

# A Gentle Introduction to Nix

Sjur Millidahl

## What is Nix

- language (functional)
  - operating system (NixOS)
  - 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
  - derivations can be built by fetching source and building it
  - fetchers can fetch a source from .tar.gz or GitHub
  - nixpkgs has recipes for **derivations** of a lot of tools
- 

## Language

- Purely functional
  - Dynamically typed
  - Weird at first
- 

## Language example

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

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

---

## nixpkgs

- A collection of derivations and functions
  - Can be used in nix-expressions
  - Gives a specific version of e.g. gcc
- 

## Back to the pure build

simple.c

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

build.sh

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

---

## An excellent suggestion, sir, with just two minor drawbacks



---

## 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/a22p8f72pghn22w168a72piscnncmmh-simple

> /nix/store/a22p8f72pghn22w168a72piscnncmmh-simple/simple
Simple!
```

---

# The nix shell

Allows any nixified package to be brought into the shell

```
> nix-shell -p python2 python3 -I nixpkgs=https://github.com/NixOS/nixpk
```

Will put me in a shell with python2 and python3

```
nix-shell> python2 --help
```

```
nix-shell> python3 --help
```

---

## 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!

@SjurMillidahl smillida@cisco.com

---