`flake.nix` Configuration Explained

Above, we created a `flake.nix` file to manage system configurations, but you might still be unclear about its structure. Let's explain the content of this file in detail.

---

1. Flake Inputs

First, let's look at the `inputs` attribute. It is an attribute set that defines all the dependencies of this flake. These dependencies will be passed as arguments to the `outputs` function after they are fetched:

```
{
  inputs = {
    # NixOS official package source, using the nixos-24.11 branch here
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.11";
  };

  outputs = { self, nixpkgs, ... }@inputs: {
    # Omitting previous configurations......
  };
}
```

Dependencies in `inputs` has many types and definitions. It can be another flake, a regular Git repository, or a local path. The section Other Usage of Flakes - Flake Inputs describes common types of dependencies and their definitions in detail.

Here we only define a dependency named `nixpkgs`, which is the most common way to reference in a flake, i.e., `github:owner/name/reference`. The `reference` here can be a branch name, commit-id, or tag.

After `nixpkgs` is defined in `inputs`, you can use it in the parameters of the subsequent `outputs` function, which is exactly what our example does.

---

2. Flake Outputs

Now let's look at `outputs`. It is a function that takes the dependencies from `inputs` as its parameters, and its return value is an attribute set, which represents the build results of the flake:

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

  inputs = {
    # NixOS official package source, here using the nixos-24.11 branch
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.11";
  };

  outputs = { self, nixpkgs, ... }@inputs: {
    # The host with the hostname `my-nixos` will use this configuration
    nixosConfigurations.my-nixos = nixpkgs.lib.nixosSystem {
      system = "x86_64-linux";
      modules = [
        ./configuration.nix
      ];
    };
  };
}
```

Flakes can have various purposes and can have different types of outputs. The section Flake Outputs provides a more detailed introduction. Here, we are only using the `nixosConfigurations` type of outputs, which is used to configure NixOS systems.

When we run the `sudo nixos-rebuild switch` command, it looks for the `nixosConfigurations.my-nixos` attribute (where `my-nixos` will be the hostname of your current system) in the attribute set returned by the `outputs` function of `/etc/nixos/flake.nix` and uses the definition there to configure your NixOS system.

Actually, we can also customize the location of the flake and the name of the NixOS configuration instead of using the defaults. This can be done by adding the `--flake` parameter to the `nixos-rebuild` command. Here's an example:

```nix
sudo nixos-rebuild switch --flake /path/to/your/flake#your-hostname
```

A brief explanation of the `--flake /path/to/your/flake#your-hostname` parameter:

1. `/path/to/your/flake` is the location of the target flake. The default path is `/etc/nixos/` .
2. `#` is a separator, and `your-hostname` is the name of the NixOS configuration. `nixos-rebuild` will default to using the hostname of your current system as the configuration name to look for.

You can even directly reference a remote GitHub repository as your flake source, for example:

```nix
sudo nixos-rebuild switch --flake github:owner/repo#your-hostname
```

---

3. The Special Parameter `self` of the `outputs` Function

Although we have not mentioned it before, all the example code in the previous sections has one more special parameter in the `outputs` function, and we will briefly introduce its purpose here.

The description of it in the [nix flake - Nix Manual] is:

> The special input named `self` refers to the outputs and source tree of this flake.

This means that `self` is the return value of the current flake's `outputs` function and also the path to the current flake's source code folder (source tree).

We are not using the `self` parameter here, but in some more complex examples (or configurations you may find online) later, you will see the usage of `self` .

> Note: You might come across some code where people use `self.outputs` to reference the outputs of the current flake, which is indeed possible. However, the Nix Manual does not provide any explanation for this, and it is considered an internal implementation detail of flakes. It is not recommended to use this in your own code!

---

4. Simple Introduction to `nixpkgs.lib.nixosSystem` Function

A Flake can depend on other Flakes to utilize the features they provide.

By default, a flake searches for a `flake.nix` file in the root directory of each of its dependencies (i.e., each item in `inputs`) and lazily evaluates their `outputs` functions. It then passes the attribute set returned by these functions as arguments to its own `outputs` function, enabling us to use the features provided by the other flakes within our current flake.

More precisely, the evaluation of the `outputs` function for each dependency is lazy. This means that a flake's `outputs` function is only evaluated when it is actually used, thereby avoiding unnecessary calculations and improving efficiency.

The description above may be a bit confusing, so let's take a look at the process with the `flake.nix` example used in this section. Our `flake.nix` declares the `inputs.nixpkgs` dependency, so that nixpkgs/flake.nix will be evaluated when we run the `sudo nixos-rebuild switch` command.

From the source code of the Nixpkgs repository, we can see that its flake outputs definition includes the `lib` attribute, and in our example, we use the `lib` attribute's `nixosSystem` function to configure our NixOS system:

nix

```nix
{
  inputs = {
    # NixOS official package source, here using the nixos-24.11 branch
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-24.11";
  };

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

The attribute set following `nixpkgs.lib.nixosSystem` is the function's parameter. We have only set two parameters here:

1. `system`: This is straightforward, it's the system architecture parameter.

2. `modules`：This is a list of modules, where the actual NixOS system configuration is defined. The `/etc/nixos/configuration.nix` configuration file itself is a Nixpkgs Module, so it can be directly added to the `modules` list for use.

Understanding these basics is sufficient for beginners. Exploring the `nixpkgs.lib.nixosSystem` function in detail requires a grasp of the Nixpkgs module system. Readers who have completed the Modularizing NixOS Configuration section can return to nixpkgs/flake.nix to find the definition of `nixpkgs.lib.nixosSystem`, trace its source code, and study its implementation.
