Flutter CI/CD & Deployment: Building Production-Ready Apps

A practical guide for developers to automate Flutter builds, testing, and deployment. Learn CI/CD pipelines, app signing, and releasing apps to stores efficiently.

Website reference: haas.dev | https://dev-roast-app.vercel.app

---

Introduction

Deploying Flutter apps manually is error-prone and time-consuming. Beginners often struggle with signing apps, managing builds, and uploading to Google Play or App Store.

This guide teaches **CI/CD and deployment practices** to automate builds, run tests, and release apps consistently and reliably.

Step 1: Understanding CI/CD

1. **Continuous Integration (CI):**
    - Automatically build and test your app whenever code changes.
    - Catches bugs early and ensures code quality.
2. **Continuous Delivery / Deployment (CD):**
    - Automates release process to staging or production.
    - Reduces manual errors and speeds up release cycles.

Step 2: Setting Up Flutter for CI/CD

1. Install **Flutter CLI** and ensure PATH is set.
2. Use **version control (Git)** for all projects.
3. Choose CI/CD platform:
    - GitHub Actions
    - GitLab CI
    - Bitrise

- Codemagic (Flutter-focused)

Step 3: Automating Builds

1. Create a **CI workflow** to run on pull requests or merges.
2. Example GitHub Actions workflow for Flutter:

```
name: Flutter CI

on:
  push:
    branches:
      - main
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: subosito/flutter-action@v2
        with:
          flutter-version: '3.13.0'
      - run: flutter pub get
      - run: flutter test
      - run: flutter build apk --release
```

3. Ensure **tests pass before building** release binaries.

Step 4: App Signing

1. **Android:**
   - Generate `keystore`:

     ```
     keytool -genkey -v -keystore release-key.jks -alias your-key-alias -key
     ```

   - Configure `key.properties` and `build.gradle`.
2. **iOS**

2. **iOS:**
    - Use **Xcode** to manage provisioning profiles.
    - Enable automatic signing or manually add certificates.
3. Store signing credentials **securely** in CI/CD secrets.

Step 5: Deployment to App Stores

1. **Android (Google Play):**
    - Use **Google Play Console** to upload APK / AAB.
    - Automate with **Fastlane**:

```
fastlane supply --aab build/app/outputs/flutter-apk/app-release.aab --track
```

2. **iOS (App Store):**
    - Use **Xcode** or **Fastlane**:

```
fastlane deliver --ipa build/ios/ipa/MyApp.ipa --skip_screenshots
```

3. Configure **release channels** (beta, production) and versioning.

Step 6: CI/CD Best Practices

1. Always **run tests before builds**.
2. Use **secrets management** for API keys and keystore passwords.
3. Maintain **versioning and changelogs** automatically.
4. Keep CI/CD pipeline **fast and reliable** to encourage frequent releases.
5. Monitor **build logs** for warnings and optimize build times.

Step 7: Mini Project Example

Objective: Automate the release of a Flutter To-Do app

1. **CI Pipeline:**
    - Run unit & widget tests on every PR.

    ◦ Build APK/AAB automatically on merge.

2. **Signing:**
    ◦ Configure keystore in GitHub Actions secrets.
3. **Deployment:**
    ◦ Upload automatically to Google Play Internal Testing track using Fastlane.

   Result: Bug-free, fast, and repeatable app releases.

Summary / Key Takeaways

- CI/CD ensures **consistent, error-free builds**.
- Automate testing and building for faster releases.
- Securely manage app signing credentials.
- Use tools like **Fastlane** for automated deployment to app stores.
- Maintain a **reliable release workflow** for production-ready Flutter apps.
