Enabling NixOS with Flakes

Compared to the default configuration method currently used in NixOS, Flakes offers better reproducibility. Its clear package structure definition inherently supports dependencies on other Git repositories, facilitating code sharing. Therefore, this book suggests using Flakes to manage system configurations.

This section describes how to use Flakes to manage NixOS system configuration, and **you don't need to know anything about Flakes in advance**.

Enabling Flakes Support for NixOS

Currently, Flakes is still an experimental feature and not enabled by default. We need to manually modify the `/etc/nixos/configuration.nix` file to enable the Flakes feature and the accompanying new nix command-line tool:

nix

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

{
  imports = [
    # Include the results of the hardware scan.
    ./hardware-configuration.nix
  ];

  # ......

  # Enable the Flakes feature and the accompanying new nix command-line tool
  nix.settings.experimental-features = [ "nix-command" "flakes" ];
  environment.systemPackages = with pkgs; [
    # Flakes clones its dependencies through the git command,
    # so git must be installed first
    git
    vim
    wget
  ];
  # Set the default editor to vim
  environment.variables.EDITOR = "vim";

```

```
  # ......
}
```

After making these changes, run `sudo nixos-rebuild switch` to apply the modifications. Then, you can use the Flakes feature to manage your system configuration.

The new nix command-line tool also offers some convenient features. For example, you can now use the `nix repl` command to open a nix interactive environment. If you're interested, you can use it to review and test all the Nix syntax you've learned before.

Switching System Configuration to `flake.nix`

After enabling the Flakes feature, the `sudo nixos-rebuild switch` command will prioritize reading the `/etc/nixos/flake.nix` file, and if it's not found, it will attempt to use `/etc/nixos/configuration.nix`.

You can start by using the official templates to learn how to write a flake. First, check what templates are available:

bash

```bash
nix flake show templates
```

Among them, the `templates#full` template demonstrates all possible usage. Take a look at its content:

bash

```bash
nix flake init -t templates#full
cat flake.nix
```

Referencing this template, create the file `/etc/nixos/flake.nix` and write the configuration content. All subsequent system modifications will be taken over by Nix Flakes. Here's an example of the content:

nix

```nix
{
  description = "A simple NixOS flake";

  inputs = {
    # NixOS official package source, using the nixos-24.11 branch here
```

```
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.11";
  };

  outputs = { self, nixpkgs, ... }@inputs: {
    # Please replace my-nixos with your hostname
    nixosConfigurations.my-nixos = nixpkgs.lib.nixosSystem {
      system = "x86_64-linux";
      modules = [
        # Import the previous configuration.nix we used,
        # so the old configuration file still takes effect
        ./configuration.nix
      ];
    };
  };
}
```

Here we defined a system named `my-nixos`, with its configuration file located at `/etc/nixos/` as `./configuration.nix`. This means we are still using the old configuration.

Now, when you execute `sudo nixos-rebuild switch` to apply the configuration, the system should not change at all because we have simply switched to using Nix Flakes, and the configuration content remains consistent with before.

> If your system's hostname is not `my-nixos`, you need to modify the name of `nixosConfigurations` in `flake.nix`, or use `--flake /etc/nixos#my-nixos` to specify the configuration name.

After the switch, we can manage the system through the Flakes feature.

Currently, our flake includes these files:

- `/etc/nixos/flake.nix`: The entrypoint for the flake, which is recognized and deployed when `sudo nixos-rebuild switch` is executed.
- `/etc/nixos/flake.lock`: The automatically generated version lock file, which records the data sources, hash values, and version numbers of all inputs in the entire flake, ensuring system reproducibility.
- `/etc/nixos/configuration.nix`: This is our previous configuration file, which is imported as a module in `flake.nix`. Currently, all system configurations are written in this file.
- `/etc/nixos/hardware-configuration.nix`: This is the system hardware configuration file, generated by NixOS, which describes the system's hardware information.

Conclusion

Up to this point, we have merely added a very simple configuration file, `/etc/nixos/flake.nix`, which has merely been a thin wrapper around `/etc/nixos/configuration.nix`, offering no new functionality and introducing no disruptive changes.

In the content of the book that follows, we will learn about the structure and functionality of `flake.nix` and gradually see the benefits that such a wrapper can bring.

> Note: The configuration management method described in this book is NOT "Everything in a single file". It is recommended to categorize configuration content into different nix files, then introduce these configuration files in the `modules` list of `flake.nix`, and manage them with Git.
> The benefits of this approach are better organization of configuration files and improved maintainability of the configuration. The section Modularizing NixOS Configuration will explain in detail how to modularize your NixOS configuration, and Other Useful Tips - Managing NixOS Configuration with Git will introduce several best practices for managing NixOS configuration with Git.
