The flutter/plugin and flutter/packages repository are more prone to out-of-band failures—persistent failures that do not originate with a PR within the repository—than flutter/engine and flutter/flutter. These are more difficult to debug than most failures since the source may not be obvious, and can be difficult to resolve since there's not necessarily anything that can be reverted to fix them. This page collects information on sources of these failures to help debug and resolve them.

This page doesn't cover flakes. While these failures are easily confused with flakes at first, they can be distinguished by being persistent across retries, as well as showing up in in-progress PRs.

# Failure Sources

## Infrastructure

This category covers anything that is a function of the CI infrastructure itself: services, machines, etc. It occurs across all repositories, but instances may be specific to one repository.

### Cirrus

Cirrus tasks are run via the Cirrus service, but with the Linux tasks running on GCP VMs. The images are controlled by a combination of `cirrus.yml` and `.ci/Dockerfile`.

Potential failures include:

- Linux VM changes: The Dockerfile is not hermitic (#84712) and can be recreated unexpectedly (e.g., #85041). If this happens, Linux tasks can have unexpected failures due to version changes of dependencies (e.g., Java, Chrome).
- Authentication issues between Cirrus and GCP (e.g., #84990).

#### Distinguishing features

- Likely to have the same failures on `stable` and `master` runs.
- May to affect many plugins, but only on one host platform.

#### Investigation & resolution tips

- When investigating failures with Cirrus, both Cirrus support and the Flutter infrastructure team may need to be involved to isolate the cause and to resolve the failure.
- Rollbacks are generally not possible

### LUCI

LUCI tasks are run on Flutter-infrastructure-managed VMs, using an out-of-repo recipe. Since LUCI images are changed by infrastructure team rollouts, and recipes are out-of-repo, almost any LUCI change is out of band. Potential failure sources include:

- Images changes.
- Recipe changes (very uncommon now that the recipe is generic).

#### Distinguishing features

- Only the Windows desktop tests are broken.
- Image changes generally result in builds not proceeding at all due to missing dependencies.
- Recipe changes generally cause setup failures or failure to start the test.

#### Distinguishing features

- Usually easy to identify since LUCI is currently only used for Windows desktop tests, and the failures are generally before the tests even run.

#### Investigation & resolution tips

- File an infrastructure ticket.
- Check the recipe file for recent changes.

### Firebase Test Lab (Also known as: "FTL")

Integration tests on Android are run in real devices through the Firebase Test Lab infrastructure by the `firebase-test-lab` plugin tool. Source. From time to time, the Firebase Test Lab will change what devices are available for testing, and our tests will start timing out.

#### Distinguishing features

- `firebase_test_lab` task starts timing out. The output is normally just: `Timed out!` (Example.)
  - These timeouts will start as "flake" tests, and get progressively worse, until no amount of "retries" helps them pass (as devices are phased out / less available).
  - `firebase_test_lab` timing out is almost always related to this. Either because of devices becoming unavailable, or by a temporary lack of resource availability.

#### Investigation & resolution tips

- Check the Deprecation List in the Firebase documentation, and see if it affects any of the devices used by the script.
- Pick another device that is more available and update the script. See a sample PR.
  - It is likely that `flutter/engine` and `flutter/flutter` have had the same problem. Use the same devices picked by them.

## Flutter

The plugins and packages repository are moving toward a pin+roller system for Flutter, but the transition is still in process. Tests that haven't been migrated yet are run against the latest version of the `master` and/or `stable` channels, so can be unexpectedly broken by updates there (#30446). Potential failure sources include:

- Breaking changes to framework features used by plugins.
- Engine changes.
- Changes to the `flutter` tool.
- Dart tool changes, such as new analyzer behaviors or formatter changes.

*The current status of the migration is:*

- *flutter/plugins: Pinned, auto-rolled version of* `master` *, latest version of* `stable` *.*
- *flutter/packages: Latest version of* `stable` *and* `master`

#### Distinguishing features

- Will almost always affect only one of `master` or `stable` (usually `master` ).
  - If `stable` , the timing would correspond to a `stable` channel release.

#### Investigation & resolution tips

- To confirm or eliminate a Flutter change as the source of an error, update `cirrus.yml` to check out a specific (older) revision, either in general or for a specific task (example).
  - This can be landed as a temporary mitigation, but it is *very important* that it be used that way only for very short periods of time to avoid hiding new issues.

- Regressions due to changes on master can be either rolled back or fixed forward, depending on the ease of rollback.

## Publishing

There are some interdependencies between packages in the plugin repository (notably federated plugins, but a few other cases as well, such as examples of one plugin that depend on another plugin). Inter-package dependencies are only tested with published versions (#52115), so publishing a package can potentially break other packages.

#### Distinguishing features

- Timing will correspond to a package being published. (With autorelease, this will be shortly after the version-updating PR lands, unless something goes wrong.)
- May have a clear reference to the published package in the failure.
  - This is not guaranteed; e.g., Android plugin Gradle changes can have transitive effects on example apps that depend on those packages.

#### Investigation & resolution tips

- Pinning a specific dependency version can confirm or eliminate this as a source of errors.
  - This should generally **not** be used as a mitigation; this category of error is often a failure that will affect clients of the package as well.

## External build dependencies

On platforms where the plugin system includes a dependency management system, there are build-time dependencies on external servers (e.g., Maven for Android, Cocoapods for iOS and macOS). Potential failure sources include:

- Temporary server outages.
- Removal of a package.

#### Distinguishing features

- The logs will be very clear that fetching a dependency failed.
  - The only challenge in identifying them quickly is that they look the same as transient server or network issue flakes, which are much more common.

#### Investigation & resolution tips

- Check for reports of outages on the relevant servers.
- Check whether the entire server is failing, or only fetching a specific package is failing.
- Server-level outages are usually short-lived and just have to be waited out; i.e., they are persistent for a matter of hours before the server issue is resolved.
- Package issues may require repository changes. E.g., for Maven, switching to another server that has the package, or to another version of the package that is still available.

## `pub`

The `publish` command periodically enables new checks. Rarely, such a check will be enabled after presubmits for a PR have run, but before it's submitted.

#### Distinguishing features

- Either the `publish` validation step or the `release` step will fail with a clear error message from the `publish` command.

**Investigation & resolution tips**

- These are usually straightforward to resolve by fixing the newly-flagged issue.