NixOS

A Brief Introduction

Arik Grahl

SysEleven

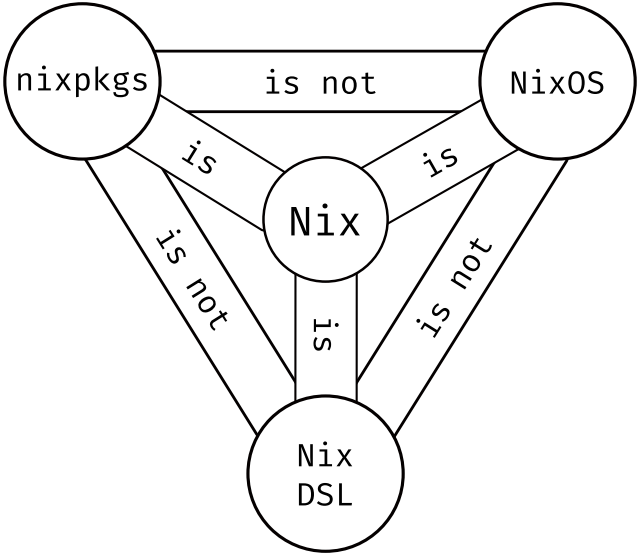
2024-10-17

echo $(whoami)

arik-grahl.de/talks/openstack-europe-10-2024

```
{ pkgs ? import <nixpkgs> {} }:
pkgs.person rec {
  meta = with pronouns; {
    name = { firstName = "Arik"; surname = "Grahl"; };
    pronouns = heHim;
  };
  work = with roles; {
    company = "SysEleven";
    role = softwareEngineer;
  };
  contact = {
    web = "www.arik-grahl.de";
    gitlab = "gitlab.com/arik.grahl";
    github = "github.com/arikgrahl";
    linkedin = "linkedin.com/in/arikgrahl";
    mastodon = "chaos.social/@arikgrahl";
  };
}
```

What is Nix?



Nix DSL (1/3)

- purely functional
  - any valid piece of Nix code is an expression returning a value
  - evaluating a Nix expression
    - yields a data structure
    - does not execute a sequence of operations

```
"foo" # ⇒ "foo"
{ x = "foo"; } # ⇒ { x = "foo"; }
let f = "foo"; in f # ⇒ "foo"
let f = x: x; in f "foo" # ⇒ "foo"
let f = x: x; in f { x = "foo"; } # ⇒ { x = "foo"; }
let f = a: b: "${a} ${b}"; in f "foo" "bar" # ⇒ "foo bar"
let f = { a, b }: "${a} ${b}"; in f { a = "foo"; b = "bar"; } # ⇒ "foo bar"
```

Nix DSL (2/3)

- lazy evaluated
  - expressions are only evaluated if result is requested
  - quick evaluation of large codebases (e.g. Nixpkgs)
  - some errors surface only when respective execution paths are taken

```
let a = {
  x = "foo";
  y = builtins.throw "exception";
}; in a.x                              # ⇒ "foo"

let a = {
  x = "foo";
  y = builtins.throw "exception";
}; in a.y                              # ⇒ error: exception
```

Nix DSL (3/3)

- purpose-built
  - domain specific language (DSL) for the Nix package manager
  - although used otherwise, explicitly not a general-purpose language

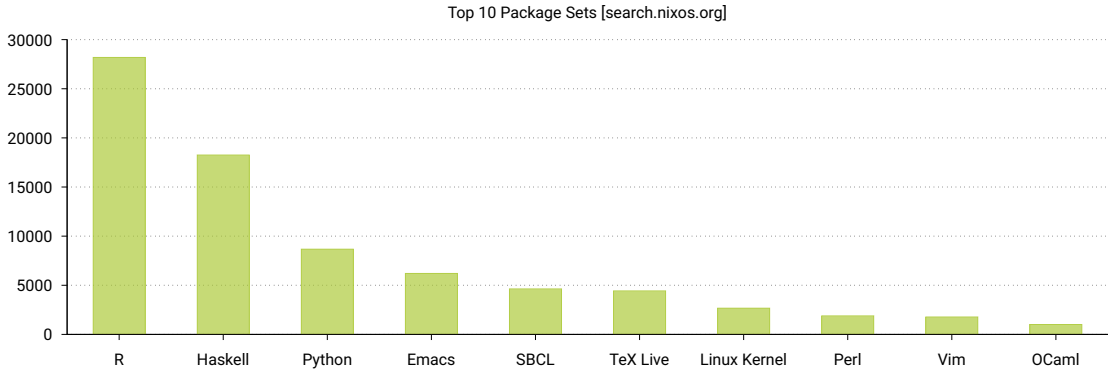
```
let pkgs = import <nixpkgs> {}; in
pkgs.buildGoModule rec {
  pname = "setup-envtest";
  version = "0.18.0";
  src = pkgs.fetchFromGitHub {
    owner = "kubernetes-sigs";
    repo = "controller-runtime";
    rev = "v${version}";
    hash = "sha256-w0AfLit5IZYTfwPYVvIPoWnIy14+4dIDIYLPKoPyOKc=";
  } + "/tools/setup-envtest";
  vendorHash = "sha256-Xr5b/CRz/DMmoc4bvrEyAZcNufLIZOY5OGQ6yw4/W9k=";
  ldflags = [ "-s" "-w" ];
}
```

Nixpkgs (1/3)

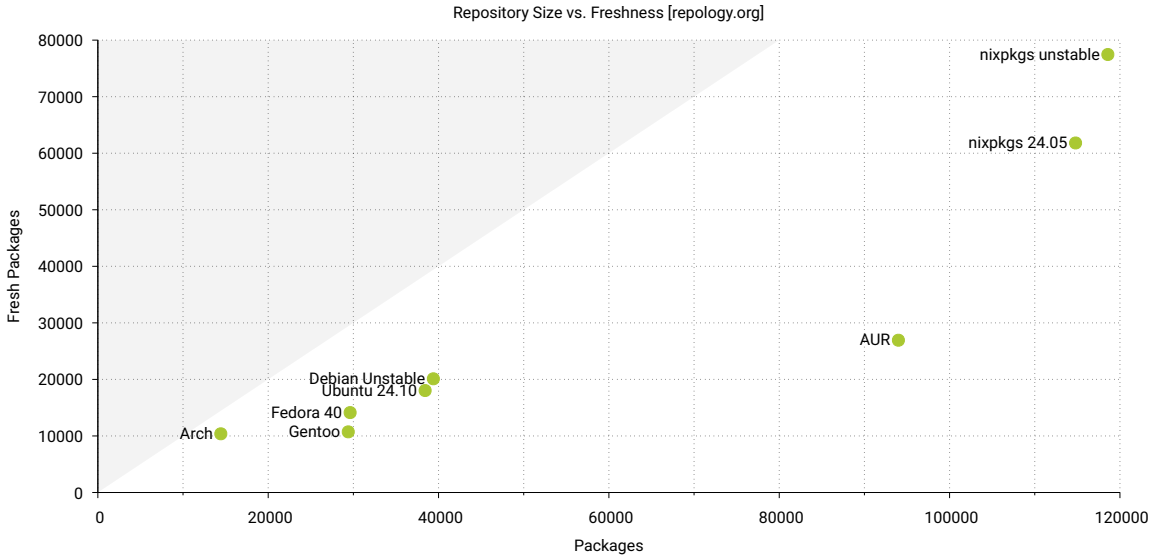
- large mono repository: github.com/NixOS/nixpkgs
  - contains all dependencies for complete build chain
    - `busybox/coreutils`
    - compiler (GCC, LLVM, …)
    - shared libraries (glibc, musl, …)
    - kernel modules
  - reproducible builds because self-contained

Nixpkgs (2/3)

- collection of over 100k packages
  - classic system packages (`rsync`, `vim`, …)
  - language specific packages (JavaScript, Python, Ruby, …)



Nixpkgs (3/3)


Repository Size vs. Freshness [repology.org]
Fresh Packages
Packages
0
10000
20000
30000
40000
50000
60000
70000
80000
0
20000
40000
60000
80000
100000
120000
nixpkgs unstable
nixpkgs 24.05
AUR
Debian Unstable
Ubuntu 24.10
Fedora 40
Gentoo
Arch

NixOS (1/4)

- Linux-based operating system on top of Nixpkgs
  - immutable design
  - atomic update model
  - initially released in 2003
  - architectures: i686, x86-64, AArch64
  - licensed under MIT

NixOS (2/4)

- complete system definition in a single configuration
  - hardware specific (e.g. partition UUIDs, MAC addresses, hostnames)
  - generic (service definition)

```
{ config, pkgs, ... }: {
  imports = [ ./hardware-configuration.nix ];
  boot.loader.systemd-boot.enable = true;
  boot.loader.efi.canTouchEfiVariables = true;
  boot.kernelPackages = pkgs.linuxPackages_latest;
  time.timeZone = "Europe/Berlin";
  environment.systemPackages = with pkgs; [
    vim
  ];
  virtualisation.podman.enable = true;
  system.stateVersion = "24.05";
}
```

NixOS (3/4)

- NixOS options
  - provide declaration of services
  - manage service lifecycle (e.g. `systemctl restart`)

```
services = {
  openssh = {
    enable = true;
    settings.PermitRootLogin = "yes";
  };
  k3s = {
    enable = true;
    role = "server";
  };
};
```

NixOS (4/4)

- generations
  - generated by configuration change or update
    - `nixos-rebuild …`
    - `nixos-rebuild … --upgrade`
  - directly active or as boot target
    - `nixos-rebuild switch`
    - `nixos-rebuild boot`
  - can be rolled back and garbage collected
    - `nixos-rebuild … --rollback`
    - `nix-collect-garbage --delete-old`
    - `nix-collect-garbage --delete-older-than 30d`

Demo: NixOS (1/5)

```
$ uname --kernel-release
6.11.0

$ sudo nixos-rebuild boot --upgrade
unpacking channels...
building Nix...
building the system configuration...

$ nixos-rebuild list-generations
Generation   Build-date           NixOS version              Kernel
119          2024-10-13 12:04:00  24.05.5667.a3f9ad65a0bf    6.11.3
118 current  2024-09-30 00:42:58  24.05.5366.fbca5e745367    6.11.0
```

Demo: NixOS (2/5)

```
$ reboot
```

```
$ nixos-rebuild list-generations
Generation   Build-date           NixOS version             Kernel
119 current  2024-10-13 12:04:00  24.05.5667.a3f9ad65a0bf   6.11.3
118          2024-09-30 00:42:58  24.05.5366.fbca5e745367   6.11.0
```

```
$ uname --kernel-release
6.11.3
```

Demo: NixOS (3/5)

```
$ kubectl get nodes
kubectl: command not found

$ cat /etc/nixos/configuration.nix
…
services.k3s = {
  enable = true;
  role = "server";
};
…
```

Demo: NixOS (4/5)

```
$ sudo nixos-rebuild switch
building Nix ...
building the system configuration ...
activating the configuration ...
setting up /etc ...
the following new units were started: k3s.service
```

```
$ nixos-rebuild list-generations
Generation   Build-date           NixOS version              Kernel
120 current  2024-10-13 12:05:23  24.05.5667.a3f9ad65a0bf  6.11.3
119          2024-10-13 12:04:00  24.05.5667.a3f9ad65a0bf  6.11.3
118          2024-09-30 00:42:58  24.05.5366.fbca5e745367  6.11.0
```

Demo: NixOS (5/5)

```
$ kubectl get nodes
NAME      STATUS   ROLES                         AGE   VERSION
nb-arik   Ready    control-plane,etcd,master   13s   v1.31.0+k3s1
```

How does Nix work? (1/4)

Fundamental theorem of software engineering (FTSE)

All problems in computer science can be solved by another level of indirection [...].

— David John Wheeler

How does Nix work? (2/4)

```
$ which rsync
/run/current-system/sw/bin/rsync

$ ls -l /run/current-system/sw/bin/rsync
/run/current-system/sw/bin/rsync → /nix/store/n1d…i5y-rsync-3.3.0/bin/rsync

$ ls -l /run/current-system/sw
/run/current-system/sw → /nix/store/iwl…x2d-system-path

$ ls -l /run/current-system
/run/current-system → /nix/store/qs5…asr-nixos-system-nb-arik-24.05.5667.a3f9ad65a0bf
```

How does Nix work? (3/4)

```
$ ldd $(which rsync)
libacl.so.1 ⇒ /nix/store/7px…0kb-acl-2.3.2/lib/libacl.so.1
libz.so.1 ⇒ /nix/store/pkl…zgc-zlib-1.3.1/lib/libz.so.1
libpopt.so.0 ⇒ /nix/store/awg…0s8-popt-1.19/lib/libpopt.so.0
liblz4.so.1 ⇒ /nix/store/1ac…zb5-lz4-1.9.4/lib/liblz4.so.1
libzstd.so.1 ⇒ /nix/store/80p…1y6-zstd-1.5.6/lib/libzstd.so.1
libxxhash.so.0 ⇒ /nix/store/cmn…zl5-xxHash-0.8.2/lib/libxxhash.so.0
libcrypto.so.3 ⇒ /nix/store/0d6…23d-openssl-3.0.14/lib/libcrypto.so.3
libc.so.6 ⇒ /nix/store/c10…4hk-glibc-2.39-52/lib/libc.so.6
libattr.so.1 ⇒ /nix/store/dwk…hwa-attr-2.5.2/lib/libattr.so.1
libdl.so.2 ⇒ /nix/store/c10…4hk-glibc-2.39-52/lib/libdl.so.2
libpthread.so.0 ⇒ /nix/store/c10…4hk-glibc-2.39-52/lib/libpthread.so.0
```

How does Nix work? (4/4)

Fundamental theorem of software engineering (FTSE)

All problems in computer science can be solved by another level of indirection,

except for the problem of too many layers of indirection.

Container

- traditional Docker build
  - is not deterministic
  - usually relies on package managers inside container to compose packages
  - provides poor layering and caching for dependencies

Demo: Container (1/3)

```go
package main

import (
    "io"
    "net/http"
)

func main() {
    client := &http.Client{}
    req, _ := http.NewRequest("GET", "https://wttr.in/Berlin", nil)
    req.Header.Set("User-Agent", "curl")

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        resp, _ := client.Do(req)
        io.Copy(w, resp.Body)
    })

    http.ListenAndServe("0.0.0.0:8080", nil)
}
```

Demo: Container (2/3)

```
{ pkgs ? import <nixpkgs> {} }:
let
  version = "1.0.0";
  bin = pkgs.buildGoModule {
    pname = "weather";
    inherit version;
    src = ./.;
    vendorHash = null;
  };
in
pkgs.dockerTools.buildLayeredImage {
  name = "registry.example.com/weather";
  tag = "v${version}";
  contents = with pkgs; [ cacert ];
  config.Entrypoint = [ "${bin}/bin/weather" ];
}
```

Demo: Container (3/3)

```
$ nix build -f weather.nix

$ ls -l ./result
result → /nix/store/g2b7rhzzzqjxwq567vjxr7kn21hhgaq7-weather.tar.gz

$ podman load < ./result
Loaded image: registry.example.com/weather:v1.0.0

$ podman run -d -p 8080:8080 registry.example.com/weather:v1.0.0
6c81bc9af3158c76186112810e38de0ec8ebdd2fab489ae83a89c55b76918964

$ curl -s http://localhost:8080 | head -n 7
Weather report: Berlin

      \  /       Partly cloudy
    _ /"".-.     +11(8) °C
      \_(   ).   → 32 km/h
      /(___(__)  10 km
                 0.1 mm
```

Conclusion

- Nix is a powerful language, satisfying various use-cases
- Nixpkgs is a large and attractive ecosystem
- NixOS is a production-ready and flexible operating system
- building containers with Nix
  - offer desired Nix attributes
  - not breaking existing tools

Further Reading

- Nix Pills: nixos.org/guides/nix-pills
- Nix Reference Manual: nix.dev/manual/nix
- NixOS Manual: nixos.org/manual/nixos
- Nix Packages: search.nixos.org/packages
- NixOS Options: search.nixos.org/options