Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Getting Started with Flutter

Creating your development environment is the first task that every developer has to go through when starting with a new platform. In some ways, the ease in which you can go from nothing to building software can be seen as a litmus test for how your experience with the platform is going to be. If the environment is difficult and painful to set up, then it might be very likely that it will be difficult and painful to work with.

The Flutter engineers must have taken this to heart because getting started with Flutter is easier than with other frameworks. You can divide the process into three distinct sections. First, you have to install the Flutter **software development kit** (**SDK**). Then, you have to install at least one platform SDK—iOS or Android, or both if you are working on a Mac. Since Flutter 2.0, you can also install a desktop SDK to develop apps for Windows, macOS, or Linux. The final stage is choosing which editor, or **integrated development environment** (**IDE**), you want to use. To make this process even easier, Flutter has a tool called Flutter Doctor that will scan your environment and offer you step-by-step guides for what you need to do to successfully complete your environment setup. This means that the Flutter team has made every effort to help you successfully install and use Flutter to develop your projects.

By the end of this chapter, you will have Flutter fully installed and will have learned how to create an app and run code on a virtual device.

In this chapter, we'll be covering the following recipes:

- How to use Git to manage the Flutter SDK
- Setting up the command line and saving path variables
- Using Flutter Doctor to diagnose your environment
- Configuring the iOS SDK

- Setting up CocoaPods (iOS only)
- Configuring the Android SDK setup
- Which IDE/editor should you choose?
- Picking the right channel
- How to choose the platform language for your app
- How to create a Flutter app
- How Flutter projects are structured
- How to run a Flutter app
- How to use Hot reload to refresh your app without recompiling

> While Flutter is compatible with Windows, macOS, and Linux, if you are interested in building applications for Apple's platforms (iOS and macOS), you will need a Mac to build your app.

Technical requirements

Building mobile applications can be a taxing task for your computer.

Your computer should have the following:

- 8 GB of **random-access memory** (**RAM**) (16 **gigabytes** (**GB**) preferred)
- 50 GB of available hard drive space
- A **solid-state drive** (**SSD**) hard drive is recommended
- At least a 2 **gigahertz** (**GHz**) + processor

If you want to build for iOS, you will also need a Mac instead of a PC.

These are not strict system requirements, but anything less than this may lead to you spending more time waiting rather than working.

How to use Git to manage the Flutter SDK

Before you can build anything, you need to download the Flutter SDK. If you go to the main Flutter website at `https://flutter.dev`, they currently recommend that you download one of their prebuilt packages for macOS, Windows, or Linux. This is certainly OK, and if you feel comfortable with this approach, you can certainly follow it. However, we can do better. Since Flutter is completely open source and hosted on GitHub, if you just clone the main Flutter repository, you'll already have everything you'll need, and you can easily change to different versions of the Flutter SDK if needed.

The packages that are available to download on Flutter's website are snapshots from the Git repository. Flutter uses Git internally to manage its versions, channels, and upgrades, so why not go straight to the source?

Installing Git

First, you need to make sure you have Git installed on your computer. If you are developing on macOS, you can skip this step.

For Windows, you can download and install Git here: `https://git-scm.com/download/win`.

You might also want to get a Git client to make working with repositories a bit easier. Tools such as `Sourcetree` (`https://www.sourcetreeapp.com`) or GitHub Desktop (`https://desktop.github.com`) can greatly simplify working with Git. They are optional, however, and this book will stick to the command line when referencing Git.

To confirm that Git is installed on Linux and macOS, if you open your Terminal and type `which git`, you should see a `/usr/bin/git` path returned. If you see nothing, then Git is not installed correctly.

How to do it...

Follow these steps to clone and configure the Flutter SDK:

1. First, choose a directory where Flutter is going to be installed. The location does not explicitly matter, but it will be simpler to install the SDK closer to the root of your hard drive.
2. On macOS, type in the following command:

```
cd $HOME
```

This ensures that the terminal is pointing to your home directory. It might be redundant since most terminal windows automatically open to the home directory when they are opened.

3. We can now install Flutter with this command:

```
git clone https://github.com/flutter/flutter.git
```

This will download Flutter and all of its associated tools, including the Dart SDK.

See also

If you do not feel comfortable using Git, you can certainly install your Flutter SDK by following the instructions at `https://flutter.dev/docs/get-started/install`.

Setting up the command line and saving path variables

Now that you have cloned the Flutter repository, there are few more steps needed to make the software accessible on your computer. Unlike apps with **user interfaces** (**UIs**), Flutter is primarily a command-line tool. Let's quickly learn how to set up the command line on macOS, Linux, and Windows in the following sections.

macOS command-line setup

To actually use Flutter, we need to save the location of the Flutter executable to your system's environment variables.

Newer Macs use the Z shell (also known as `zsh`). This is basically an improved version of the older Bash, with several additional features.

When using `zsh`, you can add a line to your `zshrc` file, which is a text file that contains the `zsh` configuration. If the file does not exist yet, you can create a new file, as follows:

1. Open the `zshrc` file with the following command:

   ```
   nano $HOME/.zshrc
   ```

   This will open a basic text editor called `nano` in your terminal window. There are other popular tools, such as `vim` and `emacs`, that will also work.

2. Type the following command at the bottom of the file:

   ```
   export PATH="$PATH:$HOME/flutter/bin"
   ```

3. If you chose to install Flutter at a different location, then replace `$HOME` with the appropriate directory.
4. Exit `nano` by typing *Ctrl* + X. Don't forget to save your file when prompted.
5. Reload your terminal session by typing the following command:

   ```
   source ~/.zshrc
   ```

6. Finally, confirm that everything is configured correctly by typing the following:

   ```
   which flutter
   ```

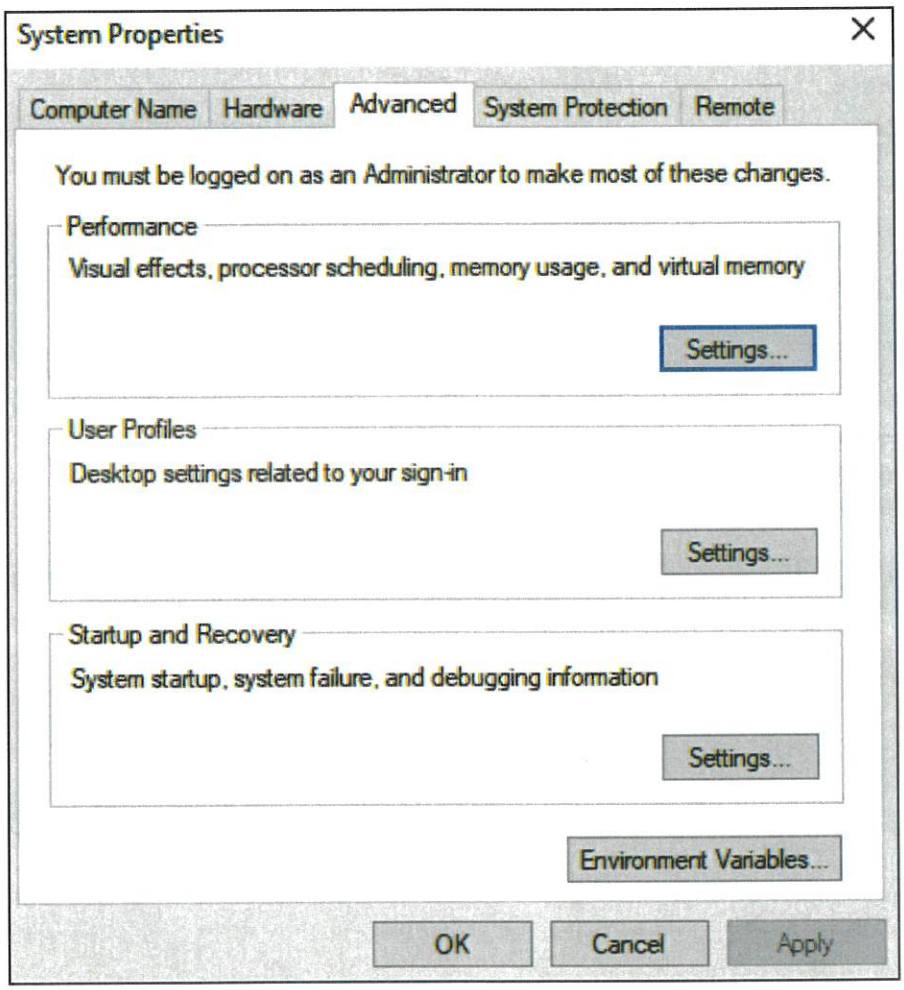
You should see the directory where you cloned (or installed) the Flutter SDK printed on the screen.

Windows command-line setup

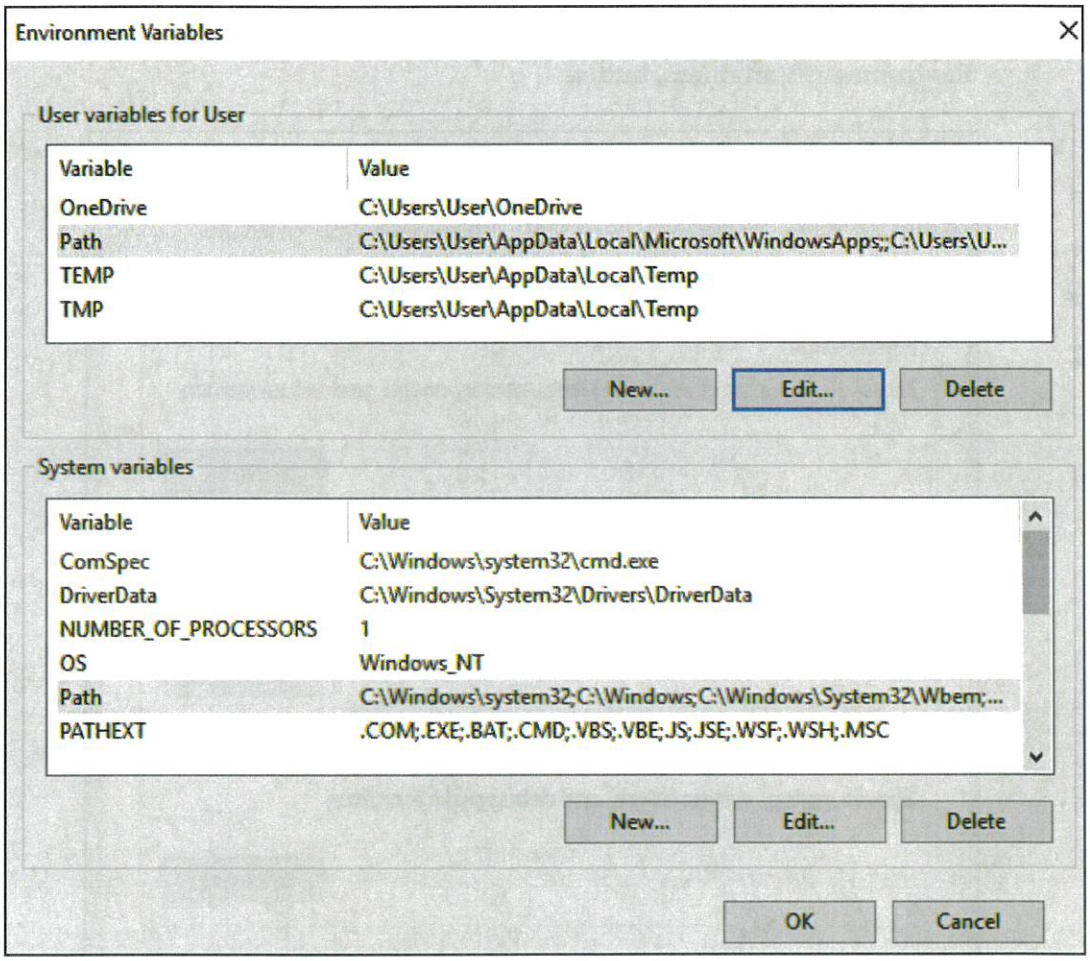
These instructions assume you are using Windows 10.

You will now set up your environment variables for Flutter on Windows:

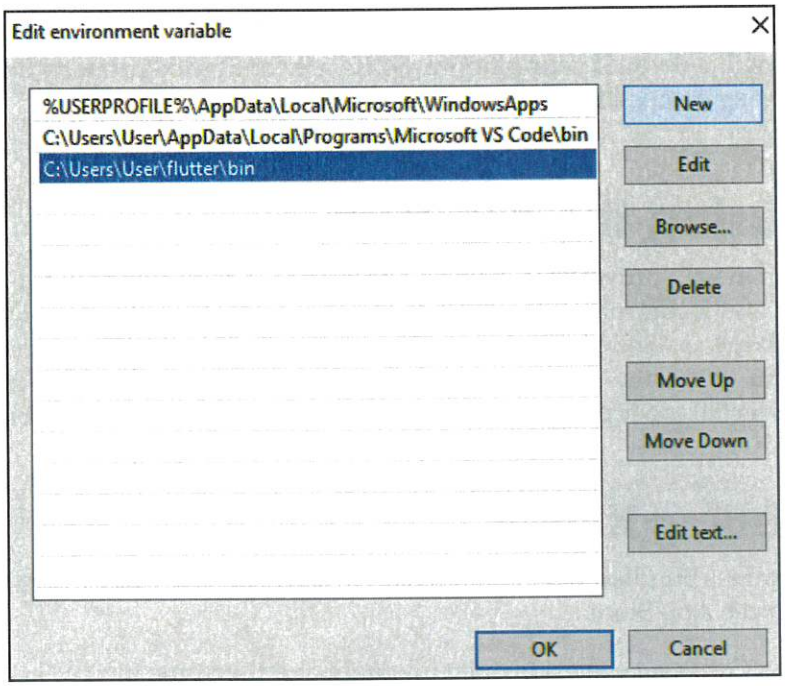
1. In the search bar at the bottom of the desktop, type `env`. You should see an **Edit the system environment variables** option appear. Select the icon to open the **System Properties** window, and at the bottom of the screen click the **Environment Variables...** button:



2. In the next dialog, select the **Path** variable in the **User variables for User** section and click the **Edit...** button:



3. Finally, add the location where you installed Flutter to your path:



4. Type `C:\Users\{YOUR_USER_NAME}\flutter\bin`, then select **OK**. Flutter should now be added to your path.
5. Restart your system.
6. Type `flutter` in the command line. You should see a message with some Flutter **command-line interface** (**CLI**) instructions. Flutter might optionally download more Windows-specific tools at this time.

Confirming your environment is correct with Flutter Doctor

Flutter comes with a tool called **Flutter Doctor** that will be your new best friend when setting up the SDK. Flutter Doctor will give you a list of everything that needs to be done to make sure that Flutter can run correctly. You are going to use Flutter Doctor as a guide during the installation process. This tool is also invaluable to check whether your system is up to date.

In your terminal window, type the following command:

```
flutter doctor
```

Flutter Doctor will tell you if your platform SDKs are configured properly and whether Flutter can see any devices, including the web browser.

Configuring the iOS SDK

The iOS SDK is mostly provided in a single application, Xcode. Xcode is one behemoth of an application and it controls all the official ways in which you will interact with Apple's platforms. As large as Xcode is, there are a few pieces of software that are missing. Two of these are community tools: CocoaPods and Homebrew. These are **package managers** or programs that install other programs. Flutter uses both of these tools in its build system.

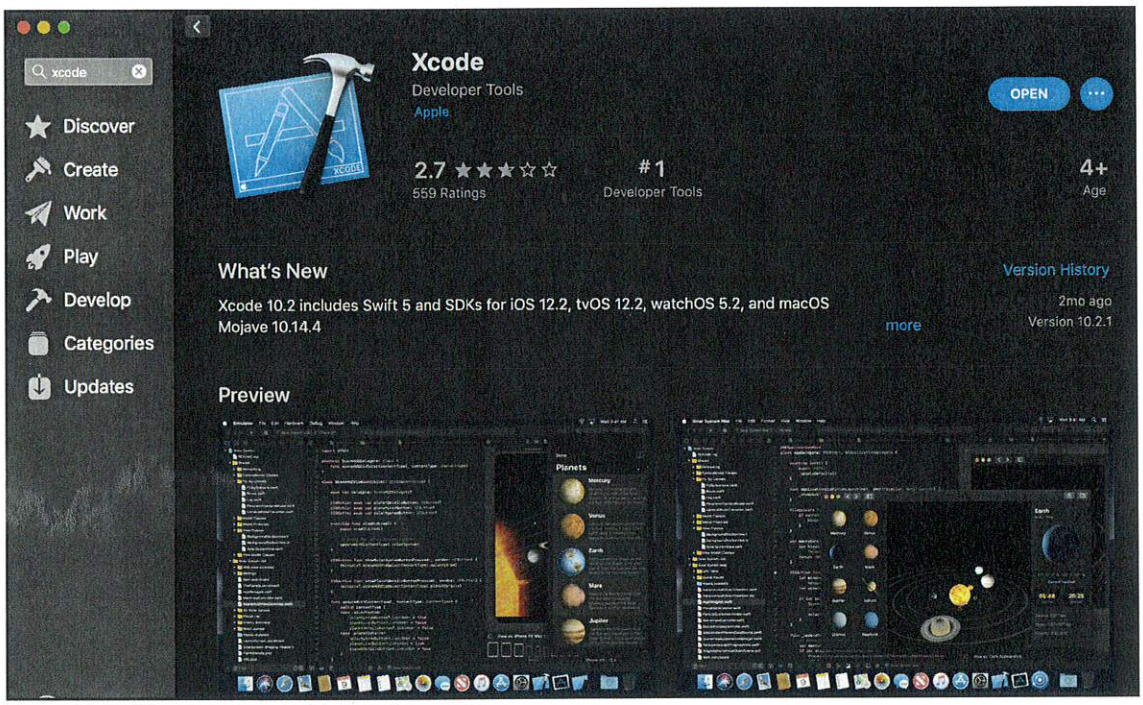
Downloading Xcode

The iOS SDK comes bundled with Apple's IDE, Xcode. The best place to get Xcode is through the Apple App Store:

1. Press *Command* + *Space* to open Spotlight and then type in `app store`:

As an alternative, you can just click on the menu in the top-left corner of your screen and select **App Store**, but keyboard shortcuts are much more fun.

2. After the App Store opens, search for `Xcode` and select **Download**:



Xcode is a rather large application, so it may take a while to download. While Xcode is installing, you can get some of the smaller tools that you'll need for development. Let's take a look at how to install these tools in the following sections.

CocoaPods

CocoaPods is a popular community lead dependency manager for iOS development. It is essentially the equivalent of npm for the web community. Flutter requires CocoaPods in its build process to link any libraries you have added to your project:

1. To install `cocoapods`, type this command:

```
sudo gem install cocoapods
```

2. Since this command requires administrator privileges, you will likely be prompted to enter your password before continuing. This should be the same password that you use to log in to your computer.
3. After `cocoapods` has finished installing, type this command:

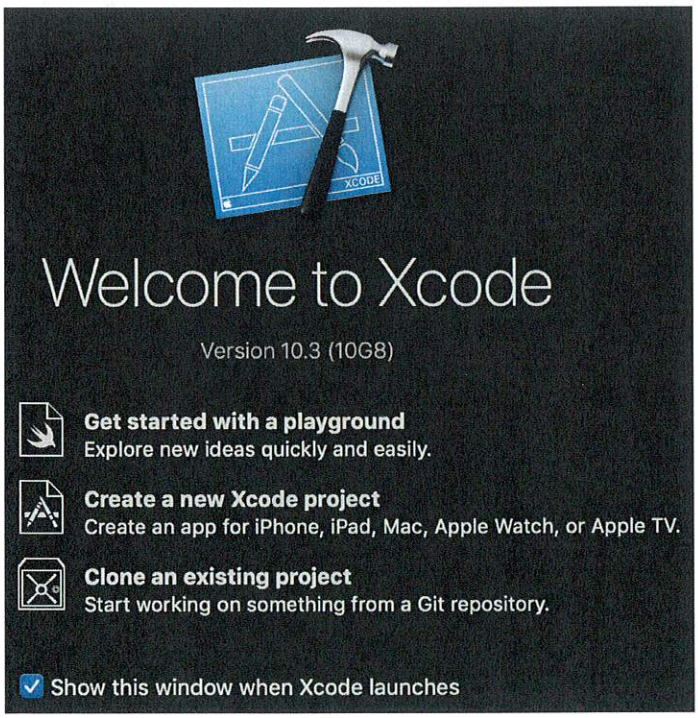
```
pod setup
```

This will configure your local version of the `cocoapods` repository, which can take some time.

Xcode command-line tools

Command-line tools are used by Flutter to build your apps without needing to open Xcode. They are an extra add-on that requires your primary installation of Xcode to be complete:

1. Verify that Xcode has finished downloading and has been installed correctly. After it is done, open the application to allow Xcode to fully configure itself. Once you see the **Welcome to Xcode** screen appear, you can close the application:



2. Type this command in the Terminal window to install the command-line tools:

```
sudo xcode-select --switch
/Applications/Xcode.app/Contents/Developer
```

3. You may also need to accept the Xcode license agreement. Flutter Doctor will let you know if this step is required. You can either open Xcode and will be prompted to accept the agreement on the first launch or you can accept it via the command line, with the following command:

```
sudo xcodebuild -license accept
```

Homebrew

Homebrew is a package manager used to install and manage applications on macOS. If CocoaPods manages packages that are specific to your project, then Homebrew manages packages that are global to your computer.

Homebrew can be installed with this command in your terminal window:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

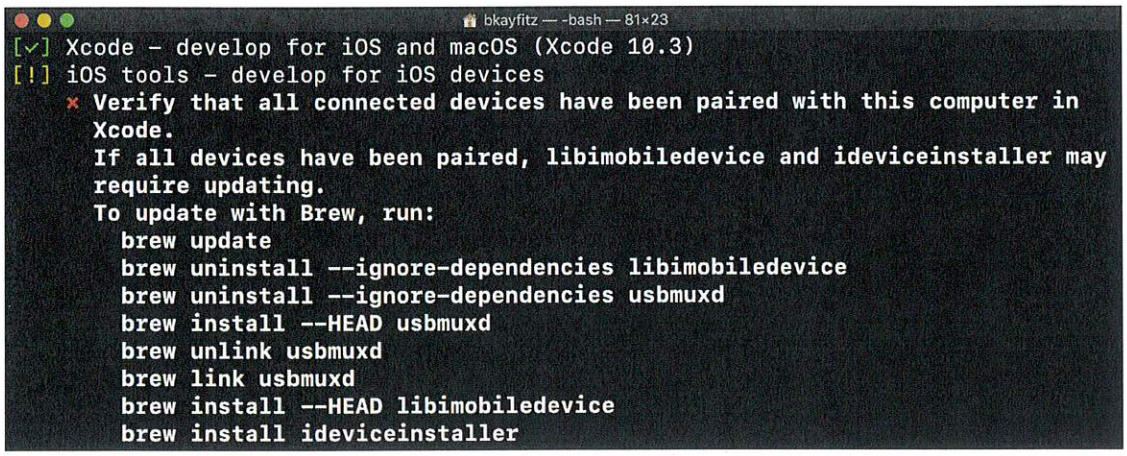
We will be using Homebrew primarily as a mechanism to get other, smaller tools.

You can also get more information about Homebrew from it's website: `https://brew.sh`.

Checking in with the Doctor

Now that we have all the platform tools for iOS, let's run Flutter Doctor one more time to make sure everything is installed correctly.

You may end up seeing this as a result:



Remember how earlier you installed Homebrew? It's now going to come in handy. You now have two options to solve this problem: you can either copy/paste each one of these `brew` commands one by one into a terminal window or you can automate this with a single shell script.

Hopefully, you prefer option 2.

1. Select and copy all the `brew` commands in *Step 2*, then enter `nano` again with this command:

```
nano update_ios_toolchain.sh
```

2. Add the following commands in the file and then exit and save `nano`:

```
brew update
brew uninstall --ignore-dependencies libimobiledevice
brew uninstall --ignore-dependencies usbmuxd
brew install --HEAD usbmuxd
brew unlink usbmuxd
brew link usbmuxd
brew install --HEAD libimobiledevice
brew install ideviceinstaller
```

3. Run this script with the following command:

```
sh update_ios_toolchain.sh
```

When the script finishes, run `flutter doctor` one more time. Everything for the iOS side should now be green.

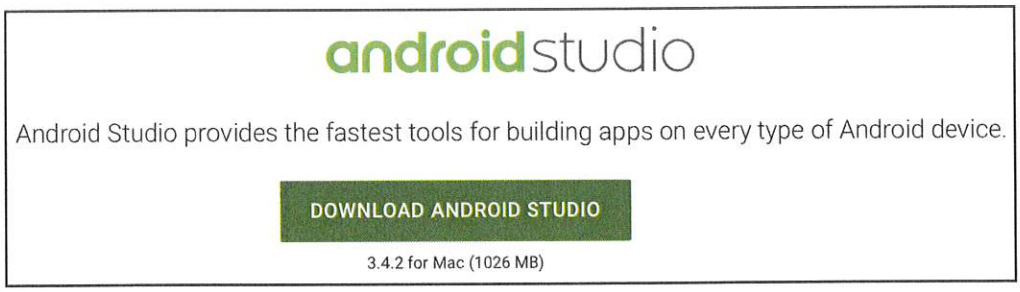
Configuring the Android SDK setup

Just like with Xcode, Android Studio and the Android SDK come hand in hand, which should make this process fairly easy. But also like iOS, Android Studio is just the starting point. There are a bunch of tiny tools that you'll need to get everything up and running.

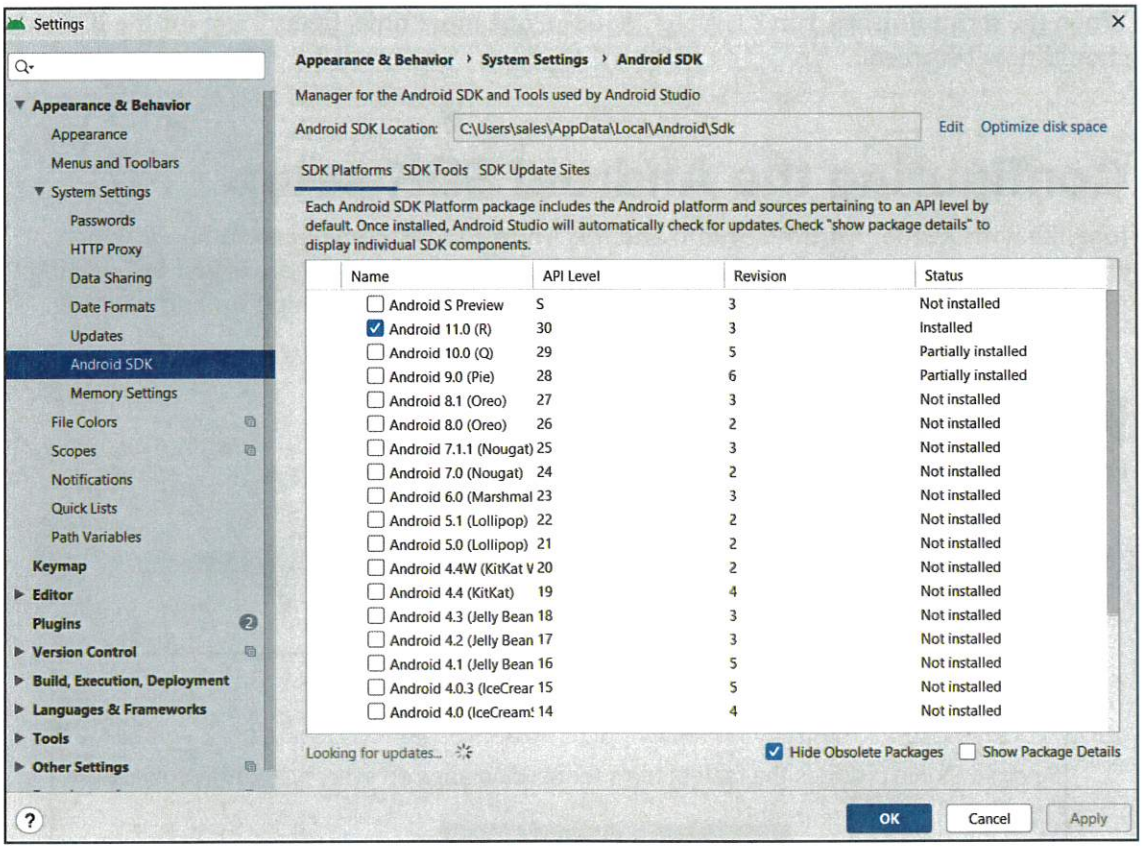
Installing Android Studio

Follow these steps to install Android Studio:

1. You can download Android Studio at `https://developer.android.com/studio`. The website will autodetect your operating system and only show the appropriate download link:



2. After Android Studio is installed, you'll need to download at least one Android SDK. From the **Android Studio** menu, select **Preferences** and then type `android` into the search field:

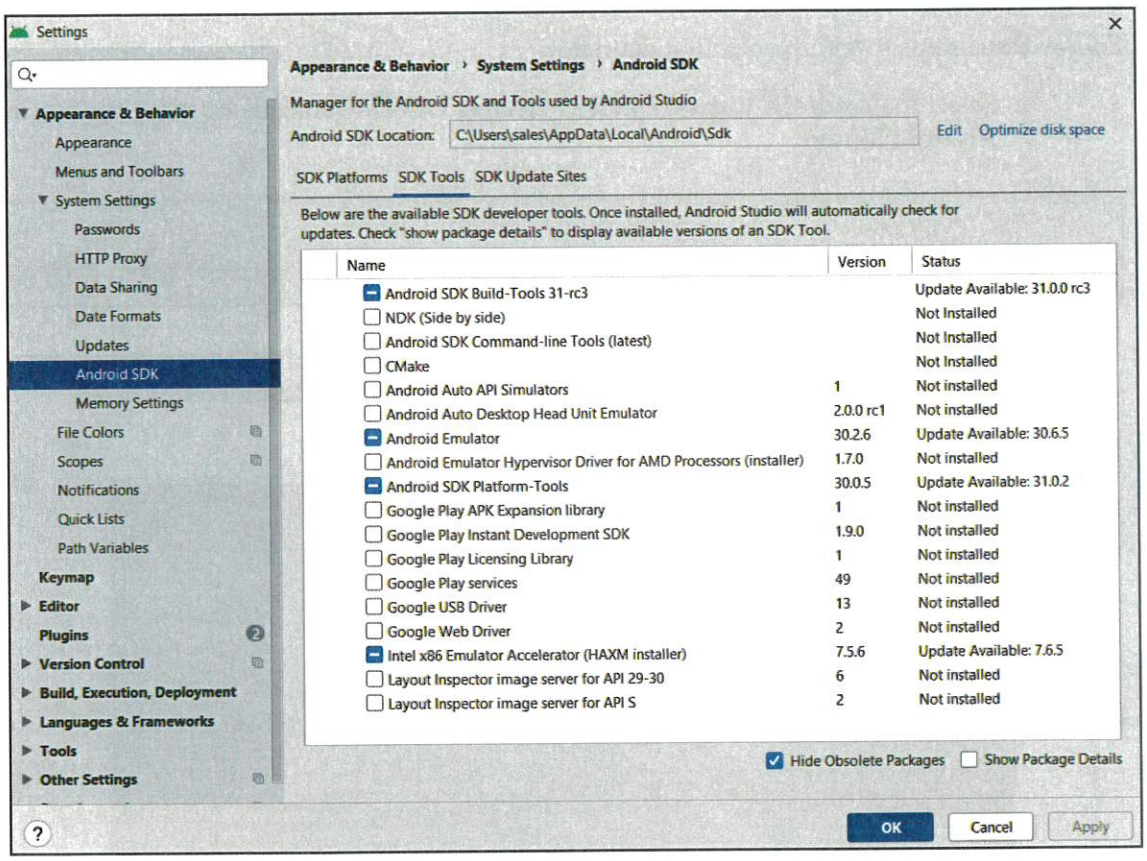


While it may be tempting to grab the most recent version of the Android SDK, you might want to choose the second most recent version, because the Flutter SDK is sometimes a bit behind Android. In most cases, it shouldn't matter, but Android is notorious for breaking compatibility, so be aware of this.

If you ever need to change your version of the Android SDK, you can always uninstall and reinstall it from this screen.

3. You will also need to download the latest build tools, emulator, SDK platform tools, SDK tools, the **Hardware Accelerated Execution Manager (HAXM)** installer, and the support library.

4. Select the **SDK Tools** tab and make sure the required components are checked. When you click the **Apply** or **OK** buttons, the tools will begin downloading:



After everything finishes installing, run `flutter doctor` to check that everything is working as expected.

Creating an Android emulator

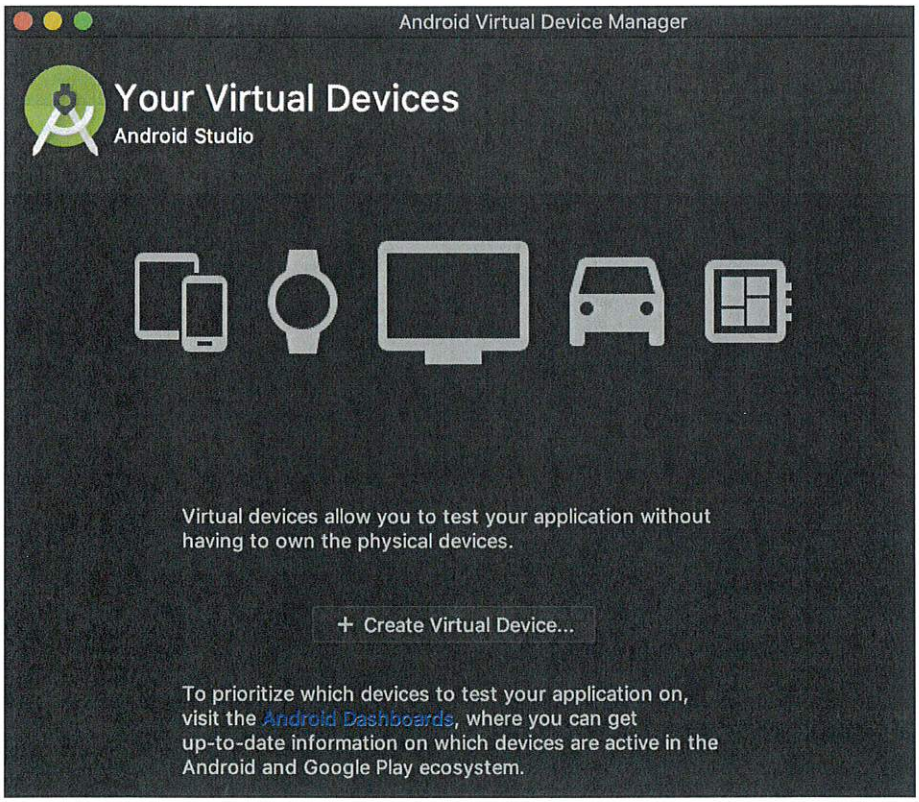
In order to run your app, you are going to need some kind of device to run it on. When it comes to Android, nothing beats the real thing. If you have access to a real Android device, it is recommended that you try to use that device for development as much as possible.

However, there are advantages to using an Android emulator (and an iOS simulator). It is often simpler to have a virtual device next to your code rather than having to carry around real devices with the required cables.

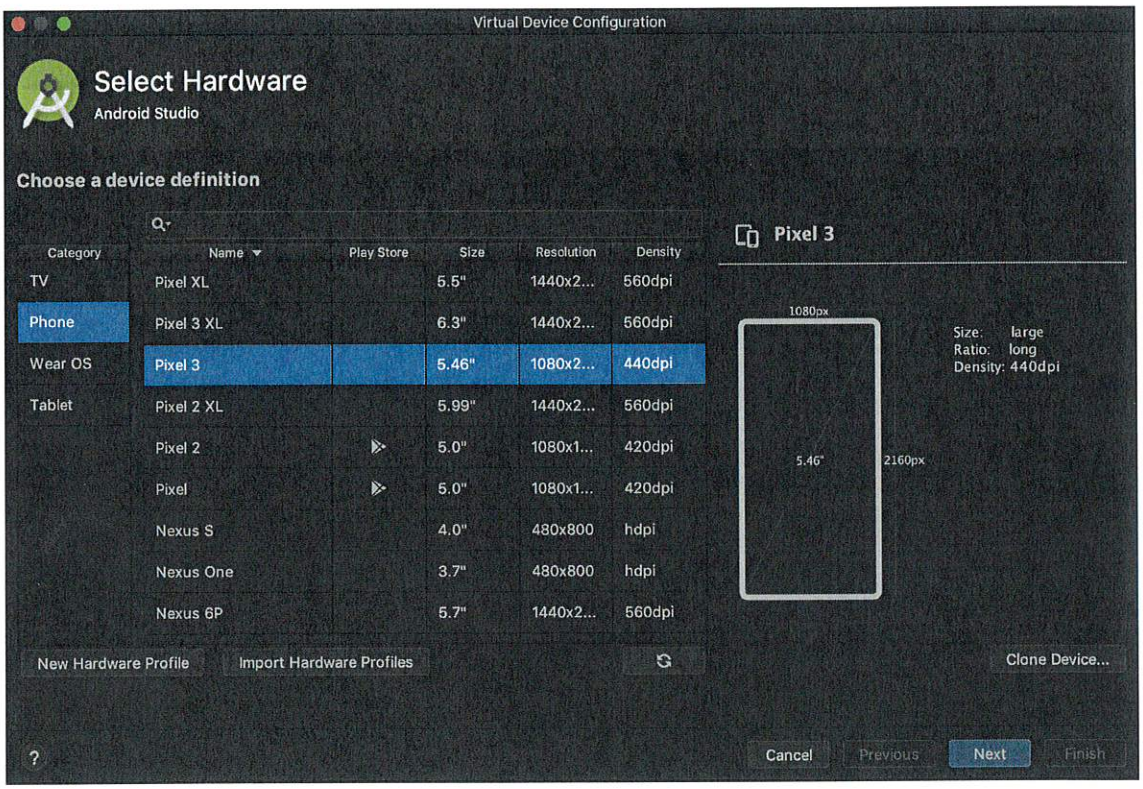
Follow these steps to set up your first emulator:

1. Select the **Android Virtual Device Manager** (**AVD Manager**) from the toolbar in Android Studio:

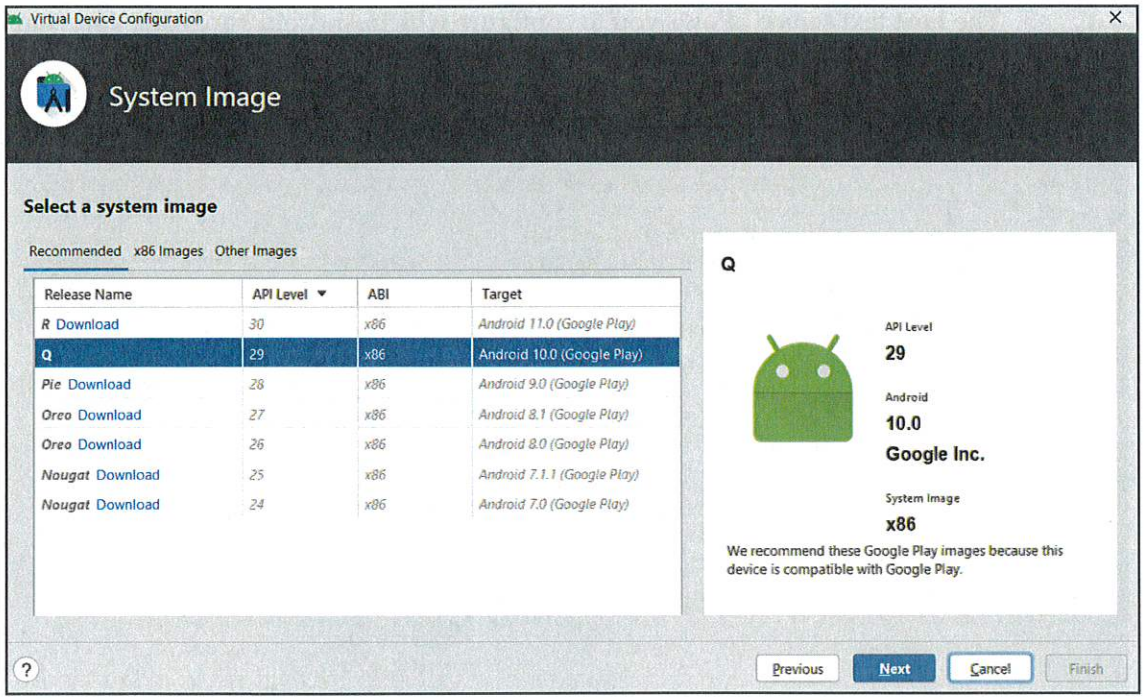
2. The first time you open the AVD Manager, you'll get a splash screen. Select the **Create Virtual Device...** button in the middle to start building your virtual device:



3. The next text screen allows you to configure which Android hardware you want to emulate. I recommend using a **Pixel** device:



4. In the next screen, you will have to pull down an Android runtime. For the most part, the most recent image will be sufficient. Each one of these images is several **gigabytes (GB)** in size, so only download what you need:



5. Click **Next** to create your emulator. You can launch the emulator if you want, but this is not necessary.
6. Once again, run `flutter doctor` to check your environment.
7. One final thing that you may have to do is accept all the Android license agreements. You can do this quickly from the terminal line with this command:

```
flutter doctor --android-licenses
```

Keep typing `y` when prompted to accept all the licenses (nobody really reads them, right?). Run `flutter doctor` one more time just for good measure. The Android SDK should now be fully configured.

Congratulations! The Flutter SDK should now be fully set up for both iOS and Android. In the next recipes in this chapter, we are going to explore some optional choices to customize your environment to fit your needs.

Which IDE/editor should you choose?

A developer's IDE is a very personal choice. In the olden days, developers would wage proverbial wars over the choice between Emacs or Vim. Today, we are apparently more cool-headed (at least some of us). Ultimately, the choice is dependent on which tool you are most productive in. If you find yourself fighting with the tool rather than just writing code, then it might not be the right choice. As with most things, it's more important to make choices based on what best fits your personal and unique style, rather than follow any prescribed doctrine.

Flutter provides official plugins for three popular IDEs:

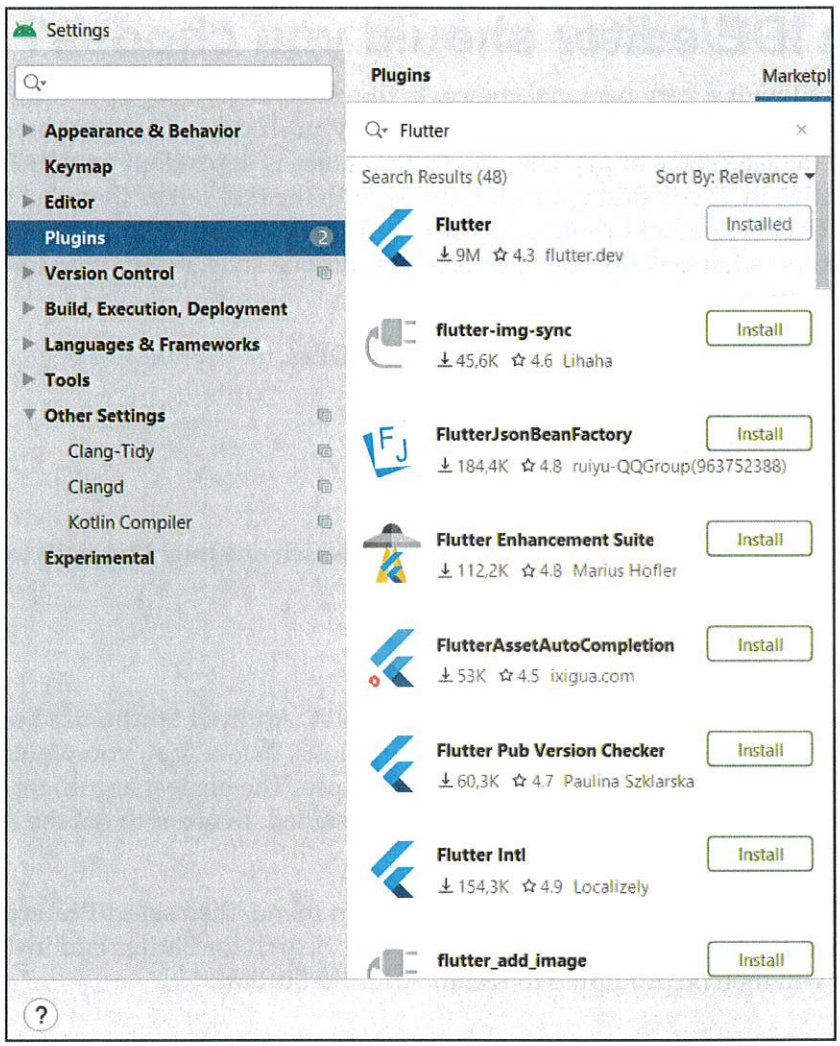
- Android Studio
- **Visual Studio Code** (**VS Code**)
- IntelliJ IDEA

Let's compare and configure all three and find out which one might be right for you.

Android Studio

Android Studio is a mature and stable IDE. Since 2014, Android Studio has been promoted as the default tool for developing Android applications. Before that, you would have had to use a variety of plugins for legacy tools such as Eclipse. The biggest argument in favor of using Android Studio is that you already have it installed. In order to get the Android SDK, you have to download Android Studio.

To add the Flutter plugin, select the **Android Studio** menu, then select **Preferences**. Click on the **Plugins** tab to open the plugins marketplace. Search for Flutter and install the plugin. You will then be prompted to restart Android Studio:



Android Studio is a very powerful tool. At the same time, the program can seem intimidating at first, with all the panels, windows, and too many options to enumerate. You will need to spend a few weeks with the program before it starts to feel natural and intuitive.

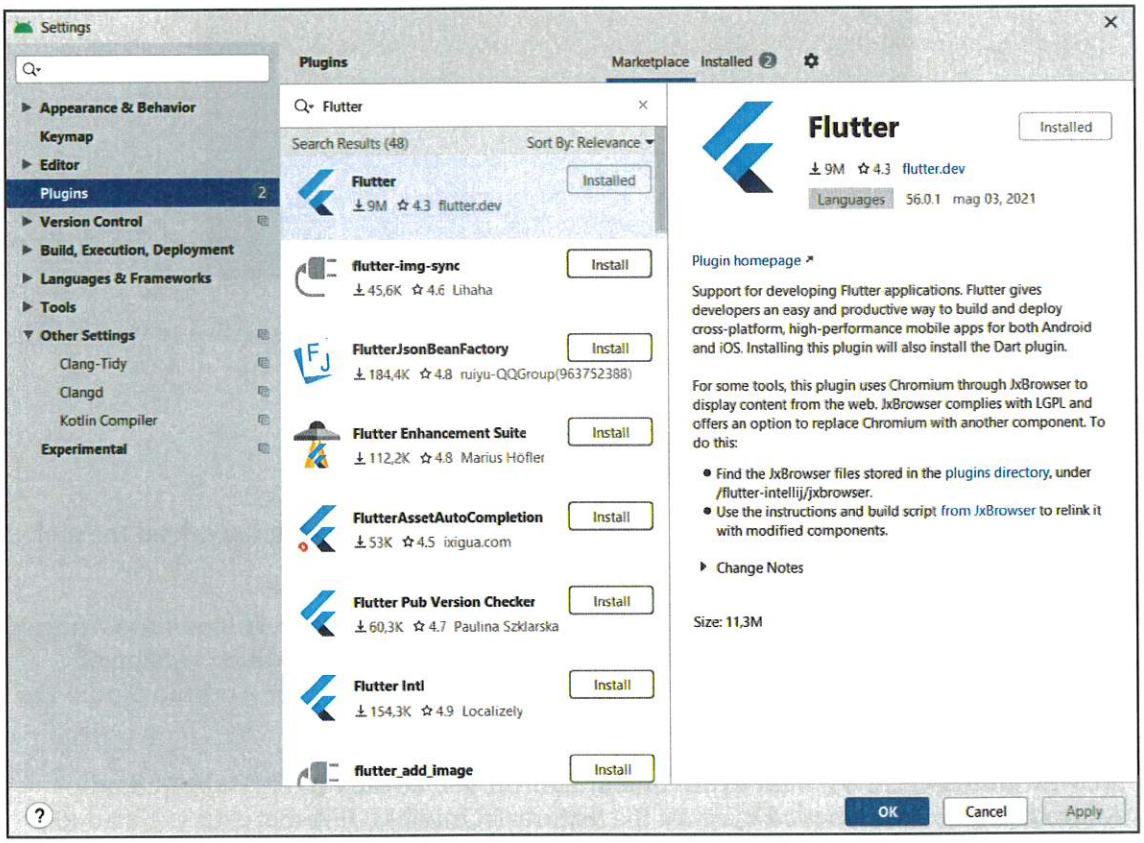
With all this power comes consequence: Android Studio is a very demanding application. On a laptop, the IDE can drain your battery very quickly, so be prepared to keep your power cable nearby. You should also make sure you have a relatively powerful computer; otherwise, you might spend more time waiting than writing code.

VS Code

VS Code is a lightweight, highly extensible tool from Microsoft that can be configured for almost any programming language, including Flutter.

You can download VS Code from `https://code.visualstudio.com`.

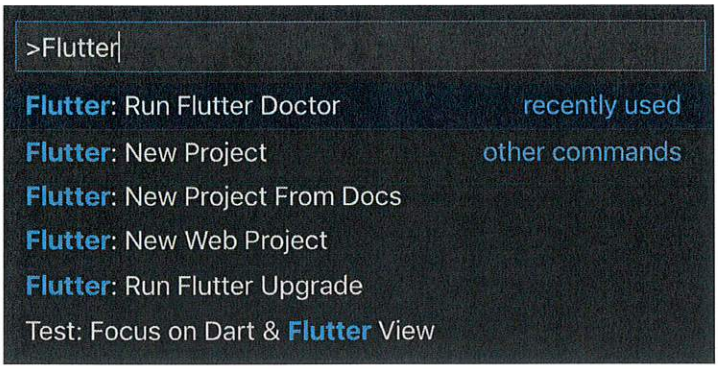
After you've installed the application, click on the fifth button in the left sidebar to open the **Extensions Marketplace**. Search for `flutter` and then install the extension:



VS Code is much kinder on your hardware than Android Studio and has a wide array of community-written extensions. You will also notice that the UI is simpler than Android Studio, and your screen is not covered with panels and menus. This means that most of the features that you would see out in the open in Android Studio are accessible through keyboard shortcuts in VS Code.

Unlike Android Studio, most of the Flutter tools in VS Code are accessible through the **Command Palette**.

Type *Ctrl* + *Shift* + *P* on Windows or *Shift* + *Command* + *P* on a Mac to open the **Command Palette** and type `>Flutter` to see the available options. You can also access the **Command Palette** through the **View** menu:



If you want a lightweight but complete environment that you can customize to your needs, then VS Code is the right tool for you.

IntelliJ IDEA

IntelliJ IDEA is another extremely powerful and flexible IDE. You can download the tool from this website: `https://www.jetbrains.com/idea/`.

If you look carefully, you'll probably notice that IntelliJ iDEA looks very similar to Android Studio, and that is no coincidence. Android Studio is really just a modified version of IntelliJ IDEA. This also means that all the Flutter tools we installed for Android Studio are the exact same tools that are available for IntelliJ IDEA.

So, why would you ever want to use IntelliJ IDEA if you already have Android Studio? Android Studio has removed many of the features in IntelliJ IDEA that aren't related to Android development. This means that if you are interested in web or server development, you are going to have to use IntelliJ IDEA to get the same experience. With Flutter now supporting the web as one of its targets, this might just be enough of a reason to reach for IntelliJ IDEA over Android Studio.