Github Down? FOD's Failing to Build?

Let's Talk About Caching

Ethan Carter Edwards
ethan@ethancedwards.com

2026-08-07

Slides, source: https://ethancedwards.com/blog/2026-def-con-nix-vegas

2026 DEF CON 34 Nix Vegas

About Me

- FLOSS lover
- Nix user, Nixpkgs contributor for almost 6 years
- Member of NGI, CUDA, and darwin-maintainers teams
- Summer of Nix 2025 fellow
- Infra, Nix @ `exa.ai` (we're hiring!! come find me please)
- CS+Philosophy @ Harvard '29
- `ethancedwards8` on GitHub, Matrix

Disclaimers

- Thank you to exa.ai for sponsoring my presentation and making it possible for me to attend DEF CON 34 Nix Vegas!
- The views and opinions expressed here are my own and do not necessarily reflect the official policy, position, or opinions of my employer, university, or any other affiliated entity.

Outline

FETCHING

build vs eval time

Build Time

```
src = fetchurl {
  url = "https://github.com/sgl-project/sglang/archive/refs/tags/v0.5.16.tar.gz";
  hash = "sha256-KAA+DCEDGdBveDMzo8mXcMRxKf0GTw8FdjRzfj1SaLY=";
};
# or
src = fetchFromGitHub {
  owner = "vllm-project";
  repo = "vllm";
  tag = "v0.24.0";
  hash = "sha256-ArmNLA71YRNpBAMlWxwBzUroMFjhyZ2ZsjX8JNc4pH4=";
};
```

Using Fixed-Output Derivations (FODs) is common to fetch source tarballs at build time

What is a Fixed-Output Derivation?

They're derivations that have a cryptographic hash that is known in advance.

Because we know the hash before building, we grant the Nix sandbox permission to use networking to download a file (something that is ordinarily disallowed). Common FOD functions are `pkgs.fetchurl`, `pkgs.fetchFromGitHub`, `fetchCargoVendor`, and more. Basically, anywhere we pre-define a hash is a FOD.

FOD Fetcher Function

Simplified version of `pkgs.fetchurl` from the manual

```
{ stdenv, curl }:

{ url, sha256 }:
stdenv.mkDerivation {
  name = baseNameOf (toString url);
  builder = ./fetch.sh;
  buildInputs = [ curl ];
  # This is a fixed-output derivation; the output must be a regular
  # file with SHA256 hash sha256.
  outputHashMode = "flat";
  outputHashAlgo = "sha256";
  outputHash = sha256;
  inherit url;
}
```

Fun Fact: FODs are also Content-Addressed. Farid Zakaria (who's speaking this week!) has a really good blog post demonstrating what this means in practice (following example from his blog).

```
derivation {
  name = "hello-nix-vegas";
  builder = "/bin/sh";
  system = builtins.currentSystem;
  args = [ "-c" ''
    echo -n "Hello Nix Vegas!" > "$out"
  '' ];
  outputHashMode = "flat";
  outputHashAlgo = "sha256";
  outputHash = "sha256-UQm8yrf/JGWh2Py4rivs6wWDgGvp0X1SyDasxI0lKmc=";
}
# nix-instantiate: /nix/store/a74f0k203pmw8yv0sjwvv0i30dmzh276-hello-nix-vegas.drv
# nix-build: /nix/store/scqg9jvc7ff5kis54mpznq35dmf0ydgq-hello-nix-vegas
```

However, if we add a new line to args:

```
derivation {
  name = "hello-world-fixed";
  builder = "/bin/sh";
  system = builtins.currentSystem;
  args = [ "-c" ''
+   echo "I will not change the hash"
    echo -n "hello world" > "$out"
  '' ];
  outputHashMode = "flat";
  outputHashAlgo = "sha256";
  outputHash = "sha256-j8XK8hu/YxYc+/SHmmKJPqo1eylmP1GUgkAp0KJcRZg=";
}
- # nix-instantiate: /nix/store/a74f0k203pmw8yv0sjwvv0i30dmzh276-hello-nix-vegas.drv
+ # nix-instantiate: /nix/store/bbjnlynw9hr3l5hs26xafc17ay1b53jq-hello-nix-vegas.drv
# nix-build: /nix/store/scqg9jvc7ff5kis54mpznq35dmf0ydgq-hello-nix-vegas
```

Eval Time

Flake Inputs

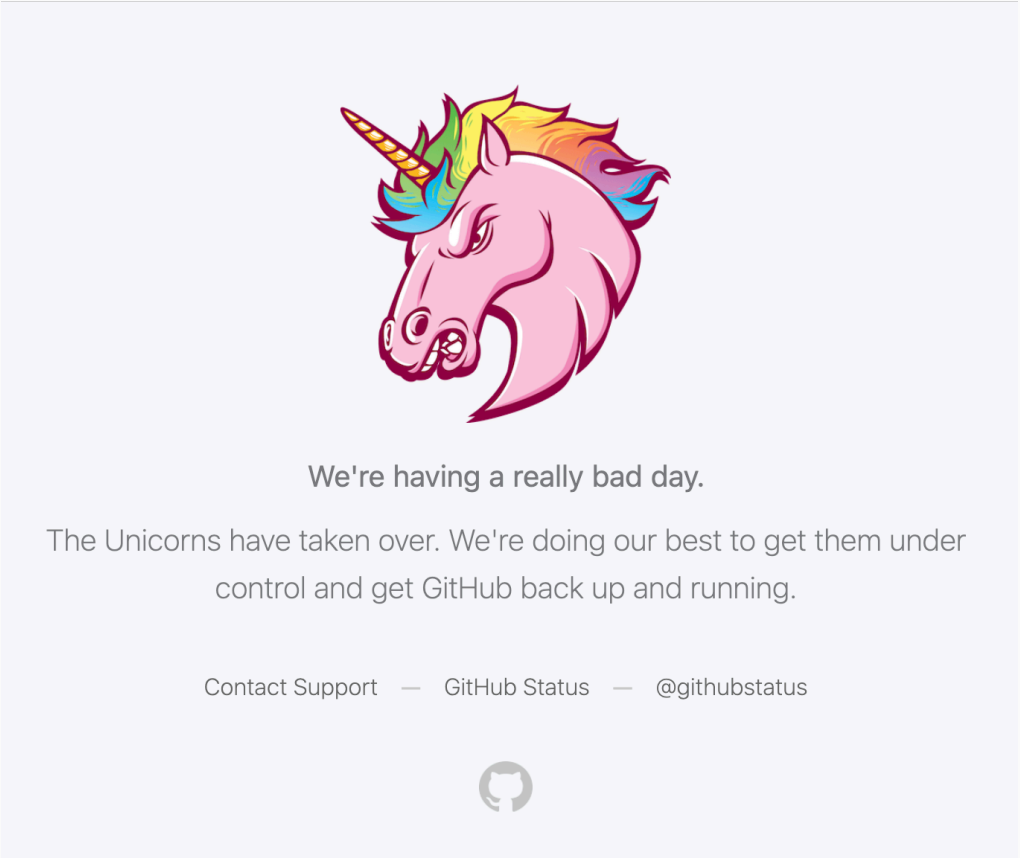
```nix
# flake inputs
{
  inputs.nixpkgs.url = "github:nixos/nixpkgs/6f095be";

  outputs = { nixpkgs, ... }: { ... };
}

# fetching for imports
{
  pkgs ? import (builtins.fetchTarball { url = "..."; sha256 = "..."; }) { },
  ...
}:
```

To fetch things at eval time, flake inputs and `builtin.fetch*` functions are often used

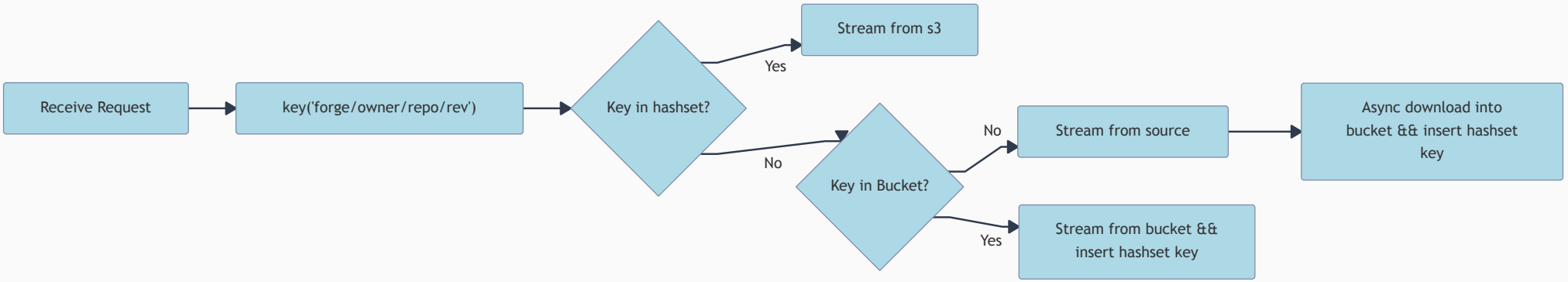
The Problem



- `GitHub is unreliable`
- We still need to ship during outages
- Rate limits are too low and lead to pain
- Caching is the solution

So what's the solution?

A pull-through s3-backed tarball cache.



- Stateless to make deployment and scaling easy
  - Hashmap pointer cache is in-memory only and is volatile
    - First request for a tarball on new pod may be slower, but after is faster
- Backed by s3, the greatest cloud service of all time
- Because we fetch (hopefully) immutable git commits/tags, once a tarball is added to the bucket it should never change. This means the hash is stable.
  - The cache tarball/hash should be identical to the upstream tarball/hash.
  - We don't support release artifacts, only commit/tag tarballs. Remember `xz`?
- Multiple git forges are supported
- To keep things simple and secure (no auth), only public repos are supported
- Yes, FODs/inputs are cached locally or in remote stores, but having an extra layer is useful
- Reference async Rust (`axum`) implementation at `https://github.com/ethancedwards8/tarball-cache`

Maintaining the hash

```nix
{ pkgs ? import <nixpkgs> { } }:

rec {
  github = pkgs.fetchurl {
    url = "https://github.com/nixos/nixpkgs/archive/78ee0aba.tar.gz";
    hash = "sha256-PNlBy3B4RNLEacMKCCtBAQC0moU51+UorUw3xg5AsnE=";
  };

  cache = pkgs.fetchurl {
    url = "https://cache.xyz/github/nixos/nixpkgs/78ee0abaa.tar.gz";
    inherit (github) hash;
  };
}
```

Bucket Layout

The bucket should be layed out roughly:

```
├── codeberg
│   └── poz
│       └── niri-nix
│           └── da8a388cfc14d55f19992c27e6870d836948bc19.tar.gz
├── github
│   └── nixos
│       └── nixpkgs
│           ├── 6b316287bae2ee04c9b93c8c858d930fd07d7338.tar.gz
│           ├── 78ee0abaa454bc057b6e5623b188b9f4b87be24a.tar.gz
│           └── 26.05.tar.gz
├── sourcehut
│   └── misterio
│       └── nix-colors
│           └── 81c0629d3a9a77e2a1d0b381a91760e34149a97d.tar.gz
```

- `git clone https://github.com/ethancedwards8/tarball-cache`
- Fill out .env.example
- `nix run`
- `curl -I https://localhost:3000/github/nixos/nixpkgs/d6566f851e850ebc4382062793123c11a3823c8f.tar.gz`

We've been using this kind of cache for 3 months with great results.

- Are immune to GitHub outages, degradations and rate limits
- Are protected from force-pushed git tags
  - Yes, the hash protects us too, but extra layers are nice
- Actually saw latency decrease because we're usually serving tarballs from within the same AWS region.
- Have a list + count of every tarball ever downloaded
- Enforce compliance in CI
- 0 cache-related outages

To be transparent, Exa does not use the handout code I linked above. Exa uses its own, non-public version that I wrote on my first day:

- Written in Go
- Supports access control + auth mechanisms
- Analytics and integration with cloudflare
  - Our prom/grafana metrics were wrong until I realized Cloudflare was caching responses
- Has denylist/allowlist support
- Supports more than just git forges

But the design is exactly the same.

Here are some alternatives and how we thought about them:

- `pkgs.fetchurl` supports the `mirror://` pattern
  - This is less explicit and would cause confusion among agents and humans.
- Use `impureEnvVars` to set `http{,s}_proxy`
  - Same as the above.
- Just using s3+cloudfront and forgoing the extra microservice
  - Reasonable, but would require more manual insertion of tarballs into the bucket. We like the pull-through pattern. Plus, this is more fun :)
- Keeping the microservice but requiring an explicit `PUT` to add new tarballs
  - Same as the above.
- `https://www.varnish.org/` - good, but afaik doesn't support persistent backends? Also its C
- Yes, FODs are cached in substituters, but having an extra safety layer is necessary at scale.
- Future work: Signing the tarballs and verifying them with a custom fetch function

Exa is hiring across Infra (we use Nix!!), Security, Full-Stack, Product, Research, etc. please come find me if you're interested (we have swag 🐐)!

We push Nix to its limits (come see my next talk) and are operating at beyond web scale.

Want the slides? Visit my website: https://ethancedwards.com/blog/2026-def-con-nix-vegas

Want the handout code? https://github.com/ethancedwards8/tarball-cache

Q/A about the talk?