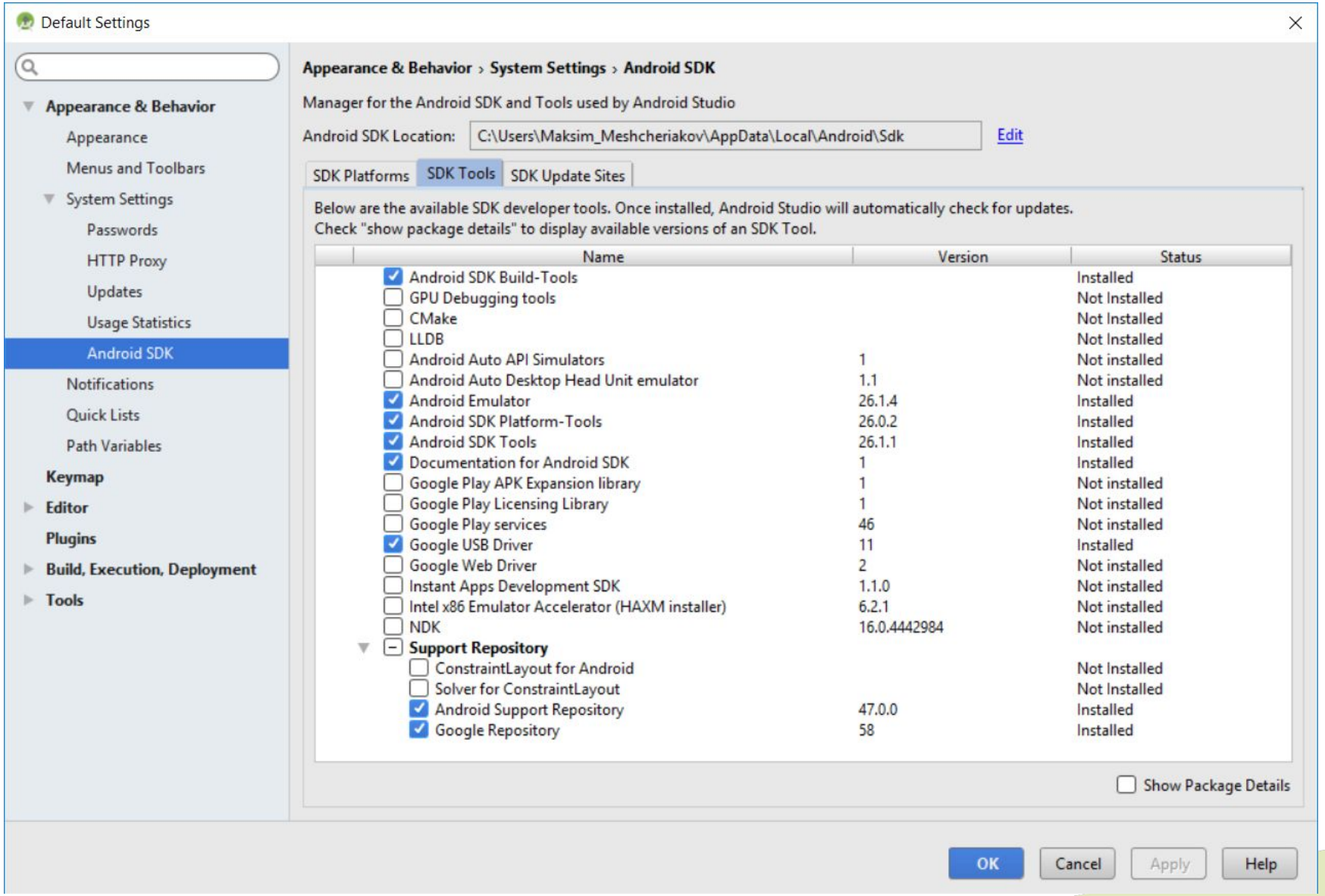
Tools >

Android >

SDK Manager >

SDK Platforms

Update tools



Tools >

Android >

SDK Manager >

SDK Tools

Appium

Business needs

- EPAM as a global IT service company needs in clear and easy to learn and implement procedure(s) of mobile test automation that will be applicable worldwide
- These procedures should be based on a limited set of tools. These tools should be easy to learn and implement as well
- Engineers can’t learn cute new tools again and again: it’s OK for personal professional development, not to meet business needs

Required features of tool

Important: we are talking about corporate-wide tool for hundreds engineers who are working on hundreds projects with their own peculiarities

- Covers main target mobile platforms
  Android, iOS
- Use the investments made
  knowledge, expertise, processes, infrastructure, software, hardware
- Not expensive
  free open source is preferable
- Easy to learn

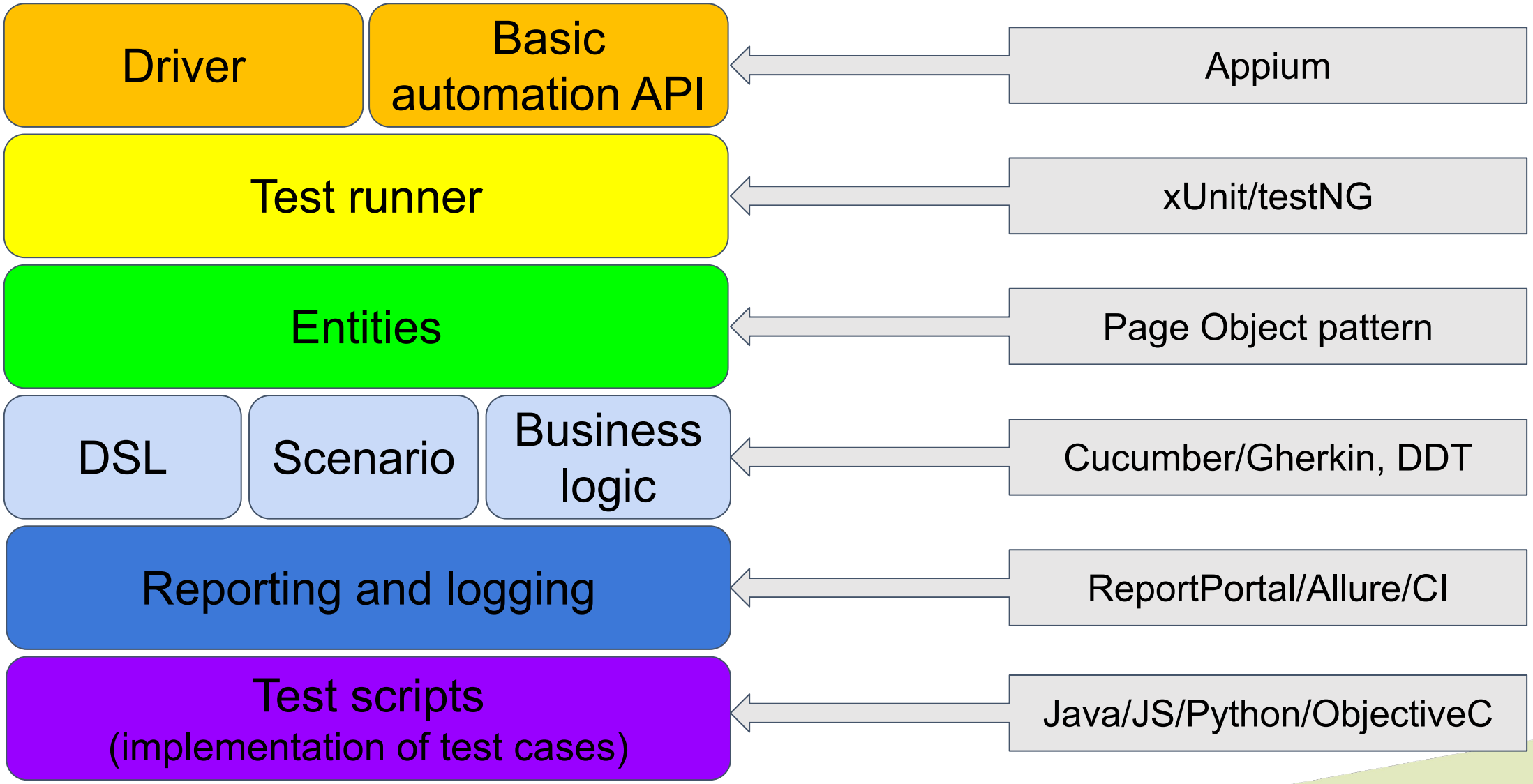
Appium advantages

Capabilities		Selenium	Appium	Espresso	XCTest UI	Calabash
Application Type Support	Mobile Web	Yes	Yes	No	No	No
	Native	No	Yes	Yes	Yes	Yes
	Hybrid	No	Yes	Yes	Yes	Yes
	Desktop Web	Yes	No	No	No	No
Supported Mobile Platforms	Android	Yes	Yes	Yes	No	Yes
	iOS	Yes	Yes	No	Yes	Yes
Supported Context	App Context	N/A	Yes	Yes	Yes	Yes
	Device Context	N/A	Yes	No	Yes	No
Scripting Development Language	Java	Yes	Yes	Yes	No	No
	Python	Yes	Yes	No	No	No
	C#	Yes	Yes	No	No	Yes
	Ruby	Yes	Yes	No	No	Yes
	Java Script	Yes	Yes	No	No	No
	Perl	Yes	Yes	No	No	No
	ObjectiveC/Swift	Yes	Yes	No	Yes	Yes

Appium advantages

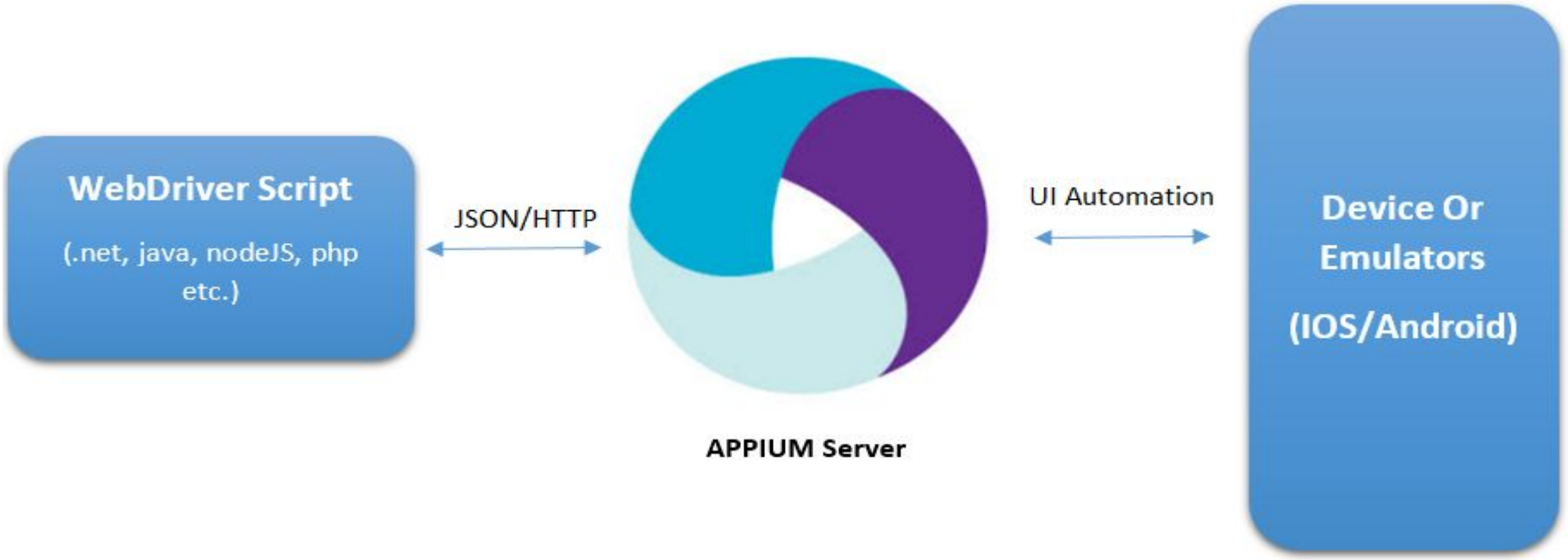
Capabilities		Selenium	Appium	Espresso	XCTest UI	Calabash
Supported Visual Object	N/A	N/A	Yes (with extension)	No	No	No
Integrated with IDEs	Eclipse	Yes	Yes	Yes	No	Yes
	Android Studio	Yes	Yes	Yes	No	Yes
	Xcode	Yes	Yes	No	Yes	Yes
	Visual Studio	Yes	Yes	No	No	Yes
	IntelliJ IDEA	Yes	Yes	Yes	No	Yes
Supported by Mobile OS Vendors (no gaps)	N/A	No	No	Yes (Google)	Yes (Apple)	No
Real devices	N/A	Yes	Yes	Yes	Yes	Yes
Simulators/Emulators	N/A	Yes	Yes	Yes	Yes	Yes
Automatic sync between test actions and App UI	N/A	No	No	Yes	No	No
Test Recorder	N/A	No	No	Yes(Version 2.2+)	Yes	No
Community Support	N/A	Active	Very Active	Google	Apple	Average
GitHub Rankings	N/A	787 Stars	2444 Stars	NA	NA (*KIF 3017 Stars)	1930 Stars

Layers of test automation harne



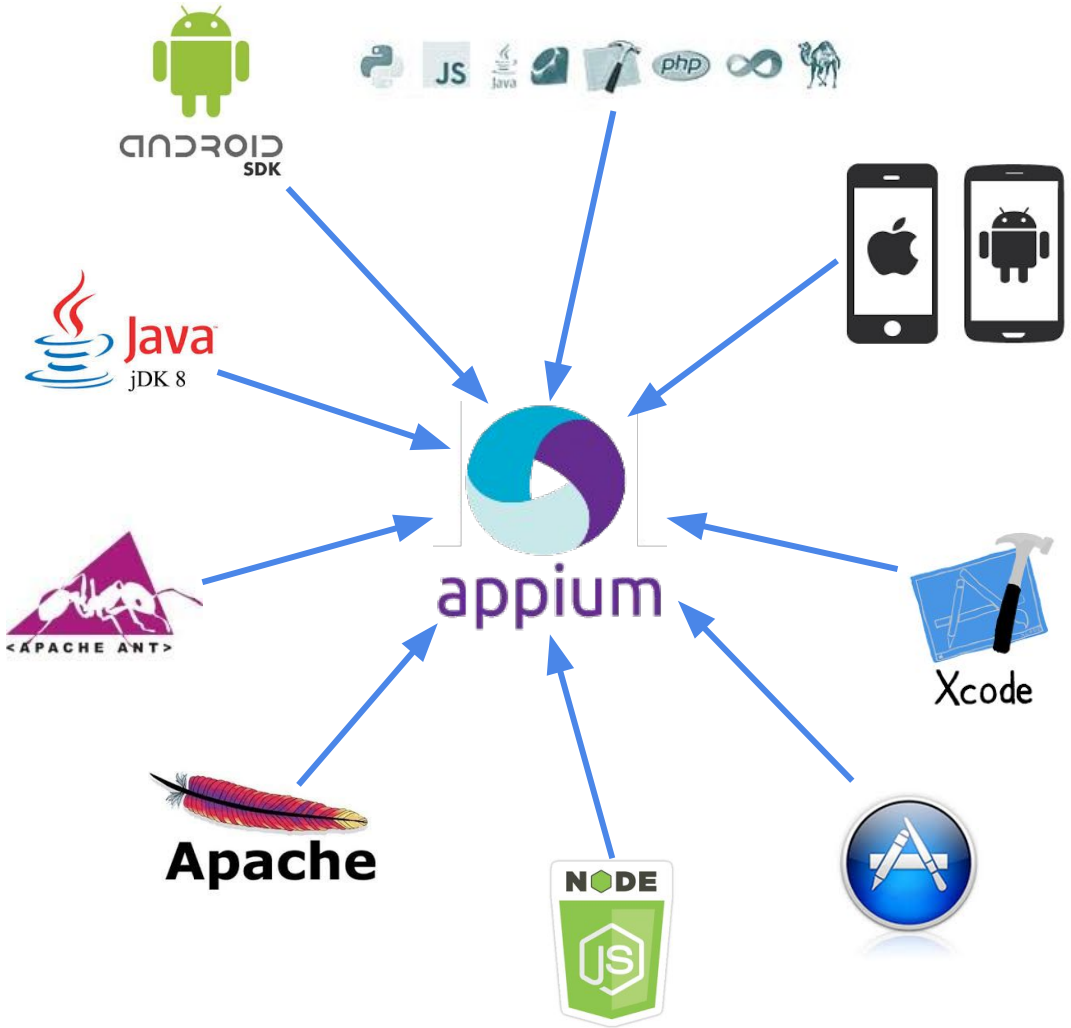
The essence of Appium

Appium server



Appium client: libraries (in Java, Ruby, Python, PHP, JavaScript, and C#) which support Appium's extensions to the WebDriver protocol

Prerequisites



About installation of JDK-8, Android SDK, mobile devices and emulators please refer to module
“General environment settings for Android platform”

The most efficient, cross-platform way to use Appium as a node module.

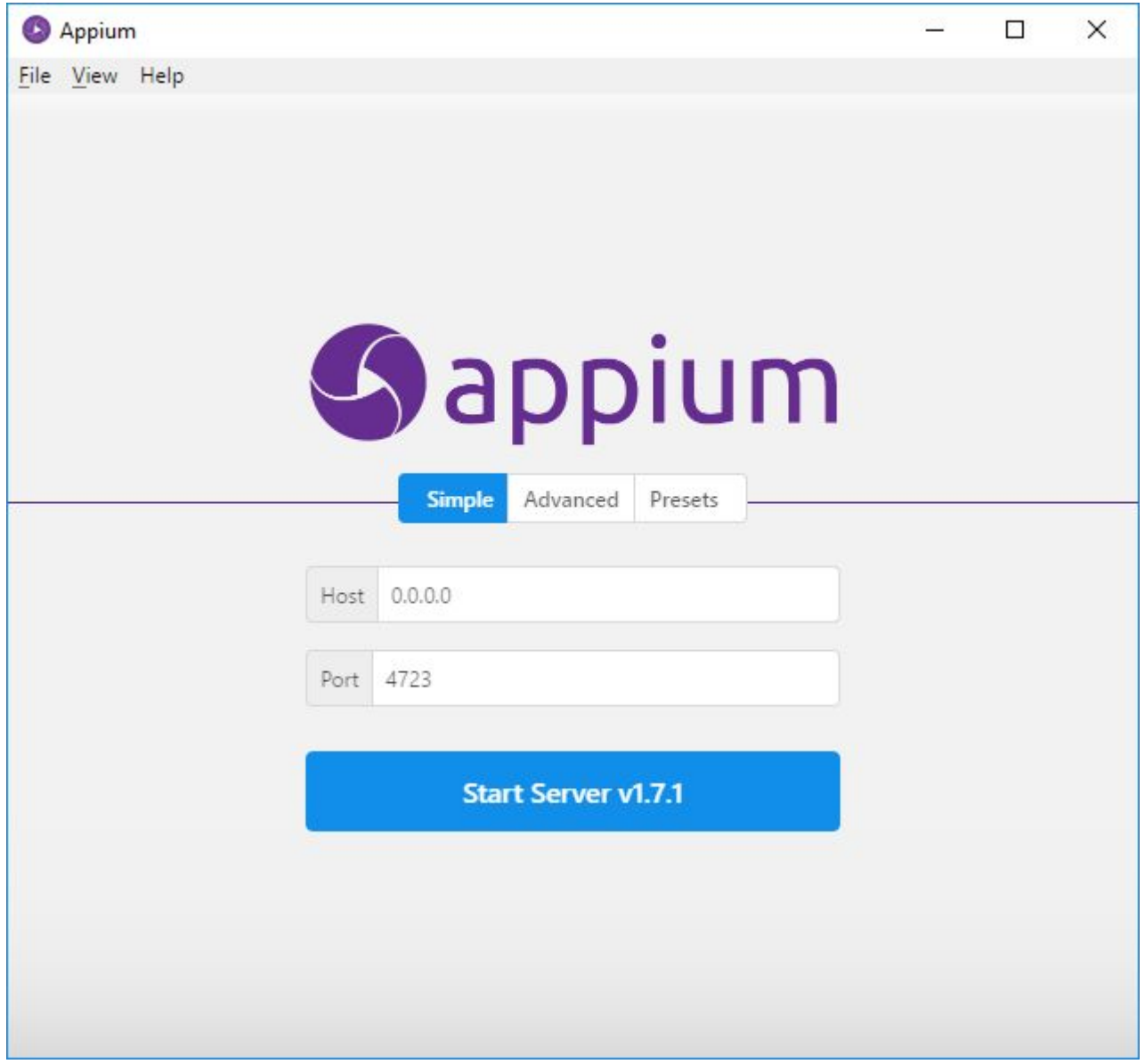
1. Download *Node.js* package suitable for your computer: https://nodejs.org/en/download/, and install it.
2. Use *appium-doctor* to check Appium preconditions:
   a. Install: > npm install -g appium-doctor
   b. Check: > appium-doctor

Appium Desktop installation

Appium *Server* and *Inspector* in desktop GUIs for Mac, Windows, and Linux

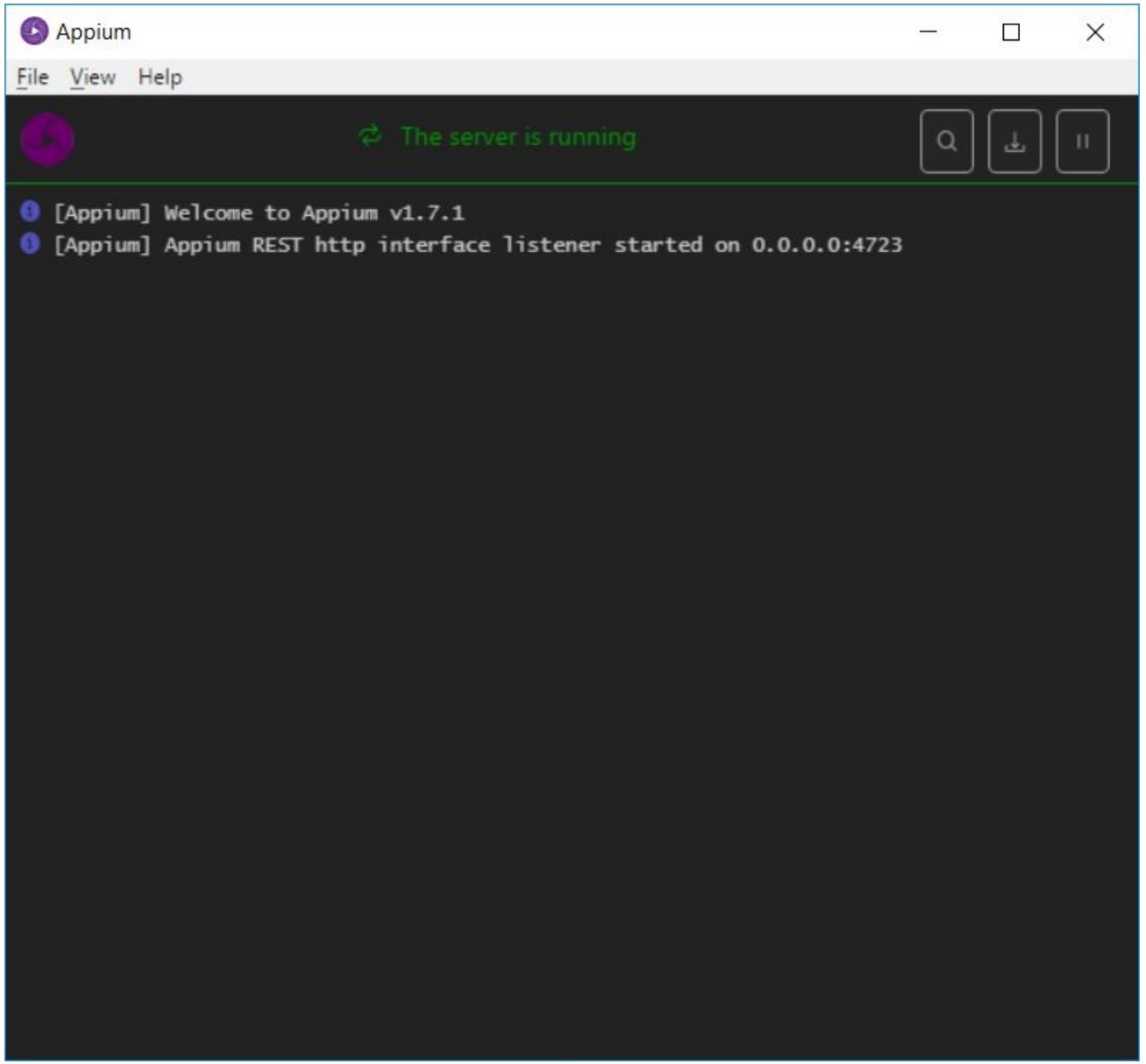
1. Download Appium Desktop from here: https://github.com/appium/appium-desktop/releases
2. Short usage instructions (scroll down to text): https://github.com/appium/appium-desktop
3. Install Appium desktop according your system rules
4. Find other Appium-related software packages here: https://github.com/appium
5. Visit appium.io to get more information

Run Appium with default settings



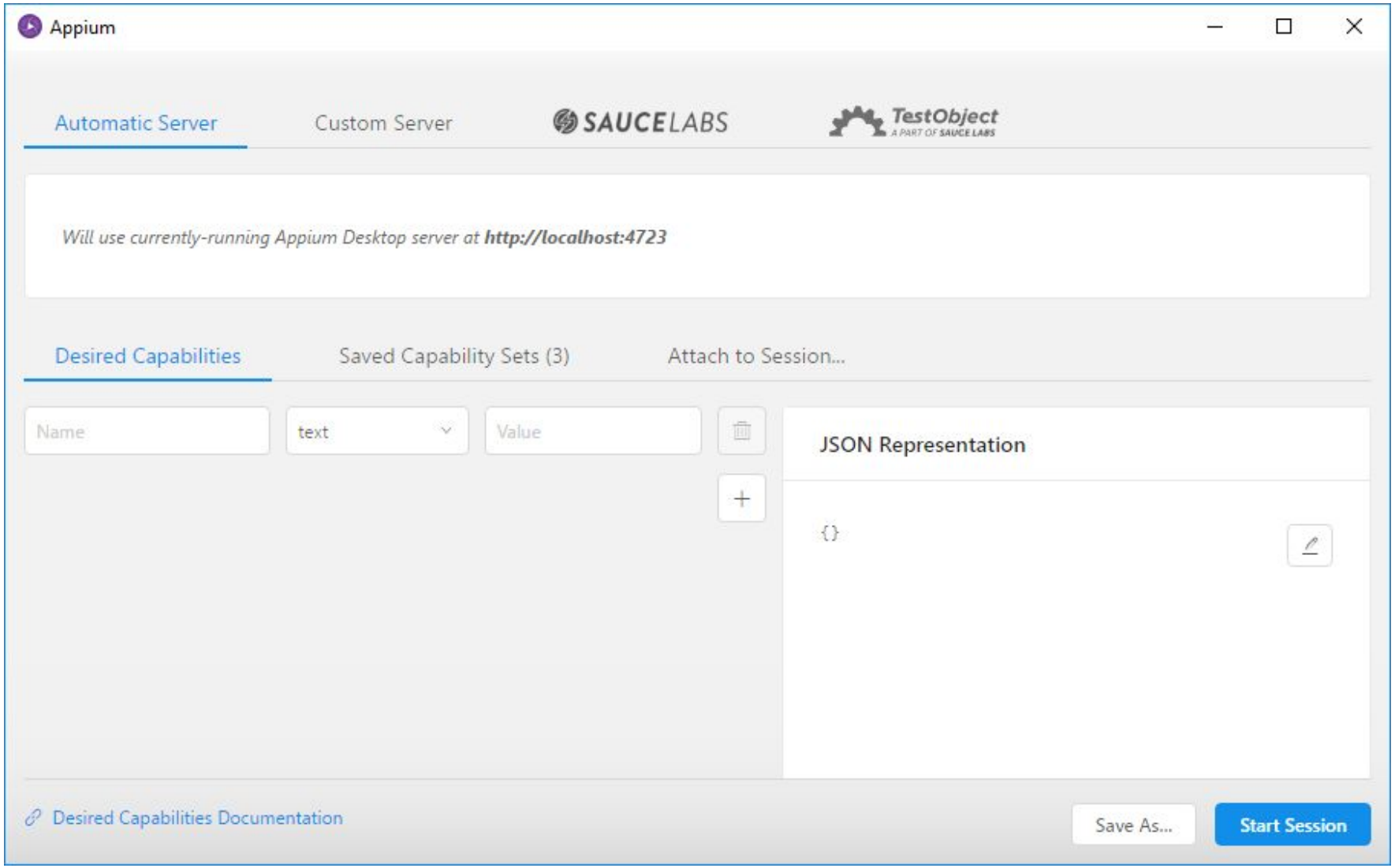
1. Start an emulator or attach a device
2. Run Appium DT by clicking on desktop
3. Use default “simple” settings: Appium server will run locally (0.0.0.0:4723)
4. Press “Start Server x.x.x”

Start Appium Inspector

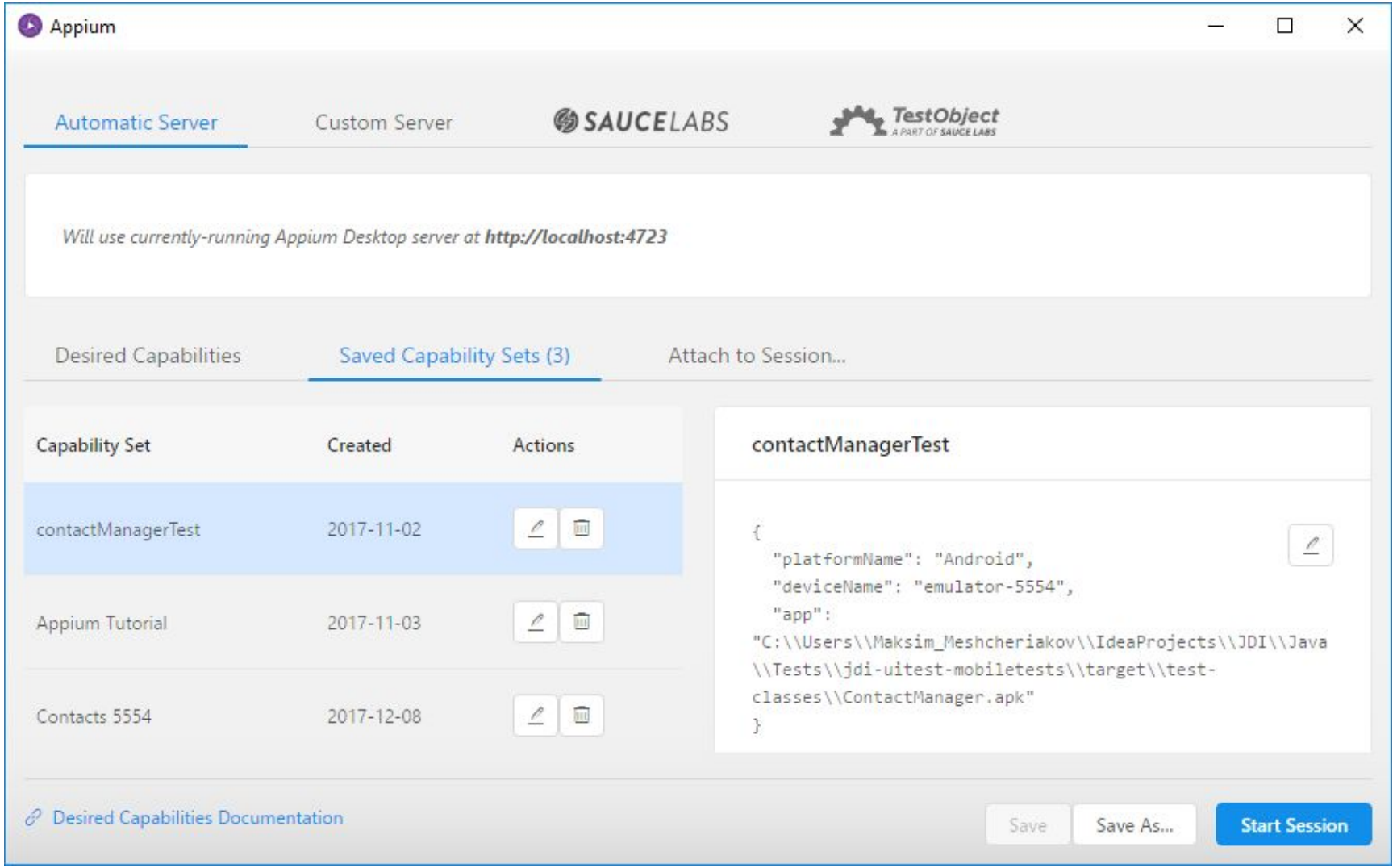


Click to start an Inspector session

Default capabilities screen

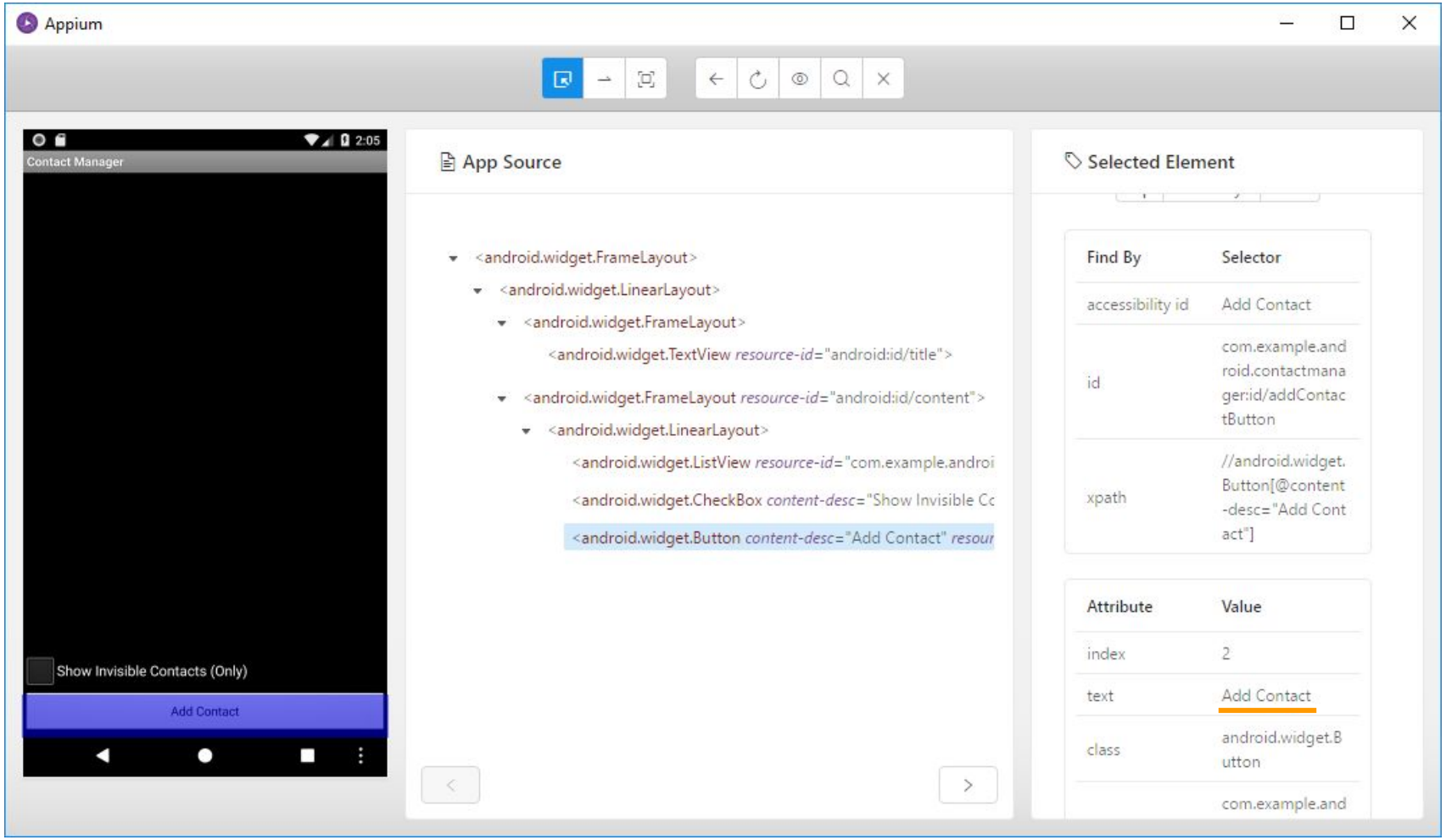


Set of required capabilities



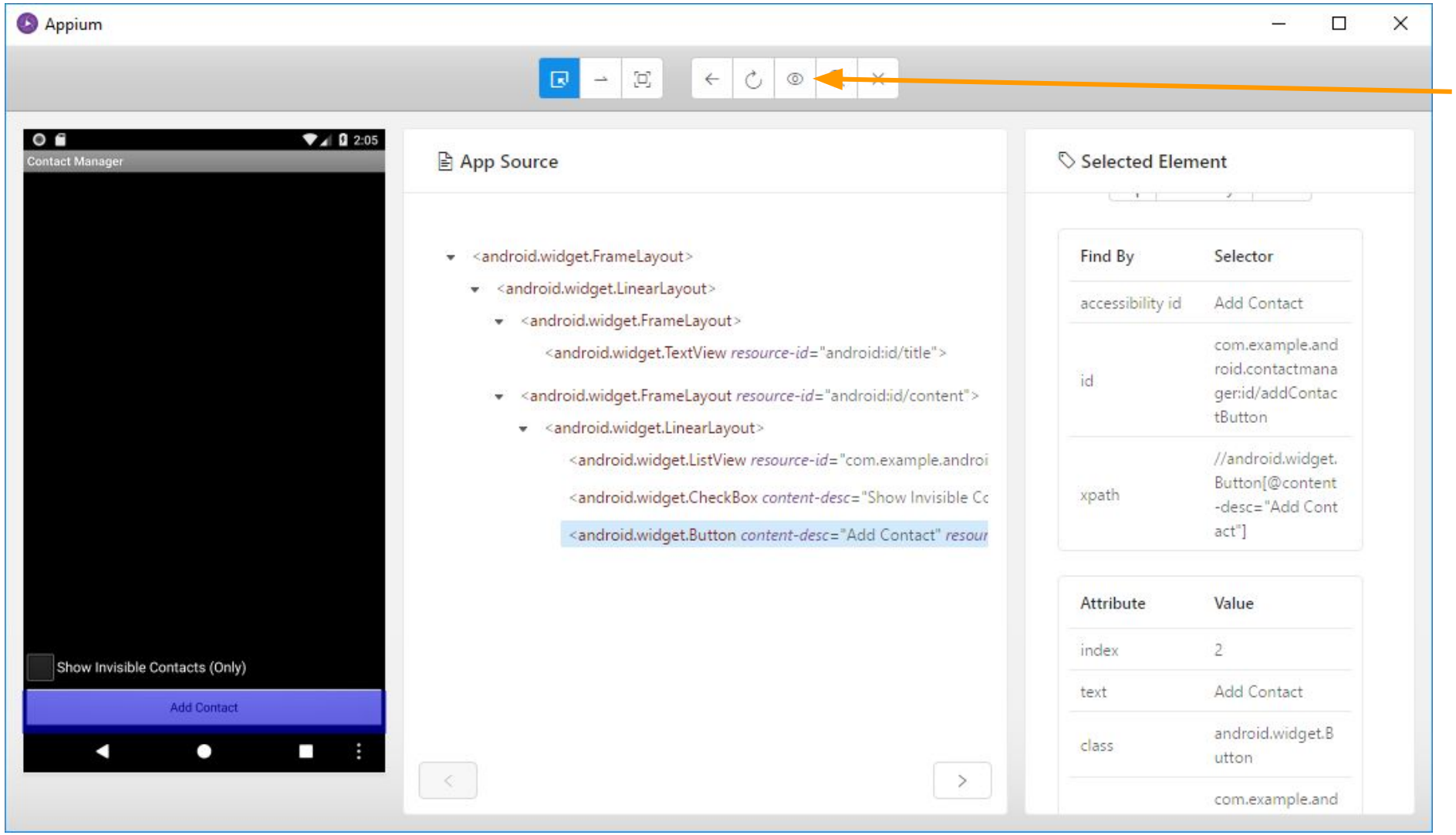
- Use “Save As” button to store capability set for further usage
- Saved sets will be available
- Click “Start Session” button to run Inspector session

Select desired element



Appium Inspector tool more convenient than Device Monitor one

Recorder



Recorder tool:

- not for production code
- help explore Appium API
- demonstration

It is a learning tool, not a robust code generation feature

Plan

- Basic tools
  - Physical devices, emulators and cloud solutions
  - Appium
- How to write auto-tests
- Basics of Mobile Cloud Services
- Tips and tricks

QUESTIONS

maxim.mescheryakov
maksim_meshcheriakov@epam.com