

[Get started](#)

[Samples & tutorials](#)

[Development](#)

▸ [User interface](#)

▸ [Data & backend](#)

▸ [Accessibility & internationalization](#)

▸ [Platform integration](#)

▸ [Packages & plugins](#)

▸ [Add Flutter to existing app](#)

▸ [Tools & techniques](#)

▸ [Migration notes](#)

[Testing & debugging](#)

[Performance & optimization](#)

[Deployment](#)

[Obfuscating Dart code](#)

[Creating flavors for Flutter](#)

[Build and release an Android app](#)

[Build and release an iOS app](#)

[Build and release a web app](#)

[Continuous deployment](#)

[Resources](#)

[Reference](#)

[Widget index](#)

[API reference](#) 

[Package site](#) 

Continuous delivery with Flutter



Contents

- [fastlane](#)
 - [Local setup](#)
 - [Running deployment locally](#)
 - [Cloud build and deploy setup](#)
 - [Reference](#)
- [Other services](#)













Follow continuous delivery best practices with Flutter to make sure your application is delivered to your beta testers and validated on a frequent basis without resorting to manual workflows.

fastlane

This guide shows how to integrate [fastlane](#), an open-source tool suite, with your existing testing and continuous integration (CI) workflows (for example, Travis or Cirrus).

Local setup

It's recommended that you test the build and deployment process locally before migrating to a cloud-based system. You could also choose to perform continuous delivery from a local machine.

1. Install fastlane `gem install fastlane` or `brew install fastlane`. Visit the [fastlane docs](#) for more info.
2. Create your Flutter project, and when ready, make sure that your project builds via
 -  `flutter build appbundle`; and
 -  `flutter build ios --release --no-codesign`.
3. Initialize the fastlane projects for each platform.
 -  In your `[project]/android` directory, run `fastlane init`.
 -  In your `[project]/ios` directory, run `fastlane init`.
4. Edit the `Appfiles` to ensure they have adequate metadata for your app.
 -  Check that `package_name` in `[project]/android/Appfile` matches your package name in `AndroidManifest.xml`.
 -  Check that `app_identifier` in `[project]/ios/Appfile` also matches `Info.plist`'s bundle identifier. Fill in `apple_id`, `itc_team_id`, `team_id` with your respective account info.
5. Set up your local login credentials for the stores.
 -  Follow the [Supply setup steps](#) and ensure that `fastlane supply init` successfully syncs data from your Play Store console. *Treat the .json file like your password and do not check it into any public source control repositories.*
 -  Your iTunes Connect username is already in your `Appfile`'s `apple_id` field. Set the `FASTLANE_PASSWORD` shell environment variable with your iTunes Connect password. Otherwise, you'll be prompted when uploading to iTunes/TestFlight.
6. Set up code signing.
 -  On Android, there are two signing keys: deployment and upload. The end-users download the .apk signed with the 'deployment key'. An 'upload key' is used to authenticate the .aab / .apk uploaded by developers onto the Play Store and re-signed with the deployment key once in the Play Store.
 - It's highly recommended to use the automatic cloud managed signing for the deployment key. For more information see the [official Play Store documentation](#).
 - Follow the [key generation steps](#) to create your upload key.
 - Configure gradle to use your upload key when building your app in release mode by editing `android.buildTypes.release` in `[project]/android/app/build.gradle`.
 -  On iOS, create and sign using a distribution certificate instead of a development certificate when you're ready to test and deploy using TestFlight or App Store.
 - Create and download a distribution certificate in your [Apple Developer Account console](#).
 - `open [project]/ios/Runner.xcworkspace/` and select the distribution certificate in your target's settings pane.
7. Create a `Fastfile` script for each platform.
 -  On Android, follow the [fastlane Android beta deployment guide](#). Your edit could be as simple as adding a `lane` that calls `upload_to_play_store`. Set the `aab` argument to `../build/app/outputs/bundle/release/app-release.aab` to use the app bundle `flutter build` already built.
 -  On iOS, follow the [fastlane iOS beta deployment guide](#). Your edit could be as simple as adding a `lane` that calls `build_ios_app` with `export_method: 'app-store'` and `upload_to_testflight`. On iOS an extra build is required since `flutter build` builds an .app rather than archiving .ipkas for release.

You're now ready to perform deployments locally or migrate the deployment process to a continuous integration (CI) system.

Running deployment locally

1. Build the release mode app.

[Get started](#)

[Samples & tutorials](#)

[Development](#)

▶ [User interface](#)

▶ [Data & backend](#)

▶ [Accessibility & internationalization](#)

▶ [Platform integration](#)

▶ [Packages & plugins](#)

▶ [Add Flutter to existing app](#)

▶ [Tools & techniques](#)

▶ [Migration notes](#)

[Testing & debugging](#)

[Performance & optimization](#)

[Deployment](#)

[Obfuscating Dart code](#)

[Creating flavors for Flutter](#)

[Build and release an Android app](#)

[Build and release an iOS app](#)

[Build and release a web app](#)

[Continuous deployment](#)

[Resources](#)

[Reference](#)

[Widget index](#)

[API reference](#)

[Package site](#)

- flutter build appbundle.
- flutter build ios --release --no-codesign. No need to sign now since fastlane will sign when archiving.

2. Run the Fastfile script on each platform.

- cd android then fastlane [name of the lane you created].
- cd ios then fastlane [name of the lane you created].

Cloud build and deploy setup

First, follow the local setup section described in ‘Local setup’ to make sure the process works before migrating onto a cloud system like Travis.

The main thing to consider is that since cloud instances are ephemeral and untrusted, you won’t be leaving your credentials like your Play Store service account JSON or your iTunes distribution certificate on the server.

Continuous Integration (CI) systems, such as Cirrus generally support encrypted environment variables to store private data.

Take precaution not to re-echo those variable values back onto the console in your test scripts. Those variables are also not available in pull requests until they’re merged to ensure that malicious actors cannot create a pull request that prints these secrets out. Be careful with interactions with these secrets in pull requests that you accept and merge.

1. Make login credentials ephemeral.

- On Android:
 - Remove the json_key_file field from Appfile and store the string content of the JSON in your CI system’s encrypted variable. Use the json_key_data argument in upload_to_play_store to read the environment variable directly in your Fastfile.
 - Serialize your upload key (for example, using base64) and save it as an encrypted environment variable. You can deserialize it on your CI system during the install phase with

```
echo "$PLAY_STORE_UPLOAD_KEY" | base64 --decode > /home/cirrus/[directory # and filename specified in your gradle].keystore
```

- On iOS:
 - Move the local environment variable FASTLANE_PASSWORD to use encrypted environment variables on the CI system.
 - The CI system needs access to your distribution certificate. fastlane’s Match system is recommended to synchronize your certificates across machines.

2. It’s recommended to use a Gemfile instead of using an indeterministic gem install fastlane on the CI system each time to ensure the fastlane dependencies are stable and reproducible between local and cloud machines. However, this step is optional.

- In both your [project]/android and [project]/ios folders, create a Gemfile containing the following content:

```
source "https://rubygems.org"

gem "fastlane"
```

- In both directories, run bundle update and check both Gemfile and Gemfile.lock into source control.
- When running locally, use bundle exec fastlane instead of fastlane.

3. Create the CI test script such as .travis.yml or .cirrus.yml in your repository root.

- Shard your script to run on both Linux and macOS platforms.
- Remember to specify a dependency on Xcode for macOS (for example osx_image: xcode9.2).
- See fastlane CI documentation for CI specific setup.
- During the setup phase, depending on the platform, make sure that:
 - Bundler is available using gem install bundler.
 - For Android, make sure the Android SDK is available and the ANDROID_HOME path is set.
 - Run bundle install in [project]/android or [project]/ios.
 - Make sure the Flutter SDK is available and set in PATH.
- In the script phase of the CI task:
 - Run flutter build appbundle or flutter build ios --release --no-codesign, depending on the platform.
 - cd android or cd ios
 - bundle exec fastlane [name of the lane]

Reference


The Flutter Gallery Project uses fastlane for continuous deployment. See the source for a working example of fastlane in action. For more information, see the Flutter framework repository’s Cirrus script.

Other services

The following are some other options available to help automate the delivery of your application.

- Codemagic CI/CD for Flutter
- Flutter CI/CD with Bitrise
- GitHub Actions- CI/CD on GitHub Get an Example Project



Get started	▼ —
Samples & tutorials	▼ —
Development	^
▶ User interface	
▶ Data & backend	
▶ Accessibility & internationalization	
▶ Platform integration	
▶ Packages & plugins	
▶ Add Flutter to existing app	
▶ Tools & techniques	
▶ Migration notes	
Testing & debugging	▼ —
Performance & optimization	▼ —
Deployment	^
Obfuscating Dart code	
Creating flavors for Flutter	
Build and release an Android app	
Build and release an iOS app	
Build and release a web app	
Continuous deployment	
Resources	▼ —
Reference	^
Widget index	
API reference 	
Package site 