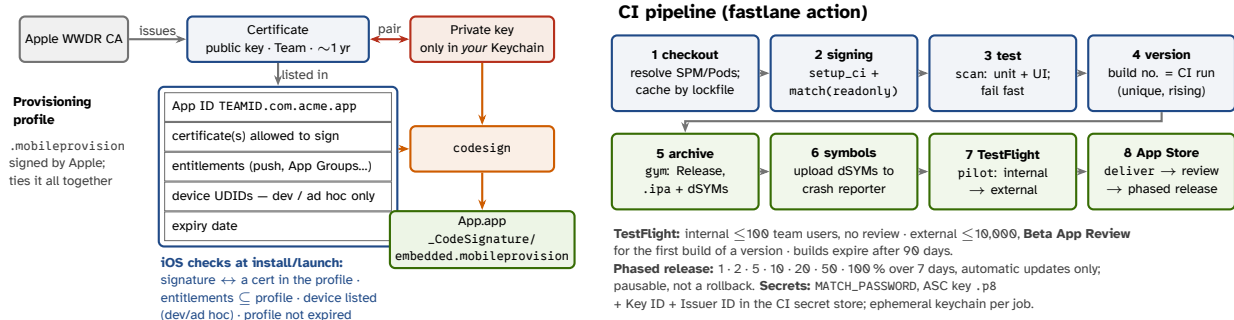


In one line: A build runs on a device only if it is **signed with a certificate's private key** and carries a **provisioning profile** that lists that certificate, the **App ID**, the **entitlements** and (dev/ad hoc) the **device**. CI breaks because the runner has none of that — **fastlane match + an App Store Connect API key** make signing and upload reproducible and non-interactive.



Profile types

Profile	Certificate	Devices	Use
Development	Apple Development	listed UDIDs	run + debug (get-task-allow)
Ad Hoc	Apple Distribution	listed UDIDs	QA outside TestFlight
App Store	Apple Distribution	none	TestFlight + App Store
Enterprise	In-House (Enterprise Program)	none	internal staff only, never public

How it works

- The **profile** picks the channel, not the certificate: one Distribution cert signs both Ad Hoc and App Store builds.
- Capability flow:** enable it on the App ID → regenerate the profile → add it to .entitlements. All three must agree.
- Automatic signing** = Xcode creates certs/profiles with a logged-in Apple ID: fine locally, flaky on CI. **Manual** = named profile + identity: reproducible.
- match** keeps certs + profiles *encrypted* (git, S3, GCS) and installs the same ones on every laptop and runner; **readonly: true** on CI so it never creates or revokes. Types: development, adhoc, appstore, enterprise.
- ASC API key** = Issuer ID + Key ID + .p8 (downloadable once); signs a short-lived ES256 JWT per request. No password, no 2FA, role-scoped, revocable. Apple ID auth needs interactive 2FA sessions that expire.
- Version vs build:** CFBundleShortVersionString (2.4.0, user-facing) vs CFBundleVersion (1387) — unique and increasing per upload of a version.
- Build once, promote the artifact:** the .ipa you tested in TestFlight is the one you submit; never rebuild for release.
- Caching:** key SPM/Pods caches on Package.resolved / Podfile.lock + Xcode version — keyed on branch only, they go stale.

Remember

“Key + Cert + Profile = permission to run; match shares them, the API key uploads them.”

Example — Fastfile (Ruby)

```
lane :beta do
  setup_ci # temp unlocked keychain on CI
  key = app_store_connect_api_key(key_id: ENV["ASC_KEY_ID"],
    issuer_id: ENV["ASC_ISSUER_ID"], key_content: ENV["ASC_P8"])
  match(type: "appstore", readonly: true, api_key: key)
  run_tests(scheme: "App") # scan
  increment_build_number(build_number: ENV["CI_RUN_NUMBER"])
  build_app(scheme: "App", export_method: "app-store") # gym
  upload_symbols_to_crashlytics # dSYMs
  upload_to_testflight(api_key: key) # pilot
end
```

Interview traps

- ! **“Works on my Mac, fails on CI”:** your Keychain has the private key; the runner doesn’t. A .cer without its private key cannot sign.
- ! **errSecInternalComponent / “User interaction is not allowed”** = locked keychain or key ACL → setup_ci or security set-key-partition-list.
- ! Automatic signing on CI keeps minting distribution certs until the account limit.
- ! Capability works in Debug, missing in the App Store build → the distribution profile was not regenerated after enabling it on the App ID.
- ! Duplicate CFBundleVersion → App Store Connect rejects the upload.
- ! Never commit .p12/.p8 in plain text or echo a secret in a job — CI logs are kept.
- ! Phased release can be paused, not rolled back; a bad build needs a new build.

Likely questions

- What’s in a provisioning profile? — App ID, certs, entitlements, UDIDs (dev/ad hoc), expiry.
- Why manual signing on CI? — deterministic; no Apple ID session, no new certs.
- Internal vs external TestFlight? — team only, no review vs up to 10k, Beta App Review.
- Why an ASC API key? — non-interactive, scoped, revocable; no 2FA.
- Stages of your pipeline? — deps → sign → test → bump → archive → dSYMs → TestFlight.