Block Storage Networking Storage Ubuntu

Alex Garnett
Senior DevOps Technical Writer

Introduction

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

Block storage is a generic term used to describe network-based storage volumes that are usually offered by hosting providers. Unlike Object Storage, block storage usually does not provide its own API for direct access. Instead, it needs to be mounted to an existing server and shared from that server. If you are using DigitalOcean's Block Storage, running an NFS server on an attached droplet is a good solution to expose a block storage volume.

In this guide, you'll go over how to install the software needed to host an NFS server, configure two NFS mounts on a server and client, and mount and unmount the remote shares.

Prerequisites

You will use two servers in this tutorial, with one sharing part of its filesystem with the other. To follow along, you will need:

- Two Ubuntu servers. This tutorial will follow our best practices for Ubuntu 22.04, but most recent Ubuntu or Debian releases should also work. Each of these should have a non-**root** user with `sudo` privileges, a firewall set up with UFW, and private networking if it's available to you.
  - For assistance setting up a non-**root** user with `sudo` privileges and a firewall, follow our Initial Server Setup with Ubuntu 22.04 guide.
  - If you're using DigitalOcean Droplets for your server and client, you can read more about setting up a private network in our documentation on How to Create a VPC. DigitalOcean

droplets created since 2019 will automatically have a private network interface enabled.

Throughout this tutorial, the server that shares its directories will be referred to as the **host** and the server that mounts these directories as the **client**. The **host** server should have a block storage volume attached to it. If you are using DigitalOcean block storage, you can follow the documentation for creating and mounting a volume. You will also need to know the IP address for both servers. Other than when initially connecting over SSH, be sure to use the *private* network address, if available.

For ideal performance, it is recommended to use Premium Dedicated Droplets (PDDs) with 10gbit networking for both the NFS server and client(s)

- Using 10gbit networking will allow the NFS performance to come close to the published volume limits.
- Without a 10gbit droplet, performance will cap out at 2gbps

Throughout this tutorial, these IP addresses will be referred to by the placeholders `host_ip` and `client_ip`. Please substitute as needed.

Step 1 – Downloading and Installing the Components

You'll begin by installing the necessary components on each server.

On the Host

On the **host** server, install the `nfs-kernel-server` package, which will allow you to share your directories. Since this is the first operation that you're performing with `apt` in this session, refresh your local package index before the installation:

```
host:$ sudo apt update
host:$ sudo apt install nfs-kernel-server
```

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

On the Client

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

```
client:$ sudo apt update
client:$ sudo apt install nfs-common
```

Note: It is also possible to connect to NFS shares on other platforms such as Windows or macOS by using built-in OS functionality. The `nfs-common` example is for Ubuntu servers.

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

Step 2 – Creating the Share Directories on the Host

In this tutorial, you'll create a general-purpose NFS mount that uses default NFS behaviour to make 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 files that were uploaded using a content management system or to create space for users to easily share project files.

Assuming your block storage is mounted to your **host** at a path like `/mnt/volume-nyc3-01`, you can make a directory to share within that volume called `nfs`.

First, make the share directory:

```
host:$ sudo mkdir -p /mnt/volume-nyc3-01/nfs
```

Since you're creating it with `sudo`, the directory is owned by the **host**'s **root** user:

```
host:$ ls -dl /mnt/volume-nyc3-01/nfs/
```

```
Output
drwxr-xr-x 2 root root 4096 Sep 27 16:19 /mnt/volume-nyc3-01/nfs/
```

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

```
host:$ sudo chown nobody:nogroup /mnt/volume-nyc3-01/nfs/
```

You're now ready to export this directory.

Step 3 – Configuring the NFS Exports on the Host Server

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

On the **host** machine, open the `/etc/exports` file in `nano` or your preferred text editor with **root** privileges:

```
host:$ sudo nano /etc/exports
```

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

/etc/exports

```
directory_to_share    client(share_option1,...,share_optionN)
```

You'll need to create a line for each of the directories that you plan to share. Be sure to change the `client_ip` placeholder shown here to your actual IP address:

/etc/exports

```
/mnt/volume-nyc3-01/nfs/    client_ip(rw,sync,no_subtree_check)
```

Here, you're using the same configuration options for both directories with the exception of `no_root_squash`. Take a look at what each of these options 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 a 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 behaviour for certain shares.

When you are finished making your changes, save and close the file. If you are using `nano`, press `Ctrl+X`, then when prompted, `Y` and then Enter. Then, to make the share available to the clients that you configured, restart the NFS server with the following command:

```
host:$ sudo systemctl restart nfs-kernel-server
```

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

Step 4 – Adjusting the Firewall on the Host

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

```
host:$ sudo ufw status
```

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

On your system, only SSH traffic is being allowed through, so you'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. However, because `ufw` also checks `/etc/services` for the port and protocol of service, you 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, you'll be specific.

Use the following command to open port `2049` on the **host**, being sure to substitute your **client** IP address:

```
host:$ sudo ufw allow from client_ip to any port nfs
```

You can verify the change by typing:

Products ›

Create a directory for your mounts:

```
client:$ sudo mkdir -p /nfs/general
```

Now that you have a location to put the remote share and you've opened the firewall, you can mount the share using the IP address of your **host** server:

```
client:$ sudo mount host_ip:/mnt/volume-nyc3-01/nfs/ /nfs/general
```

One can use the `-o nconnect=n` flag when mounting NFS share from the client to boost IOPS for certain workloads. Where "n" is the number of connections to establish between this client and the target NFS Server. The number can be from 1 to 16. You can experiment with different `nconnect` values that best suit your workload, perhaps starting with a value of `8`. Setting the `nconnect` option may provide a *slight* boost in IOPS for some workloads - specifically small write operations.

This command will mount the share from the host computer onto the **client** machine. You can double-check that it mounted successfully in several ways. You can check this with a `mount` or `findmnt` command, but `df -h` provides a more readable output:

```
client:$ df -h
```

```
Output
Filesystem                     Size  Used Avail Use% Mounted on
tmpfs                          198M  972K  197M   1% /run
```

```
/dev/vda1                        50G  3.5G   47G   7% /
tmpfs                           989M     0  989M   0% /dev/shm
tmpfs                           5.0M     0  5.0M   0% /run/lock
/dev/vda15                      105M  5.3M  100M   5% /boot/efi
tmpfs                           198M  4.0K  198M   1% /run/user/1000
10.124.0.3:/mnt/volume-nyc3-01/nfs/   25G  5.9G   19G  24% /nfs/general
```

The share you mounted appears at the bottom. 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 provides a summary of usage rather than displaying the usage for every file. The `-h` prints human-readable output.

For example:

```
client:$ du -sh /nfs/general
```

```
Output
4.0K    /nfs/general
```

This shows us that the contents of the entire home directory is using only 4K of the available space.

Step 6 – Testing NFS Access

Next, test access to the share by writing something to it.

First, write a test file to the `/mnt/volume-nyc3-01/nfs/` share:

```
client:$ sudo touch /nfs/general/test
```

Then, check its ownership:

```
client:$ ls -l /nfs/general/test
```

```
Output
-rw-r--r-- 1 nobody nogroup 0 Sep 28 18:05 /nfs/general/test
```

Because you 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.

If you want to run performance testing on your NFS mount from the client, you should follow the config instructions found at How To Benchmark DigitalOcean Volumes | DigitalOcean.

Step 7 – Mounting the Remote NFS Directories at Boot

You can mount the remote NFS share automatically at boot by adding them to `/etc/fstab` file on the **client**.

Open this file with root privileges in your text editor:

```
client:$ sudo nano /etc/fstab
```

At the bottom of the file, add a line for each of your shares. They will look like this:

/etc/fstab

```
. . .
host_ip:/mnt/volume-nyc3-01/nfs/    /nfs/general   nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1
```

Note: You can find more information about the options you are specifying here in the NFS man page. You can access this by running the following command:

```
$ man nfs
```

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

Step 8 – Unmounting an NFS Remote Share

If you no longer want a 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:

```
client:$ cd ~
client:$ sudo umount /nfs/general
```

Take note that the command is named `umount` not `unmount` as you may expect.

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

```
client:$ df -h
```

```
Output
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           198M  972K  197M   1% /run
/dev/vda1        50G  3.5G   47G   7% /
tmpfs           989M     0  989M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/vda15      105M  5.3M  100M   5% /boot/efi
tmpfs           198M  4.0K  198M   1% /run/user/1000
```

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 `#` character at the beginning of the line. You can also prevent auto-mounting by removing the `auto` option, which will allow you to still mount it manually.

Conclusion