The Flake ecosystem

Matias Zwinger

2026-04-14

OtaNix ry

Structure

- Flake infrastructure
- Meta-tools
  - flake-utils
  - flake-parts
  - flake-compat
- Flake-based tools
  - disko
  - deploy-rs

Flake infrastructure

- Official flake search: https://search.nixos.org/flakes
- 3rd party flake registries
  - FlakeHub https://flakehub.com/
  - Flakestry https://flakestry.dev/
- Good old GitHub search https://github.com/topics/flakes

Meta-tools

Flakes for developing flakes

flake-utils

https://github.com/numtide/flake-utils

- Collection of pure Nix functions for flakes
- Main use: `eachDefaultSystem` eliminates per-system boilerplate

Without flake-utils

```
{
  outputs = { self, nixpkgs }: {
    packages.x86_64-linux.default =
      nixpkgs.legacyPackages
        .x86_64-linux.hello;
    packages.aarch64-linux.default =
      nixpkgs.legacyPackages
        .aarch64-linux.hello;
    packages.x86_64-darwin.default =
      nixpkgs.legacyPackages
        .x86_64-darwin.hello;
    # repeat for every output type...
  };
}
```

With flake-utils

```
{
  inputs.flake-utils.url =
    "github:numtide/flake-utils";

  outputs = { self, nixpkgs, flake-utils }:
    flake-utils.lib.eachDefaultSystem (system:
      let
        pkgs = nixpkgs.legacyPackages.${system};
      in {
        packages.default = pkgs.hello;
        devShells.default = pkgs.mkShell {
          buildInputs = [ pkgs.hello ];
        };
      }
    );
}
```

flake-parts

https://flake.parts/

- Allows splitting the flake into modules (similar to Nix modules)

Without flake-parts

```nix
# everything in one flake.nix
{
  outputs = { self, nixpkgs, flake-utils }:
    flake-utils.lib.eachDefaultSystem (system:
      let
        pkgs = nixpkgs.legacyPackages.${system};
      in {
        packages.default = pkgs.hello;
        devShells.default = pkgs.mkShell {
          buildInputs = [ pkgs.hello ];
        };
      }
    ) // {
      nixosConfigurations.myHost =
        nixpkgs.lib.nixosSystem { ... };
      # grows unbounded...
    };
}
```

With flake-parts

```nix
# flake.nix stays small
{
  inputs.flake-parts.url =
    "github:hercules-ci/flake-parts";

  outputs = inputs@{ flake-parts, ... }:
    flake-parts.lib.mkFlake { inherit inputs; } {
      imports = [
        ./packages.nix
        ./devshells.nix
        ./nixos.nix
      ];
      systems = [
        "x86_64-linux"
        "aarch64-linux"
      ];
    };
}
```

flake-compat

https://github.com/NixOS/flake-compat

- Allows usage of flakes in systems with no flake support
- Bridges `nix-build` / `nix-shell` to `flake.nix` outputs

`default.nix`

```
(import
  (fetchTarball
    "https://github.com/NixOS/flake-compat/archive/main.
tar.gz"
  )
  { src = ./.; }
).defaultNix
```

`shell.nix`

```
(import
  (fetchTarball
    "https://github.com/NixOS/flake-compat/archive/main.
tar.gz"
  )
  { src = ./.; }
).shellNix
```

Flake-based tools

Nix{,OS} tools distributed as flakes

disko

https://github.com/nix-community/disko

- Enables declarative disk partitioning
- Replaces manual `fdisk`/`mkfs` steps during installation

disk-config.nix

```
{
  disko.devices = {
    disk.main = {
      type = "disk";
      device = "/dev/nvme0n1p1";
      content = {
        type = "gpt";
        partitions = {
          ESP = {
            size = "500M";
            type = "EF00";
            content = {
              type = "filesystem";
              format = "vfat";
              mountpoint = "/boot";
            };
          };
          root = {
            size = "100%";
            content = {
              type = "filesystem";
              format = "ext4";
              mountpoint = "/";
            };
          };
        };
      };
    };
  };
}
```

flake.nix

```
{
  inputs.disko.url =
    "github:nix-community/disko";

  outputs = { nixpkgs, disko, ... }: {
    nixosConfigurations.myHost =
      nixpkgs.lib.nixosSystem {
        modules = [
          disko.nixosModules.disko
          ./disk-config.nix
          ./configuration.nix
        ];
      };
  };
}

# Partition, format, and install
sudo nix run \
  'github:nix-community/disko#disko-install' \
  -- --flake '.#myHost' \
  --disk main /dev/sda
```

deploy-rs

https://github.com/serokell/deploy-rs

- Deploy NixOS configurations over SSH
- Rollback support

`flake.nix`

```nix
{
  inputs.deploy-rs.url =
    "github:serokell/deploy-rs";

  outputs = { nixpkgs, deploy-rs, ... }: {
    nixosConfigurations.myHost =
      nixpkgs.lib.nixosSystem { ... };

    deploy.nodes.myHost = {
      hostname = "192.168.1.67";
      profiles.system = {
        user = "root";
        path = deploy-rs.lib.x86_64-linux
          .activate.nixos
          self.nixosConfigurations.myHost;
      };
    };

    checks = builtins.mapAttrs
      (system: deployLib:
        deployLib.deployChecks
          self.deploy)
      deploy-rs.lib;
  };
}
```

Deploy

```sh
# Deploy to a node
nix run \
  'github:serokell/deploy-rs' \
  -- '.#myHost'

# Deploy a specific profile
nix run \
  'github:serokell/deploy-rs' \
  -- '.#myHost.system'

# Dry run (no activation)
deploy --dry-activate '.#myHost'
```

Questions?