Automating Google Play Publishing - Part 1

26 DEC 2017 • 1 min read

Manual publishing is a chore and a waste of time as it involves a lot of steps which are redundant and can be easily automated.

MANUAL PUBLISHING REQUIRES THESE STEPS

1. Create a signed APK for your app.
2. Login to the google play developer account.
3. Select the app and create a release.
4. Upload the APK with changelog.
5. Submit the update.
6. Repeat the above 5 steps again for a new release.

Doing this again and again can become quite irritating after a while. Also, if you are uploading multiple APKs for every release then manual publishing would only lead to headaches.

So, automating this part can help save you some time which you can use to procrastinate ;)

PREREQUISITES

1. Google play developer account.
2. An already listed app in your developer account.
3. CI (Continuous Integration) tool of your choice. I will use Travis CI for this tutorial as it is free to use with open source projects.

Signing your APK properly

In order to create a signed build of your app, you can either use AndroidStudio's inbuilt UI (Build > Generate Signed APK…) or specify your keystore details in your `build.gradle`. We will use the latter as it is will help us automate the signing process for our app.

Add this snippet to your build.grade's android section:

```
android {

    ...
```

```
    signingConfigs {
        release {
            storeFile file("someDirectory/my_keystore.jks")
            storePassword "my_store_pass_here"
            keyAlias "my_key_alias_here"
            keyPassword "my_key_pass_here"
        }
    }

    buildTypes {
        release {
            signingConfig signingConfigs.release
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard
-rules.pro'
        }
    }
}
```

You can now run `./gradlew assembleRelease` command to create a signed APK directly.

In the next part, we will discuss how to set up your google play developer account.

PS: In upcoming parts of this blog series, I will also explain how to secure your keystore details for your open source projects.

References:

1. https://medium.com/@harmittaa/travis-ci-android-example-357f6e632fc4

Automating Google Play Publishing - Part 2

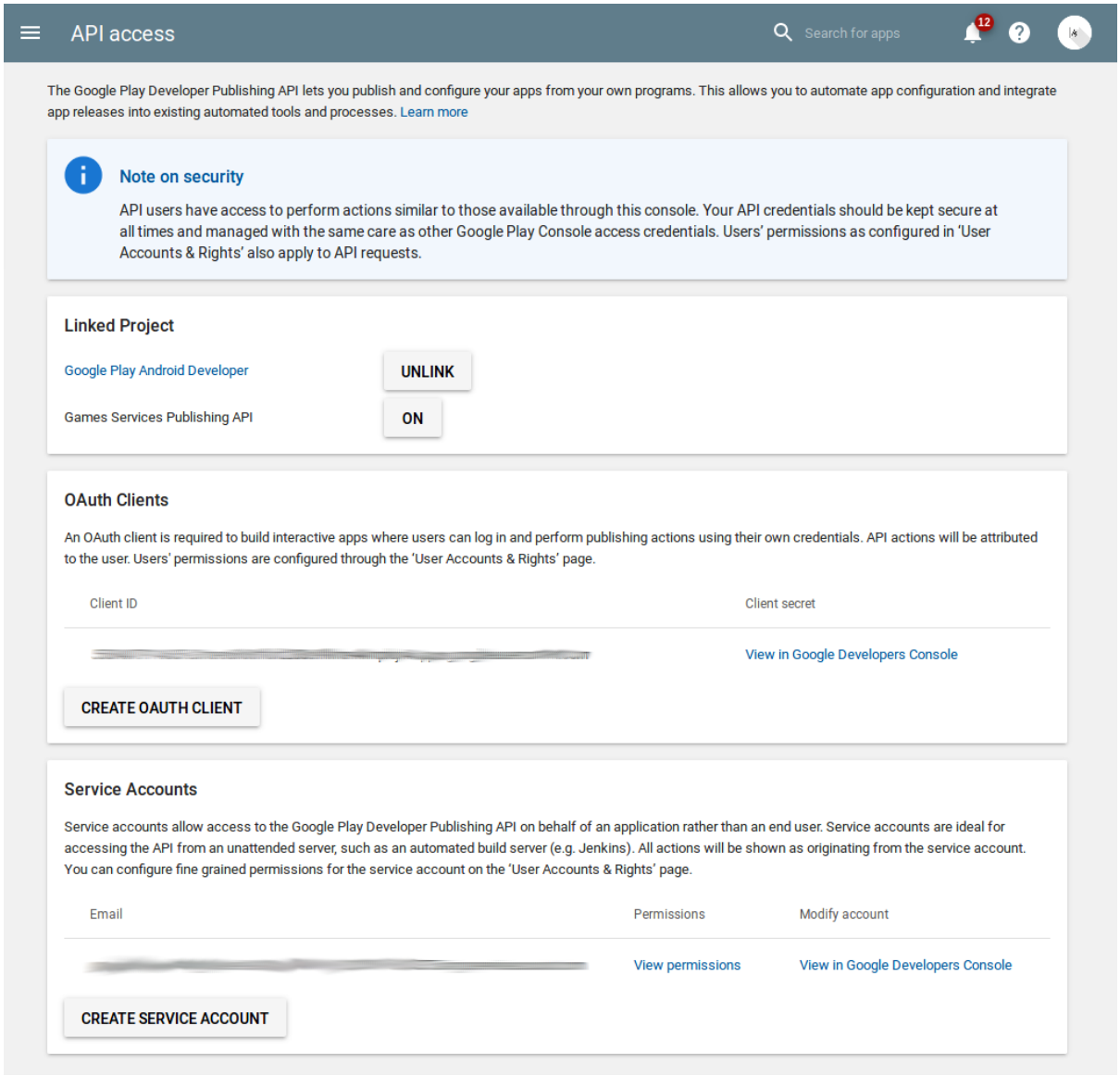
27 DEC 2017 • 1 min read

Inorder to publish the release build on google play we need to setup our developer account.

Setting google play developer account for publishing.

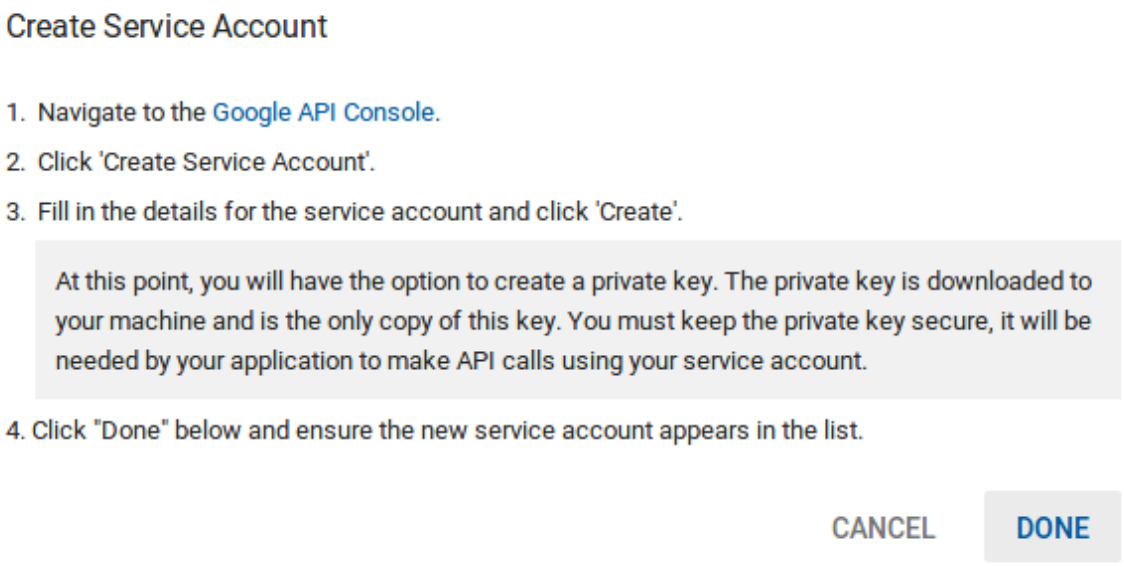
After logging into google play developer account, go into `Settings > Developer account > API access`

Now, link `Google Play Android Developer` under Linked Project section.

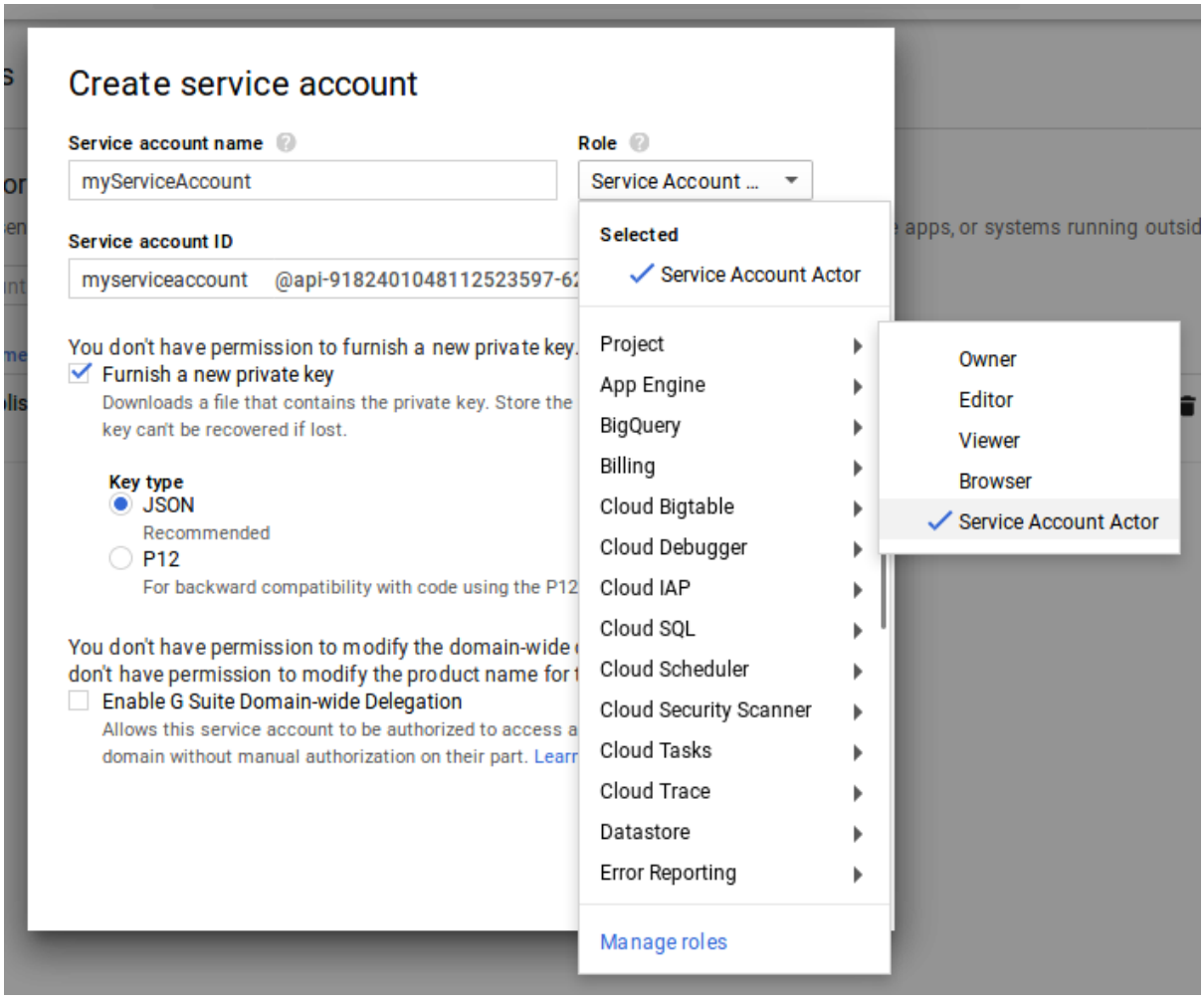


After linking the account, create a OAuth Client and Service Account. Creating OAuth Client is very easy, just click on `CREATE OAUTH CLIENT` button and it will create an OAuth Client.

And, for Service Account, after clicking on `CREATE SERVICE ACCOUNT`, it will show a dialog indicating that you need to visit `Google API Console` page and manually create a service account.



On `Google API Console` page, click the button on top named `Create Service Account`. Set the service account name and select the role as `Service Account Actor` which can be found under `Project` section in `Role`dropdown. Make sure to tick the checkbox `Furnish a new private key` and select `JSON` as key type. After all this, clicking on `CREATE` button will create this service account and download a json file which will contain service account details. Keep this json file somewhere safe as it would be required for authenticating with Google Play Access API later on.



This will create a service account for your Google Play Developer Account. Make sure to check the API Access page again to make sure that the created service account is appearing on that page.

That's all for now, in next part we will do the actual stuff and upload your signed apk on google play :)

Automating Google Play Publishing - Part 3

5 JAN 2018 • 5 mins read

For uploading APK to google play automatically, we can either use the API's provided by google itself to manually upload APK or use an existing solution. Good thing for us is that there is already an existing open source gradle plugin for publishing apps to google play, so we don't need to get our hands dirty.

The plugin we are going to use is named **gradle play publisher** that will automate all this stuff for us.

Including the plugin in your project

Add this to your project level build.gradle file.

```
buildscript {
    repositories {
        ...
    }
    dependencies {
        ...
        classpath 'com.github.triplet.gradle:play-publisher:1.2.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

And, in your app (or module) level build.gradle file.

```
apply plugin: 'com.android.application'
apply plugin: 'com.github.triplet.play'

android {
    playAccountConfigs {
        defaultAccountConfig {
            serviceAccountEmail = "service_account_email_here"
            jsonFile = file("someDirectory/my_json_key.json")
        }
    }

    defaultConfig {
```

```
        ...
        playAccountConfig = playAccountConfigs.defaultAccountConfig
    }

    signingConfigs {
        release {
            storeFile file("someDirectory/my_keystore.jks")
            storePassword "my_store_pass_here"
            keyAlias "my_key_alias_here"
            keyPassword "my_key_pass_here"
        }
    }

    buildTypes {
        release {
            signingConfig signingConfigs.release
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard
-rules.pro'
        }
    }
}

play {
    track = 'production'
}
```

You can also modify the `play` section according to your own needs.

```
play {
    track = 'production' // or 'rollout' or 'beta' or 'alpha'
    userFraction = 0.2 // only necessary for 'rollout', in this case default is 0.1
(10% of the target)
    untrackOld = true // will untrack 'alpha' while upload to 'beta'
```

```
}
```

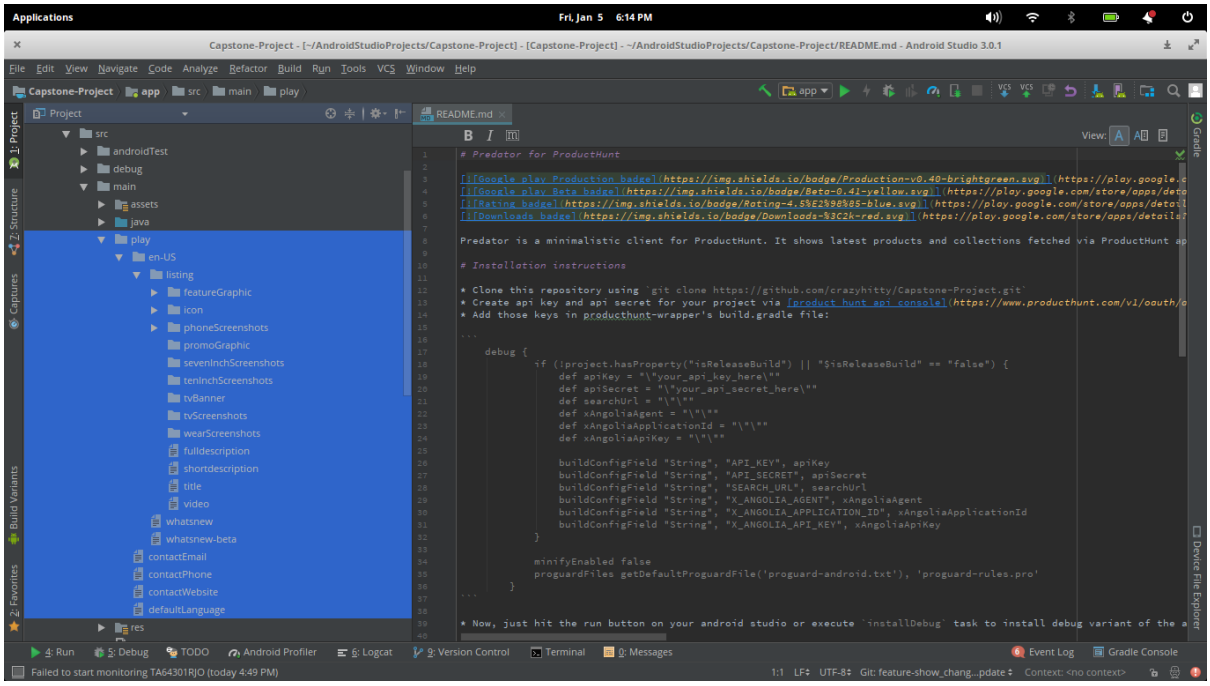
Here is the description about properties used in `play` section.

Property	Description
track	Defines on which channel the APK should be published. Can be one of the **production**, **b**
userFraction	Rollout the APK for some users only. Default value is 0.1 (which means 10%). This proper
untrackOld	Doesn't track for old APKs for version conflicts.
errorOnSizeLimit	Plugin checks if the provided details like **title**, **short description**, **long description** and **re** property to false. By default it is true.
uploadImages	Upload images for Google Play Store listing. By default it is false.

PS: In the previous blog post we discussed about how to create a service account and json key file. These properties are required inorder to authenticate with your google play account. Please check the previous blog to know more about these properties.

Preparing for release 🚀

Inorder to prepare a release, first we need to setup the appropriate metadata which will be used by the plugin.



You can either create these directories and files manually or use this gradle task to create them automatically. Make sure you are in your project directory while running this command.

```
$ ./gradlew bootstrapReleasePlayResources
```

This would create all the resources required in order to publish your APK.

PLAY STORE METADATA STRUCTURE

```
- [src]
  |
  + - [main]
      |
      + - [play]
          |
          + - [en-US] -> Locale in which your content would be published
          |   |
          |   + - [listing] -> Details about play store listing
          |   |   |
          |   |   + - fulldescription
          |   |   |
          |   |   + - shortdescription
          |   |   |
          |   |   + - title
          |   |   |
          |   |   + - video
          |   |
          |   + - whatsnew -> For showing changelog/features of the latest update
          |
          + - [de-DE]
          |   |
          |   + - [listing]
          |   |   |
          |   |   + - fulldescription
          |   |   |
          |   |   + - shortdescription
          |   |   |
          |   |   + - title
```

```
|   |   |
|   |   + - video
|   |
|   + - whatsnew
|
+ - contactEmail
|
+ - contactPhone
|
+ - contactWebsite
|
+ - defaultLanguage -> Default locale for publishing. (For example: en-US
)
```

For more info about play store metadata, check the plugin's readme section.

After setting it up, we can finally publish the app using this plugin.

```
$ ./gradlew publishApkRelease
```

This command will create a signed APK for your project and then publish it on google play with the summary of recent changes (whatsnew). Your release would now be available on Google Play Developer Console. Just check it once to make sure it release was successful or not.

PS: Make sure to bump up versionCode and versionName in your `build.gradle` file.

GRADLE TASKS PROVIDED BY THIS PLUGIN

Task	Description
publishApkRelease	Uploads signed APK and the summary of recent changes.
publishListingRelease	Uploads only images and description.
publishRelease	Uploads everything including signed APK, recent changes,
bootstrapReleasePlayResources	Retrieves current listing details from Google Play and asse

That's it for now folks. Now, you know how to publish signed APK with just a single gradle task. In next part, we will learn how to integrate CI and move the publishing part to CI.