Installing Every NixOS Package

Installing every single NixOS package in the official binary cache

12 min read | December 9, 2025

nix nixos lean

Four years ago, I installed every Arch Linux package (and someone else tried the same crazy experiment with Alpine Linux. But did I really accomplish my goal? Wellll... I slightly cheated, since I only installed non-conflicting packages rather than literally every single Arch package. Besides, installing conflicting packages would overwrite one package's files with the other's, so they wouldn't really both be installed.

The solution, as with any Linux problem, is NixOS. With NixOS, conflicting packages or even multiple versions of the same package can coexist in the Nix store, which are then activated into your environment via symlinks, hacked together with some nasty Bash and Perl scripts. Hooray for purely functional package management! Even better, search.nixos.org boasts a total of more than 120000 packages, nearly 10 times the measly 12232 packages that I installed in my Arch experiment.

But how many packages exactly are there in Nixpkgs? Is it really possible to install every one of them? Will I need 10 times more disk space than for Arch? How usable will the system be with every NixOS package installed? Will it even boot? Let's find out!

Attempt 1

First, I used `nixos-install` to make a fresh new NixOS systemd-nspawn container (to avoid inflicting unnecessary pain on any existing NixOS installations). Here are the container-specific lines in the new system's `configuration.nix`:

```
boot = {
  isNspawnContainer = true;
  loader.initScript.enable = true; # Creates /sbin/init
};
console.enable = true;
```

Next, I booted up the container with `sudo systemd-nspawn -bD /mnt`. Time to install every package! The `pkgs` attrset contains the derivations of every package, so it should be as simple as this, right?

```
environment.systemPackages = builtins.attrValues pkgs;
```

Nope! Shielded by lazy evaluation, most NixOS users have never witnessed the horrors that scuttle in the nooks and crannies of `pkgs`. They've never heard the scream of `pkgs.AAAAAASomeThingsFailToEvaluate` or gotten lost in the 200-line stack trace of

`scope.AAAAAASomeThingsFailToEvaluate` or raged at the stupidity of using both `.` in names and separators between names in `pkgs.kernelPatches.hardened.6.12.patch`. Don't underestimate `pkgs`.

Attempt 2

Yikes... maybe I should try solving an easier version of the problem first. Instead of recursively traversing `pkgs`, how about only installing every top-level package? For instance `pkgs.fish` would be included, but not `pkgs.kdePackages.dolphin`. To filter out packages marked as broken, insecure, or proprietary, the trick is to check `(builtins.tryEval x.value.drvPath).success`:

```
environment.systemPackages = builtins.attrValues pkgs
  |> builtins.map (
    x:
    if
      (builtins.tryEval x).success
      && builtins.isAttrs x
      && x ? "drvPath"
      && (builtins.tryEval x.value.drvPath).success
    then
      x
    else
      [ ]
  )
  |> lib.lists.flatten;
```

One `nixos-rebuild switch` later:

```
error: Cannot build '/nix/store/jbmc3rplwpm2xilpxdzw528i4v9lcb5b-liquidfun-1.1.0.ta
       Reason: builder failed with exit code 1.
       Output paths:
         /nix/store/4v3q7n13vb9w7qsrf5an4kz8q0f76lzl-liquidfun-1.1.0.tar.gz
       Last 11 log lines:
       >
       > ***
       > Unfortunately, we cannot download file liquidfun-1.1.0.tar.gz automaticall
       > Please go to https://github.com/google/liquidfun/releases/download/v1.1.0/
       > using either
       >   nix-store --add-fixed sha256 liquidfun-1.1.0.tar.gz
       > or
       >   nix-prefetch-url --type sha256 file:///path/to/liquidfun-1.1.0.tar.gz
       >
       > ***
       >
       For full logs, run:
         nix log /nix/store/jbmc3rplwpm2xilpxdzw528i4v9lcb5b-liquidfun-1.1.0.tar.gz
error: Cannot build '/nix/store/v1yh7qh1njidc99wmfgjx3mckqm1hqmg-liquidfun-1.1.0.dr
       Reason: 1 dependency failed.
```

Fun. So turns out `pkgs` contains quite a few packages that aren't marked as broken but still fail to build, so we need to filter them out somehow. Untrusem from exozyme suggested using Hydra, the Nix CI system. For each update to `nixos-unstable`, hydra.nixos.org conveniently lists which packages it failed to build. Hydra has an API, but I found it easier to simply Ctrl-select HTML table entries and parse that. I additionally had to filter out a few more broken packages which are marked as `hydraPlatforms = [ ];`.

This still didn't fully build due to some `pkgs.buildEnv` errors, but it's progress!

Attempt 3

After messing around in a `nix repl .#nixosConfigurations.$hostname`, I finally figured out how to recursively traverse `pkgs` with this monstrosity:

```
let
  pkgs = import <nixpkgs> { };
  lib = pkgs.lib;
  getpkgs =
    y: a: b:
    builtins.map (
      x:
      if
        (builtins.tryEval x.value).success
        && builtins.isAttrs x.value
        && !lib.strings.hasPrefix "pkgs" x.name # Ignore pkgs.pkgs*
        && !lib.strings.hasPrefix "__" x.name # Bad stuff
        && x.name != y.name # Definitely infinite recursion
        && x.name != "lib" # Ignore pkgs.lib, pkgs.agdaPackages.lib, and other evil
        && x.name != "override" # Doesn't contain packages
        && x.name != "buildPackages" # Another copy of pkgs
        && x.name != "targetPackages" # Yet another copy of pkgs
        && x.name != "formats" # Ignore the pkgs.formats library
        && x.name != "tests" # Ignore tests
        && x.name != "nixosTests" # Ignore more tests
        && x.name != "scope" # Ignore pkgs.haskell.packages.ghc910.buildHaskellPack
        && x.name != "_cuda" # Proprietary garbage
        && x.name != "vmTools" # Not a VM
        && x.name != "ghc902Binary" # Broken
        && x.name != "coqPackages_8_11" # Broken
        && x.name != "coqPackages_8_12" # Broken
        && x.name != "pypyPackages" # Broken
        && x.name != "pypy2Packages" # Broken
        && x.name != "pypy27Packages" # Broken
      then
        (
          if x.value ? "drvPath" then
            (
              if (builtins.tryEval x.value.drvPath).success then
                b
                + "|"
```

```
                + (if x.value ? "pname" then x.value.pname else "unnamed")
                + " "
                + x.value.outPath
              else
                [ ]
            )
          else if a > 10 then
            abort "Probably infinite loop?"
          else
            builtins.trace a
            <| builtins.trace x.name
            <| builtins.trace b
            <| getpkgs x (a + 1)
            # For some stupid reason x.name can contain . so use | as the separator
            <| b + "|" + x.name
        )
      else
        [ ]
    )
    <| lib.attrsToList y.value;
in
lib.strings.concatStringsSep "\n"
<| lib.lists.flatten
<| getpkgs {
  name = "pkgs";
  value = pkgs;
} 0 "pkgs"
```

And here's how to run it:

```
nix eval -f getpkgs.nix | sed 's/\\\\n/\\n/g' | sed 's/"//g' > all
```

The script took 10 minutes and 48 GB of RAM and finally spat out a 623946-line file with every Nix package. 600K? That's 50 times more packages than my Arch experiment, and 5 times as many as search.nixos.org claimed!

Actually, many of the `outPath`s are duplicated, so the true count is much lower. Since I've suffered enough trauma already from programming in Nix, I switched over to using the best language ever for filtering the package list. First, I deduplicated the file:

```
import Std.Data.HashSet

def main : List String → IO Unit
  | [inp, out] => do
    let lines := (← IO.FS.readFile inp).splitOn "\n"
    let h ← IO.FS.Handle.mk out IO.FS.Mode.write
    let mut added := Std.HashSet.emptyWithCapacity
    for line in lines do
      match line.splitOn " " with
      | [_, _, store] =>
        if ¬added.contains store then
```

```
      | _ => return ()
  | _ => IO.println "Wrong number of args"
```

Down to only 281260 packages now!

The next task was to filter out broken packages. Unfortunately, some namespaces like `pkgs.python310Packages` and `pkgs.haskell.packages.native-bignum.ghc910` are skipped by Hydra but contain tons of packages that fail to build. Since I'd rather not spend a month compiling thousands of semi-broken packages, I adjusted my goal to be installing every package in the official cache.nixos.org binary cache. So yeah, *technically* I'm slightly cheating once again but whatever.

I now had two options:

1. Use Hydra's list of successfully built packages. However, that list is really long so my copy-and-paste trick won't work and I'll have to use Hydra's API instead.
2. Directly query cache.nixos.org.

The second option seemed slightly easier, so I hacked together a program using the best language ever and its Curl bindings:

```
import Curl

open Curl

def N := 1024

def main : IO Unit := do
  try
    let mut lines := (← IO.FS.readFile "dedup").splitOn "\n"
    let h ← IO.FS.Handle.mk "cached" IO.FS.Mode.write

    while 0 < lines.length do
      IO.println s!"{lines.length} lines remaining"
      let batch := lines.take N
      lines := lines.drop N

      let curlM ← curl_multi_init
      let resps ← batch.filterMapM fun line ↦ do
        match line.splitOn " " with
        | [_, _, store] =>
          let hash := store.stripPrefix "/nix/store/" |>.takeWhile (· ≠ '-')
          let resp ← IO.mkRef { : IO.FS.Stream.Buffer}
          let curl ← curl_easy_init
          curl_set_option curl <| CurlOption.URL s!"https://cache.nixos.org/{hash}.
          curl_set_option curl <| CurlOption.NOBODY 1
          curl_set_option curl <| CurlOption.HEADERDATA resp
          curl_set_option curl <| CurlOption.HEADERFUNCTION writeBytes
          curl_multi_add_handle curlM curl
          return some (resp, line)
        | _ => return none

      while 0 < (← curl_multi_perform curlM) do
```

```
      let good_lines ← resps.filterMapM fun (resp, line) ↦ do
        let bytes := (← resp.get).data
        return if _ : bytes.size > 9 then
          -- Check if bytes starts with "HTTP/2 200"
          if bytes[7] = 50 ∧ bytes[8] = 48 ∧ bytes[9] = 48 then
            some line
          else
            none
        else
          none

      -- Don't print out extra newline if empty
      if 0 < good_lines.length then
        h.putStrLn <| "\n".intercalate good_lines
        h.flush
  catch e => IO.println s!"error: {e}"
```

This program took around 5 minutes to run since I didn't try optimizing it very much. Not bad, considering that it makes 281260 network requests!

Now we're left with only 71480 packages. I wrote a third program to generate the `configuration.nix` containing that list of packages:

```
def main := do
  let lines := (← IO.FS.readFile "cached").splitOn "\n" |>.filterMap fun (line : St
    match line.splitOn " " with
    | [path, pname, _] =>
      match path.splitOn "|" with
      | .nil => none
      | part :: parts =>
        -- Quote each part because it might contain weird characters
        some <| part ++ ".\"" ++ "\".\"".intercalate parts ++ "\""
    | _ => none
  IO.println s!"Final count: {lines.length}"
  let template ← IO.FS.readFile "configuration-template.nix"
  IO.FS.writeFile "configuration.nix" <| template.replace "999999" <| "\n".intercal
```

You can find the full `configuration.nix` here.

Finally, time for the moment of truth! `nixos-rebuild switch` !

400 GB of downloads later, I was greeted with some more fun errors:

```
pkgs.buildEnv error: /mnt/nix/store/p10qdg9nz84pl9fhskqs5pj6q28i90z2-all-cabal-hash
```

This error occurs when nixos/modules/config/system-path.nix calls pkgs/build-support/buildenv/builder.pl to build the global environment, but we can pass the parameter `ignoreSingleFileOutputs = true;` to ignore this error. The NixOS module system is pretty cryptic to me, so I asked my friend Ersei from exozyme for help with overriding a module.

1. Make a copy of `system-path.nix` with the edit I want.

configuration.nix.

Unfortunately, `pkgs.buildEnv` still wasn't happy:

```
pkgs.buildEnv error: not a directory: `/nix/store/dv8q5wfm717q4qqk6pps6dyiysqqf22c-
```

Basically, some package created the directory `bin/internal`, but now `gci` is trying to create a file at the same location. There's probably a smarter way, but my solution was to remove `gci` from the list and repeatedly run `nixos-rebuild` to catch and remove all the other offenders. OK yeah yeah I'm cheating again slightly, I'm only omitting a few more packages that no one cares about.

```
match pname with
| "hypothesis" -- Cannot build '/nix/store/4k1n7lbmkihdf1jisahqrjl3k4cbmhgf-hypothe
| "gci" -- pkgs.buildEnv error: not a directory: `/nix/store/dv8q5wfm717q4qqk6pps6d
| "openjdk-minimal-jre" -- pkgs.buildEnv error: not a directory: `/nix/store/zmcc5b
| "olvid" -- pkgs.buildEnv error: not a directory: `/nix/store/xlmxn65dw2nvxbpgvfkb
| "resources" -- pkgs.buildEnv error: not a directory: `/nix/store/36ywzl3hqg57ha34
| "temurin-bin" -- pkgs.buildEnv error: not a directory: `/nix/store/6v1bb758sn4fh2
| "temurin-jre-bin" -- pkgs.buildEnv error: not a directory: `/nix/store/kj0hj48hm4
| "semeru-bin" -- pkgs.buildEnv error: not a directory: `/nix/store/chn2lx7gnvd3ay5
| "semeru-jre-bin" -- pkgs.buildEnv error: not a directory: `/nix/store/hbazadpm1x0
| "zulu-ca-jdk" -- pkgs.buildEnv error: not a directory: `/nix/store/z16qj0jgm9vffy
| "graalvm-ce" -- pkgs.buildEnv error: not a directory: `/nix/store/1ywfss0i4rpdiyz
| "pax" -- pkgs.buildEnv error: not a directory: `/nix/store/n0w0kkr796jyiq16kff4cq
| "discord-haskell" -- pkgs.buildEnv error: not a directory: `/nix/store/hrx6j0ld4d
  => none
| _ =>
  if path.endsWith "bsd|source" then
    -- Either pkgs|openbsd|source or pkgs|netbsd|source which both clobber shadow's
    none
  else
    match path.splitOn "|" with
    | .nil => none
    | part :: parts =>
      -- Quote each part because it might contain weird characters
      some <| part ++ ".\"" ++ "\".\"".intercalate parts ++ "\""
```

And now for the real moment of truth! `nixos-rebuild switch`!

```
env: ‘php’: No such file or directory
error: your NixOS configuration path seems to be missing essential files.
To avoid corrupting your current NixOS installation, the activation will abort.

This could be caused by Nix bug: https://github.com/NixOS/nix/issues/13367.
This is the evaluated NixOS configuration path: /nix/store/vqrhc9rg3vahbpxbjyb5ribn
Change the directory to somewhere else (e.g., `cd $HOME`) before trying again.

If you think this is a mistake, you can set the environment variable
NIXOS_REBUILD_I_UNDERSTAND_THE_CONSEQUENCES_PLEASE_BREAK_MY_SYSTEM to 1
```

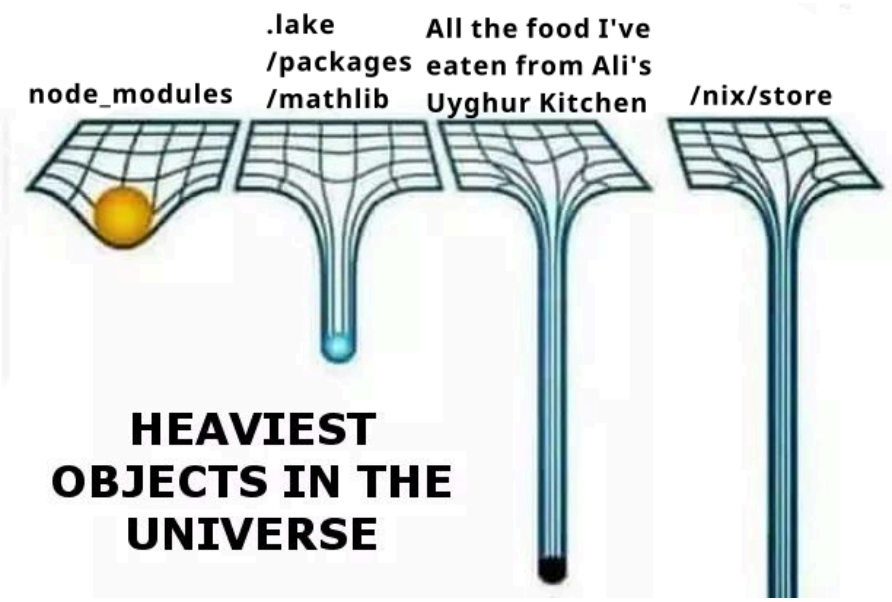
Uh... looks like even `nixos-rebuild` is terrified! Fine then, break my system.
`NIXOS_REBUILD_I_UNDERSTAND_THE_CONSEQUENCES_PLEASE_BREAK_MY_SYSTEM nixos-rebuild switch`!

And as promised, it did in fact break my system by changing my shell from Fish to a Go Fish game, so now I can't even log in!!!

Wait, why am I panicking? This is NixOS! I can just roll back by editing `/sbin/init` to boot a different system generation. So, I booted an older generation, changed my shell in `configuration.nix` to Bash, and after one last rebuild it's finally time to enjoy NixOS at its heaviest!

```
root@nixos
----------
OS: NixOS 26.05 (Yarara) x86_64
Host: MS-7C94 (1.0)
Kernel: Linux 6.17.9-300.fc43.x86_64
Uptime: 2 days, 23 hours, 22 mins
Packages: 69700 (nix-system)
Shell: fish 4.2.1
Display (DELL U2515H): 2560x1440 in 25", 60 Hz [External]
Terminal: xterm-256color
CPU: AMD Ryzen 9 3950X (32) @ 4.76 GHz
GPU: AMD Radeon RX 7900 XTX [Discrete]
Memory: 5.81 GiB / 62.70 GiB (9%)
Swap: 1.43 GiB / 8.00 GiB (18%)
Disk (/): 644.04 GiB / 1.82 TiB (35%) - btrfs
Local IP (enp42s0): 10.187.0.152/21
```

I generated that using `fastfetch --pipe 0 | aha` which of course are already installed. Fastfetch's getNixPackagesImpl actually times out, but in theory it would return 69700 which is a bit of scam since I obviously installed 71440 packages. Either way, it's still more than 5 times the number of packages I installed on Arch!



The Nix store is 1.34 TB with 744954 paths, but fortunately only takes up 690 GB on disk thanks to the magic of btrfs compression. Honestly, the total size of Nixpkgs seems smaller than I expected, and the total network usage of this experiment was about the same as a few people using NixOS for a year.

Surprisingly, the system boots quickly and just feels like a boring (although very obese) NixOS system. There's only a few oddities, such as that `ls` uses the 9base implementation rather than the one from coreutils. I have no idea what I should do with this system now.

Anyways, check out this repo for all my code and more. It should be completely reproducible if you want to try out the crazy `configuration.nix` too (but please don't if you value your sanity and disk space). Lastly, thanks again to exozyme and the other exozyme members for answering all my silly Nix questions!

← Previous: What's the Derivative of a Data Type?

October 31, 2025