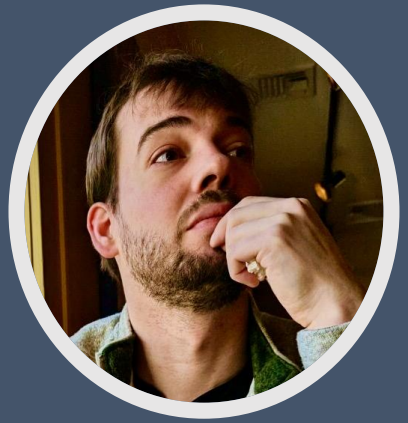
My NixOS-Powered Homelab

Josh Lee • South East Linux Fest • June 13 2025



Josh Lee

Open Source Advocate

Altinity

My path to NixOS

Nix-Shell? Home-Manager? Flakes?

```
$ nix-shell -p clickhouse
```

Nix-Shell? Home-Manager? Flakes?

```
$ nix-shell -p clickhouse
```

How do I make templated VMs without having to store multi-GB images?

How can I share parameters between VMs?

How do I keep track of the state of each VM?

Why NixOS?

Why NixOS?

- "Forced" GitOps
- Self-documenting
- Portable and reproducible
- Uses familiar tools (systemd, docker)

```nix
{ ... }:
{
  imports = [
    ./base/base.nix
    ../users/example.nix
  ];

  # disable logging to disk for generic servers
  services.journald.extraConfig = ''
    Storage=volatile;
    RuntimeMaxUse=30M;
  '';
}
```

Why Nix?

- One language for all tools (except k8s?)
- Functional ❤️
- `nix-modules` system is awesome

NixOS Generators

NixOS Generators

```
$ nixos-generate -f proxmox -c configuration.nix
```

VM-specific Configuration

VM Specific Configuration

```
imports = [
  (modulesPath + "/profiles/qemu-guest.nix")
];
```

VM Specific Configuration

```
imports = [
  (modulesPath + "/profiles/qemu-guest.nix")
];

services.qemuGuest.enable = lib.mkDefault true;
```

VM Specific Configuration

```
boot.loader.grub.enable = true;
boot.loader.grub.devices = [ "nodev" ];
```

(Important!) VM Specific Configuration

```
boot.growPartition = true;
```

Enable Remote Updates

```
nix.settings.trusted-users = [ "root" "@wheel" ];
nix.settings.experimental-features = [
    "nix-command"
    "flakes"
];
```

Use Remote Updates

```
$ nixos-rebuild --target-host user@remote-host
--use-remote-sudo
```

* Warning: does not copy configuration.nix / flake files to target host *

Managing the Menagerie

```
hostsDir = ./nix-hosts;
readHost = file: import (hostsDir + ("/" + file));
hostFiles = lib.filter (file: lib.hasSuffix ".nix" file) (lib.attrNames (builtins.readDir
hostsDir));
hostDefinitions = builtins.map (file: readHost file) hostFiles;
makeSystem = host: lib.nixosSystem {
    system = "x86_64-linux";
    modules = [ host ];
};
systems = builtins.map makeSystem hostDefinitions;
configurations = lib.listToAttrs (builtins.map (host: {
    name = host.config.networking.hostName;
    value = host;
}) systems);
```

Per-host:

Only override what's needed...

```
lib.mkDefault
lib.mkForce
```

```nix
{ ... }:
{
  imports = [
    ../templates/server.nix
    ../workloads/ingress.nix
  ];

  networking.hostName = "ingress-01";

  services.tailscale.enable = true;
  system.stateVersion = "24.05";
}
```

```nix
{ config, pkgs, modulesPath, lib, system, ... }:

{
  imports = [
    (modulesPath + "/profiles/qemu-guest.nix")
  ];

  networking.hostName = lib.mkDefault "base";

  boot.growPartition = true;

  # Enable QEMU Guest for Proxmox
  services.qemuGuest.enable = true;
  ...
}
```

Per-host:

Put `system.stateVersion` in your most-specific configuration file.

Commit it!

What about Workloads?

Deploying Services

```nix
{ config, lib, ... }:
with lib;
let
    cfg = config.home-cloud.monitoring;
    ports = {
        grafana = 2342;
        prometheus = 9000;
        node_exporter = 9100;
    };
in
{
    options.home-cloud.monitoring = {
        enable = mkEnableOption "monitoring";
    };
    config = mkIf cfg.enable {
        networking.firewall = {
            allowedTCPPorts = [ 80 443 ];
        };

        services.cadvisor.enable = true;
         services.prometheus = {
            ...
            port = ports.node_exporter;
          };
        };
     };
  };
}
```

```
{ config, lib, ... }:
with lib;
let
    cfg = config.home-cloud.monitoring;
    ports = {
        grafana = 2342;
        prometheus = 9000;
        node_exporter = 9100;
    };
in
{
    options.home-cloud.monitoring = {
        enable = mkEnableOption "monitoring";
    };
    config = mkIf cfg.enable {
        networking.firewall = {
            allowedTCPPorts = [ 80 443 ];
        };

        services.cadvisor.enable = true;
         services.prometheus = {
            ...
            port = ports.node_exporter;
          };
```

My N ocial

Deploying Docker Services

```nix
{ config, lib, ... }:
with lib;
let
   cfg = config.home-cloud.portainer;
in
{
   options.home-cloud.portainer = {
      enable = lib.mkEnableOption "portainer";
   };

   config = lib.mkIf cfg.enable {
      virtualisation.oci-containers.containers = {
         portainer = {
            image = "portainer/portainer-ce";
            ports = [
               "8000:8000"
               "9443:9443"
            ];
            volumes = [
               "appdata:/data"
               "/var/run/podman/podman.sock:/var/run/docker.sock:Z"
            ];
            extraOptions = [
               "--privileged"
            ];
         };
      };
   };
}
```

```
    };

    config = lib.mkIf cfg.enable {
      virtualisation.oci-containers.containers = {
        portainer = {
          image = "portainer/portainer-ce";
          ports = [
            "8000:8000"
            "9443:9443"
          ];
          volumes = [
            "appdata:/data"
            "/var/run/podman/podman.sock:/var/run/docker.sock:Z"
          ];
          extraOptions = [
            "--privileged"
          ];
        };
      };
    };
}
```

My Tailscale Router Config

```
networking.hostName = "tailgate";

services.tailscale.enable = true;
services.tailscale.useRoutingFeatures = "both";
system.stateVersion = "23.11";
```

Lessons Learned & Next Steps

Building an OCI Stack

Building an OCI Stack

1. Container Runtime
2. Ingress
3. DNS
4. Shared Storage
5. Monitoring

Oops, I'm building a Kubernetes...

Remember to KISS

Dedicated VMs vs Nix Services vs Containers

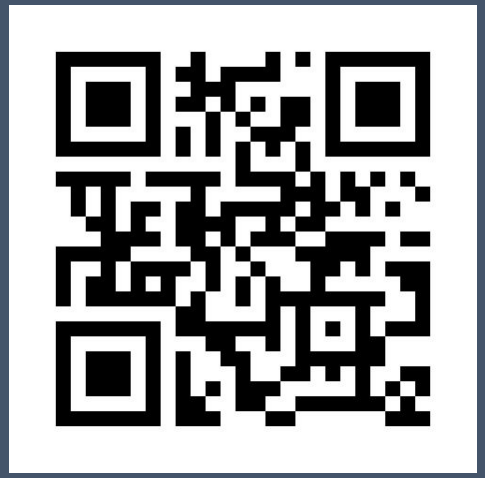
Solve a problem with Nix...

… and it’s solved “forever”

More Fun with Nix

1. Home Manager for “my environment” anywhere
2. Low-touch Thin-Clients (e.g. Nixbook)
3. Custom Installer ISO with SSH
4. MicroVMs

Q&A



My LinkedIn

Nix Homelab Resources