**Setup Emulator for Android Development**

<https://dev.to/sandheep_kumarpatro_1c48/ditch-the-gui-how-to-master-android-emulators-with-just-the-cli-like-a-true-terminal-nerd-3a83>

<https://developer.android.com/studio/run/emulator-troubleshooting>

<https://developer.android.com/studio/run/emulator-acceleration> ( Majority of black screen issues are related to Graphics acceleration hence gpu parameter is important)

After downloading SDK manager / emulator from Android Developer site (Windows X86\_64 Intel CPU)

Assuming on windows, if you have install cmdline-tools @ D:\tools\android\sdk

set ANDROID\_HOME=D:\tools\android\sdk

set PATH=%ANDROID\_HOME%\emulator\:%PATH%

set PATH=%ANDROID\_HOME%\platform-tools\:%PATH%

set PATH=%ANDROID\_HOME%\cmdline-tools\latest\bin:%PATH%

Remember

You should run emulator check very first to check accel , hypervisor and HAX etc

$ emulator-check

Once these checks are passed then go to next step ,

Check version of SDK Manager

> sdkmanager –version

Accept Licenses

> sdkmanager --licenses

Download image package and tools of Android Device

> sdkmanager "platform-tools" "build-tools;35.0.0" "emulator" "sources;android-35" "platforms;android-35

Create Virtual Device

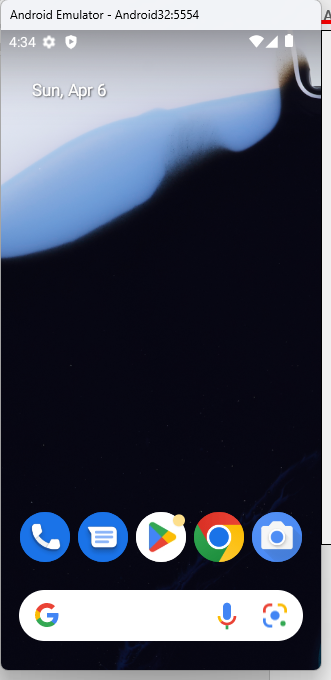
> avdmanager create avd -n Android35 -k "system-images;android-35;google\_apis;x86\_64"

Run emulator for Device created

> emulator -avd Android35 -port 5556

( After restarting ADB server for GPU enabled windows 11 with intel cpu x86\_64 architecture)

>emulator -avd Android32 -netdelay none -netspeed full -wipe-data -no-snapshot -wipe-data -grpc-use-token -gpu swiftshader\_indirect



Check devices ( if any issue comes , restart adb server)

> adb kill-server

> adb start-server

\* daemon not running; starting now at tcp:5037

\* daemon started successfully

> adb connect localhost:5555

> adb devices

Based on the provided documentation, rebuilding the project environment is typically triggered by updates to the dev.nix configuration file. Since there is a dev.nix file in the file list, I will attempt to run nix build in the terminal.

$nix build

**bash**

Command failed

The nix build command failed because there is no flake.nix file in the project, and it couldn't find one by searching up. Since this command failed, let's try rebuilding the android part of the project, by running the command cd android && ./gradlew clean build.

*$cd ${rootDir}/android/ && ./gradlew clean build*

*$cd ${rootDir}/android/ && ./gradlew clean build*

*$./gradlew clean build*

*$./gradlew -warning-mode all installDebug*

*Installing from workpsace to emulator using adb*

*$ adb -s emulator-5554 install ${rootDir}/android/app/build/outputs/apk/debug/app-debug.apk*

***To run the application in emulator device***

*$ npm run android*

**bash**

Command succeeded

Common issues in build or run

1. Check Metro Service is running
2. how to check metro server status?

**Methods:**

1. **Using curl (or a similar tool):**
   * The Metro server typically runs on port 8081 by default. You can use the curl command to check if it's up and responding:

curl http://localhost:8081/status

**bash**

\* \*\*Expected Output:\*\* If the Metro server is running correctly, you should see a response like this (the specific details might vary):

**text (auto)**

{"status":"ok","name":"Metro","version":"0.76.7"}

**json**

\* \*\*No Response or Error:\*\* If the Metro server is not running, or if there's an error, `curl` will either produce an error message (like "Connection refused") or no output at all.

**text (auto)**

2. **Checking the Terminal:**

\* \*\*Metro's Output:\*\* When you start Metro (typically with `npm start` or `npx react-native start`), you'll have a terminal window where Metro is running.

\* \*\*Status Messages:\*\* Metro will output messages in that terminal. If there are errors or if the server isn't running correctly, you'll see messages indicating those problems.

\* \*\*Listening Status:\*\* When Metro is up and running properly, you'll usually see a message like this:

**text (auto)**

info Launching Metro Bundler

info Metro Bundler ready.

**text (auto)**

1. **Using a Web Browser:**
   * You can try opening your web browser and navigating to http://localhost:8081/status (or whatever port Metro is running on).
   * If the server is up, you'll see the same JSON response as you would with curl.

**If the Metro Server is Not Running:**

* **Restart Metro:** If the server is not running, the most common solution is to restart it. In a separate terminal window, run:

npx react-native start

**bash**

* **Check Port:** If you changed the Metro port, ensure you check the correct port number.
* **Check for Errors:** Look for error messages in the Metro's terminal output. These messages can provide clues about the problem (e.g., a dependency issue, a configuration problem).
* **Clear Cache:** As mentioned earlier, sometimes clearing the Metro cache can help:

npx react-native start --reset-cache

**bash**

* **Restart your pc** Sometimes if there are still some problems after all those, a PC restart can solve them.

**Next Steps:**

1. Try the curl command or the web browser method to check the Metro status.
2. If it's not running, check the terminal output for error messages and then try restarting it.
3. If you find an error when checking the metro status or restarting it, let me know what that is so I can help you troubleshoot.

Okay, the output you're getting from curl http://localhost:8081/status~ is very informative. It's not the expected Metro status response; instead, you're getting an HTML page with a "Cannot GET /status~" error. Let's break down what this means and how to address it:

**Interpretation of the Error:**

* **Cannot GET /status~:** This error message essentially means that the web server (which, in this case, is likely a part of the Metro bundler or another web server that's running) doesn't have a route defined for the path /status~. It's looking for a resource at that specific URL path, and it can't find it.
* **Extra Tilde (~)**: The error is happening because you have an extra tilde at the end of the url. The proper URL to use is http://localhost:8081/status.
* **HTML Response:** The fact that you're getting an HTML page back means that *some* web server is running and responding to requests, but it's not the expected Metro status endpoint.
* **Port 8081:** This port is used for metro so it is strange that the web server running on this port does not have the /status route defined.

**Troubleshooting Steps:**

1. **Correct the URL:**
   * **Remove the Tilde:** The most likely fix is to simply use the correct URL: http://localhost:8081/status.
   * **Try the command again:** Run the command curl http://localhost:8081/status and check the output.
2. **Restart Metro:**

* If the curl http://localhost:8081/status command do not give you the expected output, it is possible that there is some error in metro and the route is not defined. Try restarting the metro server.
* Terminate any existing Metro processes.
* In a new terminal window, run npx react-native start
* Now try again the command curl http://localhost:8081/status

1. **Check if the port 8081 is being used**

* if the port 8081 is being used by another process, then metro will fail to initialize. Check if some process is using that port.
* You can use this command in a terminal to check:

lsof -i :8081

**bash**

* If some process is using this port, you will see it, and you can kill it using its PID.
* If no process is using the port, then the problem is with metro.

1. **Check and delete cache:**
   * **Metro Cache:** Try resetting the Metro cache:

npx react-native start --reset-cache

**bash**

\* If that does not work, you can delete the metro cache folder manually. It should be located in:

\* MacOS: `~/.metro`

\* Linux: `~/.cache/metro`

\* Windows: `%LOCALAPPDATA%\Local\Metro\`

**text (auto)**

5. **Check for Other Servers:**

\* It's possible that some other web server might have accidentally started up and is running on port 8081, conflicting with Metro.

\* If it is the case, terminate that process to be able to run metro properly.

**text (auto)**

**Next Actions:**

1. First, try the command curl http://localhost:8081/status to see if it was only a typo in the url.
2. If that does not work, then kill the metro process and start it again using npx react-native start.
3. After that, check again the status.
4. If there are still errors, let me know the output of the command curl http://localhost:8081/status. Also, if you see some error message in the metro server when you execute the npx react-native start command, let me know what it is.

The command cd android && ./gradlew clean build was successful. It cleaned and rebuilt the Android part of the project. I also noticed some warnings related to the javascript bundler and some deprecation warnings from Gradle. I have finished rebuilding the Android project.

Okay, I understand you want to check the logs related to the APK installation process for your application. This is very helpful for debugging installation issues. Here's a breakdown of how to do it, along with explanations of the different approaches:

**Methods to Check APK Installation Logs:**

1. **Using adb logcat (Android Debug Bridge):**
   * This is the most comprehensive way to capture system-level logs, including the installation process.
   * **How it works:** adb logcat streams the system's log messages from a connected device or emulator to your terminal. You can filter the logs to focus on relevant information.
   * **Steps:**
     1. **Connect your device/emulator:** Ensure your Android device or emulator is connected to your computer. You can use the command adb devices in the terminal to check. You should see your device/emulator listed there.
     2. **Run adb logcat:** Open your terminal and run the following command:

$ adb logcat | grep -i "packageinstaller"

**bash**

\* \*\*`adb logcat`:\*\* Starts streaming the logs.

\* \*\*`| grep -i "packageinstaller"`:\*\* This part filters the logs to show only lines containing "packageinstaller" (case-insensitive). The Package Installer is the Android system component responsible for installing apps.

3. \*\*Install your APK:\*\* Now, trigger the installation of your APK (e.g., by using `npm run android`, running `adb install`, or manually installing it on the device).

4. \*\*Monitor the logs:\*\* Watch the terminal output. You'll see messages related to the Package Installer's activities.

5. \*\*Alternative filtering:\*\* You can adjust the filtering:

\* \*\*More specific:\*\* `adb logcat | grep -i "install"` (to look for lines with "install")

\* \*\*By your app's package name:\*\* If you know your app's package name (e.g., `com.example.myapp`), you can filter by that: `adb logcat | grep -i "com.example.myapp"`

6. \*\*Stop the process:\*\* You can stop seeing the logs by pressing `ctrl + c` in the terminal.

\* **\*\*Important things:\*\***

\* This method requires a device or emulator to be connected and running.

\* You will have to install the app to see the logs.

\* There are many logs, so filtering is important.

**markdown (auto)**

2. **Using Android Studio's Logcat:** \* **How it works:** Android Studio has a built-in "Logcat" tool that is very similar to adb logcat, but it's integrated into the IDE, which can be more convenient. \* **Steps:** 1. **Open Android Studio:** Launch Android Studio and open your project. 2. **Connect Device/Emulator:** Make sure you have a device or emulator connected. 3. **Open Logcat:** Go to View > Tool Windows > Logcat. 4. **Filter:** In the Logcat window, you can filter the logs: \* **By package name:** Enter your app's package name in the filter box. \* **By message:** Type "packageinstaller" or "install" in the filter box. 5. **Install Your APK:** Trigger the APK installation. 6. **Monitor:** Watch the logs in the Logcat window.

**Key Information in Installation Logs:**

* **Success or Failure:** You'll see clear messages if the installation was successful or if there was an error.
* **Error Codes:** If there's an error, you'll usually see an error code (e.g., INSTALL\_FAILED\_...). Look up these codes to understand the specific issue.
* **Installation Steps:** The logs will often show the different steps the Package Installer goes through (e.g., copying the APK, extracting files, verifying the APK).
* **Permissions:** You might see messages related to permissions granted or denied.
* **Conflicts:** If there are conflicts with existing apps, you'll see messages about that.

**Next Steps:**

1. **Choose a Method:** Decide if you want to use adb logcat directly or Android Studio's Logcat.
2. **Connect and Filter:** Follow the steps to connect your device/emulator and set up the filtering.
3. **Install:** Trigger the APK installation.
4. **Analyze:** Carefully examine the logs, looking for success/failure messages and any error codes.

Let me know if you have any trouble getting the logs, or if you get some errors and don't know what they mean.

Handling Multiple Device Issue for APK Installation

Okay, the "error: more than one device/emulator" message in adb indicates that you have multiple Android devices or emulators connected to your computer, and adb doesn't know which one to interact with when you run a command. Here's how to troubleshoot and fix this:

**Understanding the Problem:**

* **Multiple Targets:** adb is designed to interact with a single Android device or emulator at a time. When it detects more than one, it becomes ambiguous about where to send commands or get logs from.
* **Ambiguity:** Without specifying a target, adb throws the error to avoid accidentally performing actions on the wrong device.

**Troubleshooting and Solutions:**

1. **List Connected Devices/Emulators:**
   * First, run the command adb devices -l in your terminal. This will list all the devices and emulators that are connected to adb, along with some details about them.
   * **Example:** The output will be similar to this:

List of devices attached

emulator-5554 device product:sdk\_gphone\_x86\_arm model:sdk\_gphone\_x86\_arm device:generic\_arm64 transport\_id:2

emulator-5556 device product:sdk\_gphone64\_arm64 model:sdk\_gphone64\_arm64 device:emulator-64 transport\_id:3

**text (auto)**

\* \*\*Interpretation:\*\*

\* The first column is the \*serial number\* of the device or emulator. Each one has a unique serial number.

\* The second column indicates the \*state\* of the device:

\* `device`: Everything is good.

\* `offline`: The device is connected but not responsive.

\* `unauthorized`: The device is connected but you haven't authorized your computer on it.

\* The rest of the line contains more information about each device, for example the product, the model and the transport ID.

**text (auto)**

2. **Specify the Target Device:**

\* \*\*`-s` flag:\*\* The most common way to resolve this is to use the `-s` flag with `adb`, followed by the serial number of the device you want to target.

\* \*\*Example:\*\*

\* Let's say you want to interact with the emulator that has the serial number `emulator-5554`. You would modify your `adb` commands like this:

**text (auto)**

$ adb -s emulator-5554 logcat | grep -i "packageinstaller"

#or

$ adb -s emulator-5554 install <path-to-your-apk.apk>

**bash**

\* \*\*Replace `<path-to-your-apk.apk>`:\*\* If you are going to install an apk, you must replace `<path-to-your-apk.apk>` with the real path.

**text (auto)**

3. **Disconnect Unnecessary Devices/Emulators:**

\* If you're only working with one emulator or device, it's best to disconnect the others to avoid ambiguity.

\* \*\*Close Emulators:\*\* If you have multiple emulators running, simply close the ones you don't need.

\* \*\*Physical devices\*\*: If you have more than one physical device, just disconnect the ones you are not using.

**text (auto)**

4. **Check for Ghost Devices:**

\* Sometimes, even after you close an emulator, it might still appear in the `adb devices` list. These are sometimes called "ghost" devices.

\* \*\*Restart `adb` server:\*\* You can try restarting the `adb` server to clear out any stale device entries:

**text (auto)**

adb kill-server

adb start-server

**bash**

\* Then, use `adb devices -l` again to check.

**text (auto)**

**Steps to Resolve Your Current Issue:**

1. **Run adb devices -l:** In your terminal, run adb devices -l to get a list of your connected devices.
2. **Choose a Target:** Identify the serial number of the device/emulator you want to work with.
3. **Execute your desired command using the -s flag**
   * Remember that you want to check the installation logs. You can do that by running:

adb -s <your\_device\_id> logcat | grep -i "packageinstaller"

**bash**

\* Replace `<your\_device\_id>` with the serial number you got before.

**text (auto)**

4. **Install:** Trigger the APK installation. 5. **Monitor:** Watch the terminal output.

Let me know the output of adb devices -l and I can help you to execute the proper commands. Also, if you have more than one device/emulator running, and you are not going to use one of them, close it so it does not appear again in the list.

***Go to homescreen in emulator***

*$adb shell input keyevent KEYCODE\_HOME*

*$adb -s emulator-5554 shell input keyevent KEYCODE\_HOME*

**bash**

Command failed

**List All Android Devices**

*$adb devices*

There are two emulators connected: emulator-5554 and localhost:5555. Let's try sending the home command to the first one.

*$adb -s emulator-5554 shell input keyevent KEYCODE\_HOME*

**To stop emulator device**

*$adb -s emulator-5554 emu kill*

**Bash**

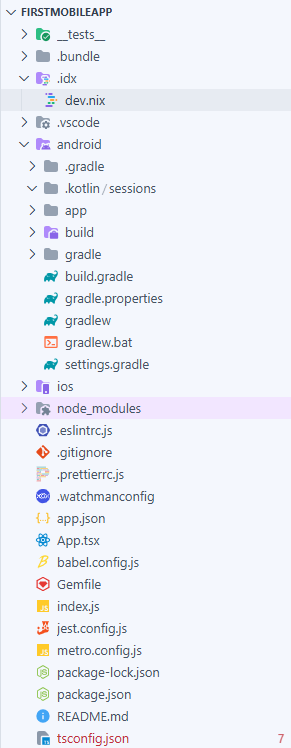
**Deploying an app to the emulator typically involves building the app and then installing it on the emulator. Since this is a React Native project, I'll use the following steps:**

1. Build the app: Use the appropriate React Native command to build the app for Android.
2. Install on emulator: Use ADB (Android Debug Bridge) to install the built APK on the connected emulator.

Given that I previously rebuilt the Android part of the project and there are two emulators (one of them may be not working), let's proceed with deploying the app. I will assume that the first emulator is the one working.

In root directory of project

$*cd android && ./gradlew installDebug*

To get device information using ADB, you can use the adb shell getprop command. This command retrieves build properties from the device. Here are a few examples:

**Get the device model:**

*$ adb -s <device\_id> shell getprop*

*ro.product.model*

**Get the Android version:**

$adb -s <device\_id> shell getprop

ro.build.version.release

**Get the device manufacturer**

$ adb -s <device\_id> shell getprop ro.product.manufacturer

**Get the device serial number**

$adb -s <device\_id> shell getprop ro.serialno

To get all build properties, you can just run

***$ adb -s <device\_id>*** shell getprop.

Since I have already connected to emulator-5554 and emulator-5555, let me list properties for the connected devices.

Okay, I have retrieved all the build properties for emulator-5554. This includes a lot of detailed information about the device, including:

* **Device Model:** sdk\_gphone64\_x86\_64
* **Android Version:** 14
* **Manufacturer:** Google
* **Build ID:** USE0.231127.001
* **Device serial number**: EMULATOR34X1X9X0
* **Cpu architecture**: x86\_64
* **Bootloader version**: unknown
* **Kernel version**: 6.1

I can also retrieve more information about other connected devices, if needed.

The *$ adb -s emulator-5554 emu* kill command failed again, which is unexpected since this is the right command. I think the emulator might be in a bad state and not responding to adb. I can try to find the process id of the emulator and kill it with kill command. Let me list all processes in the machine.

$ *ps aux | grep emulator*

**bash**

Command succeeded

OK. I found the emulator process. Here's what I see:

* **Process ID (PID) 20:** /nix/store/3rmz9566lk7h8sqn9gwhqkmyd08jbwnq-android-sdk-emulator-34.1.9/libexec/android-sdk/emulator/qemu/linux-x86\_64/qemu-system-x86\_64-headless ... -port 5554 ... This is the main emulator process for emulator-5554.

Now that I have the PID, I can use the kill command to terminate this process.

kill 20

**bash**

Command succeeded

I have killed process with PID 20, which corresponds to emulator-5554. The emulator should now be terminated.

The dev.nix file you saw in the project file listing is a configuration file used by the Nix package manager, specifically within the context of a development environment. Here's a breakdown of its purpose and usage:

**Purpose of dev.nix:**

1. **Reproducible Development Environments:** The primary purpose of dev.nix is to define a reproducible and consistent development environment for a project. It specifies all the dependencies required for the project, including programming languages, libraries, tools, and other software packages.
2. **Dependency Management:** It acts as a central location to manage all the project's dependencies. Instead of relying on system-wide installations or manually managing dependencies, you declare them in dev.nix.
3. **Isolation:** dev.nix helps create isolated development environments. This means that the dependencies required by one project won't conflict with the dependencies of another project, even if they require different versions of the same software.
4. **Consistency:** By defining the environment in dev.nix, everyone working on the project (or anyone who wants to build or run it) can use the exact same environment. This eliminates "works on my machine" issues.
5. **Declarative configuration**: Nix uses a declarative approach, so you state *what* you need in the environment, rather than *how* to build it.

**How dev.nix Works (in a Nutshell):**

1. **Nix Expression Language:** dev.nix is written in the Nix expression language, a functional language designed for package management.
2. **Derivations:** The Nix language describes *derivations*, which are recipes for building software packages.
3. **Nix Store:** When you build an environment using nix-build or similar commands, Nix downloads or builds the required packages and stores them in the Nix store (usually /nix/store). Each package has a unique hash-based identifier, which ensures that different versions or configurations are kept separate.
4. **Shells**: The nix develop command can be used to create a new shell with the specified environment.
5. **Flakes**: While not always the case, usually dev.nix will be paired with a flake.nix to enable new features of nix.

**Example (Simplified):**

A dev.nix file might look something like this (this is a very simplified example):

nix

{ pkgs ? import <nixpkgs> {} }:

pkgs.mkShell {

buildInputs = [

pkgs.python3

pkgs.nodePackages.yarn

pkgs.git

];

}

**text (auto)**

This example defines an environment with Python 3, Yarn, and Git as dependencies.

**Benefits for Development:**

* **Faster Onboarding:** New developers can set up the project environment quickly and easily, often with a single command (nix-shell or nix develop).
* **Reduced Errors:** The consistent environment helps prevent errors caused by incorrect or incompatible dependency versions.
* **Improved Collaboration:** Everyone on the team uses the same environment, which simplifies debugging and testing.
* **CI/CD**: Using nix in CI/CD can be very powerful and beneficial because you are ensuring that the same dependencies and environments will be used, which will prevent unexpected behavior in your pipelines.

**In summary:** The dev.nix file is a powerful tool for managing development environments. It promotes consistency, reproducibility, and isolation, making it easier to develop and maintain software projects. It can be extremely helpful for complex projects.

To change default emulator in project

Changing the default emulator in the project typically involves modifying the environment variables or configuration files that the development tools use to determine which emulator to target.

Since this is a react native project, the main file that controls this behavior is package.json. In the scripts section, you can modify the script for android:

{

"scripts": {

"android": "react-native run-android"

}

}

**json**

The command react-native run-android will run the app on the default emulator. You can add the --deviceId flag to choose which device to use, for example:

{

"scripts": {

"android": "react-native run-android --deviceId emulator-5556"

}

}

**json**

This will run the application in the emulator-5556 emulator, if available. This is a hardcoded value, so if the emulator changes, the user needs to change it here.

**Common Errors**

Run build command in ${root}/android folder of project

***./gradlew -warning-mode all installDebug***

The error message *com.android.builder.testing.api.DeviceException*: No connected devices! typically arises when you're trying to run Android tests or deploy an app using Android tools, but the system cannot detect any connected Android devices or emulators.

Here's a breakdown of the likely causes and how to address them in this environment:

**Likely Causes:**

1. **No Emulator Running:** The most common reason is that you haven't started any Android emulators.
2. **Emulator Not Connected to ADB:** Even if an emulator is running, it might not be properly connected to the Android Debug Bridge (ADB). ADB is the tool used to communicate with Android devices and emulators.
3. **No real device connected:** If you are trying to deploy to a real device and it is not connected, you will get this message.
4. **ADB server not running**: If adb is not running, no devices will be listed.
5. **Multiple ADB Servers:** In some cases, multiple ADB servers might be running and interfering with each other.

**Troubleshooting Steps:**

1. **Start an Emulator:**
   * Since there are no available AVDs, we will need to start an emulator from scratch. Let's try to start the Pixel2 avd.
2. **Restart adb**: Let's restart the adb server.

emulator -avd Pixel2 -no-window -no-snapshot-load -no-boot-anim -accel on -verbose -gpu swiftshader\_indirect -netfast

To list all devices and emulators

$adb devices

$adb start-server

$adb kill-server

$emulator -list-avds

$ emulator -avd Android32 -no-window **-no-snapshot**-load -no-boot-anim -accel on -verbose -gpu swiftshader\_indirect -netfast

*To launch android emulator , I always use cold boot ( i.e. without snapshot ) and ensure atleast use 3 versions older of current version to avoid common issues. Right now version 35 is there , so I am using Android version 32 to avoid issues.*

**$emulator -avd Android32 -no-snapshot-load -no-boot-anim -accel on -verbose -gpu swiftshader\_indirect -netfast -window**

A blank black Android emulator screen is a common issue, and thankfully, there are several troubleshooting steps you can take to resolve it.

Here's a breakdown of the most common causes and their fixes:

**1. Cold Boot:**

* **Problem:** Sometimes, the emulator gets stuck in a bad state, especially after a crash or unexpected shutdown. A simple restart might not be enough.
* **Solution:** Perform a cold boot.
  1. Close the emulator.
  2. In the terminal, try to run: emulator -avd <your\_avd\_name> -no-snapshot-load (replace <your\_avd\_name> with the actual name of your AVD). This command launches the emulator without loading a saved state.
  3. If you do not know the name of your AVD, you can run: emulator -list-avds.
  4. Then try again using the command: emulator -avd <your\_avd\_name> -no-snapshot-load.

**2. Graphics Issues:**

* **Problem:** The emulator might have trouble rendering graphics correctly, especially with hardware acceleration.
* **Solutions:**
  + **Check Graphics Settings:**
    1. Open Android Virtual Device Manager (AVD Manager).
    2. Edit the AVD that's having issues.
    3. Click "Show Advanced Settings."
    4. Under "Emulated Performance," change "Graphics" to "Software - GLES 2.0."
    5. Try launching the emulator again.
  + **Update Graphics Drivers:** Outdated or corrupted graphics drivers can cause rendering problems. Make sure your computer's graphics drivers are up to date.
  + **Disable Hardware Acceleration:** If software rendering works, it means there is an issue with hardware acceleration. You may need to disable hardware acceleration in the emulator's settings temporarily.
  + **Change renderer:** Try changing the renderer used in the emulator. By adding -gpu swiftshader\_indirect or -gpu angle\_indirect when starting the emulator.

**3. Insufficient RAM:**

* **Problem:** The emulator requires a decent amount of RAM to run smoothly. If your computer is low on RAM, the emulator might struggle to display anything.
* **Solutions:**
  + **Close Unnecessary Programs:** Close any other programs that are consuming a lot of RAM.
  + **Allocate More RAM:**
    1. In the AVD Manager, edit your AVD.
    2. Click "Show Advanced Settings."
    3. Increase the "RAM" setting. Be cautious not to allocate more RAM than your system has available.
  + **Restart Computer:** If there is some sort of RAM issue you can try restarting your computer.

**4. Corrupted AVD:**

* **Problem:** The files associated with your AVD might have become corrupted.
* **Solution:**
  1. **Delete and Recreate:** The easiest fix is often to delete the problem AVD and create a new one.
  2. In AVD Manager, delete your AVD, and create a new one.

**5. Android SDK/Emulator Issues:**

* **Problem:** There might be a problem with the Android SDK or the emulator itself.
* **Solutions:**
  + **Update the Android SDK:** Make sure your Android SDK is up to date.
  + **Update the Emulator:** Update the Android Emulator from the SDK Manager.

**6. Android Studio/IDE Issues:**

* **Problem:** If you're launching the emulator from Android Studio or a similar IDE, there might be a problem with the IDE itself.
* **Solutions:**
  + **Restart the IDE:** Try restarting Android Studio or your IDE.
  + **Invalidate Caches/Restart:** If you are using Android Studio try:
    1. Go to "File" -> "Invalidate Caches / Restart."
    2. Choose "Invalidate and Restart."

**7. Conflicting Software:**

* **Problem:** Other software, particularly virtualization software (like Hyper-V or VirtualBox), might conflict with the emulator.
* **Solution:**
  + **Disable Conflicting Software:** If you have other virtualization software running, try disabling it temporarily.

**Troubleshooting Steps:**

1. **Start with a Cold Boot:** This is the easiest and often most effective fix.
2. **Check Graphics:** Try changing the graphics settings to "Software - GLES 2.0."
3. **RAM:** Make sure you have enough RAM and close unnecessary programs.
4. **Recreate the AVD:** If nothing else works, try deleting and recreating the AVD.
5. **Update Software:** Update your Android SDK and the emulator.

If you can provide more details about the specific error messages or behavior you're seeing, I might be able to give you more tailored advice!

Common emulator / devices / launch issues

In root directory of project

$npx install-peerdeps --dev react-native-windows

I have successfully installed the react-native-windows package along with its peer dependencies using npx install-peerdeps --dev react-native-windows. This command automatically handled installing the correct versions of react, @types/react, and react-native as development dependencies.

$ npm run android –verbose

Output for npx react-native doctor

$ npx react-native doctor

Common

✓ Node.js - Required to execute JavaScript code

✓ npm - Required to install NPM dependencies

✓ Metro - Required for bundling the JavaScript code

Android

✖ Adb - No devices and/or emulators connected. Please create emulator with Android Studio or connect Android device.

✖ JDK - Required to compile Java code

- Version found: javac 21

- Version supported: >= 17 <= 20

✖ Android Studio - Required for building and installing your app on Android

✓ ANDROID\_HOME - Environment variable that points to your Android SDK installation

✓ Gradlew - Build tool required for Android builds

✓ Android SDK - Required for building and installing your app on Android

Errors: 3

Warnings: 0

Attempting to fix 3 issues...

Android

✖ Adb

Read more about how to download Adb at https://reactnative.dev/docs/running-on-device?os=linux&platform=android&guide=native#method-1-using-adb-reverse-recommended-2

✖ JDK

Read more about how to download JDK at https://reactnative.dev/docs/environment-setup?os=linux&platform=android&guide=native#jdk-studio

✖ Android Studio

Read more about how to download Android Studio at https://reactnative.dev/docs/environment-setup?os=linux&platform=android&guide=native#android-studio