



GitLab CI/CD for Flutter Apps

Step-by-Step Guide for Automated Test, Beta, and Production Releases

Note: This guide is intended as an editable Word template. Placeholders such as Bundle ID, Package Name, Team ID, repository URLs, and variable values must be adapted to the respective app before publication.

Element	Description / Example
Goal	Automatically test the Flutter app via GitLab CI/CD and publish it to TestFlight and the Google Play Internal Track.
Production	Production releases are triggered via Git tags and started manually in GitLab.
Tools	GitLab CI/CD, fastlane, App Store Connect API, Google Play Service Account, fastlane match.
Important	iOS requires a macOS runner. Android can be built on a normal Docker runner.

1. Target Workflow

For merge requests and pushes, the pipeline should reliably check whether the Flutter app still builds and can be tested. After a merge into the main branch, beta releases are published automatically. Production releases are intentionally only offered for a Git tag and then started manually.

```
Merge Request:
  flutter analyze
  flutter test

Merge in main:
  Android → Google Play Internal Track
  iOS → TestFlight

Git tag, e.g., v1.0.0:
  Android → Google Play Production
  iOS → App Store Review / Production
```

Note: GitLab does not publish the app itself. GitLab only starts the jobs. The actual store logic is handled by fastlane.

2. Requirements

- ☐ Flutter app is located in a GitLab repository.
- ☐ Google Play Console app has been created.
- ☐ App Store Connect app has been created.
- ☐ Android upload keystore is available.
- ☐ Google Play service account is available and has the required permissions in the Play Console.
- ☐ App Store Connect API key is available.
- ☐ Private GitLab repository for fastlane match is available.
- ☐ macOS runner with Xcode, Flutter, Ruby, and GitLab Runner is set up.
- ☐ Bundle ID / app identifier and Android application ID are finalized.

3. Basic Structure in the Flutter Project

For a consistent workflow, fastlane and the Gemfile should be located in the root of the Flutter project, not only in the ios or android folder.

```
flutter-app/
├─ android/
├─ ios/
├─ lib/
├─ test/
├─ fastlane/
│  └─ Fastfile
│  └─ Matchfile
├─ Gemfile
├─ Gemfile.lock
├─ pubspec.yaml
└─ .gitlab-ci.yml
```

4. Set Up Gemfile and Bundler

Create a Gemfile in the root of the Flutter project. CocoaPods should be run through Bundler so that the pipeline does not suddenly use an outdated or broken global CocoaPods version.

Gemfile

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

gem "fastlane"
gem "cocoapods", "~> 1.16.2"
gem "ffi"
```

```
bundle install
git add Gemfile Gemfile.lock
git commit -m "Add fastlane and cocoapods"
```

Note: In the pipeline, always use `bundle exec pod install`, not just `pod install`.

5. Prepare Android

5.1 Store the Android Upload Keystore in GitLab

The Android release build requires an upload keystore. It is not committed to the repository, but stored in GitLab as a Base64 variable. The following command copies the keystore value to the clipboard:

macOS

```
base64 -i upload-keystore.jks | pbcopy
```

Linux

```
base64 -w 0 upload-keystore.jks
```

Element	Description / Example
ANDROID_KEYSTORE_BASE64	Base64 content of upload-keystore.jks
ANDROID_KEYSTORE_PASSWORD	Store password
ANDROID_KEY_ALIAS	Alias of the upload key
ANDROID_KEY_PASSWORD	Key password

The pipeline uses this to generate `android/app/upload-keystore.jks` and `android/key.properties` at runtime.

```
echo "$ANDROID_KEYSTORE_BASE64" | base64 -d > android/app/upload-keystore.jks
cat > android/key.properties <<EOF
storePassword=$ANDROID_KEYSTORE_PASSWORD
keyPassword=$ANDROID_KEY_PASSWORD
keyAlias=$ANDROID_KEY_ALIAS
storeFile=upload-keystore.jks
EOF
```

5.2 Android Signing in build.gradle

The release build must read the key.properties file. Example for android/app/build.gradle in Groovy format:

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')

if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
    signingConfigs {
        release {
            keyAlias keystoreProperties['keyAlias']
            keyPassword keystoreProperties['keyPassword']
            storeFile keystoreProperties['storeFile'] ? file(keystoreProperties['storeFile']) : null
            storePassword keystoreProperties['storePassword']
        }
    }

    buildTypes {
        release {
            signingConfig signingConfigs.release
        }
    }
}
```

5.3 Google Play Service Account

A service account is required for uploading to Google Play. The most important permission is not granted in the Google Cloud Console, but in the Google Play Console.

Element	Description / Example
Google Cloud Console	Under IAM & Admin → Service Accounts → create a service account and download the JSON key. If a role is required, Viewer is usually sufficient for this setup. →→
Google Play Console	Invite the service account under Users and permissions.
Recommended role	Release Manager for the specific app.
Required permissions	View app, manage testing tracks, create/edit/publish releases, manage production releases.
Not required	Financial data, orders, user management, admin rights.

5.4 GitLab Google Play Integration

In GitLab, go to Settings → Integrations → Google Play and enter the package name and the service account JSON key. GitLab then typically provides variables such as SUPPLY_PACKAGE_NAME and SUPPLY_JSON_KEY_DATA.

Note: If “Test settings” fails with Connection failed, check the following: correct package name, app exists in the Play Console, service account was invited in the Play Console, Google Play Android Developer API is enabled in the Google Cloud Console, permissions are set correctly.

6. Prepare iOS

6.1 App Store Connect API Key

For CI/CD, you should not use an Apple ID login with 2FA, but an App Store Connect API key.

Element	Description / Example
Path	App Store Connect → Users and Access → Integrations → App Store Connect API
Issuer ID	Displayed in App Store Connect.
Key ID	Displayed for the API key.
Private Key	.p8 file, can only be downloaded once.
Role	App Manager is usually more suitable for uploads and release processes than Developer. Administrator is usually too much.

Enter it in GitLab under Settings → Integrations → Apple App Store Connect. If this integration is used, no custom variables need to be created.

6.2 Separate match Repository

Fastlane match should use its own private repository for certificates and provisioning profiles. Do not use the app repository.

```
App code:
git@gitlab.com:your-group/your-flutter-app.git

iOS signing:
git@gitlab.com:your-group/ios-certificates.git
```

Warning: During the first push, the default branch in the certificate repository must not be protected in a way that prevents your user from writing to it. Otherwise, half-finished states can occur: the Apple portal has been changed, but the match repository does not contain the changes.

6.3 Matchfile

fastlane/Matchfile

```
git_url(ENV["MATCH_GIT_URL"])
storage_mode("git")
type("appstore")
```

Note: For TestFlight and the App Store, always use appstore, not development. Development is for local debug and device builds.

6.4 Initialize match

```
bundle exec fastlane match init
bundle exec fastlane match appstore --app_identifier com.yourcompany.yourapp --team_id TEAMID
```

You choose MATCH_PASSWORD yourself. It is not your Apple password, not your GitLab password, and not your Mac password. It only encrypts the match repository.

7. Fastfile

The following Fastfile shows a complete basic setup for Android Internal, Android Production, iOS TestFlight, and iOS Production. Adapt the app identifier, team ID, and release behavior.

```
default_platform(:ios)

# Build number: e.g., CI_PIPELINE_IID + offset
def build_number
  offset = (ENV["BUILD_NUMBER_OFFSET"] || "0").to_i
  pipeline_number = (ENV["CI_PIPELINE_IID"] || "1").to_i
  (offset + pipeline_number).to_s
end

# Visible app version: e.g., 0.0.60, 0.0.61, ...
def build_name
  patch_offset = (ENV["VERSION_PATCH_OFFSET"] || "0").to_i
  pipeline_number = (ENV["CI_PIPELINE_IID"] || "1").to_i
  "0.0.#{patch_offset + pipeline_number}"
end

def android_aab_path
  File.expand_path("../build/app/outputs/bundle/release/app-release.aab", __dir__)
end

def ios_ipa_path
  ipa = Dir[File.join(Dir.pwd, "build/ios/ipa/*.ipa")].first
  UI.user_error!("No IPA found in build/ios/ipa") unless ipa
  ipa
end

platform :android do
  desc "Build and upload Android app to Google Play Internal Track"
  lane :internal do
    sh("flutter pub get")
    sh("flutter build appbundle --release --build-name=#{build_name} --build-number=#{build_number}")

    upload_to_play_store(
      package_name: ENV["SUPPLY_PACKAGE_NAME"],
      json_key_data: ENV["SUPPLY_JSON_KEY_DATA"],
      aab: android_aab_path,
      track: "internal",
      release_status: "completed"
    )
  end

  desc "Build and upload Android app to Google Play Production"
  lane :production do
    sh("flutter pub get")
    sh("flutter build appbundle --release --build-name=#{build_name} --build-number=#{build_number}")

    upload_to_play_store(
      package_name: ENV["SUPPLY_PACKAGE_NAME"],
      json_key_data: ENV["SUPPLY_JSON_KEY_DATA"],
      aab: android_aab_path,
      track: "production",
      release_status: "completed"
    )
  end
end
```

```
platform :ios do
  private_lane :app_store_api_key_from_gitlab do
    app_store_connect_api_key(
      key_id: ENV["APP_STORE_CONNECT_API_KEY_KEY_ID"],
      issuer_id: ENV["APP_STORE_CONNECT_API_KEY_ISSUER_ID"],
      key_content: ENV["APP_STORE_CONNECT_API_KEY_KEY"],
      is_key_content_base64: true,
      in_house: false
    )
  end

  desc "Build and upload iOS app to TestFlight"
  lane :testflight do
    setup_ci(
      keychain_name: "fastlane_tmp_keychain",
      keychain_password: ENV["MATCH_KEYCHAIN_PASSWORD"]
    )

    api_key = app_store_api_key_from_gitlab

    match(
      type: "appstore",
      readonly: is_ci,
      app_identifier: ENV["IOS_APP_IDENTIFIER"],
      team_id: ENV["IOS_TEAM_ID"]
    )

    sh("flutter pub get")
    sh("flutter build ipa --release --build-name=#{build_name} --build-number=#{build_number} --export-
method=app-store")
    sh("find build -name '*.ipa'")

    upload_to_testflight(
      api_key: api_key,
      ipa: ios_ipa_path
    )
  end

  desc "Build and submit iOS app to App Store Review"
  lane :production do
    setup_ci(
      keychain_name: "fastlane_tmp_keychain",
      keychain_password: ENV["MATCH_KEYCHAIN_PASSWORD"]
    )

    api_key = app_store_api_key_from_gitlab

    match(
      type: "appstore",
      readonly: is_ci,
      app_identifier: ENV["IOS_APP_IDENTIFIER"],
      team_id: ENV["IOS_TEAM_ID"]
    )

    sh("flutter pub get")
    sh("flutter build ipa --release --build-name=#{build_name} --build-number=#{build_number} --export-
method=app-store")

    upload_to_app_store(
      api_key: api_key,
```

```

ipa: ios_ipa_path,
submit_for_review: true,
automatic_release: true,
skip_metadata: true,
skip_screenshots: true,
force: true
)
end
end

```

8. GitLab CI/CD Variables

Variable	Example	Note
ANDROID_KEYSTORE_BASE64	...	Base64 content of upload-keystore.jks
ANDROID_KEYSTORE_PASSWORD	...	Store password
ANDROID_KEY_ALIAS	upload	Alias of the upload key
ANDROID_KEY_PASSWORD	...	Key password
ENV_FILE_CONTENT	API_URL=...	Content of the .env file if .env is used as an asset
MATCH_GIT_URL	git@gitlab.com:group/ios-certificates.git	Separate private match repository
MATCH_PASSWORD	...	Encryption password for match
MATCH_KEYCHAIN_PASSWORD	...	Password for the temporary fastlane keychain on the runner
IOS_APP_IDENTIFIER	com.yourcompany.yourapp	iOS Bundle ID
IOS_TEAM_ID	ABCDE12345	Apple Team ID
BUILD_NUMBER_OFFSET	50	Offset for build number
VERSION_PATCH_OFFSET	50	Offset for visible patch version

Note: If variables are marked as protected, main and the tags used must also be protected accordingly. For production tags, allow v* as protected tags, for example.

9. macOS Runner for iOS

iOS builds require Xcode and therefore a macOS runner. The runner should have the tag macos and be allowed to run on protected branches if main is protected.

Element	Description / Example
Executor	shell
Tag	macos
Required tools	Xcode, Flutter, Homebrew Ruby, Bundler, Git, access to the match repository
Important	The runner does not necessarily load your .zshrc. Therefore, set PATH explicitly in .gitlab-ci.yml.

```
export PATH="/opt/homebrew/opt/ruby/bin:/opt/homebrew/bin:$PATH"
export PATH="$HOME/Developer/flutter/bin:$PATH"
which ruby
ruby -v
which flutter
flutter --version
```

10. .gitlab-ci.yml

This example pipeline covers the complete workflow: tests for merge requests/main, beta releases on main, and manual production releases for Git tags.

```
stages:
  - test
  - beta
  - production

variables:
  HOMEBREW_NO_AUTO_UPDATE: "1"
  BUNDLE_PATH: "vendor/bundle"

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'
    - if: '$CI_COMMIT_TAG'

flutter_test:
  stage: test
  image: ghcr.io/cirruslabs/flutter:3.41.7
  script:
    - printf "%s" "$ENV_FILE_CONTENT" > .env
    - flutter pub get
    - flutter analyze --no-fatal-infos --no-fatal-warnings
    - flutter test --exclude-tags golden
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'

flutter_golden_test:
  stage: test
  image: ghcr.io/cirruslabs/flutter:3.41.7
  script:
    - printf "%s" "$ENV_FILE_CONTENT" > .env
    - flutter pub get
    - flutter test --tags golden
  artifacts:
    when: always
    paths:
      - test/golden/failures/
  allow_failure: true
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'

android_internal:
  stage: beta
```

```

image: ghcr.io/cirruslabs/flutter:3.41.7
before_script:
  - apt-get update && apt-get install -y ruby-full build-essential
  - gem install bundler
  - bundle install
  - echo "$ANDROID_KEYSTORE_BASE64" | base64 -d > android/app/upload-keystore.jks
  - |
    cat > android/key.properties <<EOF
    storePassword=$ANDROID_KEYSTORE_PASSWORD
    keyPassword=$ANDROID_KEY_PASSWORD
    keyAlias=$ANDROID_KEY_ALIAS
    storeFile=upload-keystore.jks
    EOF
script:
  - printf "%s" "$ENV_FILE_CONTENT" > .env
  - bundle exec fastlane android internal
rules:
  - if: '$CI_COMMIT_BRANCH == "main"'

ios_testflight:
  stage: beta
  tags:
    - macos
  before_script:
    - export PATH="/opt/homebrew/opt/ruby/bin:/opt/homebrew/bin:$PATH"
    - export PATH="$HOME/Developer/flutter/bin:$PATH"
    - export LANG=en_US.UTF-8
    - export LC_ALL=en_US.UTF-8
    - cd "$CI_PROJECT_DIR"
    - flutter pub get
    - gem install bundler -v 4.0.11
    - bundle install
    - cd ios && bundle exec pod install && cd ..
  script:
    - cd "$CI_PROJECT_DIR"
    - printf "%s" "$ENV_FILE_CONTENT" > .env
    - bundle exec fastlane ios testflight
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'

android_production:
  stage: production
  image: ghcr.io/cirruslabs/flutter:3.41.7
  before_script:
    - apt-get update && apt-get install -y ruby-full build-essential
    - gem install bundler
    - bundle install
    - echo "$ANDROID_KEYSTORE_BASE64" | base64 -d > android/app/upload-keystore.jks
    - |
      cat > android/key.properties <<EOF
      storePassword=$ANDROID_KEYSTORE_PASSWORD
      keyPassword=$ANDROID_KEY_PASSWORD
      keyAlias=$ANDROID_KEY_ALIAS
      storeFile=upload-keystore.jks
      EOF
  script:
    - printf "%s" "$ENV_FILE_CONTENT" > .env
    - bundle exec fastlane android production
  rules:
    - if: '$CI_COMMIT_TAG'
      when: manual

```

```
ios_production:
  stage: production
  tags:
    - macos
  before_script:
    - export PATH="/opt/homebrew/opt/ruby/bin:/opt/homebrew/bin:$PATH"
    - export PATH="$HOME/Developer/flutter/bin:$PATH"
    - export LANG=en_US.UTF-8
    - export LC_ALL=en_US.UTF-8
    - cd "$CI_PROJECT_DIR"
    - flutter pub get
    - gem install bundler -v 4.0.11
    - bundle install
    - cd ios && bundle exec pod install && cd ..
  script:
    - cd "$CI_PROJECT_DIR"
    - printf "%s" "$ENV_FILE_CONTENT" > .env
    - bundle exec fastlane ios production
  rules:
    - if: '$CI_COMMIT_TAG'
      when: manual
```

Warning: For stable CI/CD, you should pin the Flutter image, e.g. `ghcr.io/cirruslabs/flutter:3.41.7`, instead of permanently using `stable`. Otherwise, a Flutter update can suddenly break your pipeline.

11. Start the Production Release

Production jobs only appear for a Git tag. Because of `when: manual`, they must also be started deliberately.

```
git checkout main
git pull origin main
git tag v1.0.0
git push origin v1.0.0
```

Then in GitLab: CI/CD → Pipelines → open the pipeline for tag v1.0.0 → start `android_production` and `ios_production` with the play button.

12. Versioning: Build Number and Visible Version

`CI_PIPELINE_IID` is the project-internal pipeline number in GitLab. If the pipeline was newly set up, it might be at 11, for example, even though your app is already at build 61. That is why the Fastfile uses offsets.

Element	Description / Example
BUILD_NUMBER_OFFSET	Adds to <code>CI_PIPELINE_IID</code> and results in the build number or Android <code>versionCode</code> .
VERSION_PATCH_OFFSET	Adds to <code>CI_PIPELINE_IID</code> and results in the visible patch version, e.g., 0.0.61.
Example	<code>CI_PIPELINE_IID=11</code> and <code>offset=50</code> → 61

```
Pipeline 11 → Version 0.0.61, build 61
Pipeline 12 → Version 0.0.62, build 62
Pipeline 13 → Version 0.0.63, build 63
```

13. Troubleshooting

Could not locate Gemfile: The job is not running in the root directory or the Gemfile has not been committed. In the pipeline, output `pwd`, `ls -la`, and `find . -maxdepth 3 -name Gemfile`.

Generated.xcconfig must exist: `flutter pub get` must run before `cd ios && bundle exec pod install`.

CocoaPods version too old: Add `cocoapods` to the Gemfile and use `bundle exec pod install`.

No runners match tag macos: Register a macOS runner with the tag `macos` and allow it for protected branches.

git@gitlab.com: Permission denied (publickey): The Mac or runner does not have SSH access to the match repository. Test with `ssh -T git@gitlab.com`.

Could not create another Distribution certificate: Apple certificate limit reached. Import an existing `.p12/.cer/profile` or revoke an old unused distribution certificate.

Provisioning profile does not include signing certificate: Provisioning profile and distribution certificate do not match. Clean up the profiles for the app and run `match appstore --force` again.

No ipa or pkg file given: Usually a follow-up error: the IPA was not generated because of an export error. Check the previous lines. `find build -name *.ipa` helps with debugging.

fastlane_tmp_keychain is annoying on the Mac: Remove the temporary keychain from the search list or delete it. It may be created again during the next CI run.

Inconsistent JVM Target Compatibility: Android plugin conflict between Java and Kotlin. First, update the plugins. Pragmatic fix: `kotlin.jvm.target.validation.mode=warning` in `android/gradle.properties`.

android.app.Activity not found: Do not use a global `JavaCompile` override in `android/build.gradle`. This can damage the Android classpath of the plugin projects.

IconData cannot be extended because final: Old `lucide_icons` is incompatible with newer Flutter versions. Update/replace the package or pin the Flutter version in CI.

.env missing: Do not commit `.env`. Instead, store `ENV_FILE_CONTENT` as a GitLab variable and run `printf "%s" "$ENV_FILE_CONTENT" > .env` before every Flutter command.

14. Final Checklist for a New Flutter App

- ☐ App created in Google Play Console and App Store Connect.
- ☐ Android `applicationId` and iOS Bundle ID finalized.
- ☐ Google service account created and authorized in the Play Console.
- ☐ GitLab Google Play integration set up.
- ☐ App Store Connect API key created with appropriate permissions.
- ☐ GitLab Apple App Store Connect integration set up.
- ☐ Android upload keystore stored as a GitLab variable.
- ☐ Separate private `ios-certificates` repository created.
- ☐ `match appstore` successfully run locally and pushed to the match repository.
- ☐ Gemfile, Gemfile.lock, Fastfile, and Matchfile committed.

- ☐ macOS runner with the tag macos is running.
- ☐ .gitlab-ci.yml inserted and variables set.
- ☐ Merge into main tested: Android Internal and iOS TestFlight run successfully.
- ☐ Production tested via Git tag.

Note: Recommendation: For mobile apps, initially always start production releases manually via tag. This keeps the process automated, but controlled.