Michael Maclean

Using private flakes in NixOS

THURSDAY, JUNE 06, 2024

I've been using Nix and NixOS for a while now, but I'd got into a way of working where it was doing what I needed to do, so I haven't felt the need to modernise it with the new features that have come along since I started using it. On the other hand, I did want to use some code that I'm not ready to open source yet and I wanted to find a neat way to do that. It's likely that none of this post will be new or interesting to a lot of people who use Nix and NixOS, but I haven't seen it written down anywhere in exactly this way.

Until now, to make this code work, I had been using a pretty awful and in no way hermetic solution with a local git checkout and some overlays. For some time I had imagined there must be a better way to do this using Flakes. I had searched around a couple of times to figure out how this might work but I never found a way to meet two preferences I had:

1. The code should be pulled from GitHub from a private repository without having to be proxied somehow
2. I shouldn't have to have any credentials for GitHub stored on the box in the clear

While I was at EMF, I dropped into the nix.camp tent a couple of times, and finally managed to work it out after a couple of pointers from Matt. I haven't really used Flakes an awful lot so it took a couple of tries.

Key changes

There were three key points:

Don't use /etc/nixos any more

The configuration for my machine was still stored in `/etc/nixos`, which was how I'd installed it in the first place. With Flakes, you can keep the configuration anywhere on the filesystem. With the config in my home directory, the rebuild command becomes:

```
sudo nixos-rebuild switch --flake .#
```

Don't use sudo with nixos-rebuild

The next problem was how to use the `ssh` credentials I have as my normal user, without having to copy them to root. It turns out this is also fairly simple, though the name of the option is a little non-obvious given my use of it – `--use-remote-sudo`. By using this, you don't need the whole command to be run with `sudo`, and instead it will ask for the password after it's completed the work in the Nix store when it needs to apply the changes to the system. With this, the command is:

```
sudo nixos-rebuild switch --flake .# --use-remote-sudo
```

Use SSH agent forwarding

I'm not normally keen on this feature of SSH, and I don't have it switched on permanently, but in this particular case it works for me. For this to work I need to have `ssh-agent` running locally, I need to use `ssh -A` to connect, and then everything should work. If it doesn't work for some reason I get some errors when it's retrieving the flakes. This saves me having to keep something like a GitHub access token or deploy key on the machine.

The service's flake

In my service's repo, my `flake.nix` looks like this (all my services are written in Rust):

```
{
  inputs = {
    naersk.url = "github:nix-community/naersk/master";
    nixpkgs.url = "github:NixOS/nixpkgs/nixpkgs-unstable";
    utils.url = "github:numtide/flake-utils";
  };

  outputs = { self, nixpkgs, utils, naersk }:
    utils.lib.eachDefaultSystem (system:
      let
        pkgs = import nixpkgs { inherit system; };
        naersk-lib = pkgs.callPackage naersk { };
      in
      {
        packages.default = naersk-lib.buildPackage ./.;
        devShell = with pkgs; mkShell {
          buildInputs = [ cargo cargo-edit rustc rustfmt pre-commit rustPackages.
          RUST_SRC_PATH = rustPlatform.rustLibSrc;
        };
      }
    );
}
```

System configuration

First off, I added a very basic `flake.nix` inspired by the documentation. It includes my own package, and then imports the existing `configuration.nix`.

```
{
  description = "Flake-based NixOS configuration";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.05";
    myPackage.url = "git+ssh://git@github.com/mgdm/myPackage.git";
  };

  outputs = { self, nixpkgs, ... }@inputs: {
    nixosConfigurations.metis = nixpkgs.lib.nixosSystem {
      system = "x86_64-linux";
      specialArgs = { inherit inputs; };
      modules = [
        ./configuration.nix
      ];
    };
  };
}
```

And then in my `configuration.nix`, I made a few changes.

First off, I added inputs at the top:

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

To make referring to the package involve a little less typing, I gave it a shorter name:

```
let
  myPackage = inputs.myPackage.packages."${pkgs.system}".default;
in
{
  nix = {
    extraOptions = "experimental-features = nix-command flakes";
  };

  # ...
```

Then later I can reference that in `environment.systemPackages`:

```
{
  environment.systemPackages = with pkgs; [ vim ripgrep git tmux myPackage ];
}
```

I have a `systemd` service configured for this in `configuration.nix`, but my next move is to investigate how to provide a module in the service's `flake.nix` and move that configuration there. Eventually I'll look into building this remotely so I can keep the configuration on my local computer.

If there are better ways to do any of this, I'd be happy to hear about them.

Have something to say? Give me a shout on Bluesky, Fediverse, or drop me an email.