Setting up distributed builds

Contents

Nix can speed up builds by spreading the work across multiple computers at once.

Introduction

In this tutorial, you'll set up a separate build machine and configure your local machine to offload builds to it.

What will you learn?

You'll learn how to

- Create a new user for remote build access from a local machine to the remote builder
- Configure remote builders with a sustainable setup
- Test remote builder connectivity and authentication
- Configure the local machine to automatically distribute builds

Skip to main content

What do you need?

- Familiarity with the Nix language
- Familiarity with the Module system
- A *local machine* (example hostname: `localmachine`)

  The computer with Nix installed that distributes builds to other machines.

- A *remote machine* (example hostname: `remotemachine`)

  A computer running NixOS that accepts build jobs from the *local machine*. Follow Provisioning remote machines via SSH to set up a remote NixOS system.

How long will it take?

- 25 minutes

Create an SSH key pair

The *local machine*'s Nix daemon runs as the `root` user and will need the *private* key file to authenticate itself to remote machines. The *remote machine* will need the *public* key to recognize the *local machine*.

On the *local machine*, run the following command as `root` to create an SSH key pair:

```
# ssh-keygen -f /root/.ssh/remotebuild
```

Note

The name and location of the key pair files can be freely chosen.

Set up the remote builder

In the NixOS configuration directory of the *remote machine*, create the file `remote-builder.nix`:

```
{
  users.users.remotebuild = {
```

Skip to main content

```
    group = "remotebuild";

    openssh.authorizedKeys.keyFiles = [ ./remotebuild.pub ];
  };

  users.groups.remotebuild = {};

  nix.settings.trusted-users = [ "remotebuild" ];
}
```

Copy the file `remotebuild.pub` into this directory.

This configuration module creates a new user `remotebuild` with no home directory. The `root` user on the *local machine* will be able to log into the remote builder via SSH using the previously generated SSH key.

Add the new NixOS module to the existing configuration of the *remote machine*:

```
{
  imports = [
    ./remote-builder.nix
  ];

  # ...
}
```

Activate the new configuration as root:

```
nixos-rebuild switch --no-flake --target-host root@remotemachine
```

Test authentication

Make sure that the SSH connection and authentication work. On the *local machine*, run as `root`:

```
# ssh remotebuild@remotemachine -i /root/.ssh/remotebuild "echo hello"
Could not chdir to home directory /home/remotebuild: No such file or directory
hello
```

If the `hello` message is visible, the authentication works. The `Could not chdir to ...` message confirms that the remote user has no home directory.

This test login also adds the host key of the remote builder to the `/root/.ssh/known_hosts`

Skip to main content

Set up distributed builds

Note

If your *local machine* runs NixOS, skip this section and configure Nix through module options.

Configure Nix to use the remote builder by adding to the Nix configuration file as `root`:

```
# cat << EOF >> /etc/nix/nix.conf
builders = ssh-ng://remotebuild@remotebuilder $(nix-instantiate --eval -E builtins.
builders-use-substitutes = true
```

Detailed explanation

To activate this configuration, restart the Nix daemon:

Linux macOS

On Linux with `systemd`, run as `root`:

```
# systemctl cat nix-daemon.service
```

Skip to main content

NixOS

If your *local machine* runs NixOS, in its configuration directory create the file `distributed-builds.nix`:

```
{ pkgs, ... }:
{
  nix.distributedBuilds = true;
  nix.settings.builders-use-substitutes = true;

  nix.buildMachines = [
    {
      hostName = "remotebuilder";
      sshUser = "remotebuild";
      sshKey = "/root/.ssh/remotebuild";
      system = pkgs.stdenv.hostPlatform.system;
      supportedFeatures = [ "nixos-test" "big-parallel" "kvm" ];
    }
  ];
}
```

Detailed explanation

Add the new NixOS module to the existing machine configuration:

```
{
  imports = [
    ./distributed-builds.nix
  ];

  # ...
}
```

Activate the new configuration as `root`:

```
# nixos-rebuild switch
```

Test distributed builds

Try building an new derivation on the *local machine*:

```
$ nix-build --max-jobs 0 -E << EOF
(import <nixpkgs> {}).writeText "test" "$(date)"
```

Skip to main content

```
  /nix/store/9csjdxv6ir8ccnjl6ijs36izswjgchn0-test.drv
building '/nix/store/9csjdxv6ir8ccnjl6ijs36izswjgchn0-test.drv' on 'ssh://remotebui
Could not chdir to home directory /home/remotebuild: No such file or directory
copying 0 paths...
copying 1 paths...
copying path '/nix/store/hvj5vyg4723nly1qh5a8daifbi1yisb3-test' from 'ssh://remoteb
/nix/store/hvj5vyg4723nly1qh5a8daifbi1yisb3-test
```

The resulting derivation changes on every invocation because it depends on the current system time, and thus can never be in the local cache. The `--max-jobs 0` command line argument forces Nix to build it on the remote builder.

The last output line contains the output path and indicates that build distribution works as expected.

Optimise the remote builder configuration

To maximise parallelism, enable automatic garbage collection, and prevent Nix builds from consuming all memory, add the following lines to your `remote-builder.nix` configuration module:

```
 {
   users.users.remotebuild = {
     isNormalUser = true;
     createHome = false;
     group = "remotebuild";

     openssh.authorizedKeys.keyFiles = [ ./remotebuild.pub ];
   };

   users.groups.remotebuild = {};

-  nix.settings.trusted-users = [ "remotebuild" ];
+  nix = {
+    nrBuildUsers = 64;
+    settings = {
+      trusted-users = [ "remotebuild" ];
+
+      min-free = 10 * 1024 * 1024;
+      max-free = 200 * 1024 * 1024;

+      max-jobs = "auto";
+      cores = 0;
+    };
+  };

+  systemd.services.nix-daemon.serviceConfig = {
+    MemoryAccounting = true;
```

Skip to main content

```
+   };
}
```

Tip

Refer to the Nix reference manual for details on the options available in `nix.settings`.

Remote builders can have different performance characteristics. For each `nix.buildMachines` item, set the `maxJobs`, `speedFactor`, and `supportedFeatures` attributes correctly for each different remote builder. This helps Nix on the *local machine* distributing builds the optimal way.

Tip

Refer to the NixOS option documentation on `nix.buildMachines` for details.

Set the `nix.buildMachines.*.publicHostKey` field to each remote builder's public host key to secure build distribution against man-in-the-middle scenarios.

Next steps

- Configure Nix to use a custom binary cache on each remote builder
- Setting up post-build hooks to upload store objects to a binary cache

To set up multiple builders, repeat the instructions in the Set up the remote builder section for each remote builder. Add all new remote builders to the `nix.buildMachines` attribute shown in the Set up distributed builds section.

Alternatives

- nixbuild.net - Nix remote builders as a service
- Hercules CI - Continuous integration with automatic build distribution
- garnix - Hosted continuous integration with build distribution

Skip to main content

References

- Nix reference manual: Settings for distributed builds