NFS In OpenStack

And How To Set Up NFS on Ubuntu

Introduction

NFS, or **N**etwork **F**ile **S**ystem, is a distributed file system protocol that allows you to mount remote *Host* directories on your *Client* server. NFS is an RPC based protocol, making Remote Procedure Calls between the Client and Host. This lets an OpenStack environment leverage storage space in a remote location and facilitates easy access to that storage space from multiple Client servers. NFS works well for directories that will have to be accessed regularly.

In this guide, we'll cover how to configure NFS mounts on an Ubuntu 12.04 server. This should also work on 14.04.

Prerequisites

During this guide, we will be configuring directory sharing between two Ubuntu servers. These can be of any size. For each of these servers, you will have to have an account set up with sudo privileges.

For the purposes of this guide, we are going to refer to the server that is going to be sharing its directories as the Host and the server that will mount these directories as the Client. If you are used to iSCSI, the Client corresponds to the SCSI Initiator and the Host corresponds to the SCSI Target; the Initiator sends requests to the Target.

In order to keep these servers straight throughout the guide, I will be using the following IP addresses as stand-ins for the Host and Client values:

```
Host:   192.168.10.3    (glenng@nfs-server)
Client: 192.168.10.11   (glenng@openstack1)
```

You should substitute the values above with your own Host and Client values.

Setup

Download and Install the Components

Before we can begin setting up NFS, we need to install the necessary components on both our Host and Client servers.

Host Server

On the Host server (the NFS Server side), we need to install the nfs-kernel-server package, which will allow us to listen for Client requests and 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
```

Now let's install the NFS server component:

```
glenng@nfs-server:~$ sudo apt-get install nfs-kernel-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nfs-kernel-server
0 upgraded, 1 newly installed, 0 to remove and 80 not upgraded.
Need to get 124 kB of archives.
After this operation, 560 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main nfs-kernel-
server amd64 1:1.2.5-3ubuntu3.1 [124 kB]
Fetched 124 kB in 0s (154 kB/s)
Selecting previously unselected package nfs-kernel-server.
(Reading database ... 115458 files and directories currently installed.)
Unpacking nfs-kernel-server (from .../nfs-kernel-server_1%3a1.2.5-
3ubuntu3.1_amd64.deb) ...
Processing triggers for ureadahead ...
Processing triggers for man-db ...
Setting up nfs-kernel-server (1:1.2.5-3ubuntu3.1) ...

Creating config file /etc/exports with new version

Creating config file /etc/default/nfs-kernel-server with new version
 * Exporting directories for NFS kernel daemon...  [ OK ]
 * Starting NFS kernel daemon                      [ OK ]
glenng@nfs-server:~$
```

Notice that the NFS kernel daemon started. Once these packages are installed, you can switch over to the client computer.

Client Server

On the client machine, we're going to have to install a package called nfs-common, which provides NFS functionality without having to include the 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

glenng@openstack1:~$ sudo apt-get update
  .....
  .....
glenng@openstack1:~$ sudo apt-get install nfs-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
 .....
Setting up nfs-common (1:1.2.8-6ubuntu1.1) ...

Creating config file /etc/idmapd.conf with new version

Creating config file /etc/default/nfs-common with new version
```

```
Adding system user `statd' (UID 122) ...
Adding new user `statd' (UID 122) with group `nogroup' ...
Not creating home directory `/var/lib/nfs'.
statd start/running, process 13292
gssd stop/pre-start, process 13326
idmapd start/running, process 13373
Processing triggers for libc-bin (2.19-0ubuntu6.3) ...
Processing triggers for ureadahead (0.100.0-16) ...
glenng@openstack1:~$
```

Note the creation of the /etc/idmapd and /etc/default/nfs-common files.

The /etc/idmapd.conf file is where you configure the NFS Domain name that your server is using: this allows the client and server to understand usernames and so on.

The /etc/default/nfs-common file, add an entry of "NEED_IDMAPD=yes" to get the ID Mapd services to start after a reboot.

Creating the NFS Environment

Create the Share Directory on the Host Server

Go back to the Host server. We're going to experiment with sharing a directory during this guide. The directory we're going to share is a general-purpose directory that we're creating specifically for NFS so that we can demonstrate the proper procedures and settings. This will be located at /var/nfs.

```
sudo mkdir /var/nfs
```

Now, we have a new directory designated specifically for sharing with remote clients. Let's open the permission for our simple use case.

```
sudo chmod 777 /var/nfs
```

Configure the NFS Exports on the Host Server

Now that we have our directories created and assigned, we can dive into the NFS configuration file to set up the sharing of these resources. For a client to be able to access this directory, it must be exported to them (made visible).

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

```
sudo vi /etc/exports
```

The files that you see will have some comments that will show you the general structure of each configuration line. Basically, the syntax for NFS version 3 is something like:

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

So we want to create a line for each of the directories that we wish to share. Since in this example or client has an IP of 192.168.10.11, our lines will look like this:

```
/var/nfs    192.188.10.11(rw,sync,no_subtree_check,no_root_squash)
```

We've explained everything thus far except for the specific options we've enabled. Let's go over those now.

- **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.
- **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 supposed to be a security feature by not allowing a root account on the client to use the filesystem of the host as root. This directive disables this for certain shares.

When you are finished making your changes, save and close the file.

Next, you should create the NFS table that holds the exports of your shares by typing:

```
sudo exportfs -a
```

This tells NFS to export “all” the configured shares such that any client may see what we are sharing.

Note that the NFS service is not actually running yet; we need to start it by typing:

```
sudo service nfs-kernel-server start
```

Now the NFS Server is running and your shares will be available to the clients that you configured.

Create the Mount Points and Mount Remote Shares on the Client Server

Now that your host server is configured and making its shared directory(s) available, an NFS Client can mount the directories being shared.

Keep in mind that the Host has specific shared directory locations, known as **share points**; the Client is going to mount these share points at client specific **mount points**. The share points are specific to the NFS Server and all clients must use them as defined by the server. The general syntax for doing this is:

```
mount <share point> <mount point>
```

Okay, so let's prep our Client. Go back top the client machine. If we’re going to mount the remote shares, we need to create some mount points. The mount points are client specific: they are only of meaning to this client. We'll use the traditional /mnt as a starting point and create a directory called nfs under it to keep our mounts consolidated. Then we'll create an actual directory that corresponds with their location on the host server share points. We can create each directory, and the necessary parent directories, by typing this:

```
sudo mkdir -p /mnt/nfs/var/nfs
```

Now that we have some place to store our mount points, we can mount the share point(s) via addressing our Host server, which in this guide is 192.168.10.3, like this:

```
sudo mount 192.168.10.3:/var/nfs /mnt/nfs/var/nfs
glenng@openstack1:~$ ls -l /mnt/nfs/var/nfs
total 0
glenng@openstack1:~$
```

Anything the client wants to do after mounting is done at the mount point. Let’s write something at our mount point, which ends up on our share point.

```
glenng@openstack1:~$ ls -l / > /mnt/nfs/var/nfs/client_listing.txt
```

Let’s go back to our Host server and look at the shared directory.

```
glenng@nfs-server:~$ ls -l /var/nfs
total 4
-rw-rw-r-- 1 glenng glenng 1399 Jul 15 18:49 client_listing.txt
glenng@nfs-server:~$
```

Remember, /var/nfs is on the Host server (the NFS Server) and we are seeing what the NFS Client has placed herein. We can even look into the file at the server side:

```
glenng@nfs-server:~$ ls -l /var/nfs
total 4 -rw-rw-r-- 1 glenng glenng 1399 Jul 23 15:02 client_listing.txt
glenng@nfs-server:~$ cat /var/nfs/client_listing.txt
total 80
drwxr-xr-x   2 root root  4096 Feb 27 12:28 bin
drwxr-xr-x   3 root root  4096 Feb 27 12:30 boot
drwxr-xr-x  14 root root  3980 Jul 23 14:12 dev
drwxr-xr-x 100 root root  4096 Jul 23 15:02 etc
rwxr-xr-x   3 root root  4096 Nov 15  2013 home
lrwxrwxrwx   1 root root    33 Feb 27 12:29 initrd.img -> /boot/initrd.img-
3.8.0-36-generic
lrwxrwxrwx   1 root root    33 Nov 18  2013 initrd.img.old -> /boot/initrd.img-
3.8.0-33-generic
dr wxr-xr-x  19 root root  4096 Feb 27 12:28 lib
drwxr-xr-x   2 root root  4096 Nov 18  2013 lib64
drwx------   2 root root 16384 Nov 15  2013 lost+found
drwxr-xr-x   3 root root  4096 Nov 15  2013 media
drwxr-xr-x   3 root root  4096 Jul 23 15:01 mnt
drwxr-xr-x   2 root root  4096 Nov 15  2013 opt
dr-xr-xr-x  97 root root     0 Jul 23 14:12 proc
drwx------   5 root root  4096 Jan 22 16:45 root
drwxr-xr-x  18 root root   720 Jul 23 14:46 run
drwxr-xr-x   2 root root  4096 Feb 27 12:28 sbin
drwxr-xr-x   2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x   2 root root  4096 Nov 15  2013 srv
dr-xr-xr-x  13 root root     0 Jul 23 14:12 sys
drwxrwxrwt   4 root root  4096 Jul 23 14:17 tmp
drwxr-xr-x  10 root root  4096 Nov 15  2013 usr
drwxr-xr-x  12 root root  4096 Jun  9 14:54 var
lrwxrwxrwx   1 root root    29 Feb 27 12:29 vmlinuz -> boot/vmlinuz-3.8.0-36-
generic
lrwxrwxrwx   1 root root    29 Nov 18  2013 vmlinuz.old -> boot/vmlinuz-3.8.0-
33-generic
glenng@nfs-server:~$
```

This is how multiple clients are all working with data at a common storage point: a "share point".

And thus, we now have a rudimentary NFS Client/Server now setup and operational.

NFS And OpenStack

For OpenStack users, let’s explore what happens. Much of what will transpire is built upon what we have already done: the NFS share point (aka, NFS share) that we created on our NFS Host is again used. The share point is specified in the Cinder configuration.

Going back to our NFS client server, let us manually unmount the share.

glenng@openstack1:~$ sudo umount /mnt/nfs/var/nfs glenng@openstack1:~$

Adding OpenStack

Now let's install OpenStack.

```
glenng@openstack1:~$ git clone https://github.com/openstack-dev/devstack.git
Cloning into 'devstack'... … ...
```

I'll next create a local.conf file. In my local.conf, I include the NFS driver information:

```
enabled_backends = genericNfs [genericNfs]
volume_backend_name=NFS
volume_driver=cinder.volume.drivers.nfs.NfsDriver
nfs_shares_config = /etc/cinder/nfs_shares
```

Note here that I have also specified the "nfs_shares_config" file. This file contains our NFS share point information. This is how Cinder will automatically perform what we manually performed previously; i.e., mounting the share at the NFS Server.

For this to work, we must go ahead and create the /etc/cinder directory and create our shares file. Doing so will let everything be in place when Cinder starts up.

```
glenng@openstack1:~/devstack$ sudo mkdir /etc/cinder
glenng@openstack1:~/devstack$ sudo chown glenng /etc/cinder
glenng@openstack1:~/devstack$ echo '192.168.10.3:/var/nfs' >
/etc/cinder/nfs_shares
```

Notice the above <NFS Server IP>:<NFS share location> is the same as when we manually performed the mounting. We didn't specify a mount point here in our shares file because Cinder will create that on its own.

Now run the stack setup.

```
glenng@openstack1:~/devstack$ ./stack.sh
  . . .
  . . .
```

Okay, we have completed the OpenStack install. Let's look at what Cinder has done for mounting:

```
glenng@openstack1:~$ mount
/dev/vda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev) rpc_
pipefs on /run/rpc_pipefs type rpc_pipefs (rw)
cgroup on /sys/fs/cgroup type tmpfs (rw,uid=0,gid=0,mode=0755)
/opt/stack/data/swift/drives/images/swift.img on
/opt/stack/data/swift/drives/sdb1 type xfs
(rw,noatime,nodiratime,nobarrier,logbufs=8)
```

```
192.168.10.3:/var/nfs on
/opt/stack/data/cinder/mnt/6eeadc9c244dec996c2dc101eb0f8759 type nfs
(rw,vers=4,addr=192.168.10.3,clientaddr=192.168.10.11)
```

We can see that Cinder created a mount point and mounted the NFS Server. Let's look inside our mount point.

```
glenng@openstack1:~$ ls -l
/opt/stack/data/cinder/mnt/6eeadc9c244dec996c2dc101eb0f8759
total 4
-rw-rw-r-- 1 glenng glenng 1399 Jul 23 15:02 client_listing.txt
glenng@openstack1:~$
```

All there is the file we created earlier.

NFS and OpenStack In Action

Our NFS environment is set up. Let's create a volume in Cinder!

```
glenng@openstack1:~$ cinder create 1
+--------------------------------+---------------------------------------+
|            Property            |                 Value                 |
+--------------------------------+---------------------------------------+
|          attachments           |                  []                   |
|       availability_zone        |                 nova                  |
|            bootable            |                false                  |
|           created_at           |      2014-07-23T20:23:02.000000       |
|          description           |                 None                  |
|           encrypted            |                False                  |
|               id               | 6b56292a-2d94-4c52-8884-5be3535025f0 |
|            metadata            |                  {}                   |
|              name              |                 None                  |
|     os-vol-host-attr:host      |                 None                  |
| os-vol-mig-status-attr:migstat |                 None                  |
| os-vol-mig-status-attr:name_id |                 None                  |
|  os-vol-tenant-attr:tenant_id  |   58ea66e241fa4506b0736b04277d3c98    |
|              size              |                   1                   |
|          snapshot_id           |                 None                  |
|          source_volid          |                 None                  |
|             status             |               creating                |
|            user_id             |   9ed23b53acba4837a8c185f341ec0613    |
|          volume_type           |                 None                  |
+--------------------------------+---------------------------------------+
glenng@openstack1:~$ cinder list
+--------------------------------------+-----------+------+------+-------------
+----------+-------------+ |                  ID                  |   Status  |
Name | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+------+------+-------------
+----------+-------------+
| 6b56292a-2d94-4c52-8884-5be3535025f0 | available | None |  1   |     None
|  false   |             |
+--------------------------------------+-----------+------+------+-------------
+----------+-------------+
glenng@openstack1:~$
```

And if we look at our mount point again:

```
glenng@openstack1:~$ls –l
/opt/stack/data/cinder/mnt/6eeadc9c244dec996c2dc101eb0f8759
total 4
-rw-rw-r-- 1 glenng glenng       1399 Jul 23 15:02 client_listing.txt
```

```
-rw-rw-rw- 1 root   root   1073741824 Jul 23 16:23 volume-6b56292a-2d94-4c52-
8884-5be3535025f0
glenng@openstack1:~$
```

We see Cinder’s volume. Looking at the NFS Server:

```
glenng@nfs-server:~$ ls -l /var/nfs
total 4
-rw-rw-r-- 1 glenng glenng       1399 Jul 23 15:02 client_listing.txt
-rw-rw-rw- 1 root   root   1073741824 Jul 23 16:23 volume-6b56292a-2d94-4c52-
8884-5be3535025f0
glenng@nfs-server:~$
```

And there it is!