Enterprise developer

Private app publishing guide

Nov, 2017

Introduction

This document is designed as a step by step supplement to the current help center guide on private apps to help enterprise customers navigate publishing private Android apps and also describe best practices.

Private apps are similar to any other app on Google Play with the exception of them being restricted in distribution to the target enterprise(s). On that basis, all requirements, policies and best practices for distributing apps on Google Play apply. Please refer to the Google Play site for Android developers for full details.

1. Sign up

Sign up for a Google Play developer account.

You'll need to pay a $25 signup fee with a credit card to complete the sign up. You will be sent a receipt for this payment which you can use for your expenses.

Naming your developer account

The name of your developer account will be visible to your users. Even if you're exclusively developing private apps, you should use this opportunity to make it clear to your users that these apps are coming from your own development team.

Also, be aware that the developer name is included in the meta-data that is reviewed for compliance to the Google Play Policy. You should ensure that it does not contain the trademarks of any other company, including Google. See section 7 below about trademarked and copyrighted content for more details.

Who should the developer account owner be?

Google recommends signing up for a developer account using a Google Account that isn't owned by a single user, and storing the credentials in a safe place, such as a shared corporate password store. There can be only one owner of the developer account, and this cannot be changed, so to avoid having to migrate your apps to a new developer account if an employee leaves, hold these details centrally.

Then for day to day use of the developer account, setup employees' individual accounts as delegated administrators of the Developer Account, rather than using the shared credentials of the owner. See details here.

If you're using Google Cloud Identity or GSuite, you should use one of your domain administrator accounts.

If you're using Managed Google Play Accounts, Google recommends that you:

1. create that account with a username in your corporate domain, e.g. admin@acme.com. This will demonstrate to Google that the developer account genuinely belongs to your company.
2. Avoid use of any 3rd party brands in the username and your developer account name, including "Google" and "Play". Otherwise, this may be interpreted as a violation of Google Play Policy.

How many developer accounts do I need?

In general, you should only need to sign up for one developer account to publish all of your private apps in your organization, because:

- You can delegate access to other users, so multiple users / developers can publish different apps, and you can set different privileges for different users within Play Console, helping to separate control of apps that might be developed by different teams.
- There are no limitations on the number of apps that can be published.
- There is no expiration date of your account.

There may be reasons that you need to have multiple developer accounts in your organization, and this is possible. For example, you want to have complete separation of apps developed by different teams.

2. Generate your app signing certificate

Your apps need to be digitally signed with a certificate before they can be installed on Android. Every update of your app needs to be signed with the same certificate. If you lose your certificate, then you will not be able to update your app. You will need to create a new app and migrate your users to that. For this reason, Google recommends to manage your app signing keys in Google Play. Find out more details here.

You can still sign the app yourself if you choose, but you should take care to protect your signing key ...

How should it be protected?

Once you've created your keystores that contain your certificates (debug and prod), Google recommends you back these up to a safe place that can be retrieved by key members of your organization. By doing this you ensure that the same keys can be used to sign your app updates throughout the lifecycle of your app even if employees leave or machines get refreshed. Find out more about securing your key here.

Who should own it / manage it?

Make sure there is always a member of your team who has access to the keystore files in your shared storage. Ideally this should be a team, so that if any one person leaves the company, then the team still maintains access.

3. Build and sign your APK

If you've previously distributed your APK through another enterprise app store platform, there are a few things that you should be aware of and check before distributing your APK through Google Play:

- Package names must be globally unique in Google Play, not just unique within your enterprise or developer account. If your package name is the same as a package name already in Google Play then you will need to recompile your app with a different package name.
- Google Play requires your app to be byte aligned to optimize RAM usage. You can use the zipalign tool for this. Learn more.
- Google Play requires that the key you use to sign your APK must have a long validity period. See this article for the latest requirement.

4. Define your app

Please refer to detailed instructions about publishing a private app here.

As part of publishing a private app, you can choose to publish your app to multiple organizations, as needed. You might wish to do this if you have multiple EMMs or EMM tenants, each of which is bound to managed Google Play separately. Each EMM binding has a different Organization ID, which you will need to add to the list of Organizations that can get your app.

Bear in mind the Play publishing policies listed at the Play Developer Policy Center.

5. Decide to host the APK with Google or self-host

See this existing help article

The APK binary for private apps can be hosted in one of two ways. In both cases, the app can only be distributed to users within your organization. The decision you make here is likely to be about where you wish the APK to be hosted, however, there are many benefits to hosting APKs with Google.

Google hosted

Developer uploads private APKs to Google Play and then Google Play is responsible for serving those APKs to users within the enterprise. By hosting your APKs on Google Play, you get a number of benefits:

1. Reliability and service
    a. High-uptime Google servers
    b. Automated device compatibility testing
    c. Faster downloads
    d. Compressed patch updates, rather than downloading the whole APK each time your app is updated.
    e. Benefit from Cloud Test Lab pre-launch reports which can help you identify runtime issues across a range of Android devices before your app hits the field.
2. Easy administration
    a. Google-managed servers, so you don't have to manage or support your own server
    b. Alpha and beta channels to manage new app pilot programs and new app versions
    c. Automatic generation of APK store-listing meta-data, which allows compatibility with Android Wear, Android Auto, and Android TV
3. Security
    a. Google's enterprise-grade security, including SSL downloads.
    b. Malware security scanning.
    c. Alerts if we detect security vulnerabilities in versions of SDKs that you're using, such as Heartbleed in OpenSSL.

Self-hosted

Developer uploads JSON formatted meta-data to Google Play and hosts the APK on private servers. Downloads of that APK can be authorized from that server using a Java Web Token (JWT) which is provided by Google Play when the install is triggered.

Generating the JSON metadata

In the first step you'll need to generate a JSON file, which has details both about your server and the APK. Google provides a Python script that can be used to generate the JSON file. Your EMM might provide a tool for this as part of their platform.

The script requires the following be installed available in your path environment variable:

- OpenSSL
- JDK
- Python 2.x
- Android Asset Packaging Tool (aapt) – Included in the Android SDK Build Tools

The script requires the APK file that was just uploaded to your server and the URL that it can be retrieved from. Running the command as follows will display the JSON file to your console screen:

```
python externallyhosted.py -apk=<path/to/apk.apk>
-externallyHostedUrl="https://www.example.com/test.apk"
```

You can use this method and copy it, or redirect the output to a file by appending " > filename.json" to the end of the command (without the quotes):

```
python externallyhosted.py -apk=<path/to/apk.apk>
-externallyHostedUrl="https://www.example.com/test.apk" > filename.json
```

Or you can do both to see what it looks like first and then save it to a file. There isn't any limit on the number of times you can run it.

Constraints on self-hosted apps

Google also imposes some constraints on serving self-hosted apps to protect Android users from APKs that have not been through Google's thorough app scanning system:

1. Android Auto second-screen projection is disabled. This is because all Auto-targeted apps must go through a specific review by Google to ensure that they're not distracting to drivers.
2. To remotely install self-hosted apps on devices, the device must be in managed device mode (aka Device Owner). This typically applies to devices that are owned and fully managed by the company. It's not possible to remotely push install a self-hosted app to a work profile.
3. If a user wants to install self-hosted apps, they must have a work profile on their device, or have a managed device.
4. Self-hosted apps are published to the production channel only, and not to an alpha or beta channel.

6. Create a release

Play developer console publishing help center
Video overview of release management

Choose which channel (alpha/beta/production) you want your app to start in. If you want your app to be generally available to your users, then choose the production channel. NOTE: only production channel is available for self-hosted APKs.

Create a release for that channel (Manage Releases > Manage <channel> > Create release) and upload your APK.

Alpha / beta testing

Google recommends that you define Alpha and Beta groups of users to test your app before wide rollout. For example:

- Use an alpha group for your development and/or QA team. They'll get the earliest builds and do comprehensive testing both in the lab and in the field.
- Use a beta group for your early adopters in your business. You should choose users who a) don't mind software being a little bit buggy, and b) will give you great feedback and file bugs.
  Note: Google Play provides crash reporting, which you can use to identify critical bugs by version, but you need to implement your own feedback mechanism if you want more detailed feedback from your users.

Alpha / beta testing is available for any enterprise using managed Google Play. You just have to target your alpha or beta release to your organization ID(s) and then manage your alpha and beta user groups through your EMM. Learn more about targeting alpha / beta versions to organizations here; and see your EMM's documentation for assigning users to alpha or beta tests.

f you want to host your APK on your own servers using a "self-hosted APK" then these can only be published in the production channel.

7. Are you using trademarked or copyrighted content?

If you're using trademarked graphics or text, like a company name or logo, or you're using licensed or copyrighted intellectual property, then you should notify Google that you have permission prior to publishing your app, otherwise this may be interpreted as a violation of Google Play Policy, even if it is your own company brand. You can do that here. You should wait for a response after submitting your permission docs before submitting your app.

Remember, you wouldn't want a 3rd party developer using your trademark, so we need to confirm that your developer account has permission to use it.

Also, do not republish an APK that you've acquired from a 3rd party or external source without providing permission documentation, as above.

8. Publish your app

Hit the publish button and work through any errors and warnings that are flagged up by Google Play.

Play will automatically check for security vulnerabilities and notify you if any of these are found, through the app security improvement program. In some cases Play will block publishing of the app if it detects a vulnerability that has passed the program remediation time limits. Policy compliance will also be checked.

9. Distribute your app

It normally takes around 2 hours for an app to go through the approval process and be available in your EMM console, however, sometime it can take longer than this.

Head back to your EMM console. You will first need to approve the private app, in the same way as you would any public app. Then you will be able to distribute your new private app to your users.

Other situations

Updates

You can update your app by creating a new release within developer console. Just make sure you create a new release against the correct app. Upload your APK.

Staged rollouts

You should consider using percentage rollouts to control the update of your app when you release to production. This allows you to spot any issues that arise in the field but also minimize the impact of any bug that might have crept into your production release. Learn more here.

If you want explicit control of who gets an early version of the app, then you should use alpha / beta testing.

Version Codes

When updating your app you need to assign a new versionName, which your users see, and also a versionCode, which Google Play uses to determine which version of the app is most recent. Google Play only updates an app on a device to a **higher** versionCode. Learn more about setting versionCodes in Android here.

When assigning versionCodes for apps you distribute through Google Play, you should consider spacing your versionCodes by a large number, like 1000 per minor version. This gives you flexibility to iterate your app within the same version name or ID that you share with your users. Bear in mind though, that the highest versionCode in Google Play is 2,100,000,000.

For example:

- you have just finished development of version 1.1 of your app.
- You assign versionCode 11,000 to it.
- You deploy it using Google Play to your alpha test group and find some bugs.
- You do a bug fix release, and make this versionCode 11,001 and then make more incremental changes, getting to versionCode 11,026.
- You then move this to your beta group as versionCode 11,100, do some more iteration to versionCode 11,105.
- Finally push to production at 10%, as versionCode 11,500.
- You realize that there's a problem and need to roll back to 1.0, which you build as 11,501 and release.
- You respin v1.1 with the fix and successfully roll it out to all users as versionCode 11,600.
- All this time the version ID or name that is shown to users is v1.1 (apart from rollback to v1.0), and you've preserved your version code convention so that 12,000 can be your v1.2, which might already be in the test pipeline with your alpha group, so you don't want the various respins of v1.1 affecting QA of v1.2.
- You've also given yourself more wiggle room in versionCodes across the base of your various test groups in case you need to distribute interim versions.

Rollbacks

Google Play does not allow straight rollbacks to an older version of the app, since it enforces that every overwrite of an app needs to be to a higher versionCode. If you have a problem during an update that you need to rollback, then:

1) Halt the current rollout (see more here).
2) Create a new release with the old version of the APK, but increment the versionCode of the APK to a value higher than the broken version.
3) Publish the rollback release to the same target users as the broken version.

If you need to rollback in a hurry, rather than waiting for Google Play's auto-update mechanism then your EMM should provide you with the ability to re-push your app immediately to users. This will force an immediate app update.

You might also consider remotely uninstalling the app and then push installing it again, which you should also be able to do through your EMM, however, you should be aware that this will delete any app data that the app had stored on the device. The app would therefore need to be setup again from scratch.

Developer wants to distribute an app to multiple customers without making it available on the public Play Store

2 options here

1. Using private app publishing, it's possible for any Google Play developer to target up to 20 enterprises using their Organization IDs, which the enterprise can get by

logging into play.google.com/work with their admin user account and opening “Admin settings”.
2. If the developer wishes to target a large number of customers, they should strongly consider publishing a public app and then use managed configuration for any customisations per customer.

Scripting / programmatically publishing apps

Google Play allows you to programmatically publish and update your apps directly from your internal build tools. Publishing apps is supported using the Google Play Custom App Publishing API, as an alternative to using the Play Console. And once you’ve defined your app the first time, you can make updates, and add new releases, including alpha and beta using the Google Play Publishing API.