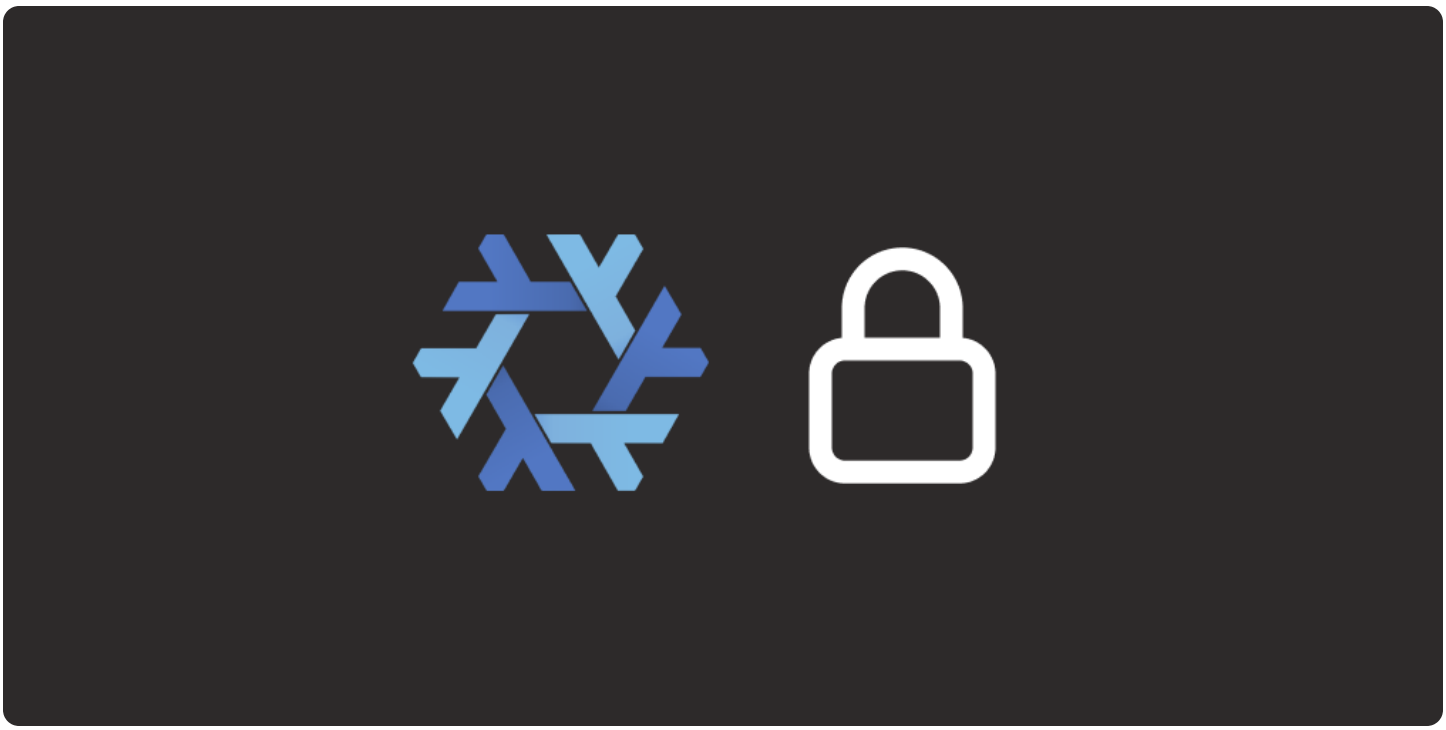
Manage Secrets in NixOS

October 27, 2022 · 6 min · sekun | Suggest Changes



How to avoid hardcoding secrets in Nix

Table of Contents

Recently, I experimented with running NixOS on a DigitalOcean droplet (which I will probably write about in the future), and migrated some of my toy projects from App Platform. During the migration process, I realized that I would have to somehow

handle the DB certificate, and other sensitive credentials. I can't just hardcode these!

One of the more popular projects for problems like this is agenix. Their README for how to use it was a bit confusing (for me) so hopefully this post will be of use to others.

There are two parts to this post: 1) creating secrets with `agenix`, and 2) reading said secrets in a remote NixOS server. Of course you could use it for different things, and my example isn't exactly the simplest.

> **Note**: I'm not saying this is the best way to do things. Please let me know if there are any glaring issues you find!

Prerequisites

Here's a checklist for what's needed:

- `agenix` CLI present in your computer
- `agenix`'s module to the system you want secrets to be read in
- An SSH key pair present in the machine you'll create the secrets in as well as in the machine you want the secrets to be read in

Installing `agenix` is documented well in their README so feel free to consult it.

About `agenix`

Age files

Secrets are encrypted and stored by `agenix` in these things called *age files*. These files have the `.age` format which is created by the `agenix` CLI. For you to create an age file, the CLI looks for a `secrets.nix` file in the current directory for the *rules* to determine who is allowed to decrypt it.

So, what are these rules?

Rules

Let's see what the `agenix` CLI says:

```
$ agenix
agenix - edit and rekey age secret files

agenix -e FILE [-i PRIVATE_KEY]
agenix -r [-i PRIVATE_KEY]

options:
-h, --help                show help
-e, --edit FILE           edits FILE using $EDITOR
-r, --rekey               re-encrypts all secrets with specified recipients
-i, --identity            identity to use when decrypting
-v, --verbose             verbose output

FILE an age-encrypted file

PRIVATE_KEY a path to a private SSH key used to decrypt file

EDITOR environment variable of editor to use when editing FILE

RULES environment variable with path to Nix file specifying recipient public keys.
Defaults to './secrets.nix'

agenix version: 0.13.0
age binary path: /nix/store/kfasn0129ac0xn8wfvf7mq38rxhbc725-rage-0.8.1/bin/rage
age version: rage 0.8.1
```

Specifically:

> RULES environment variable with path to Nix file specifying recipient public keys. Defaults to './secrets.nix'

In the aforementioned file, we're able to say whose key can decrypt what age file. Let's create this file first.

```
# We'll store the rules, and age files in the `secrets` folder
mkdir secrets

touch secrets.nix
```

For the example later, I'll need two files: one containing the DB CA certificate, and the DB password. So I'll create a rule for each age file I'm going to generate later on.

Here's how it looks like:

```
# secrets/secrets.nix

let
  peepeepoopoo = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIB/Oxx/jZS7TRqjp2kwaYavzcxxKFT
in {
  "emojiedDBPassword.age".publicKeys = [ peepeepoopoo ];
  "emojiedDBCACert.age".publicKeys = [ peepeepoopoo ];
}
```

When decrypting/creating `emojiedDBPassword.age` for example, `agenix` looks for the private key pair of the public key that was supplied. Otherwise, it prohibits the user from doing so, and will complain about there not being any matching keys.

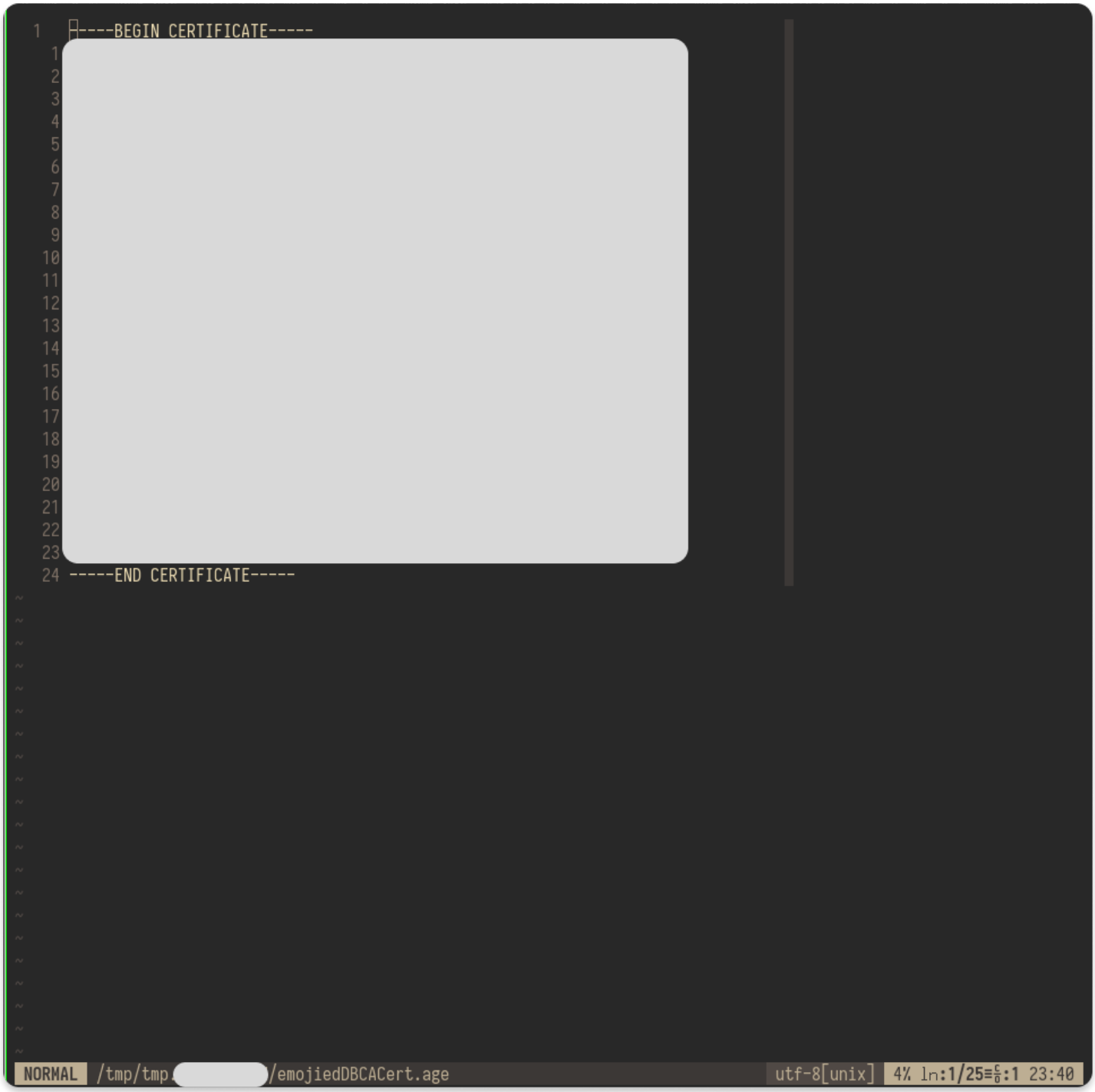
> **Note**: You don't need the rules file for decrypting an age file because the permissions are already encoded in the age file. You should only have it present if you're using the CLI for creating/updating an age file.

Creating a secret

We'll need the `agenix` CLI to create an age file containing our encrypted secret, and run this:

```
$ agenix -e emojiedDBCACert.age
```

This opens a text editor for you to put the secret in.



Updating a secret's public keys

If you need to add public keys for existing age files, update the `secrets.nix` file accordingly, and run `agenix -r`. Make sure you're in the same directory as the `secrets.nix`, and age files.

It would show you something like this if it succeeds:

```
sekun@ichi /s/S/d/secrets (main)> agenix -r
rekeying emojiedDBCACert.age...
rekeying emojiedDBPassword.age...
```

Now our age files are ready to be used!

Reading secrets in Nix

I mentioned earlier that my use case is for supplying the DB CA certificate, and DB password to the `emojied` app for it to connect with the DB properly. I won't get into the details of how I set up the server. Rather, I'll just include the parts necessary. If you wish to read the full config files, check the repo although `peepeepoopoo` here is a throwaway server. Hence not in the repo.

What I need in the remote server called `peepeepoopoo` (where `emojied` is running) are the ff:

- `agenix` NixOS module
- Age files
- SSH key permitted to decrypt said age files

`flake.nix` :

```
{
  description = "Example";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-22.05";
    emojiedpkg.url = "github:sekunho/emojied";
    agenix.url = "github:ryantm/agenix";
  };

  outputs = {
    self,
    nixpkgs,
    emojiedpkg,
    agenix
  }:
    let
      system = "x86_64-linux";
      pkgs = nixpkgs.legacyPackages.${system};
      emojied = emojiedpkg.packages.${system}.emojied;
    in {
      nixosConfigurations.peepeepoopoo = nixpkgs.lib.nixosSystem {
        inherit system;
```

```
        modules = [
          emojiedpkg.nixosModule
          ./hosts/peepeepoopoo/configuration.nix
          agenix.nixosModules.age
        ];

        # Applies the configuration.nix function to these arguments
        specialArgs = {
          inherit pkgs;
          inherit emojied;
        };
      };
    };
}
```

All that's left to do is specify where the age files are located, and referencing the decrypted age files' paths.

`hosts/peepeepoopoo/configuration.nix` :

```
{ modulesPath, lib, config, pkgs, ... }: {
  imports = lib.optional (builtins.pathExists ./do-userdata.nix) ./do-userdata.nix +
    (modulesPath + "/virtualisation/digital-ocean-config.nix")
  ];

  programs.ssh = {
    startAgent = true;

    extraConfig = ''
      AddKeysToAgent yes
    '';
  };

  nix = {
    package = pkgs.nixVersions.nix_2_9;

    extraOptions = ''
      experimental-features = nix-command flakes
    '';

    settings.trusted-public-keys = [
      "hydra.iohk.io:f/Ea+s+dFdN+3Y/G+FDgSq+a5NEWhJGzdjvKNGv0/EQ="
      "iohk.cachix.org-1:DpRUyj7h7V830dp/i6Nti+NEO2/nhblbov/8MW7Rqoo="
      "nix-community.cachix.org-1:mB9FSh9qf2dCimDSUo8Zy7bkq5CX+/rkCWyvRCYg3Fs="
    ];

    settings.substituters = [
      "https://cache.iog.io"
```

```
    "https://iohk.cachix.org"
    "https://nix-community.cachix.org"
  ];
};

age = {
  # We're letting `agenix` know where the locations of the age files will be
  # in the server.
  secrets = {
    emojiedDBPassword.file = "/root/secrets/emojiedDBPassword.age";
    emojiedDBCACert.file = "/root/secrets/emojiedDBCACert.age";
  };

  # Private key of the SSH key pair. This is the other pair of what was supplied
  # in `secrets.nix`.
  #
  # This tells `agenix` where to look for the private key.
  identityPaths = [ "/root/.ssh/id_ed25519" ];
};

# List services that you want to enable:
services = {
  emojied = {
    enable = true;
    port = "3000";
    dbHost = "<REPLACE_WITH_YOUR_OWN>";
    dbName = "<REPLACE_WITH_YOUR_OWN>";
    dbPort = "<REPLACE_WITH_YOUR_OWN>";
    dbUser = "<REPLACE_WITH_YOUR_OWN>";
    dbPoolSize = "5";
    dbPasswordFile = config.age.secrets.emojiedDBPassword.path;
    dbCACertFile = config.age.secrets.emojiedDBCACert.path;
  };

  openssh = {
    enable = true;
    permitRootLogin = "prohibit-password";
    passwordAuthentication = false;
  };
};

networking = {
  firewall = {
    enable = true;
    allowedTCPPorts = [ 22 3000 ];
  };
};

# List packages installed in system profile. To search, run:
# $ nix search wget
```

```
  environment = {
    systemPackages = with pkgs; [];

    loginShellInit = ''
      export SSH_AUTH_SOCK="$XDG_RUNTIME_DIR/ssh-agent.socket"
    '';
  };

  system.stateVersion = "22.05";
}
```

Now we can move the `secrets` folder, and the SSH key (if it's not already there) from our host machine to the remote server.

```
$ scp secrets root@<SERVER_IP>:/root/
$ scp ~/.ssh/<YOUR_KEY> root@<SERVER_IP>:/root/.ssh/id_ed25519
$ scp ~/.ssh/<YOUR_KEY>.pub root@<SERVER_IP>:/root/.ssh/id_ed25519.pub
```

Then hit build and apply the config:

```
$ nixos-rebuild switch \
  --flake .#peepeepoopoo \
  --target-host root@<SERVER_IP> \
  --build-host localhost
copying 3 paths...
copying path '/nix/store/lali119kww58c4df3b1w61yzg5an1mr7-system-units' to 'ssh://ro
copying path '/nix/store/g1izspgbn34hmlll2hby5qapx90nm43p-etc' to 'ssh://root@<SERVE
copying path '/nix/store/739zphavd4d1vjfnd8v2b1bpm0dzwxz6-nixos-system-unnamed-22.05
updating GRUB 2 menu...
stopping the following units: emojied.service
activating the configuration...
[agenix] creating new generation in /run/agenix.d/1
[agenix] decrypting secrets...
decrypting '/root/secrets/emojiedDBCACert.age' to '/run/agenix.d/1/emojiedDBCACert'.
decrypting '/root/secrets/emojiedDBPassword.age' to '/run/agenix.d/1/emojiedDBPasswo
[agenix] symlinking new secrets to /run/agenix (generation 1)...
[agenix] chowning...
setting up /etc...
reloading user units for root...
setting up tmpfiles
starting the following units: emojied.service
the following new units were started: run-agenix.d.mount
```

Which gives me this beautiful creation:

The abomination now works!

nix flakes