# How NixOS is built

Jul 31, 2025

I recently got the urge to better understand how NixOS is built and how secure the build pipeline is. So, I started looking at all the build systems involved, the infrastructure they run on, how everything is managed and which build jobs are running in these systems. This is my attempt at documenting the above.

Table of contents:



## Repos and infrastructure

I'll focus on two main repositories, NixOS/nixpkgs and NixOS/infra. Nixpkgs is the main repository where all the packages and modules of NixOS are defined,


Sharing my thoughts on Decentralization, Security and Operations.

---



---



---

while infra is the repository that specifies most of the infrastructure used by the NixOS foundation.

Anyone with a GitHub account can contribute to Nixpkgs, there are no special groups one needs to belong in to open a PR. There are about 200 committers currently with access to merge PRs.

The infra repository contains Terraform/OpenTofu files that define the cloud infrastructure used and nix expressions for configuring the hosts. The two main cloud services used in infra are AWS S3 for object storage and Fastly that acts as a CDN and a cache for the S3 buckets. Again, anyone can contribute to the infra repository, but it's generally managed by the Infra team.

## Getting the NixOS ISO

Let's start from the beginning of a user's journey to installing NixOS. We first need to download the NixOS ISO, so we visit the download page, scroll to the bottom of the page and find the link for the ISO. Right now this is: https://channels.nixos.org/nixos-25.05/latest-nixos-minimal-x86_64-linux.iso.

What is this `channels.nixos.org` website and where is it defined and configured?

### Defining domains, S3 buckets and CDN

#### Domains

We can use `dig` to figure out where the nixos.org nameservers are hosted.

```
$ dig NS nixos.org +short
ns-61-b.gandi.net.
ns-161-c.gandi.net.
```

ns-177-a.gandi.net.
Turns out they're hosted on `gandi.net`. Let's move on to see where this is defined.

On NixOS/infra there's a `dns` directory with a flake-module.nix in it, where we see that DNSControl is used to configure domains. Looking at creds.json there's an entry to setup Gandi credentials which matches what we discovered earlier. Furthermore, at dnsconfig.js we include four more files, one for each of the following domains:

01: nixcon.org
02: nix.dev
03: nixos.org
04: ofborg.org

Following this breadcrumb, we arrive at dns/nixos.org where `channels.nixos.org` is defined as a CNAME to a Fastly domain. The exact same setup is used for `cache.nixos.org`, the NixOS binary cache from where users download prebuilt binaries.

**S3 buckets and CDN**

In infra/terraform we find the Terraform files for managing the S3 buckets and the Fastly CDN setup. For channels in particular, we define a S3 bucket with a website endpoint enabled which we can view in nix-channels.s3-website-us-east-1.amazonaws.com.

Further down the file, there's a definition for a Fastly Service which uses the public website endpoint of the bucket as a backend. Again, we have a similar setup for `cache.nixos.org`.

To recap, `channels.nixos.org` points to Fastly which caches responses and acts as a global CDN,

which in turn uses an AWS S3 bucket as the source for channels and packages.

## Building the packages

Now that we know the flow for getting the iso and packages, let's figure out how they're built in the first place.

### Nixpkgs and GitHub Actions

As mentioned previously, Nixpkgs is the main repository where packages and NixOS modules are defined and built from. In order for a package to be available via `cache.nixos.org` it must first be added to Nixpkgs. Nixpkgs has a lot of automated checks that prevent common problems, help with organizing Pull Requests (PRs) and Issues and generally make the life of contributors and reviewers easier.

The actions are defined in nixpkgs/.github/workflows. One thing that is interesting to note is that almost all actions run on an ARM build of Ubuntu 24.04 and just install Nix as one of their steps, they don't run on NixOS itself. I assume we're running on ARM because it's about two thirds of the price of running on x86-64 according to GitHub pricing. The monthly cost for all the actions in July of 2025 came out to a bit over 14500 USD which GitHub covers in its entirety.

Some actions run on a periodic schedule, while others run each time someone opens or updates a PR.

### Periodic actions

There are two main actions that run on a schedule. The first one is periodic merges which merges the master branch into staging every few hours. The other one is labels which runs every ten minutes and automatically applies labels to PRs and issues via this script.

**Pull Request actions**

Every time someone opens a new pull request or a pull request is updated, some actions run. In particular these actions revolve around different areas of the PR, from linting or building the code changes to applying labels and pinging the maintainers of a package to get a review.

The lint workflow lints the changes and also calls nixpkgs-vet which is a software specifically made to enforce NixOS RFC 140.

Eval, as the name suggests, evaluates the derivations (in all supported systems), calculates some statistics for later use and verifies things still evaluate. These statistics might include what paths were added or changed in a PR. Using PR #429664 as an example, we get the following artifact out of the diff between the PR and the previous master branch for x86_64.


```
{
  "added": [],
  "changed": [
    "python312Packages.llm-ollama.x86_64-linux",
    "python313Packages.llm-ollama.x86_64-linux",
    "release-checks"
  ],
  "removed": []
}
```

This information is later used by the previously mentioned labels workflow to add related labels to a PR.

Build, as the name suggests, builds `docs`, `shell`, `lib`, `tarball` but doesn't actually build the package. A point of interest here is the fact that the build action uses Cachix, a hosted Nix binary cache. This cache is only meant to be used by the Nixpkgs CI and shouldn't be trusted.

Reviewers figures out which contributors should get a ping to review the PR.

**Nixpkgs and OfBorg**

You might have noticed that we never actually build any packages or ran any NixOS VM tests in the GitHub Actions previously. This is because these actions happen on a different system, OfBorg. The name comes from Star Trek.

OfBorg automatically builds any PRs that follow a certain naming scheme or you can trigger builds/tests/evals manually. The results of these actions then get reported back to GitHub.

Up until the end of 2024, OfBorg was the main CI for Nixpkgs, however the company was sponsoring the hosts decided to end the sponsorship. This event triggered the creation of a big part of the previously mentioned GitHub Actions workflows.

Nowadays, the OfBorg x86-64 builders are hosted in tetaneutral, while the Darwin builders are hosted in MacStadium. The core/orchestrator service (`core01.ofborg.org`) is hosted in Hetzner Cloud.

There's also an adjacent project to view build logs that's deployed on `core01.ofborg.org` as well.

The OfBorg infrastructure is managed via ofborg/ofborg-infrastructure.

### Hydra

While OfBorg builds packages, runs tests and reports the results back to the GitHub PR, it never actually signs the packages or uploads them to `cache.nixos.org`. Building packages that are meant to end up in the official cache requires trusted hardware that only a few people have access to. This is the role of hydra.nixos.org.

Hydra is yet another CI service for Nix based projects. It's written in Perl and supports a typical CI architecture with multiple job runners. I believe it was the first CI system for Nix, the initial commit was on the 10th of October 2008!

Similarly with OfBorg, the Hydra builders had to quickly be scrapped and recreated at the end of 2024. Nowadays, the builders and the machine that hosts the actual Hydra application are Hetzner dedicated machines. You can see a list of all the current and past builders here.

Hydra only builds packages *after* a PR has been reviewed and merged.

#### Projects, jobsets and jobs

Builds in Hydra are organized in the following structure:

Project -> Jobset -> Job

To get back to the initial question of how the ISO is built, let's look at the NixOS project. We have a jobset for each release (release-25.05, staging, unstable-small, etc). Next, if we search for iso there's a job called nixos_iso_minimal. The configuration of the `release-25.05` jobset calls nixos/release-combined.nix from Nixpkgs.

This nix file specifies all the jobs that Hydra should build for this jobset. One of these jobs is nixos.iso_minimal, which is our entry point for building the ISO.

Similarly, for Nixpkgs there's a Nixpkgs project with a jobset called `trunk` that builds the latest committed version of a package.

**Signing and uploading to cache.nixos.org**

Anything that Hydra builds is signed and uploaded to the S3 bucket for `cache.nixos.org`.

We can verify this by querying the cache. This is the latest build of a package I maintain. On the details tabs we see:

Output store paths
/nix/store/zi6ypq21r8534cx53bx8rx930k749xs8-pytho
/nix/store/clz5dsk4cxsakbw9zw634riwhpjwhjk7-pytho

Let's query the cache for the narinfo file of this build:

$ curl https://cache.nixos.org/clz5dsk4cxsakbw9zw63

StorePath: /nix/store/clz5dsk4cxsakbw9zw634riwhpjw
URL: nar/0xpnlv3q605ljcid91kn72awipjh58x9f3iaqrj1c
Compression: xz
FileHash: sha256:0xpnlv3q605ljcid91kn72awipjh58x9
FileSize: 22620

NarHash: sha256:10g4imc7s0yhlx042ly44pfr98z1kjc5
NarSize: 102152
References: 9yh9ak97gn659bk4d3n411fx6c0ng7s2-p
Deriver: 06ni1ca50ndxh2pgxy0kznbyv07pxmyr-pytho
Sig: cache.nixos.org-1:5FnqydQlPAWi9bVeDoN2MVx

The signing and uploading to the cache behaviour is configured in nixos/infra.

My understanding is that builds of Hydra started getting signed in June of 2015. Moreover, the key has likely never been rotated before! There is an ongoing effort to support multiple key signatures at the same time and rotate the old key.

### My security fears

I don't mind Perl that much, but I think Hydra shows its age since it's littered with practices that today are considered antipatterns. A simple example, here's a remote endpoint that shells out to git. If that `die()` line that validates the two revs was missing, it would be a command injection vector. This pattern can be seen all over Hydra.

It took me a couple of hours to find a (minor?) security issue in Hydra when I started looking. I reported it upstream, but it was already reported by someone else. I think it will be out of embargo towards the end of August.

## Conclusion

I believe this is the complete flow for how NixOS is built. Nixpkgs is the source of truth for packages and modules, GitHub Actions and OfBorg help with catching errors and automation, Hydra is the authoritative builder that has the keys to the kingdom.

Hydra is definitely the scariest part of the whole pipeline for me. Both the fact that the signing key hasn't been rotated in ten years (how many people had access to it?) and the general vibes I get from the Hydra source code are creeping me out.

I *love* the fact that it's possible to figure out all of the above by just looking at the code in GitHub, past issues and discourse threads. Ideally, I would prefer if things were documented better, but I feel the need to emphasize how transparent everything is, *if* you're willing to invest the time. Improvements can definitely be made, but even the state we're in is a great starting point.

I'm looking forward to the next steps in this journey of better understanding the security of NixOS. I'm sure there's a lot more things to discover.

Tags: nix nixos
Source: GitHub

---

Comments on Fediverse.

---