

How to Implement GoogleMaps with Jetpack Compose

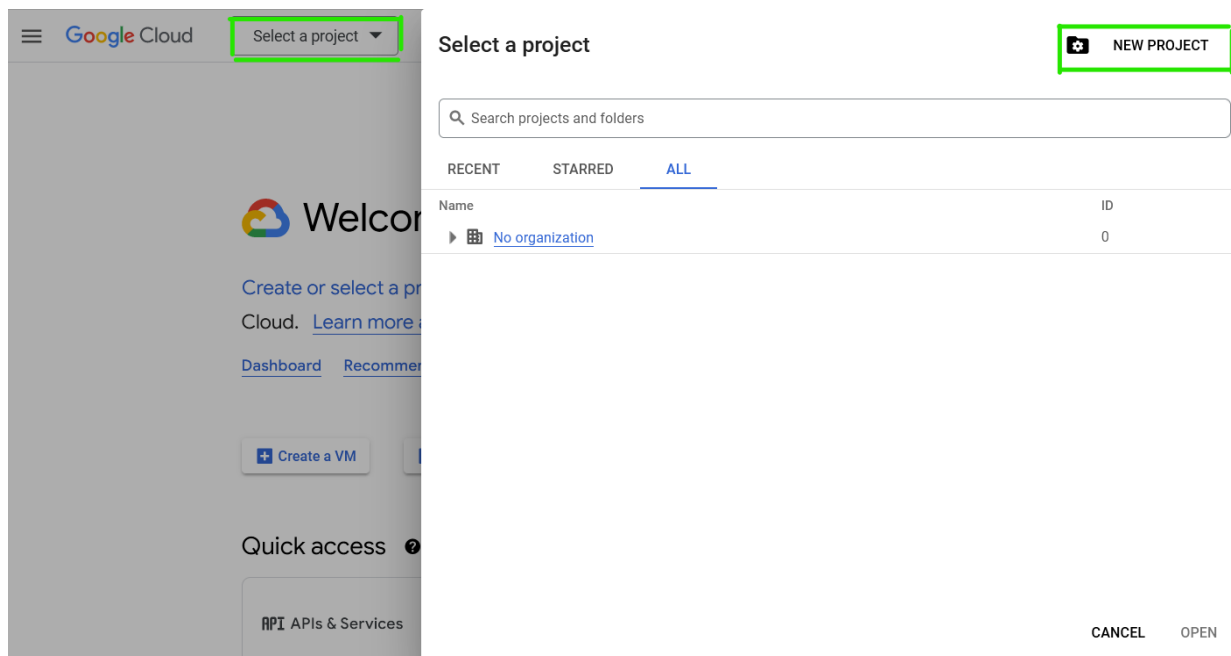
APR 2023

Sam Van Haren, Adam Rodrigues
BCIT - SSD Program

Step 1 - Setup a GoogleCloud account and API key

To access GoogleMaps in your application you first require an API key. To get an API key you first need access to GoogleCloud, so let's start there.

This [link](#) will bring you to the GoogleCloud homepage where you can use your google account (register if you do not yet have one) to create a new project by selecting **New Project** from the **Select a project** dropdown.



After creating a new project, you will need to enable billing on your project in order to have access to the API keys. Navigate to the **Billing tab** under Quick Access, and follow the steps to add billing information. Charges will not be made to your account unless you go over the given credits (which is most likely definitely not going to happen)

Once billing has been added, navigate back to your dashboard and go through these steps:

- Navigate to the [Google Maps Platform](#)
- Click on the Credentials tab and select your project
- Click **Create Credentials > API Key**
- Save this API Key, we'll need it in a minute


We should restrict our API key so no one has access to our GoogleCloud account, but first we will need to create our application before going through that process. You will need your application **package name** and **SHA-1 certificate fingerprint** in order to do this.

Create an Empty Compose Activity in Android Studio.

Package Name : in the format of com.example.YourProjectName

SHA-1 certificate : Navigate to the gradle tab on the right hand side, under the

*Tasks/android and double click on signingReport

If you cannot see this folder, go to the gradle settings icon  and under Experimental uncheck the **Only include test tasks in the Gradle task list ... then click Apply and Ok. If you still do not see the file structure, rebuild the app.*

Now that you have the package name and SHA-1 certificate, navigate back to the credentials dashboard and select your Maps API key and follow these steps:

- Select the Android App restriction
- Add your package name and SHA-1 certificate
- Enable Restrict Key and select **Maps SDK for Android**
- Save your changes

Key Restriction Screenshots for reference

- ☐ None
- ☐ Websites
- ☐ IP addresses
- ☒ Android apps
- ☐ iOS apps

Android restrictions

Restricts API key usage to specified Android apps. Add the package name and SHA-1 certificate fingerprint for each app.

+ ADD

Filter Enter property name or value				?
<input type="checkbox"/>	Status	Package name	Fingerprint	Edit
<input type="checkbox"/>		com.example.googlemapsdemo	0F:53:D4:CA:3E:9B:48:57:	

API restrictions

API restrictions specify the enabled APIs that this key can call

- ☐ Don't restrict key
This key can call any API
- ☒ Restrict key

1 API

Selected APIs:

Maps SDK for Android

Note: It may take up to five minutes for settings to take effect.

SAVE CANCEL

Android restrictions

Restricts API key usage to specified Android apps. Add the package name and SHA-1 certificate fingerprint for each app.

Add Android app

Package name *

SHA-1 certificate fingerprint *

CANCEL

Filter Type to filter

☐ Google Cloud Storage JSON API

☐ Maps Elevation API

☐ Maps Embed API

☐ Maps JavaScript API

☒ Maps SDK for Android

☐ Maps SDK for iOS

☐ Maps Static API

☐ Places API

☐ Roads API

CANCEL

Now that we have our API key ready to go, let's add it to our application and get the SDK set up.

To begin setting up the SDK start by adding the following code snippet to your top-level settings.gradle file if it is not already there.

```
pluginManagement {
    repositories {
        google()
        mavenCentral()
        gradlePluginPortal()
    }
}
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
}
```

Next, in your module level build.gradle (Module :app) under dependencies add the following implementations:

```
implementation 'com.google.maps.android:maps-compose:2.11.2'
implementation 'com.google.android.gms:play-services-maps:18.1.0'
```

Then ensure that in the same module level build.gradle file the following SDKs are at least these versions :

```
android {
    compileSdk 31

    defaultConfig {
        minSdk 19
        // ...
    }
}
```

Time to add your API key! It's best to have your API key in your local storage, so let's also install a gradle plug-in to help with this.

This time, add the following code snippet to your project-level build.gradle file (Project:ProjectName)

```
plugins {  
    // ...  
    id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin'  
    version '2.0.1' apply false  
}
```

Next open the module level build.gradle file and add the following to the plugins element:

```
id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin'
```

Save your changes and sync your project with gradle. After that is done, open local.properties in the Project directory and replace YOUR_API_KEY with the API key your grabbed earlier

```
MAPS_API_KEY=YOUR_API_KEY
```

Save your changes once again, and add your API key to the AndroidManifest.xml. Do this by updating the android: value attribute in the **com.google.android.geo.API_KEY** like so :

```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="${MAPS_API_KEY}" />
```

Now your application is set up to use the GoogleMaps SDK! Use this walkthrough to help get yourself set up to use Google Maps in your jetpack compose application, and here's a demo with what we did with it once we had it all set up!

<https://github.com/adamrodrigues11/android-google-maps-demo>

Step 2 - Implementing GoogleMaps in JetPack Compose Application

//Include demo steps? Or just show?