Flutter's Build Infrastructure

This directory exists to support building Flutter on our build infrastructure.

Flutter build results are available at:

- https://flutter-dashboard.appspot.com/#/build
  - Aggregate dashboard of the separate CI systems used by Flutter.
- https://cirrus-ci.com/github/flutter/flutter/master
  - Testing is done on PRs and submitted changes on GitHub.
- https://ci.chromium.org/p/flutter/
  - Additional testing and processing are done after changes are submitted.

Flutter infra requires special permissions to retrigger builds on the build dashboard. File an infra ticket to request permission.

The Cirrus-based bots run the `test.dart` script for each PR and submission. This does testing for the tools, for the framework, and (for submitted changes only) rebuilds and updates the master branch API docs staging site. For tagged dev and beta builds, it also builds and deploys the gallery app to the app stores. It is configured by the .cirrus.yml.

We also have post-commit testing with actual devices, in what we call our devicelab.

LUCI (Layered Universal Continuous Integration)

A set of recipes are run on Windows, Linux, and Mac machines. The configuration for how many machines and what kind are managed internally by Google. File an infra ticket if you suspect changes are needed there. Both of these technologies are highly specific to the LUCI project, which is the successor to Chromium's infra and the foundation to Flutter's infrastructure.

Prerequisites

To work on this infrastructure you will need:

- depot_tools
- Python package installer: `sudo apt-get install python-pip`
- Python coverage package (only needed for `training_simulation`): `sudo pip install coverage`

To run `prepare_package.dart` locally:

- Make sure the `depot_tools` is in your `PATH`. If you're on Windows, you also need an environment variable called `DEPOT_TOOLS` with the path to `depot_tools` as value.
- Run `gsutil.py config` (or `python3 %DEPOT_TOOLS%\gsutil.py` on Windows) to authenticate with your auth token.
- Create a local temp directory. `cd` into it.
- Run `dart [path to your normal Flutter repo]/dev/bots/prepare_package.dart --temp_dir=. --revision=[revision to package] --branch=[branch to deploy to] --publish`.
- If you're running into `gsutil` permission issues, check with @Hixie to make sure you have the right push permissions.

Getting the code

The following will get way more than just recipe code, but it *will* get the recipe code:

```
mkdir chrome_infra
cd chrome_infra
fetch infra
```

More detailed instructions can be found here.

Most of the functionality for recipes comes from `recipe_modules`, which are unfortunately spread to many separate repositories. After checking out the code search for files named `api.py` or `example.py` under `infra/build`.

Editing a recipe

Flutter has several recipes depending on the test. The recipes share common actions through `recipe_modules`. Searching the builder config in infra will indicate the recipe used for a test.

Recipes are just Python with some limitations on what can be imported. They are documented by the luci/recipes-py GitHub project.

The typical cycle for editing a recipe is:

1. Check out the recipes project using `git clone https://flutter.googlesource.com/recipes`.
2. Make your edits (probably to files in `//recipes/recipes`).
3. Update the tests. Run `recipes.py test train` to update the existing expected output to match the new output. Verify completely new test cases by altering the `GenTests` method of the recipe. The recipe is required to have 100% test coverage.
4. Run `led get-builder 'luci.flutter.prod:BUILDER_NAME' | led edit -p 'revision="GIT_HASH"' | led edit-recipe-bundle | led launch`, where `BUILDER_NAME` is the builder name (e.g. `Linux Engine`), and `GIT_HASH` is the hash to build (which is important for the engine but not for the framework).
    - If `led` fails, ensure that your `depot_tools` checkout is up to date.
5. To submit a CL, you need a local branch first (`git checkout -b [some branch name]`).
6. Upload the patch (`git commit`, `git cl upload`), and open the outputted URL to the CL.
7. Use "Find owners" to get reviewers for the CL

The infra config repository

The infra repository contains configuration files for the dashboard, builder groups, scheduling, and individual builders. Edits to this may require changes other internal Google repositories - e.g., to change the operating system or number of machines. If you want to do that, file an infra ticket with your proposed changes.

Each configuration file in that repository has a link in the top comments to a schema that describes available properties.

Android Tools

The Android SDK and NDK used by Flutter's Chrome infra bots are stored in Google Cloud. During the build, a bot runs the `download_android_tools.py` script that downloads the required version of the Android SDK into `dev/bots/android_tools`.

To check which components are currently installed, download the current SDK stored in Google Cloud using the `download_android_tools.py` script, then `dev/bots/android_tools/sdk/tools/bin/sdkmanager --list`. If you find that some components need to be updated or installed, follow the steps below:

How to update Android SDK on Google Cloud Storage

1. Run Android SDK Manager and update packages `$ dev/bots/android_tools/sdk/tools/android update sdk` Use `android.bat` on Windows.
2. Use the UI to choose the packages you want to install and/or update.
3. Run `dev/bots/android_tools/sdk/tools/bin/sdkmanager --update`. On Windows, run `sdkmanager.bat` instead. If the process fails with an error saying that it is unable to move files (Windows makes files and directories read-only when another process is holding them open), make a copy of the `dev/bots/android_tools/sdk/tools` directory, run the `sdkmanager.bat` from the copy, and use the `--sdk_root` option pointing at `dev/bots/android_tools/sdk`.
4. Run `dev/bots/android_tools/sdk/tools/bin/sdkmanager --licenses` and accept the licenses for the newly installed components. It also helps to run this command a second time and make sure that it prints "All SDK package licenses accepted".
5. Run upload_android_tools.py -t sdk `$ dev/bots/upload_android_tools.py -t sdk`

How to update Android NDK on Google Cloud Storage

1. Download a new NDK binary (e.g. android-ndk-r10e-linux-x86_64.bin)
2. cd dev/bots/android_tools `$ cd dev/bots/android_tools`
3. Remove the old ndk directory `$ rm -rf ndk`
4. Run the new NDK binary file `$ ./android-ndk-r10e-linux-x86_64.bin`
5. Rename the extracted directory to ndk `$ mv android-ndk-r10e ndk`
6. Run upload_android_tools.py -t ndk `$ cd ../..` `$ dev/bots/upload_android_tools.py -t ndk`

Flutter codelabs build test

The Flutter codelabs exercise Material Components in the form of a demo application. The code for the codelabs is similar to, but distinct from, the code for the Shrine demo app in Flutter Gallery.

The Flutter codelabs build test ensures that the final version of the Material Components for Flutter Codelabs can be built. This test serves as a smoke test for the Flutter framework and should not fail. If it does, please address any issues in your PR and rerun the test. If you feel that the test failing is not a direct result of changes made in your PR or that breaking this test is absolutely necessary, escalate this issue by submitting an issue to the MDC-Flutter Team.

Unpublishing published archives

Flutter downloadable archives are built for each release by our continuous integration systems using the `prepare_package.dart` script, but if something goes very wrong, and a release is published that wasn't intended to be published, the `unpublish_package.dart` script may be used to remove the package or packages from the channels in which they were published.

For example To remove a published package corresponding to the git hash `d444a455de87a2e40b7f576dc12ffd9ab82fd491`, first do a dry run of the script to see what it will do:

```
$ dart ./unpublish_package.dart --temp_dir=/tmp/foo --revision
d444a455de87a2e40b7f576dc12ffd9ab82fd491
```

And once you've verified the output of the dry run to be sure it is what you want to do, run:

```
$ dart ./unpublish_package.dart --confirm --temp_dir=/tmp/foo --revision
d444a455de87a2e40b7f576dc12ffd9ab82fd491
```

and it will perform the actions. You will of course need to have access to the cloud storage server and have `gsutil` installed to perform this operation. Only runs on Linux or macOS systems.

See `dart ./unpublish_package.dart --help` for more details.

Once the package is unpublished, it will not be available from the website for download, and will not be rebuilt (even though there is a tagged revision in the repo still) unless someone forces the packaging build to run again at that revision to rebuild the package.