Provisioning remote machines via SSH

Contents

- Provisioning remote machines via SSH
- Next steps

It is possible to replace any Linux installation with a NixOS configuration on running systems using `nixos-anywhere` and `disko`.

Introduction

In this tutorial, you will deploy a NixOS configuration to a running computer.

What will you learn?

You'll learn how to

- Specify a minimal NixOS configuration with a declarative disk layout and SSH access
- Check that a configuration is valid
- Deploy and update a NixOS configuration on a remote machine

What do you need?

- Familiarity with the Nix language
- Familiarity with the Module system

For a successful unattended installation, ensure for the *target machine* that:

- It is a QEMU virtual machine running Linux
  - With `kexec` support

Skip to main content

    - With at least 1 GB of RAM

  This may also be a live system booted from USB, such as the NixOS installer.

- The IP address is configured automatically with DHCP
- You can login via SSH
    - With public key authentication (prefered), or password
    - As user `root` or another user with `sudo` permissions

The *local machine* only needs a working Nix installation.

We call the *target machine* `target-machine` in this tutorial. Replace it with the actual hostname or IP address.

Prepare the environment

Create a new project directory and enter it with your shell:

```
mkdir remote
cd remote
```

Specify dependencies on `nixpkgs`, `disko`, and `nixos-anywhere`:

```
$ nix-shell -p npins
[nix-shell:remote]$ npins init
[nix-shell:remote]$ npins add github nix-community disko
[nix-shell:remote]$ npins add github nix-community nixos-anywhere
```

Create a new file `shell.nix` which provides all needed tooling using the pinned dependencies:

```
let
  sources = import ./npins;
  pkgs = import sources.nixpkgs {};
in

pkgs.mkShell {
  nativeBuildInputs = with pkgs; [
    npins
    nixos-anywhere
    nixos-rebuild
  ];
  shellHook = ''
    export NIX_PATH="nixpkgs=${sources.nixpkgs}:nixos-config=$PWD/configuration.nix
```

Skip to main content

```
  '';
}
```

Now exit the temporary environment and enter the newly specified one:

```
[nix-shell:remote]$ exit
$ nix-shell
```

This shell environment is ready to use well-defined versions of Nixpkgs with `nixos-anywhere` and `nixos-rebuild`.

> **Important**
>
> Run all following commands in this environment.

Create a NixOS configuration

The new NixOS configuration will consist of the general system configuration and a disk layout specification.

The disk layout in this example describes a single disk with a master boot record (MBR) and EFI system partition (ESP) partition, and a root file system that takes all remaining available space. It will work on both EFI and BIOS systems.

Create a new file `single-disk-layout.nix` with the disk layout specification:

```
{ ... }:

{
  disko.devices.disk.main = {
    type = "disk";
    content = {
      type = "gpt";
      partitions = {
        MBR = {
          priority = 0;
          size = "1M";
          type = "EF02";
        };
        ESP = {
          priority = 1;
          size = "500M";
          type = "EF00";
          content = {
```

Skip to main content

```
            mountpoint = "/boot";
          };
        };
        root = {
          priority = 2;
          size = "100%";
          content = {
            type = "filesystem";
            format = "ext4";
            mountpoint = "/";
          };
        };
      };
    };
  };
}
```

Create the file `configuration.nix`, which imports the disk layout definition and specifies which disk to format:

Tip

If you don't know the target disk's device identifier, list all devices on the *target machine* with `lsblk`:

```
$ ssh target-machine lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0   256G  0 disk
├─sda1   8:1    0 248.5G  0 part /nix/store
│                               /
└─sda2   8:2    0   7.5G  0 part [SWAP]
sr0     11:0    1  1024M  0 rom
```

In this example, the disk name is `sda`. The block device path is then `/dev/sda`. Note that value for later.

```
{ modulesPath, ... }:

let
  diskDevice = "/dev/sda";
  sources = import ./npins;
in
{
  imports = [
    (modulesPath + "/profiles/qemu-guest.nix")
    (sources.disko + "/module.nix")
    ./single-disk-layout.nix
  ];
```

Skip to main content

```
15
16   boot.loader.grub = {
17     devices = [ diskDevice ];
18     efiSupport = true;
19     efiInstallAsRemovable = true;
20   };
21
22   services.openssh.enable = true;
23
24   users.users.root.openssh.authorizedKeys.keys = [
25     "<your SSH key here>"
26   ];
27
28   system.stateVersion = "24.11";
29 }
```

Important

Replace `/dev/sda` with your disk block device path.

Replace the `<your SSH key here>` string with the SSH public key that you want to use for future logins as user `root`.

Detailed explanation

Test the disk layout

Check that the disk layout is valid:

```
nix-build -E "((import <nixpkgs> {}).nixos [ ./configuration.nix ]).installTest"
```

This command runs the complete installation in a virtual machine by building a derivation in the `installTest` attribute provided by the `disko` module.

Deploy the system

To deploy the system, build the configuration and the corresponding disk formatting script, and run `nixos-anywhere` using the results:

Skip to main content

> **Important**
>
> Replace `target-host` with the hostname or IP address of your *target machine*.

```
toplevel=$(nixos-rebuild build --no-flake)
diskoScript=$(nix-build -E "((import <nixpkgs> {}).nixos [ ./configuration.nix ]).d
nixos-anywhere --store-paths "$diskoScript" "$toplevel" root@target-host
```

> **Note**
>
> If you don't have public key authentication: Set the environment variable `SSH_PASS` to your password then append the `--env-password` flag to the `nixos-anywhere` command.

`nixos-anywhere` will now log into the target system, partition, format, and mount the disk, and install the NixOS configuration. Then, it reboots the system.

Update the system

To update the system, run `npins` and re-deploy the configuration:

```
npins update nixpkgs
nixos-rebuild switch --no-flake --target-host root@target-host
```

`nixos-anywhere` is not needed any more, unless you want to change the disk layout.

- Setting up an HTTP binary cache
- Setting up post-build hooks

References

- **`nixos-anywhere`** project page
- **`disko`** project repository
- Collection of disk layout examples