# Ubuntu 20.04系统下9步配置NFS

假设有2台电脑连接组成的集群：

Host（lijian-cug）: 192.168.1.86

Client（计算节点lijian-1; lijian-2）:

lijian-1: 192.168.1.150

lijian-2: 192.168.1.225

注：lustre系统比NFS系统在读写方面，性能普遍提高10%~30%。

## 第1步：下载和安装部件

在Host (lijian-cug)上：

apt-get update

apt-get install nfs-kernel-server

这将允许共享某路径。安装完这些程序后，切换到Client服务器。

在Client (lijian-1, lijian-2)上：

apt-get update

apt-get install nfs-common

现在2个服务器上都有了必要的软件，可以配置2台电脑了。

## 第2步：在Host服务器上创建共享路径

1、创建一个通用目的的挂载(mount)—默认的NFS行为，在Client服务器上拥有su权限，但难以与host交互。

在Host上创建一个共享路径：

sudo mkdir /home/lijian/nfs -p

ls -la /home/lijian/nfs

后面执行mpirun时，将可执行程序拷贝到此路径。

需要更改路径的所属权，匹配信任：

sudo chown nobody:nogroup /home/lijian/nfs

该路径可用于输出。

2、在Host上创建home路径，可以被Client服务器访问；还允许Client服务器上的信任管理员访问。

## 第3步：在Host服务器上配置NFS输出

gedit /etc/exports

内容或语法结构是：

/etc/exports

directory\_to\_share client(share\_option1,...,share\_optionN)

修改IP匹配Client：

/etc/exports

/home/lijian/nfs 192.168.1.150(rw,sync,no\_subtree\_check)

/home/lijian 192.168.1.150(rw,sync,no\_root\_squash,no\_subtree\_check)

/home/lijian/nfs 192.168.1.225(rw,sync,no\_subtree\_check)

/home/lijian 192.168.1.225(rw,sync,no\_root\_squash,no\_subtree\_check)

保存，退出。重启NFS服务器：

systemctl restart nfs-kernel-server

在实际使用新的共享之前，需要确保访问共享路径是被防火墙准则允许的。

## 第4步：调整Host上的防火墙

sudo ufw status

屏幕输出：

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

OpenSSH (v6) ALLOW Anywhere (v6)

系统仅允许SSH访问，因此需要增加NFS交通的准则。

增加Host上的打开端口2049，确保代替了Client的IP地址：

ufw allow from 192.168.1.86 to any port nfs

核实改变：sudo ufw status

屏幕输出：

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

2049 ALLOW 192.168.1.86

OpenSSH (v6) ALLOW Anywhere (v6)

## 第5步：在Client上创建挂载点

现在Host服务器的配置和共享已完成，开始准备Client。

将Host的共享路径，挂载到一个Client上的空路径。注意：如果挂载路径下有文件，挂载后将隐藏，确保挂载路径是空的。

在Client上创建空路径：

sudo mkdir -p /home/lijian/nfs

## 第6步：在Client服务器上挂载路径

192.168.1.86是Host端的IP地址。

执行：

sudo mount 192.168.1.86:/home/lijian/nfs /home/lijian/nfs

上述命令将Host上的共享路径挂载到Client上的空路径了。

# 使挂载永久生效

$ cat /etc/fstab

#MPI CLUSTER SETUP

manager:/home/lijian/nfs /home/lijian/nfs nfs

df -h 底部将显示挂载路径的情况。

du -sh /home/lijian/nfs 显示该路径的实际使用情况。

## 第7步：测试NFS访问

在Client上创建文件：

sudo touch /home/lijian/nfs/general.test

ls -l /home/lijian/nfs/general.test

输出：

-rw-r--r-- 1 nobody nogroup 0 Aug 1 13:31 /home/lijian/nfs/general.test

## 第8步：在Boot下挂载远程NFS路径

在Client服务器上，以root特权打开：gedit /etc/fstab

在文件底部添加：

/etc/fstab

. . .

192.168.1.86:/home/lijian/nfs /home/lijian/nfs nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0

Client服务器将自动挂载boot下的远程分区，这需要一定时间连接，共享路径可获取。

## 第9步：卸载NFS远程共享

cd ~

sudo umount /home/lijian/nfs

df -h

完成！

# 9步配置NFS（英文）

https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-16-04

## Introduction

NFS, or Network File System, is a distributed file system protocol that allows you to mount remote directories on your server. This lets you manage storage space in a different location and write to that space from multiple clients. NFS provides a relatively quick and easy way to access remote systems over a network and works well in situations where the shared resources will be accessed regularly.

In this guide, we’ll cover how to configure NFS mounts.

## Prerequisites

We will be using two servers in this tutorial: one will share part of its filesystem with the other.  
To follow along, you will need:

* **Two Ubuntu 16.04 servers, each with a non-root user with sudo privileges** and private networking enabled.
  + For assistance setting up a user with these privileges, follow our [Initial Server Setup with Ubuntu 16.04](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04) guide.
  + For help setting up private networking, see [How To Set Up And Use DigitalOcean Private Networking](https://www.digitalocean.com/community/tutorials/how-to-set-up-and-use-digitalocean-private-networking).

Throughout the tutorial, we refer to the server that shares its directories as the **host** and the server that mounts these directories as the **client**. In order to keep them straight, we’ll use the following IP addresses as stand-ins for the host and client values:

* **Host**: 203.0.113.0
* **Client**: 203.0.113.256

You should replace these values with your own host and client ip addresses.

Step 1 — Downloading and Installing the Components

We’ll begin by installing the necessary components on each server.

On the Host

On the host server, we will install the nfs-kernel-server package, which will allow us to share our directories. Since this is the first operation that we’re performing with apt in this session, we’ll refresh our local package index before the installation:

* sudo apt-get update
* sudo apt-get install nfs-kernel-server

Once these packages are installed, switch to the client server.

On the Client

On the client server, we need to install a package called nfs-common, which provides NFS functionality without including unneeded server components. Again, we will refresh the local package index prior to installation to ensure that we have up-to-date information:

* sudo apt-get update
* sudo apt-get install nfs-common

Now that both servers have the necessary packages, we can start configuring them.

Step 2 — Creating the Share Directories on the Host

We’re going to share two separate directories, with different configuration settings, in order to illustrate two key ways that NFS mounts can be configured with respect to superuser access.

Superusers can do anything anywhere on their system. However, NFS-mounted directories are not part of the system on which they are mounted, so by default, the NFS server refuses to perform operations that require superuser privileges. This default restriction means that superusers on the client cannot write files as root, re-assign ownership, or perform any other superuser tasks on the NFS mount.

Sometimes, however, there are trusted users on the client system who need to be able to do these things on the mounted file system but who have no need for superuser access on the host. The NFS server can be configured to allow this, although it introduces an element of risk, as such a user *could* gain root access to the entire host system.

Example 1: Exporting a General Purpose Mount

In the first example, we’ll create a general-purpose NFS mount that uses default NFS behavior to makes it difficult for a user with root privileges on the client machine to interact with the host using those client superuser privileges. You might use something like this to store the files uploaded using a content management system or to create space for users to easily share project files.

First, make a share directory called nfs:

* sudo mkdir /var/nfs/general -p

Since we’re creating it with sudo, the directory is owned by root here on the host.

* ls -la /var/nfs/general

Output

4 drwxr-xr-x 2 root root 4096 Jul 25 15:26 .

NFS will translate any root operations on the client to the nobody:nogroup credentials as a security measure. Therefore, we need to change the directory ownership to match those credentials.

* sudo chown nobody:nogroup /var/nfs/general

This directory is now ready for export.

Example 2: Exporting the Home Directory

In our second example, the goal is to make user home directories stored on the host available on client servers, while allowing trusted administrators of those client servers the access they need to conveniently manage users.

To do this, we’ll export the /home directory. Since it already exists, we don’t need to create it. We won’t change the permissions, either. If we *did*, it would cause all kinds of issues for anyone with a home directory on the host machine.

Step 3 — Configuring the NFS Exports on the Host Server

Next, we’ll dive into the NFS configuration file to set up the sharing of these resources.

Open the /etc/exports file in your text editor with root privileges:

* sudo nano /etc/exports

The file has comments showing the general structure of each configuration line. The syntax is basically:

/etc/exports

directory\_to\_share client(share\_option1,...,share\_optionN)

We’ll need to create a line for each of the directories that we plan to share. Since our example client has an IP of 203.0.113.256, our lines will look like the following. Be sure to change the IPs to match your client:

/etc/exports

/var/nfs/general 203.0.113.256(rw,sync,no\_subtree\_check)

/home 203.0.113.256(rw,sync,no\_root\_squash,no\_subtree\_check)

We’re using the same configuration options for both directories with the exception of no\_root\_squash. Let’s take a look at what each one means.

* **rw**: This option gives the client computer both read and write access to the volume.
* **sync**: This option forces NFS to write changes to disk before replying. This results in a more stable and consistent environment since the reply reflects the actual state of the remote volume. However, it also reduces the speed of file operations.
* **no\_subtree\_check**: This option prevents subtree checking, which is a process where the host must check whether the file is actually still available in the exported tree for every request. This can cause many problems when a file is renamed while the client has it opened. In almost all cases, it is better to disable subtree checking.
* **no\_root\_squash**: By default, NFS translates requests from a root user remotely into a non-privileged user on the server. This was intended as security feature to prevent a root account on the client from using the file system of the host as root. no\_root\_squash disables this behavior for certain shares.

When you are finished making your changes, save and close the file. Then, to make the shares available to the clients that you configured, restart the NFS server with the following command:

* sudo systemctl restart nfs-kernel-server

Before you can actually use the new shares, however, you’ll need to be sure that traffic to the shares is permitted by firewall rules

Step 4 — Adjusting the Firewall on the Host

First, let’s check the firewall status to see if it’s enabled and if so, to see what’s currently permitted:

* sudo ufw status

Output

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

OpenSSH (v6) ALLOW Anywhere (v6)

On our system, only SSH traffic is being allowed, so we’ll need to add a rule for NFS traffic.

With many applications, you can use sudo ufw app list and enable them by name, but nfs is not one of those. Because ufw also checks /etc/services for the port and protocol of a service, we can still add NFS by name. Best practice recommends that you enable the most restrictive rule that will still allow the traffic you want to permit, so rather than enabling traffic from just anywhere, we’ll be specific.

Use the following command to open port 2049 on the host, being sure to substitute your client’s ip address:

* sudo ufw allow from 203.0.113.256 to any port nfs

You can verify the change by typing:

* sudo ufw status

You should see traffic allowed from port 2049 in the output:

Output

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

2049 ALLOW 203.0.113.256

OpenSSH (v6) ALLOW Anywhere (v6)

This confirms that UFW will only allow NFS traffic on port 2049 from our client machine.

Step 5 — Creating the Mount Points on the Client

Now that the host server is configured and serving its shares, we’ll prepare our client.

In order to make the remote shares available on the client, we need to mount the host directory on an empty client directory.

**Note:** If there are files and directories in your mount point, as soon as you mount the NFS share, they’ll be hidden. Be sure if you mount in a directory that already exists that the directory is empty.

We’ll create two directories for our mounts:

* sudo mkdir -p /nfs/general
* sudo mkdir -p /nfs/home

Step 6 — Mounting the Directories on the Client

Now that we have some place to put the remote shares and we’ve opened the firewall, we can mount the shares by addressing our host server, which in this guide is 203.0.113.0, like this:

* sudo mount 203.0.113.0:/var/nfs/general /nfs/general
* sudo mount 203.0.113.0:/home /nfs/home

These commands should mount the shares from the host computer onto the client machine. You can double-check that they mounted successfully in several ways. You can check this with a plain mount or findmnt command, but df -h will give you more human readable output illustrates how disk usage is displayed differently for the nfs shares:

* df -h

Output

Filesystem Size Used Avail Use% Mounted on

udev 238M 0 238M 0% /dev

tmpfs 49M 628K 49M 2% /run

/dev/vda1 20G 1.2G 18G 7% /

tmpfs 245M 0 245M 0% /dev/shm

tmpfs 5.0M 0 5.0M 0% /run/lock

tmpfs 245M 0 245M 0% /sys/fs/cgroup

tmpfs 49M 0 49M 0% /run/user/0

203.0.113.0:/home 20G 1.2G 18G 7% /nfs/home

203.0.113.0:/var/nfs/general 20G 1.2G 18G 7% /nfs/general

Both of the shares we mounted appear at the bottom. Because they were mounted from the same file system, they show the same disk usage. To see how much space is actually being used under each mount point, use the disk usage command du and the path of the mount. The -s flag will provide a summary of usage rather than displaying the usage for every file. The -h will print human readable output.

For example:

* du -sh /nfs/home

Output

36K /nfs/home

This shows us that the contents of the entire home directory is using only 20K of the available space.

Step 7 — Testing NFS Access

Next, let’s test access to the shares by writing something to each of them.

Example 1: The General Purpose Share

First, write a test file to the /var/nfs/general share.

* sudo touch /nfs/general/general.test

Then, check its ownership:

* ls -l /nfs/general/general.test

Output

-rw-r--r-- 1 nobody nogroup 0 Aug 1 13:31 /nfs/general/general.test

Because we mounted this volume without changing NFS’s default behavior and created the file as the client machine’s root user via the sudo command, ownership of the file defaults to nobody:nogroup. Client superusers won’t be able to perform typical administrative actions, like changing the owner of a file or creating a new directory for a group of users, on this NFS-mounted share.

Example 2: The Home Directory Share

To compare the permissions of the General Purpose share with the Home Directory share, create a file Home Directory the same way:

* sudo touch /nfs/home/home.test

Then look at the ownership of the file:

* ls -l /nfs/home/home.test

Output

-rw-r--r-- 1 root root 0 Aug 1 13:32 /nfs/home/home.test

We created home.test as root via the sudo command, exactly the same way we created the general.test file. However, in this case it is owned by root because we overrode the default behavior when we specified the no\_root\_squash option on this mount. This allows our root users on the client machine to act as root and makes the administration of user accounts much more convenient. At the same time, it means we don’t have to give these users root access on the host.

Step 8 — Mounting the Remote NFS Directories at Boot

We can mount the remote NFS shares automatically at boot by adding them to /etc/fstab file on the client.

Open this file with root privileges in your text editor:

* sudo nano /etc/fstab

At the bottom of the file, we’re going to add a line for each of our shares. They will look like this:

/etc/fstab

. . .

203.0.113.0:/var/nfs/general /nfs/general nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0

203.0.113.0:/home /nfs/home nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0

**Note:** More information about the options we are specifying here can be found in the man page that describes NFS mounting in the fstab with the man nfs command.

The client server will automatically mount the remote partitions at boot, although it may take a few moments for the connection to be made and the shares to be available.

Step 9 — Unmounting an NFS Remote Share

If you no longer want the remote directory to be mounted on your system, you can unmount it by moving out of the share’s directory structure and unmounting, like this:

* cd ~
* sudo umount /nfs/home
* sudo umount /nfs/general

This will remove the remote shares, leaving only your local storage accessible:

* df -h

Output

Filesystem Size Used Avail Use% Mounted on

/dev/vda 59G 1.3G 55G 3% /

none 4.0K 0 4.0K 0% /sys/fs/cgroup

udev 2.0G 12K 2.0G 1% /dev

tmpfs 396M 320K 396M 1% /run

none 5.0M 0 5.0M 0% /run/lock

none 2.0G 0 2.0G 0% /run/shm

none 100M 0 100M 0% /run/user

If you also want to prevent them from being remounted on the next reboot, edit /etc/fstab and either delete the line or comment it out by placing a # symbol at the beginning of the line. You can also prevent auto-mounting by removing the auto option, which will allow you to mount it manually.

## Conclusion

In this tutorial, we created an NFS host and illustrated some key NFS behaviours by creating two different NFS mounts, which we shared with our NFS client. If you’re looking to implement NFS in production, it’s important to note that the protocol itself is not encrypted. In cases where you’re sharing files that are intended to be publicly accessible, that doesn’t cause any serious problems.

If you’re using NFS for private data, however, you’ll need to decide how you want to protect that data. You might be able to route NFS over SSH or a VPN connection to create a more secure experience, but this often comes with a serious loss of performance. If performance is an issue, consider [SSHFS](https://www.digitalocean.com/community/tutorials/how-to-use-sshfs-to-mount-remote-file-systems-over-ssh). It’s slightly slower than unencrypted NFS traffic, but usually much faster than tunnelled NFS. Kerberos authenticated encryption for NFS is another option to explore.