Google Play Android Publisher Plugin

Plugin Information
View Google Play Android Publisher on the plugin site for more information.

Older versions of this plugin may not be safe to use. Please review the following warnings before using an older version:

- Users with Overall/Read access can enumerate credential IDs

Enables Jenkins to upload Android apps (APK files) and related info to Google Play.

Features

- Uploading APK files to Google Play
  - This includes apps which use Multiple APK support
  - ProGuard mapping.txt files can also be associated with each APK, for deobfuscating stacktraces
- Uploading APK expansion (.obb) files
  - With the option to re-use expansion files from existing APKs, e.g. for patch releases
- Assigning apps to alpha, beta or production release tracks
  - This includes a build step for moving existing APKs to a different track
  - e.g. You can upload an alpha in one job, then later have another job promote it to beta
- Staged rollout of apps to any release track
- Assigning release notes to uploaded files, for various languages
- Changing the build result to failed if the configuration is bad, or uploading APKs fails for some reason
- Every configuration field supports variable and token expansion, allowing release notes to be dynamically generated, for example
- Integration with the Google OAuth Plugin, so that credentials can be entered once globally and shared between jobs
  - Multiple Google Play accounts are also supported via this mechanism

Potential upcoming features

- Uploading screenshots and text for the app
- Updating recent changes text on an existing APK (JENKINS-32109)

Requirements

Jenkins

Jenkins version 2.60.3 or newer is required.

Google Play publisher account

For the initial setup only, you must have access to the Google account which owns the Google Play publisher account.

This is required to enable API access from Jenkins to your Google Play account.

Note that having admin access is not enough; you need the account owner.
You can see who the account owner is under Settings User accounts & rights in the Google Play developer console.

Please note

- Any APKs uploaded will be published by Google Play immediately; they will not be held in a draft or pending state
- The app being uploaded must already exist in Google Play; you cannot use the API to upload brand new apps

Setup

One-time: Set up Google Play credentials

The following initial setup process is demonstrated in this video: https://www.youtube.com/watch?v=txdPSJF94RM
(note that Google has changed the Google API Console (twice) since this video was recorded; steps 3–13 in the "Create Google service account" section below have the updated info)

Install plugin

Install this plugin via the Jenkins plugin manager.
Or if installing the plugin via other means, ensure that the prerequisite Google OAuth Plugin, Token Macro Plugin and their dependencies are also installed.

Create Google service account

To enable automated access to your Google Play account, you must create a service account:

1. Sign in to the Google Play developer console as the account owner
2. Select Settings Developer account API access
3. Under Service Accounts, click "Create Service Account"
4. Follow the link to the Google API Console
5. Click the "Create service account" button
6. Give the service account any name you like, e.g. "Jenkins"
7. Choose Service Accounts > Service Account User for the "Role" field
8. Enable "Furnish a new private key"
9. Choose "JSON" as the key type (P12 works as well, but JSON is a little simpler)
10. Click the "Save" button
11. Note that a .json file is downloaded, named something like "api-xxxxxxxxx-xxxxx-xxxx.json"
12. Close the dialog that appears
13. Copy the email address of the new user (something like "jenkins@api-xxxxxxxxx-xxxxx-xxxx.iam.gserviceaccount.com")
14. You can now close the page

Assign permissions to the service account

1. Return to the Google Play developer console page
2. Click "Done" on the dialog
3. Note that the service account has associated with the Google Play publisher account
    a. If it hasn't, follow these additional steps before continuing:
    b. Click "Users & permissions" in the menu
    c. Click "Invite new user"
    d. Paste in the email address you copied above
    e. Continue from step 5
4. Click the "Grant access" button for the account (e.g. "jenkins@api-xxxxxxxxx-xxxxx-xxxx.iam.gserviceaccount.com")
5. Ensure that at least the following permissions are enabled:
    - **View app information** — this is always required
    - **Manage production releases** — if you want to upload APKs to production
    - **Manage testing track releases** — if you want to upload APKs to alpha, beta, or internal
6. Click "Add user" (or "Send invitation", as appropriate)
7. You can now log out of the Google Play publisher account

Add the service account credentials to Jenkins:

1. Navigate to your Jenkins instance
2. Select "Credentials" from the Jenkins sidebar
3. Choose a credentials domain and click "Add Credentials"
4. From the "Kind" drop-down, choose "Google Service Account from private key"
5. Enter a name for the credential — the actual value is not important
6. Choose the "JSON key" type
7. Upload the .json file that was downloaded by the Google API Console
8. Click "OK" to create the credential

Jenkins now has the required credentials and permissions in order to publish to Google Play.

Once you've set up a job (see the next section) and confirmed that uploading works, either delete the downloaded JSON file or ensure that it's stored somewhere secure.

Per-job configuration

Freestyle job configuration

Uploading an APK

The following job setup process is demonstrated in this video: https://www.youtube.com/watch?v=iu-bLY9-jkc

1. Create a new free-style software project
2. Ensure, via whatever build steps you need, that the APK(s) you want to upload will be available in the build's workspace
3. Add the "Upload Android APK to Google Play" post-build action
4. Select the credential name from the drop-down list
    - The credential must belong to the Google Play account which owns the app to be uploaded
5. Enter paths and/or wildcards pointing to the APK or APKs to be uploaded
    - This can be an Ant-style `**/*-release.apk` pattern, or a comma-separated list of filenames, relative to the root of the workspace
6. Choose the track to which the APKs should be deployed
    - You can optionally specify a rollout percentage, as well as ProGuard obfuscation mapping files
7. Optionally choose "Add language" to associate release notes with the uploaded APK(s)
    - You add entries for as many or as few of your supported language as you wish, but each language must already have been added to your app, under the "Store Listing" section in the Google Play Developer Console.

APK expansion files

You can optionally add up to two expansion files for each APK being uploaded.

A list of expansion files can be specified in the same way as APKs, though note that they must be named in the format `[main|patch].<expansion-version>.<package-name>.obb`.

See the inline help for more details.

Moving existing APK(s) to another release track

If you have already uploaded an app to the alpha track (for example), you can later use Jenkins to re-assign that version to the beta or production release track.
Under the "Build" section of the job configuration, add the "Move Android APKs to a different release track" build step and configure the new release track.

You can tell Jenkins **which** APKs to be moved by either entering the APK version codes directly, or by providing the APK files, from which the plugin will read the application ID and version codes for you.

Pipeline job configuration

As of version 1.5, this plugin supports the Pipeline Plugin syntax. You can generate the required Pipeline syntax via the Snippet Generator, but some examples follow.

Note that you should avoid using these steps in a `parallel` block, as the Google Play API only allows one concurrent "edit session" to be open at a time.

Uploading an APK

The `androidApkUpload` step requires at least the Google Play credential ID, a list of APK(s) to upload, and the track to assign them to:

```
androidApkUpload googleCredentialsId: 'My Google Play account', apkFilesPattern: '**/*.apk', trackName:
'production'
```

Uploading the ProGuard mapping file(s) to be associated with the APK(s) just requires one more more patterns in addition:

```
androidApkUpload googleCredentialsId: 'My Google Play account', apkFilesPattern: '**/*.apk',
                 deobfuscationFilesPattern: '**/mapping.txt', trackName: 'production'
```

Performing a staged rollout requires the rollout percentage as a string; the percentage sign is optional:

```
androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '**/*.apk', trackName: 'production',
rolloutPercentage: '50%'
```

Adding "recent changes" text requires specifying a list:

```
androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '**/*.apk', trackName: 'alpha',
                 recentChangeList: [
                     [language: 'en-GB', text: "Please test the changes from Jenkins build ${env.
BUILD_NUMBER}."],
                     [language: 'de-DE', text: "Bitte die Änderungen vom Jenkins Build ${env.BUILD_NUMBER}
testen."]
                 ]
```

To upload expansion files, reusing those from the previous upload where possible:

```
androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '**/*.apk', trackName: 'production',
                 expansionFilesPattern: '**/patch.obb',
                 usePreviousExpansionFilesIfMissing: true
```

Moving existing APK(s) to another release track

The `androidApkMove` step requires at least the Google Play credential ID and the track to move APK(s) to, plus either an application ID and version code (s), or APK file(s) to read this information from.

Moving APKs from alpha to beta, specifying the application ID and version codes:

```
androidApkMove googleCredentialsId: 'My Google Play account',
               applicationId: 'com.example.app',
               versionCodes: '1050, 2050, 3050',
               trackName: 'beta'
```

Moving APKs from beta to staged rollout, specifying the application ID and version codes via the uploaded APKs:

```
androidApkMove googleCredentialsId: 'My Google Play account',
               fromVersionCode: false,
               apkFilesPattern: '**/*.apk',
               trackName: 'production',
               rolloutPercentage: '5'
```

Feedback wanted

If you're a user of expansion files, especially if you use multiple APKs, it would be helpful to know how you normally use expansion files, and whether what the plugin provides is adequate.
Do you normally have separate main/patch expansion files for each of the APKs, or is it more common for all APKs to share the same expansion files?

Currently the former case isn't possible when trying to re-use expansion files from existing APKs (without having you explicitly specify which expansion files from which old APKs should be re-used with which newly-uploaded APKs).

Let me know via the maintainer email at the top of the page!

Troubleshooting

Error messages from the plugin (many of which come directly from the Google Play API) should generally be self-explanatory.
If you're having trouble getting a certain config to work, try uploading the same APKs manually to Google Play. There you'll likely see the reason for failure, e.g. a version code conflict or similar.

Otherwise, please file a bug report or send an email with details, including the build console log output; see info box at the top of the page.

Some known error messages and their solutions are shown below:

GoogleJsonResponseException: 401 Unauthorized

This means that the Google service account does not have permission to make the changes that you requested.

Make sure that you followed the setup instructions above, and confirm that the service account you are using in this Jenkins job has the appropriate permissions for the app that you are trying to change.

GoogleJsonResponseException: 500 Internal Server Error

Unfortunately, the Google Play API sometimes is not particularly reliable, and will throw generic server errors for no apparent reason.

In these cases you can try running your build again, or wait a few hours before retrying, if the problem persists.

Please also consider contacting Google Play Developer Support to help make them aware that people use the Google Play API, and that it should preferably work in a reliable manner.

This plugin already recognises some temporary Google Play API server problems and works around them; more workarounds may be added in future, e.g. automatically retrying when a generic server error is encountered.

Unable to retrieve an access token with the provided credentials

If you see this error message, look further down the error log to see what is causing it. Below are a couple of common causes:

Invalid JWT: Token must be a short-lived token and in a reasonable timeframe

Ensure that the time is correctly synchronised on the build machine, and then try again.

java.net.SocketTimeoutException: connect timed out

This likely means your build machine is behind an HTTP proxy.

In this case, you should set up Jenkins as documented on the JenkinsBehindProxy page.

This plugin only makes secure (HTTPS) requests, so you need to make sure that the `-Dhttps.proxyHost=<hostname>` and `-Dhttps.proxyPort=<port>` Java properties are set when starting Jenkins. Add the appropriate http versions of those flags too, if unsecured HTTP requests also need to go through the proxy.

Frequently asked questions

What if I want to upload APKs with multiple, different application IDs (i.e. build flavours)?

Using the build flavours feature of the Android Gradle build system, it's possible to have a single Android build which produces multiple APKs, each with a different application ID.
e.g. You could have application IDs "com.example.app" and "com.example.app.pro" for free and paid versions.

As these will often be built in a single Jenkins job, people have wondered why this plugin will refuse to upload APKs with differing application IDs in a single freestyle job.

However, as far as Google Play is concerned, these are completely separate apps. This is correct and, as such, they should be uploaded in separate Jenkins builds: one per application ID.

If the plugin did allow this and you were to attempt to upload, say three, completely different APKs in one Jenkins build, this would require opening and committing three separate "edit sessions" with the Google Play API. If any one of these were to fail — maybe because of an invalid APK, versionCode conflict, or due to an API failure (which, unfortunately, is not uncommon with the Google Play API) — you would end up with your Google Play account in an inconsistent state. Your Jenkins build would be marked as failed, but one or more applications will have actually been uploaded and published to Google Play, and you would have to fix the situation manually. Also, you would not be able to simply re-run the build, as it would fail due to already-existing APKs.

The best practice in this case would be to have one job that builds the different flavours (i.e. the APKs with different application IDs) and then, if the build is successful, it would archive the APKs and start multiple "downstream" Jenkins builds which individually publish each of the applications.
This can be achieved, for example, with the Parameterized Trigger Plugin and the Copy Artifacts Plugin, i.e. the "upload" job could be generic, and would receive the APK information via parameter.

Alternatively, if you have version 1.5 of this plugin, and use the Pipeline Plugin, you should be able to use the `androidApkUpload` step multiple times within a single build.

Version history

Version 2.0 (July 17, 2019)

- Upgraded to v3 of the Google Play Developer Publishing API (thanks to Joe Hansche)
- Fixed various potential NullPointerExceptions if no APKs had been uploaded already (thanks to Kazuhide Takahashi)
- Increased minimum Jenkins version to 2.60.3

Version 1.8 (June 3, 2018)

- Enabled ability to upload to the "internal" track (thanks to Serge Beauchamp)
- Allowed arbitrary percentage values to be used for staged rollouts
- Fixed potential NullPointerException if something went wrong (JENKINS-49789)

Version 1.7 (February 26, 2018)

- Fix security issue

Version 1.6 (January 18, 2018)

- Added ability to upload ProGuard mapping files (JENKINS-38731)
- Made it possible to specify the Google Play credential dynamically (JENKINS-38613)
- Updated Google Play API dependency to hopefully make uploads a bit more stable
- Improved various error messages and configuration validation
- Increased minimum Jenkins version to 2.32
- Thanks to Christopher Frieler and Dragan Marjanovic

Version 1.5 (September 8, 2016)

- Added support for the Pipeline Plugin

Version 1.4.1 (August 27, 2015)

- Ensured that the required Token Macro Plugin is installed (see JENKINS-29887)

Version 1.4 (July 30, 2015)

- Fixed the Google Play credential being forgotten when editing a job's configuration (see JENKINS-26542)
- Improved error messages when Google Play credentials are missing or misconfigured (see JENKINS-25933)
- Improved error messages shown when reading info from an APK fails
- Fixed or improved various aspects of job configuration validation
- Added additional logging about the credential ID and app ID being used
- Integrated the Token Macro Plugin to give more options for specifying "Recent Changes" text (or any other field)
    - For example, with version 1.11 of token-macro, it will be possible to use something like `${FILE, path="changes-en.txt"}`, to read a changelog from the workspace and use its contents as release notes when uploading an APK to Google Play

Version 1.3.1 (April 1, 2015)

- Improved the logging of which and how many APK files are to be uploaded/re-assigned

Version 1.3 (March 22, 2015)

- Added new build step to enable moving existing APKs to a different release track

Version 1.2 (November 1, 2014)

- Added ability to automatically recover from API failures when committing changes, if possible (see JENKINS-25398)

Version 1.1 (October 27, 2014)

- Added ability to upload APK expansion files
- Upgraded APK parser library to fix some potential crashes

Version 1.0 (October 4, 2014)

- Initial release