2026-08-09 · 7 MIN READ

nixpkgs-multiverse: every version that ever existed

Enter the Nixpkgs multiverse. All the versions that ever existed, all in one place.

I bumped the `nixpkgs` release for my NixOS configuration to refresh many of my packages and found that a package I depended on at a particular version is no longer available.

The package was "version bumped forward" in a way that broke some of my tooling. It's late and I don't want to fix it, so I just add another `nixpkgs` input pinned to the commit that had the version I want. This works, but it is miserable in a way that compounds.

NIX

```nix
{
  inputs = {
    nixpkgs-unstable.url = "github:NixOS/nixpkgs/nixos-unstable";
    nixpkgs-25_11.url = "github:NixOS/nixpkgs/nixos-25.11";
    nixpkgs-25_05.url = "github:NixOS/nixpkgs/nixos-25.05";
  };
}
```

The need for the most recent package is so common that I had keep an overlay that would inject `unstable` as a package set for me to easily pull from.

NIX

```nix
unstable-packages = final: _prev: {
  unstable = import inputs.nixpkgs-unstable {
    system = final.stdenv.hostPlatform.system;
    config.allowUnfree = true;
    overlays = [inputs.nix-vscode-extensions.overlays.default];
  };
};
```

If I have a need for a particular version of a package and it's not present in my current `nixpkgs`, I am left searching for the commit and pinning it.

> 1 Thankfully sites like nixhub.io (https://www.nixhub.io/) or lazamar's search (https://lazamar.co.uk/nix-versions/) make this a little easier.

Every pin is a whole extra `nixpkgs` in the file. Flake inputs are fetched **eagerly** even if not used. A flake with three `nixpkgs` inputs whose output references only the *first* are all materialised.

Nix lets us easily create a closure that reproduces a specific version of a package, but Nixpkgs makes it hard to hold one package still while everything else moves.

Each Nixpkgs input to a flake is a distinct universe. If we can have multiple Nixpkgs as input to achieve fetching a particular package, why not **have every version that ever existed always available**? 🤯

§ nixpkgs-multiverse

nixpkgs-multiverse (https://github.com/fzakaria/nixpkgs-multiverse) is one flake input that gives you **all of them** at once.

```
$ nix run 'github:fzakaria/nixpkgs-multiverse#versions.python3."3.6.2"' --
Python 3.6.2

$ nix run 'github:fzakaria/nixpkgs-multiverse#versions.python3."3.8.9"' --
Python 3.8.9

# We can also get the latest version of a package.
$ nix run 'github:fzakaria/nixpkgs-multiverse#latest.python3' -- --version
Python 3.14.6
```

We can query the flake for all the versions of a package that **ever existed in Nixpkgs**.

```
$ nix eval --json --apply 'f: f "python3"' \
   github:fzakaria/nixpkgs-multiverse#multiverse.x86_64-linux.versionsOf
[
  "3.5.3",
  "3.6.2",
  # 53 other versions omitted for brevity
  # ...
  "3.13.13",
  "3.14.6"
]
```

If we want a specific complete revision of Nixpkgs we can use the `at` function.

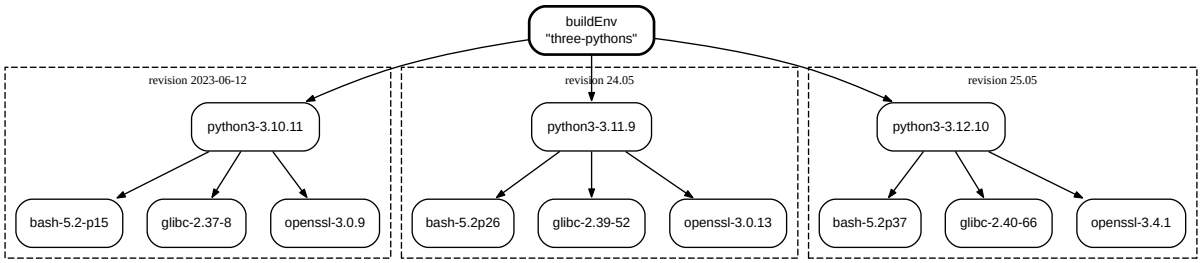
NIX

```
let
  mv = multiverse.multiverse.x86_64-linux;
  # newest revision the index knows, as a real Nixpkgs
  pkgs_tip = mv.tip;
  # by release
  pkgs_24_11 = mv.at "24.11";
  # newest revision on or before that date
  pkgs_2022_03_15 = mv.at "2022-03-15";
  # by commit
  pkgs_aae12a743f75 = mv.at "aae12a743f75";
in {
  packages = [
      pkgs_tip.python3
      pkgs_24_11.python3
      pkgs_2022_03_15.python3
      pkgs_aae12a743f75.python3
    ];
}
```

That is access to all the versions of all the packages that ever existed in Nixpkgs. You can mix them all together in one shell, one package or a build environment.

How is it possible to have multiple Python versions? That is the whole point of Nix itself. Every package immaculately describes its dependencies using a hash via the intensional model (/2025/03/08/demystifying-nix-s-intensional-model).

> **2** The hash is a unique identifier for the exact set of inputs that were used to build it. If you change any input, the hash changes and you get a new package.



(/assets/graphviz/c2754b42502fbbb6.svg)

Nixpkgs already supports multiple versions of a package in a single revision (i.e. `python39`, `python312`, `gcc12`) as separate attributes. We took this to its logical conclusion of making them all available easily.

§ What's the magic?

Our `flake.nix` deliberately has no inputs: `inputs = { }`. Inputs are fetched eagerly, and we have 1,393 of them. We need to fetch them lazily, only when

something actually references a revision. To do this, we fetch revisions with `builtins.fetchTree`, pinned by `narHash`, only when needed.

Two files do all the work: `revisions.json` and `versions.json`.

`revisions.json` is one ordered array of every revision from Nixpkgs (https://github.com/NixOS/nixpkgs), 1,393 as of this writing, from 2017 to 2026.

> 3 A NixOS release is not special; it is a commit that happens to carry a `release` label.

JSON

```
[
  {
    "rev": "0eeebd64de89…",
    "date": "2023-06-12",
    "channel": "nixos-unstable",
    "narHash": "sha256-2xT+Jmk3m…"
  },
  {
    "rev": "afb2b21ba489…",
    "date": "2025-05-23",
    "channel": "release",
    "release": "25.05",
    "narHash": "sha256-rWtXrcIzU5wm…"
  }
]
```

We limit our commits to those that were actually built and cached by Hydra, so we only include commits that were either a release or a `nixos-unstable` channel bump.

How do we know which revisions to pick for the `nixos-unstable` ones?

We rely on the nix-releases S3 bucket (https://nix-releases.s3.amazonaws.com/) to tell us which commits actually became published builds. The S3 bucket uses the commit hash as the directory name, so we can list the bucket and get a complete list of all revisions that were actually built.

`index/versions.json` is the map from (attribute, version) to a revision:

JSON

```json
{
  "revisionCount": 1393,
  "attrs": {
    "python3": { "3.8.9": 412, "3.12.10": 1204 }
  }
}
```

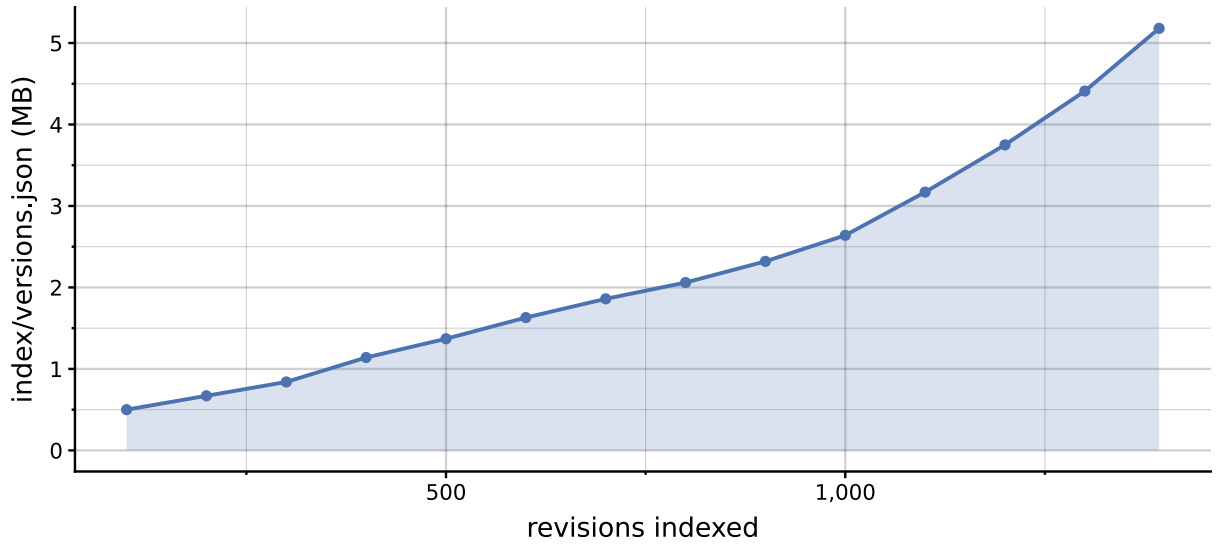
That integer is an offset into `revisions.json`. It is the most recent revision that shipped that version.

§ Sparse data

At this many revisions, it turns out that how you encode the data matters a lot. My first encoding stored every revision a version appeared in.

Although it was simple, it was a disaster in terms of size for these JSON files. As you might expect, most versions of most packages are unchanged across many revisions. The size of our `versions.json` file was growing linearly with the number of revisions.

By storing only the *newest* revision that shipped a version, we can keep the file small and still answer the question "which revision had this version". Here is how it actually grows as revisions get indexed:



(/assets/plotnine/bb1965e463779551.svg)

5.18 MB covering **1,393 revisions** and 289,521 distinct (attribute, version) pairs.

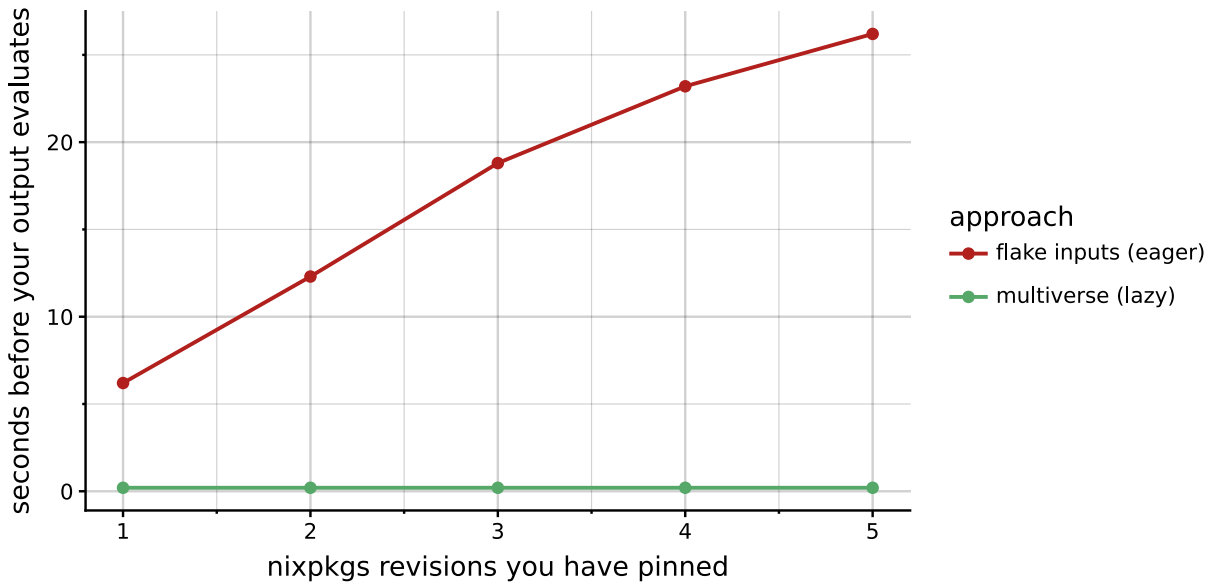
§ Performance

The key design rule for our flake:

> Cost is per **revision touched**, not per package.

If we were to add revisions as inputs, evaluating our flake would explode. Each flake in our measurement below has N `nixpkgs` inputs and an output that references **only the first one**; the timing is how long before that output evaluates.

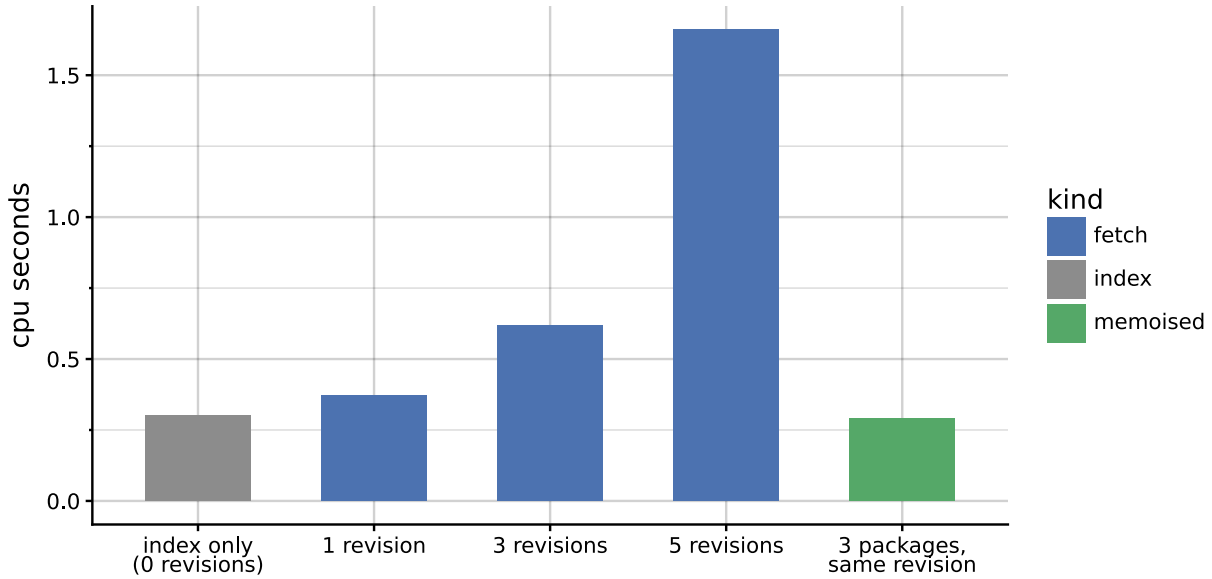
> 4 Everything is `git+file` against a local clone, so there is no network latency.



(/assets/plotnine/3ccac799768d6a6d.svg)

Five `nixpkgs` pins that are not used cost **26 seconds** before the output evaluates. Each input costs about 5 seconds, and the input is fetched and materialised even if never used. In contrast, the green line is `nixpkgs-multiverse` with **1,393 revisions available**, which is a flat 0.20s to parse the JSON. 🤩

Revisions are memoised, so pulling 3 packages out of one revision costs the same as pulling one.



(/assets/plotnine/db726a030d67b90f.svg)

§ Why I like this

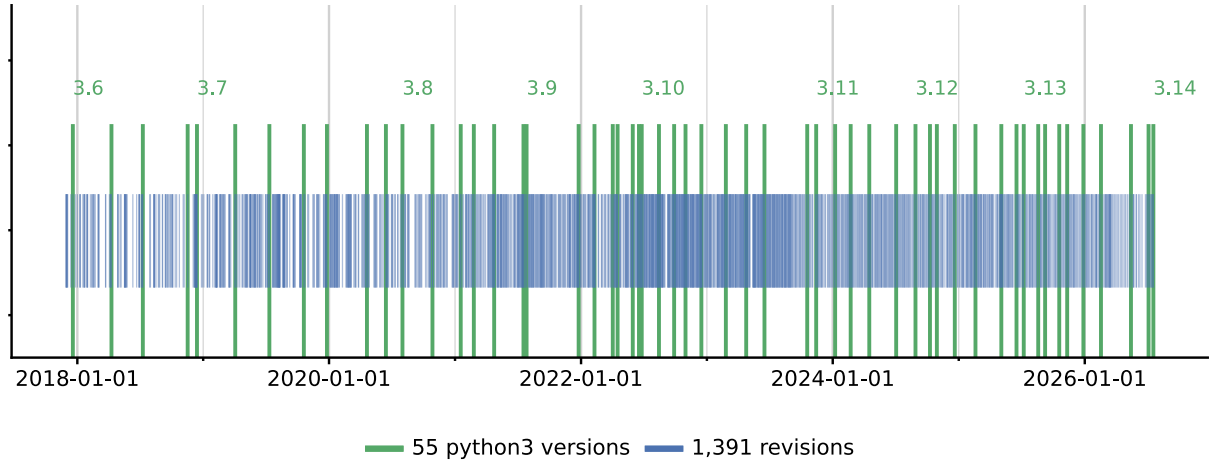
That concept that the `/nix/store` can hold many graphs of the same package is core to understanding Nix. The popularity and rise of flakes made it even more apparent that we can mix multiple revisions of Nixpkgs together.

The thing I keep coming back to is that Nixpkgs history *already is* the multiverse. Every version that ever existed is already built, already cached, already reachable. It was just addressed by commit hash instead of by version number, which is exactly backwards from how anyone thinks about it.

The whole project is 5 MB of JSON and about 200 lines of Nix. It does not build anything, mirror anything, or host anything. It is a phone book.

NIX

```nix
inputs.multiverse.url = "github:fzakaria/nixpkgs-multiverse";
```



(/assets/plotnine/4dcff788eb3cd44a.svg)
