

GitLab CI/CD für Flutter Apps

Schritt-für-Schritt-Anleitung für automatische Test-, Beta- und Production-Releases

Hinweis: Diese Anleitung ist als editierbare Word-Vorlage gedacht. Platzhalter wie Bundle ID, Package Name, Team ID, Repository-URLs und Variablenwerte müssen vor der Veröffentlichung auf die jeweilige App angepasst werden.

Element	Beschreibung / Beispiel
Ziel	Flutter-App per GitLab CI/CD automatisch testen und in TestFlight sowie Google Play Internal Track veröffentlichen.
Production	Production-Releases werden über Git Tags ausgelöst und manuell in GitLab gestartet.
Tools	GitLab CI/CD, fastlane, App Store Connect API, Google Play Service Account, fastlane match.
Wichtig	iOS benötigt einen macOS Runner. Android kann auf einem normalen Docker Runner gebaut werden.

1. Ziel-Workflow

Die Pipeline soll bei Merge Requests und Pushes zuverlässig prüfen, ob die Flutter-App noch baut und getestet werden kann. Nach einem Merge in den main-Branch werden Beta-Releases automatisch veröffentlicht. Production-Releases werden bewusst erst bei einem Git Tag angeboten und anschließend manuell gestartet.

```
Merge Request:
  flutter analyze
  flutter test

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

Git Tag, z. B. v1.0.0:
  Android → Google Play Production
  iOS → App Store Review / Production
```

Hinweis: GitLab veröffentlicht die App nicht selbst. GitLab startet nur die Jobs. Die eigentliche Store-Logik übernimmt fastlane.

2. Voraussetzungen

- ☐ Flutter-App liegt in einem GitLab-Repository.
- ☐ Google Play Console App ist angelegt.
- ☐ App Store Connect App ist angelegt.
- ☐ Android Upload Keystore ist vorhanden.
- ☐ Google Play Service Account ist vorhanden und in der Play Console berechtigt.
- ☐ App Store Connect API Key ist vorhanden.
- ☐ Privates GitLab-Repository für fastlane match ist vorhanden.
- ☐ macOS Runner mit Xcode, Flutter, Ruby und GitLab Runner ist eingerichtet.
- ☐ Bundle ID / App Identifier und Android Application ID sind final gesetzt.

3. Grundstruktur im Flutter-Projekt

Für einen einheitlichen Workflow sollten fastlane und der Gemfile im Root des Flutter-Projekts liegen, nicht nur im ios- oder android-Ordner.

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

4. Gemfile und Bundler einrichten

Lege im Root des Flutter-Projekts einen Gemfile an. CocoaPods sollte über Bundler ausgeführt werden, damit die Pipeline nicht plötzlich eine zu alte oder kaputte globale CocoaPods-Version nutzt.

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"
```

Hinweis: In der Pipeline immer `bundle exec pod install` verwenden, nicht einfach `pod install`.

5. Android vorbereiten

5.1 Android Upload Keystore in GitLab hinterlegen

Der Android Release Build braucht einen Upload Keystore. Dieser wird nicht ins Repository committed, sondern als Base64-Variable in GitLab gespeichert. Folgender Befehl kopiert den Keystore-Wert in die Zwischenablage:

macOS

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

Linux

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

Element	Beschreibung / Beispiel
ANDROID_KEYSTORE_BASE64	Base64-Inhalt der upload-keystore.jks
ANDROID_KEYSTORE_PASSWORD	Store-Passwort
ANDROID_KEY_ALIAS	Alias des Upload Keys
ANDROID_KEY_PASSWORD	Key-Passwort

Die Pipeline erzeugt daraus zur Laufzeit `android/app/upload-keystore.jks` und `android/key.properties`.

```
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

Der Release-Build muss die key.properties-Datei lesen. Beispiel für android/app/build.gradle im 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

Für den Upload zu Google Play wird ein Service Account benötigt. Die wichtigste Berechtigung wird nicht in der Google Cloud Console vergeben, sondern in der Google Play Console.

Element	Beschreibung / Beispiel
Google Cloud Console	Unter IAM&Admin → Service Accounts → Service Account erstellen und JSON-Key herunterladen. Falls eine Rolle erforderlich ist, reicht für dieses Setup meist Viewer.
Google Play Console	Service Account unter Users and permissions einladen.
Empfohlene Rolle	Release Manager für die konkrete App.
Benötigte Rechte	App ansehen, Testing-Tracks verwalten, Releases erstellen/bearbeiten/veröffentlichen, Production-Releases verwalten.
Nicht nötig	Finanzdaten, Bestellungen, Nutzerverwaltung, Admin-Rechte.

5.4 GitLab Google Play Integration

In GitLab unter Settings → Integrations → Google Play den Package Name und den Service-Account-JSON-Key eintragen. Danach stellt GitLab typischerweise Variablen wie SUPPLY_PACKAGE_NAME und SUPPLY_JSON_KEY_DATA bereit.

Hinweis: Wenn „Test settings“ mit Connection failed abbricht, prüfe: richtiger Package Name, App existiert in der Play Console, Service Account wurde in der Play Console eingeladen, in Google Cloud Console ist Google Play Android Developer API aktiviert, Rechte sind korrekt gesetzt.

6. iOS vorbereiten

6.1 App Store Connect API Key

Für CI/CD sollte kein Apple-ID-Login mit 2FA verwendet werden, sondern ein App Store Connect API Key.

Element	Beschreibung / Beispiel
Pfad	App Store Connect → Users and Access → Integrations → App Store Connect API
Issuer ID	Wird in App Store Connect angezeigt.
Key ID	Wird beim API Key angezeigt.
Private Key	.p8-Datei, kann nur einmal heruntergeladen werden.
Rolle	App Manager ist für Uploads und Release-Prozesse in der Regel sinnvoller als Developer. Administrator ist meist zu viel.

In GitLab unter Settings → Integrations → Apple App Store Connect eintragen. Wenn diese Integration genutzt wird, müssen keine eigenen Variablen angelegt werden.

6.2 Separates match-Repository

Fastlane match sollte ein eigenes privates Repository für Zertifikate und Provisioning Profiles nutzen. Nicht das App-Repository verwenden.

```
App-Code:  
git@gitlab.com:deine-gruppe/deine-flutter-app.git  
  
iOS Signing:  
git@gitlab.com:deine-gruppe/ios-certificates.git
```

Warnung: Beim ersten Push darf die Default Branch im Zertifikats-Repo nicht so geschützt sein, dass dein User nicht schreiben darf. Sonst entstehen halb fertige Zustände: Apple Portal wurde geändert, aber das match-Repo enthält die Änderungen nicht.

6.3 Matchfile

fastlane/Matchfile

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

Hinweis: Für TestFlight und App Store immer appstore verwenden, nicht development. Development ist für lokale Debug- und Device-Builds.

6.4 match initialisieren

```
bundle exec fastlane match init  
bundle exec fastlane match appstore --app_identifizier com.deinefirma.deineapp --team_id TEAMID
```

Das MATCH_PASSWORD denkst du dir selbst aus. Es ist nicht dein Apple-Passwort, nicht dein GitLab-Passwort und nicht dein Mac-Passwort. Es verschlüsselt nur das match-Repository.

7. Fastfile

Das folgende Fastfile zeigt einen vollständigen Grundaufbau für Android Internal, Android Production, iOS TestFlight und iOS Production. Passe App Identifier, Team ID und Release-Verhalten an.

```
default_platform(:ios)

# Build Number: z. B. 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

# Sichtbare App-Version: z. B. 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	Beispiel	Hinweis
ANDROID_KEYSTORE_BASE64	...	Base64-Inhalt der upload-keystore.jks
ANDROID_KEYSTORE_PASSWORD	...	Store-Passwort
ANDROID_KEY_ALIAS	upload	Alias des Upload Keys
ANDROID_KEY_PASSWORD	...	Key-Passwort
ENV_FILE_CONTENT	API_URL=...	Inhalt der .env-Datei, falls .env als Asset verwendet wird
MATCH_GIT_URL	git@gitlab.com:gruppe/ios-certificates.git	Separates privates match-Repo
MATCH_PASSWORD	...	Verschlüsselungspasswort für match
MATCH_KEYCHAIN_PASSWORD	...	Passwort für temporäre fastlane Keychain auf dem Runner
IOS_APP_IDENTIFIER	com.deinefirma.deineapp	iOS Bundle ID
IOS_TEAM_ID	ABCDE12345	Apple Team ID
BUILD_NUMBER_OFFSET	50	Offset für Build Number
VERSION_PATCH_OFFSET	50	Offset für sichtbare Patch-Version

Hinweis: Wenn Variablen als protected markiert sind, müssen auch main und die verwendeten Tags passend geschützt sein. Für Production Tags z. B. v* als Protected Tags erlauben.

9. macOS Runner für iOS

iOS Builds brauchen Xcode und damit einen macOS Runner. Der Runner sollte den Tag macos haben und auf protected branches laufen dürfen, wenn main geschützt ist.

Element	Beschreibung / Beispiel
Executor	shell
Tag	macos

Benötigte Tools	Xcode, Flutter, Homebrew Ruby, Bundler, Git, Zugriff auf das match-Repo
Wichtig	Der Runner lädt nicht zwingend deine .zshrc. Deshalb PATH in der .gitlab-ci.yml explizit setzen.

```
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

Diese Beispiel-Pipeline bildet den vollständigen Workflow ab: Tests bei Merge Request/main, Beta-Releases bei main und manuelle Production-Releases bei 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
```

Warnung: Für stabile CI/CD solltest du das Flutter-Image pinnen, z. B. ghcr.io/cirruslabs/flutter:3.41.7, statt dauerhaft stable zu verwenden. Sonst kann ein Flutter-Update deine Pipeline plötzlich brechen.

11. Production Release starten

Production-Jobs erscheinen nur bei einem Git Tag. Durch when: manual müssen sie zusätzlich bewusst gestartet werden.

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

Danach in GitLab: CI/CD → Pipelines → Pipeline für Tag v1.0.0 öffnen → android_production und ios_production mit dem Play-Button starten.

12. Versionierung: Build Number und sichtbare Version

CI_PIPELINE_IID ist die projektinterne Pipeline-Nummer in GitLab. Wenn die Pipeline neu eingerichtet wurde, kann sie z. B. bei 11 stehen, obwohl deine App bereits bei Build 61 ist. Deshalb nutzt das Fastfile Offsets.

Element	Beschreibung / Beispiel
BUILD_NUMBER_OFFSET	Addiert zur CI_PIPELINE_IID und ergibt die Build Number bzw. Android versionCode.

VERSION_PATCH_OFFSET	Addiert zur CI_PIPELINE_ID und ergibt die sichtbare Patch-Version, z. B. 0.0.61.
Beispiel	CI_PIPELINE_ID=11 und Offset=50 → 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: Der Job läuft nicht im Root oder der Gemfile ist nicht committed. In der Pipeline pwd, ls -la und find . -maxdepth 3 -name Gemfile ausgeben.

Generated.xcconfig must exist: Vor cd ios && bundle exec pod install muss flutter pub get laufen.

CocoaPods version zu alt: cocoapods in den Gemfile aufnehmen und bundle exec pod install verwenden.

No runners match tag macos: macOS Runner mit Tag macos registrieren und für protected branches erlauben.

git@gitlab.com: Permission denied (publickey): Der Mac oder Runner hat keinen SSH-Zugriff auf das match-Repo. ssh -T git@gitlab.com testen.

Could not create another Distribution certificate: Apple-Zertifikatslimit erreicht. Vorhandenes .p12/.cer/Profile importieren oder altes ungenutztes Distribution Certificate revoke.

Provisioning profile does not include signing certificate: Provisioning Profile und Distribution Certificate passen nicht zusammen. Profiles für die App bereinigen und match appstore --force erneut ausführen.

No ipa or pkg file given: Meist Folgefehler: IPA wurde wegen eines Exportfehlers nicht erzeugt. Vorherige Zeilen prüfen. find build -name *.ipa hilft beim Debuggen.

fastlane_tmp_keychain nervt auf dem Mac: Temporäre Keychain aus der Suchliste entfernen oder löschen. Sie kann beim nächsten CI-Run erneut angelegt werden.

Inconsistent JVM Target Compatibility: Android-Plugin-Konflikt zwischen Java und Kotlin. Primär Plugins aktualisieren. Pragmatiker-Fix: kotlin.jvm.target.validation.mode=warning in android/gradle.properties.

android.app.Activity nicht gefunden: Keinen globalen JavaCompile-Override in android/build.gradle verwenden. Das kann den Android-Classpath der Plugin-Projekte beschädigen.

IconData cannot be extended because final: Altes lucide_icons ist mit neuerem Flutter inkompatibel. Package aktualisieren/ersetzen oder Flutter-Version in CI pinnen.

.env missing: .env nicht committen, sondern ENV_FILE_CONTENT als GitLab Variable speichern und vor jedem Flutter-Befehl printf "%s" "\$ENV_FILE_CONTENT" > .env ausführen.

14. Abschluss-Checkliste für eine neue Flutter-App

- ☐ App in Google Play Console und App Store Connect angelegt.
- ☐ Android applicationId und iOS Bundle ID final gesetzt.
- ☐ Google Service Account erstellt und in Play Console berechtigt.
- ☐ GitLab Google Play Integration eingerichtet.

- ☐ App Store Connect API Key mit passenden Rechten erstellt.
- ☐ GitLab Apple App Store Connect Integration eingerichtet.
- ☐ Android Upload Keystore als GitLab Variable gespeichert.
- ☐ Separates privates ios-certificates-Repo erstellt.
- ☐ match appstore lokal erfolgreich ausgeführt und ins match-Repo gepusht.
- ☐ Gemfile, Gemfile.lock, Fastfile und Matchfile committed.
- ☐ macOS Runner mit Tag macos läuft.
- ☐ .gitlab-ci.yml eingefügt und Variablen gesetzt.
- ☐ Merge in main getestet: Android Internal und iOS TestFlight laufen durch.
- ☐ Production über Git Tag getestet.

Hinweis: Empfehlung: Production-Releases bei mobilen Apps zunächst immer manuell per Tag starten. So bleibt der Prozess automatisiert, aber kontrolliert.