Nix party tricks

Alexander Flurie

<2022-07-12 Tue>

Outline

Preamble

Extremely abbreviated intro to nix

Demo Overview

First party trick: nix for managing development environments

Second party trick: nix for managing ec2s

Third party trick: nix for managing lambda runtimes

Preamble

Nix is magic

Surprise, this is a magic show!

Nix is a special kind of magic for specifying pretty much any output you could want.

Nothing up my sleeve

Behold, a fresh VM.

Extremely abbreviated intro to nix

Nix the First: Language

Main features:

- functional
- dynamic
- lazy
- base language is tiny
- Haskell influence (though much divergence since)

Nix the First: Language (con't)

Quirky type system:

- strings have native multiline support
- URIs
- paths (relative and absolute)
- no advanced objects, everything is a set (map)
- first-class functions

Nix the Second: Package Manager

nixpkgs

- Fundamental unit: the derivation
- Built with and extends Nix language
- Largest, most active package repository of its kind
- Many smaller ecosystems, especially by language (2nix)

Nix the Second: Package Manager (con't)

The Dirty Secret:

Nix the Third: Linux Distribution

NixOS

- Built on top of nixpkgs and systemd
- Familiar to users of gentoo and arch
- Adds in modules for system-level configurability

Nix the Fourth: *misc* tooling

- home-manager (nix for $HOME)
- nix-darwin (nix for macOS)
- cachix (arbitrary caching for nix derivations)
- Hercules CI (CI/CD for nix derivations)

Demo Overview

Purpose

- survey of a bunch of common problems and demonstrate solutions with nix
- whirlwind tour of some great nix ecosystem tooling
- code is public

Let's install nix!

- Go to `https://nixos.org`
- select Download
- Follow multi-user installation instructions (unless you're on something weird like WSL)

First party trick: nix for managing development environments

Misc tools for environment management

- direnv: automate environment switching in shell
- devshell: manage all your development tools per-project with a simple configuration file

Let's install direnv!

- Go to `https://direnv.net/#basic-installation`
- Follow the NixOS instructions (because I'm not installing Homebrew, boo!) for non-NixOS systems
- Hook direnv into shell

Oops, we need git, too

We *could* install git the usual way on macOS... (by installing the Xcode command line tools) ... but what if we didn't have to? nixpkgs to the rescue! And this time we don't even need to "install" it!

```
nix-shell -p git
```

Let's grab the code

```
git clone
https://github.com/flurie/nix-party-tricks.git
```

. . . and then let the magic take hold

direnv holds a lot of power, so be careful with what you allow.

Using nix with direnv provides an additional level of security.

Time to take the ride.

```
direnv allow
```

Tour our new powers

Figure 1: I'm in devshell! I'm in normal shell!

Enter AWS with train

Set the stage for more magic

```
cp -r "$PRJ_ROOT"/support/.aws ~/.aws
```

Create some new creds and never have to look at them!

```
Log in to AWS
```

```
Create new programmatic IAM credentials
```

```
Download the csv to our devshell root
```

Time to test the thing out

```
aws sts get-caller-identity
```

Second party trick: nix for managing ec2s

Preamble: terraform to stand up the host

```
# $PRJ_ROOT/terraform/ec2
terraform init
terraform apply
```

Misc tools for deployment management

- cachix (arbitrary caching for nix derivations)
- deploy-rs (deploy NixOS to anywhere from anywhere)

NixOS on AWS three ways

#1: ec2 user data

NixOS on AWS three ways - #1

```
# main.tf
resource "aws_instance" "nixos" {

  # ...some parts omitted

  root_block_device {
    # need this to be big enough to build things
    volume_size = 20
  }

  user_data = <<END
### https://nixos.org/channels/nixos-22.05 nixos

{ config, pkgs, modulesPath, ... }:
{
  # nix uses same string interpolation as terraform, so we must escape it here
  imports = [ "$${modulesPath}/virtualisation/amazon-image.nix" ];
  ec2.hvm = true;
  system.stateVersion = "22.05";
  environment.systemPackages = with pkgs; [ nix-direnv direnv git ];
  networking.hostName = "nixos-aws";
}
END
}
```

NixOS on AWS three ways - #1

We can now enter the machine.

Make sure to use the IP given by terraform.

```
ssh -i /tmp/nixos-ssh.pem root@{IP}
```

NixOS on AWS three ways - #1

Let's pull down the party tricks repo here...

```
git clone
https://github.com/flurie/nix-party-tricks.git
```

NixOS on AWS three ways - #1

. . . and activate the devshell!

```
cd nix-party-tricks && direnv allow
```

NixOS on AWS three ways

#2: deploy-rs

NixOS on AWS three ways - #2

```
deploy = {
  nodes = {
    "aws" = {
      sshUser = "root";
      sshOpts = [ "-i" "/tmp/nixos-ssh.pem" ];
      hostname = "nixos-aws";
      profiles.hello = {
        path = deploy-rs.lib.x86_64-linux.activate.custom
          nixpkgs.legacyPackages.x86_64-linux.hello "./bin/hello";
      };
      profiles.system = {
        path = deploy-rs.lib.x86_64-linux.activate.nixos
          self.nixosConfigurations.aws;
      };
    };
  };
};
```

NixOS on AWS three ways - #2

Deploying to hostname, make sure it's in our /etc/hosts

```
sudo echo "{IP}  nixos-aws" >> /etc/hosts
```

NixOS on AWS three ways - #2

First deploy: “hello world”

```
deploy .#aws.hello
```

NixOS on AWS three ways - #2

Second deploy: NixOS system running nginx

```
{
  services.nginx = { enable = true; };
  networking.firewall.allowedTCPPorts = [ 80 ];
}
```

Let's deploy!

```
deploy .#aws.system
```

NixOS on AWS three ways - #2

Now we should get the nginx splash page in a browser

`http://nixos-aws`

Third party trick: nix for managing lambda runtimes

Preamble: more terraform for my lambda

Introducing a great nix feature: remote builders