Adding Binary Cache Servers

We have introduced the concepts of Nix Store and binary cache. Here, we will see how to add multiple cache servers to speed up package downloads.

Why Add Cache Servers

Nix provides an official cache server, https://cache.nixos.org, which caches build results for most commonly used packages. However, it may not meet all users' needs. In the following cases, we need to add additional cache servers:

1. Add cache servers for some third-party projects, such as the nix-community cache server https://nix-community.cachix.org, which can significantly improve the build speed of these third-party projects.
2. Add cache server mirror sites closest to the user to speed up downloads.
3. Add a self-built cache server to speed up the build process of personal projects.

How to Add Cache Servers

In Nix, you can configure cache servers using the following options:

1. substituters: It is a string list, and each string is the address of a cache server. Nix will attempt to find caches from these servers in the order specified in the list.
2. trusted-public-keys: To prevent malicious attacks, The require-sigs option is enabled by default. Only caches with signatures that can be verified by any public key in `trusted-public-keys` will be used by Nix. Therefore, you need to add the public key corresponding to the `substituters` in `trusted-public-keys` .
   1. cache mirror's data are directly synchronized from the official cache server. Therefore, their public keys are the same as those of the official cache server, and you can use the public key of the official cache server without additional configuration.
   2. This entirely trust-based public key verification mechanism transfers the security responsibility to users. If users want to use a third-party cache server to speed up the build process of a certain library, they must take on the corresponding security risks and decide whether to add the public key of that cache server to `trusted-`

`public-keys`. To completely solve this trust issue, Nix has introduced the experimental feature [ca-derivations](https://nixos.org), which does not depend on `trusted-public-keys` for signature verification. Interested users can explore it further.

You can configure the `substituters` and `trusted-public-keys` parameters in the following ways:

1. Configure in `/etc/nix/nix.conf`, a global configuration that affects all users.
   1. You can use `nix.settings.substituters` and `nix.settings.trusted-public-keys` in any NixOS Module to declaratively generate `/etc/nix/nix.conf`.
2. Configure in the `flake.nix` of a flake project using `nixConfig.substituters`. This configuration only affects the current flake.
3. Temporarily set through the `--option` parameter of the `nix` command, and this configuration only applies to the current command.

Among these three methods, except for the first global configuration, the other two are temporary configurations. If multiple methods are used simultaneously, later configurations will directly override earlier ones.

However, there are security risks in temporarily setting `substituters`, as explained earlier regarding the deficiencies of the security verification mechanism based on `trusted-public-keys`. To set `substituters` through the second and third methods, you need to meet one of the following conditions:

1. The current user is included in the `trusted-users` parameter list in `/etc/nix/nix.conf`.
2. The `substituters` specified temporarily via `--option substituters "http://xxx"` are included in the `trusted-substituters` parameter list in `/etc/nix/nix.conf`.

Based on the above information, the following are examples of the three configuration methods mentioned earlier.

Firstly, declaratively configure system-level `substituters` and `trusted-public-keys` using `nix.settings` in `/etc/nixos/configuration.nix` or any NixOS Module:

```nix
{
  lib,
  ...
}: {

  # ...

  nix.settings = {
    # given the users in this list the right to specify additional substituters
    #    1. `nixConfig.substituters` in `flake.nix`
    #    2. command line args `--options substituters http://xxx`
    trusted-users = ["ryan"];

    substituters = [
      # cache mirror located in China
      # status: https://mirror.sjtu.edu.cn/
      "https://mirror.sjtu.edu.cn/nix-channels/store"
      # status: https://mirrors.ustc.edu.cn/status/
      # "https://mirrors.ustc.edu.cn/nix-channels/store"

      "https://cache.nixos.org"
    ];

    trusted-public-keys = [
      # the default public key of cache.nixos.org, it's built-in, no need to add
      "cache.nixos.org-1:6NCHdD59X431o0gWypbMrAURkbJ16ZPMQFGspcDShjY="
    ];
  };

}
```

The second method is to configure `substituters` and `trusted-public-keys` using `nixConfig` in `flake.nix`:

> As mentioned earlier, it is essential to configure `nix.settings.trusted-users` in this configuration. Otherwise, the `substituters` we set here will not take effect.

```nix
{
  description = "NixOS configuration of Ryan Yin";

  # the nixConfig here only affects the flake itself, not the system configurati
  nixConfig = {
    # override the default substituters
    substituters = [
      # cache mirror located in China
      # status: https://mirror.sjtu.edu.cn/
      "https://mirror.sjtu.edu.cn/nix-channels/store"
      # status: https://mirrors.ustc.edu.cn/status/
```

```
        # "https://mirrors.ustc.edu.cn/nix-channels/store"

        "https://cache.nixos.org"

        # nix community's cache server
        "https://nix-community.cachix.org"
      ];
      trusted-public-keys = [
        # nix community's cache server public key
        "nix-community.cachix.org-1:mB9FSh9qf2dCimDSUo8Zy7bkq5CX+/rkCWyvRCYg3Fs="
      ];
    };

    inputs = {
      nixpkgs.url = "github:nixos/nixpkgs/nixos-24.11";

      # omitting several configurations...
    };

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

            {
              # given the users in this list the right to specify additional subst
              #    1. `nixConfig.substituters` in `flake.nix`
              nix.settings.trusted-users = [ "ryan" ];
            }
            # omitting several configurations...
         ];
        };
      };
    };
  }
```

Finally, the third method involves using the following command to temporarily specify `substituters` and `trusted-public-keys` during deployment:

```bash
sudo nixos-rebuild switch --option substituters "https://nix-community.cachix.or
```

Choose one of the above three methods for configuration and deployment. After a successful deployment, all subsequent packages will preferentially search for caches from domestic mirror sources.

> If your system 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.

The `extra-` Prefix for Nix Options Parameters

As mentioned earlier, the `substituters` configured by the three methods will override each other, but the ideal situation should be:

1. At the system level in `/etc/nix/nix.conf`, configure only the most generic `substituters` and `trusted-public-keys`, such as official cache servers and domestic mirror sources.
2. In each flake project's `flake.nix`, configure the `substituters` and `trusted-public-keys` specific to that project, such as non-official cache servers like nix-community.
3. When building a flake project, nix should **merge** the `substituters` and `trusted-public-keys` configured in `flake.nix` and `/etc/nix/nix.conf`.

Nix provides the `extra-` prefix to achieve this **merging** functionality.

According to the official documentation, if the value of the `xxx` parameter is a list, the value of `extra-xxx` will be appended to the end of the `xxx` parameter:

In other words, you can use it like this:

```nix
{
  description = "NixOS configuration of Ryan Yin";

  # the nixConfig here only affects the flake itself, not the system configurati
  nixConfig = {

    # will be appended to the system-level substituters
    extra-substituters = [
      # nix community's cache server
      "https://nix-community.cachix.org"
    ];

    # will be appended to the system-level trusted-public-keys
    extra-trusted-public-keys = [
      # nix community's cache server public key
      "nix-community.cachix.org-1:mB9FSh9qf2dCimDSUo8Zy7bkq5CX+/rkCWyvRCYg3Fs="
    ];
  };

  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs/nixos-24.11";

    # omitting several configurations...
  };

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

          {
            # given the users in this list the right to specify additional subst
            #    1. `nixConfig.substituters` in `flake.nix`
            nix.settings.trusted-users = [ "ryan" ];

            # the system-level substituters & trusted-public-keys
            nix.settings = {
              substituters = [
                # cache mirror located in China
                # status: https://mirror.sjtu.edu.cn/
                "https://mirror.sjtu.edu.cn/nix-channels/store"
                # status: https://mirrors.ustc.edu.cn/status/
                # "https://mirrors.ustc.edu.cn/nix-channels/store"
                "https://cache.nixos.org"
              ];

              trusted-public-keys = [
                # the default public key of cache.nixos.org, it's built-in, no n
                "cache.nixos.org-1:6NCHdD59X431o0gWypbMrAURkbJ16ZPMQFGspcDShjY="
              ];
            };

          }
          # omitting several configurations...
       ];
      };
    };
  };
}
```

Accelerate Package Downloads via a Proxy Server

> Referenced from Issue: [roaming laptop: network proxy configuration - NixOS/nixpkgs](#) Although it's mentioned earlier that a transparent proxy running on your router or local machine can completely solve the issue of slow package downloads in NixOS, the configuration is rather cumbersome and often requires additional hardware.

Some users may prefer to directly speed up package downloads by using a HTTP/Socks5 proxy running on their machine. Here's how to set it up. Using methods like `export HTTPS_PROXY=http://127.0.0.1:7890` in the Terminal will not work because the actual work is done by a background process called `nix-daemon`, not by commands directly executed in the Terminal.

If you only need to use a proxy temporarily, you can set the proxy environment variables with the following commands:

```bash
sudo mkdir /run/systemd/system/nix-daemon.service.d/
cat << EOF >/run/systemd/system/nix-daemon.service.d/override.conf
[Service]
Environment="https_proxy=socks5h://localhost:7891"
```

```
EOF
sudo systemctl daemon-reload
sudo systemctl restart nix-daemon
```

After deploying this configuration, you can check if the environment variables have been set by running `sudo cat /proc/$(pidof nix-daemon)/environ | tr '\0' '\n'` .

The settings in `/run/systemd/system/nix-daemon.service.d/override.conf` will be automatically deleted when the system restarts, or you can manually delete it and restart the nix-daemon service to restore the original settings.

If you want to permanently set the proxy, it is recommended to save the above commands as a shell script and run it each time the system starts. Alternatively, you can use a transparent proxy or TUN and other global proxy solutions.

> There are also people in the community who permanently set the proxy for nix-daemon in a declarative way using `systemd.services.nix-daemon.environment` . However, if the proxy encounters problems, it will be very troublesome. Nix-daemon will not work properly, and most Nix commands will not run correctly. Moreover, the configuration of systemd itself is set to read-only protection, making it difficult to modify or delete the proxy settings. So, it is not recommended to use this method.

> When using some commercial or public proxies, you might encounter HTTP 403 errors when downloading from GitHub (as described in nixos-and-flakes-book/issues/74). In such cases, you can try changing the proxy server or setting up access-tokens to resolve the issue.

Loading comments...