Nix - Zero to full cloud deployment in 30 minutes

Adam Hose

August 9, 2018

Outline

Who am I?

- Likes programming in Nix, Python, Go, Rust, Lisp
- Loves UNIX
- I'm a señor developer at Enuma Technologies
- First used Nix back in 2015
- NixOS member since 2017

Nix - what is it?

- A package manager
- A build system
- A language

What do we expect from package managers?

- Manages software builds
  - Build manifests
- Manages package repositories
  - Debian: universe/multiverse/non-free
  - Npm: Registry
- Create redistributable packages
- Dependency management
- Upgrades/downgrades

Problems with traditional package management

- Underspecified dependencies
  - E.g. Program `x` actually depends on `y` but not in the manifest
- Rolling back?
  - No atomicity
- Major upgrades
  - Often breaks entirely
  - Broken ABIs
- Packages are either installed or uninstalled
  - Need another package? You need to alter system state.
- Docker as a development environment
  - Need another tool in your Docker container? Time for a break...
- Ultimate trust
  - `nix-build --check`

Nixpkgs - The packages

- Available on Github `https://github.com/nixos/nixpkgs`
- Huge package tree

More packages than Debian/Ubuntu/Arch

- Very up to date

Packages are ~85% up to date

- Mostly free software
- Accepts unfree packages (but must be user enabled)
- Pull request based workflow on Github
  - Around 200-300 monthly contributors
  - Last week we saw ~600 commits from 140 authors
- Some fully autogenerated ecosystems
  - Emacs
  - Haskell
  - NodeJS
- Comes with lots of abstractions
  - Language specific
  - Source fetchers
  - Library functionality

How nix deals with these issues

- No unspecified dependencies
  - Dependency not in inputs? Not available at build time.
- Immutable package store
  - No more in-place upgrades
- Atomic installs/uninstalls/upgrades/downgrades
  - Using symlinks
- Pure package builds

Nix - An introduction

- Reproducible deterministic builds
  - Easier to debug
  - No more "works on my machine"
- Packages built in isolation (sandboxed)
  - Only specified inputs are available
- All inputs are hashed
  - If any input changes it is considered to be a distinct evaluation
- All outputs are stored by hash

`/nix/store/<hash>-packagename-version/`

- Source based with binary cache
- Unprivileged installs
- Both Linux (`x86_64` / `aarch64`) and `OSX` are fully supported

Nix - the language

- Purely functional
  - Always returns the same answer given the same inputs
  - Evaluation has no side effects
- Lazy eval - Like haskell!
  - A good fit for package trees where you want to go from a few leafs (user installed packages) to many dependencies
- Untyped - With a few exceptions
  - paths, urls, bool, int, lists, functions and attrsets

Nix - the language

- Hello world

```
let
  name = "Codeaholics";
"Hello ${name}"
```

- Functions

```
let
  fn = (a: b: a + b);
in fn 5 5
```

- Expressions

```
let
  x = if x > 5 then x else throw "x is too small";
in x 5
```

Nix - the language

- Attribue sets (maps)

```
{
  foo="bar";
}
```

- Lists

```
[ "foo" "bar" ]
```

- Currying (partial application)

```
let
  fn = (a: b: a * b);
  mul5 = fn 5;
in mul5 5;
```

Nix - The build system

- Clear separation between build time and runtime
  - No need for complicated multi-stage builds
- Each package is composed of a `derivation`
  - A derivation is the package description
  - Lists all input `derivations` (packages)
  - A `derivation` can depend on one or more `outputs`
- One build results in one or more `outputs`
  - `dev`
  - `man`
  - `bin`

Nix - The build process

`/nix/store/` 2i4vyzq4i9j7l8d2g3fdal97h4mi5sy3-openssh `-7.7p1/`

- The OpenSSH derivation + all of it's input are instantiated
- A hash is calculated over the instantiated derivation
- A nix build environment (sandbox) is created for the package
- Each build `phase` from the derivation runs. `unpackPhase`, `patchPhase`, `buildPhase`, `installPhase`, etc
- All binaries are patched
  - Shared libraries point to absolute store path
  - Shebangs are patched
- Package is being written to the nix store

Nix - installing packages

- Install a package into your user profile

```
nix-env -iA nixpkgs.hello
```

- Global package installs

```
# /etc/nixos/configuration.nix
environment.systemPackages = [
  pkgs.hello
];
```

Nix - magical superpowers

- Start a new shell with a package

```
nix-shell -p hello
```

- Magical superpowers

```
nix-shell -p 'python3.withPackages(ps: with ps; [
  ipython tensorflow numpy requests
])' --run ipython
```

Nix - magical superpowers

- Self-documenting scripts

```
#!/usr/bin/env nix-shell
#! nix-shell -i python3 -p python3 python3Packages.requests
import requests

if __name__ == '__main__':
    print(requests.get('https://codeaholics.io/'))
```

Nix - magical superpowers

- Overrides are a breeze

```
somePackage.overrideAttrs(oldAttrs: {
  name = "overriden-${oldAttrs.version}";

  buildInputs = oldAttrs.buildInputs ++ [ pkgs.firefox ];

  patches = [ (fetchpatch {
    url = "https://github.com/path/to.patch";
    sha256 = "1n1x1f7xgci7wqm0xjbxxlxxd1kq3866a3xnv7dfz2512z
  }) ];
})
```

- Cross compilation at your finger tips

Live demo

- Goals
  - Making a declarative environment
  - Pinning nixpkgs
  - Writing our project
  - Writing a derivation Add missing inputs
  - Build a Docker container
- Tooling
  - direnv

Demo time!

- Live coding a service

NixOS - Fully declarative operating system

- A module system built on top of Nix
- Fully declarative service management
- Atomic upgrades and rollbacks
- Composable reusable systems

NixOS - An example configuration

```
{ config, pkgs, ... }:

{
  imports = [ ./hardware-configuration.nix ];

  boot.loader.systemd-boot.enable = true;
  boot.loader.efi.canTouchEfiVariables = true;

  services.redis.enable = true;
  services.redis.port = 9001;

  users.extraUsers.adisbladis = {
    uid = 1000;
    isNormalUser = true;
    extraGroups = [ "wheel" ];
  };
}
```

Nixops

- Fully declarative cloud deployment tool
- Deploys to AWS/Azure/GCP/hetzner/bare-metal
- Replaces ansible/salt/puppet/chef

Nixops - deploying a service

- Example config

```
{ timeserver =
    { resources, ... }:
    {
      deployment.targetEnv = "ec2";
      deployment.ec2.region = "us-west-2";
      deployment.ec2.instanceType = "t2.small";
      deployment.ec2.keyPair = "my-aws-keypair";

      systemd.services.geth = {
description = "Time as a service";
serviceConfig.Type = "simple";
serviceConfig.Restart = "on-failure";
serviceConfig.User = "nobody";
wantedBy = [ "multi-user.target" ];
after = [ "network.target" ];
      };};}
```

Production users

- EU
  - 3 TOP 500 computers!
- Intel
- Target
- Pinterest
- Zalora
- Packet.net

Takeaways

- Understand how and why Nix isolation works
- Have a rough idea of the Nix ecosystem
- Know how to start using Nix
- Enlightenment - hopefully

Questions?