

Flake stabilisation proposal

Preface

FIXME: this page hasn't been reviewed by Lix Core team members, so it's effectively a draft/suggestion/pre-RFC/dream, whatever. It's not an official design document, but thought has been put into making it good, anyway.

Problem Statement

Flakes are a mess. They are extremely popular (so it's very painful to discard them), but they are also deeply flawed in so many ways, and their compat story is non-existent. Let's go through a few things that are *traditionally* associated with flakes, but they don't need to be.

- 2.4 CLI is obvious. There's no reason why it ever had to be tied so much to flakes. It should be stabilized independently (and probably before flakes)
- Pure eval. Again, never should've been flake-gated
- Channels deprecation
- NIX_PATH deprecation
- builtins.fetchTree deprecation/refactoring/stabilization (TODO: research this more)

On the last 2 points, see this: <https://samuel.dionne-riel.com/blog/2024/05/07/its-not-flakes-vs-channels.html>

Overall, flakes **did too much at once**. We can sort those out one by one. Deprecating NIX_PATH and channels would be a bit tricky, but we can try to re-use flake registries for the same functionality.

Also, flakes have a very bad backwards-compatibility story. Worse than that, we are a CppNix fork, so we want to provide a migration path for a reasonable amount of time. CppNix also completely doesn't have forward compatibility. This means that doing any changes to the `flake.nix` or to `flake.lock` will break flakes for CppNix users. This is really bad, it essentially means we're removing flakes outright, so this isn't something that we want to do.

With those preparations out of the way, we can now get to the flakes.

Flake Components

Flakes themselves have many moving parts.

- `flake.nix` schema: `description`, `nixConfig`, `inputs` and `outputs`

- `inputs` are super static. Changing anything about them will break a lot of stuff
- `outputs` is extensible. Changing the predefined attributes isn't great and can break things
- `description` and `nixConfig` are arbitrary, and can contain bogus info (FIXME: is this true?) We can use this to introduce new functionality without changing other fields, but this is a crime, so let's try to avoid that
- `inputs` URL parser
- `flake.lock` format

The most cursed part is how tightly connected all of that is. `flake.lock` records the inputs to `builtins.fetchTree`. These inputs are parsed from `flake.nix`. The real abstraction here is `inputs` URL parser. Everything else is implementation details that leak out into public interfaces.

So the situation is tricky. Code changes leak out, there are no useful versioning mechanisms, we need to make changes in such a way as to not break upstream, and the adoption is large enough that we don't want to break things. But thankfully, there is a way to deal with it, largely inspired but Opentofu's approach.

The Plan

Stage 0: Fork the Interfaces

First, we must fork the interfaces. Instead of having ossified `flake.nix` and `flake.lock` interfaces that we have no control over - we fork them into different files. Naming is TBD, but let's use `flake.lix` and `flake.lick` in this discussion. More specifically, the procedure looks like this:

- We change all of the flake-related code to use `flake.lix` and `flake.lick` files instead
- We add new internal structures for `flake.lix` and `flake.lick`. For starters, we can have the same structure, but fix the versioning story: `flake.lick` should have SemVer versioning instead of monotonic uint (that would make experimenting and/or forking the format so much easier, because SemVer allows "metadata" info added to the actual version), and `flake.lix` should have the `version` top-level element, too. `flake.lix` is computable, and so it's very non-trivial and depends on many factors: we **must** version it. Also SemVer. The versions have to be managed separately
- We add the migration code. It would look at `flake.nix` and `flake.lock` and create corresponding `flake.lix` and `flake.lick`

This completely changes the compatibility story, because we no longer have to think about upstream usage: we only read, never modify the files the upstream uses. Together with adding sane versioning, we can isolate the versioning to just our project, and make changes (including backwards-incompatible ones) in a sane manner.

Stage 1: Eating Spaghetti

Next, we need to decouple implementation, `flake.lix` and `flake.lick` from each other. For the latter two, we already have separate version on "manifest" file and the lockfile; it's a good start. Let's discuss what needs to be done to unveil this spaghetti:

- Implementation and `flake.lix`
 - TODO: does it make sense to use `builtins.fetchTree` for inputs, or do we need a separate interface?
 - Parametrized URLs are similar to [Terraform](#), but they have an extreme amount of edge cases to cover. The actual parameters should be separate arguments; no need to try to embed them into a URL
 - `follows` mechanism is horrible. It is extremely rare that you want to respect downstream lockfile in practice. Let's just not do that
 - `inputs` is a special case among special cases; it can't contain any logic, and it also uses C++ code for [trust on first use](#). There's no reason to be so locked into C++: it may be reasonable to expose the toggle to do trust on first use to the user, and have `inputs` be regular NixLang, and possibly even its interpretation be in NixLang (TBD about that)
- Implementation and `flake.lick`
 - The implementation completely bleeds through to the lockfile: it saves all of the arguments for `builtins.fetchTree` and uses that for reproducibility
 - To verify that the contents are actually the same, we need a checksum; `narHash` is the checksum. TBD if we want checksum to have more complex structure (algo/version/w.e. as well as value) or if lockfile versioning is enough
 - Instead of saving all of the arguments for the particular fetcher, we need to have an abstract `version` that we can compute from fetcher contents (TBD if in NixLang or in C++)
- `flake.lix` and `flake.lick`
 - As pointed out above, parametrisation of URLs is a blatant abstraction violation; the interface for parameters in `flake.lix` and `flake.lick` should match
 - `flake.lix` should contain "version range", and `flake.lick` should contain the "resolved version". The entire specifics are tricky for e.g. git

Stage 2: Improving the Interfaces

There's a lot that can be done here. Cross-compilation, version resolution, newer fields, and more - all of that belongs to this stage.

Stage 3: Maintenance

This path is backwards-compatible throughout, so we can maintain an upgrade path without much issue. We can have a directory with subdirectories for each major version. Those subdirectories will also handle upgrading the lockfile; then, we'll always have a path to upgrade from CppNix flakes to Lix flakes: you just execute all of the existing upgrades in order.

Truly backwards-incompatible changes would be adding absolutely necessary metadata, without which the previous version is useless. `npm` has this: their oldest lockfile (you can call it "v0") didn't

have a `version` field, and it also didn't record checksums. It simply doesn't contain any metadata that better lockfiles do, so the only way to move forward is to extract whatever you can from it, and generate a newer-version lockfile from scratch with that data.

As long as we only need the `version` and `checksum` (which seems to be the case), the only source of breaking changes I see is security vulnerabilities. If e.g. NAR hashing is proven to be vulnerable - it's probably for the best to not rely on the already existing hashes at all.

Notes

This plan doesn't have to be executed as sequentially as it's described. Really, we can have something like a from-scratch rewrite for flakes and include it in the first `flake.lix` + `flake.lock` versions. Or we could only add the versioning code. Or we could add versioning and version resolution, or versioning and cross-compilation, or literally anything else, as long as versioning is definitely present.

Revision #1

Created 28 October 2024 11:48:55 by kfearsoff

Updated 28 October 2024 20:36:49 by kfearsoff