A Gentle Introduction to Nix

Sjur Millidahl

TechZone 22

nix

What is Nix

- Operating system (NixOS)
- Language (functional, declarative)
- Package manager

Pure functions

```
     |---------|
a -> | machine | -> b
     |---------|
```

- A mapping of a’s to b’s
- Every a always results in the same b
- Morally equal to a HashMap a b

Building software

```
            |-----|
simple.c -> | gcc | -> .exe
            |-----|
```

- The machine has gcc “inside it”
- Altering gcc in the machine alters the build function
- Hence impure build

Nix

```
     gcc -> |---------|
            | machine | -> .exe
simple.c -> |---------|
```

- gcc is now an argument
- The build function is pure again
- Hence the build is pure

What do you mean gcc is an argument

- gcc is called a derivation here
- A derivation is a tool or lib that is built with nix

- Can be found in a repo called nixpkgs
- Can be built from source

How to Nixify a build

- Express tools and libs as arguments
- Such arguments must be derivations
- Either find derivation in nixpkgs
- OR transform it to a derivation yourself

Language

- Purely functional
- Declarative
- Lazy
- Dynamically typed
- Weird at first

Language example

example.nix

```nix
let
  increase = x: x + 1;
  myList = [ (increase 2) "world" false ];

in { result = "Hello ${builtins.elemAt myList 1}"; }
```

bash

```bash
> nix eval -f example.nix
```

output

```
{ result = "Hello world"; }
```

---

nixpkgs

- A collection of derivations and util-functions
- Can be used in nix-expressions
- Gives a specific version of e.g. gcc
- https://search.nixos.org/packages

Back to the pure build

simple.c

```c
void main() {
  puts("Simple!");
}
```

build.sh

```
> gcc -o simple simple.c
```

The nix way

```
> $gcc -o simple $src
```

.nix example

build.nix

```
let
  nixpkgs = (import (builtins.fetchTarball {
    url =
      "https://github.com/NixOS/nixpkgs/archive/d1c3fea7ecbed758168787fe
    sha256 = "sha256:0ykm15a690v8lcqf2j899za3j6hak1rm3xixdxsx33nz7n3swsy
  })) { };

  pureBuildFunction = pkgs : src : system :
    with pkgs;
     derivation {
      name = "simple";
      builder = "${bash}/bin/bash";
      args = [ ./builder.sh ];
      inherit src system gcc coreutils;
    };

in pureBuildFunction nixpkgs ./simple.c "x86_64-darwin"
```

---

build-script

builder.sh

```
export PATH="$coreutils/bin:$gcc/bin"
mkdir $out
gcc -o $out/simple $src
```

Let Nix build our code

```
> nix-build build.nix
/nix/store/a22p8f72pghn22w168a72pisicnncmmh-simple

> /nix/store/a22p8f72pghn22w168a72pisicnncmmh-simple/simple
Simple!
```

The Nix shell

Allows any nixified package to be brought into scope

```
> nix-shell --pure -p python2 python3
```

Will put me in a shell with python2 and python3

```
nix-shell> python2 --version
Python 2.7.18

nix-shell> python3 --version
Python 3.10.6
```

Upsides

- reproducible builds
- does not alter your entire system (only your nix-store)
- you can have every version of python available without conflicts
- efficient caching
- build small, reproducible docker-images
- easily override e.g. gcc with an unmerged PR

Downsites

- language can be weird
- long build times from empty caches
- docs can be.. sparse
- disk use can be.. significant

No time to talk about

- Nix flakes
- NixOS

Thanks!

Talks: A gentle introduction to Nix

@SjurMillidahl

smillida@cisco.com
