2014

Hands On Lab

Ubuntu NFS Version 4.0 Server

Table of Contents

Introduction

The Network File System (NFS) is a distributed file system protocol that was originally developed by Sun Microsystems. It allows a client computer to "mount" network folders from a server so that the resulting mount appears and behaves as a local file system to the client.

NFS builds on the Open Network Computing Remote Procedure Call system and is currently an open standard that is defined as an RFC, which allows anyone to implement it. The current version is NFS v4 and is the version we will be working with during this lab.

Goals

This lab will introduce you to the concepts of NFS v4 services from a client and server perspective. By the end of this document, you will have built an Ubuntu 13.10 NFS Server and created a client configuration to test any shared directories for the network client you choose to set up.

Packages, Resources and Prerequisites

The packages involved in our lab are:

- nfs-kernel-server
- rpcbind
- nfs-common

The resources you will be accessing during the course of this lab are:

- Ubuntu 13.10 Server: Test Client
  - You will use this to test your server and shares once completed
- Ubuntu 13.10 Server: NFS Version 4.0 Server
  - This is the server that you will deploy, install and configure NFS on, create your sample shares and add client entries to

Prerequisites to this lab:

- A LinuxAcademy.com Lab+ Subscription
- Internet Access and SSH Client
  - You will need to connect to the public IP of the server in order to configure NFS, the only method of connectivity is over SSH
    - SSH client can be Windows (i.e. Putty) or from another Linux system shell
- Login Information (Provided When The Server Starts Up)

Document Conventions

Just a couple of housekeeping items to go over so you can get the most out of this Hands On Lab without worrying about how to interpret the contents of the document.

When we are demonstrating a command that you are going to type in your shell while connected to a server, you will see a box with a dark blue background and some text, like this:

```
linuxacademy@ip-10-0-0-0:~$ sudo apt-get install package
[sudo] password for linuxacademy: <PASSWORD PROVIDED>
```

That breaks down as follows:

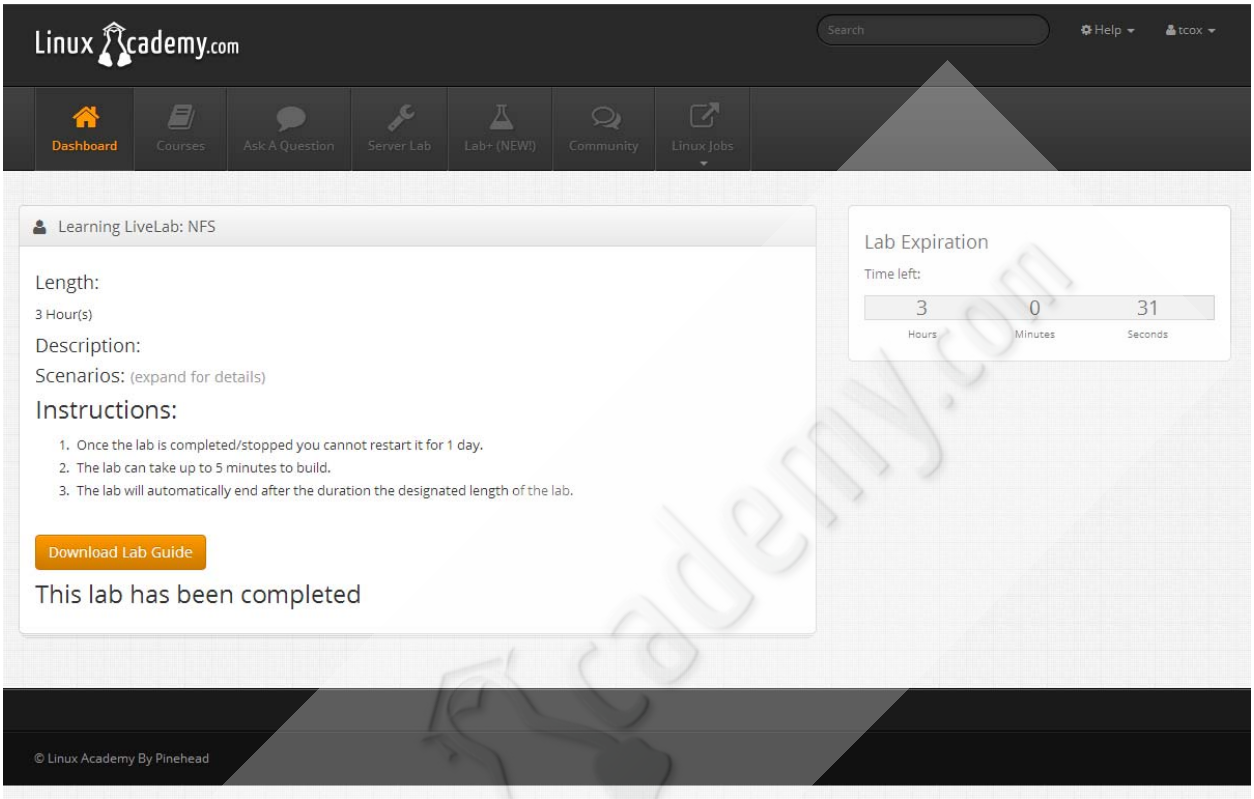
- The white text that looks something like "linuxacademy@ip-10-0-0-0:~$: "should be interpreted as the console prompt your cursor will be sitting at when you are logged into the server. You are not typing that into the console, it is just there. Note that the portion that says "ip-10-0-0-0" will be different for you based on the IP address of your system.
- The bold yellow text in any command example is the actual command you will type in your shell.
- Any lines subsequent to the bold yellow command that you type in is to be considered as the response you can expect from your command entry.

If you do not see the command prompt as the first line in the example, then all the white text is an example of a text, script or configuration file and is intended to be typed in its entirety (in the event you are instructed to create a previously non-existent file) or to compare against the contents of the file on your system.

One final convention, if you see a "~" at the end of a line in a command or text example, that will indicate that the line overflowed the end of line. Meaning you should just keep typing without hitting enter to start a new line until the natural end of the command.

General Process

When you are ready to begin the Hands On Lab, log into your Linux Academy Lab+ subscription and navigate to the "Live Labs" section on the Dashboard. Once you choose the "Ubuntu NFS Version 4.0 Server" Lab from the list, you will see the screen below:

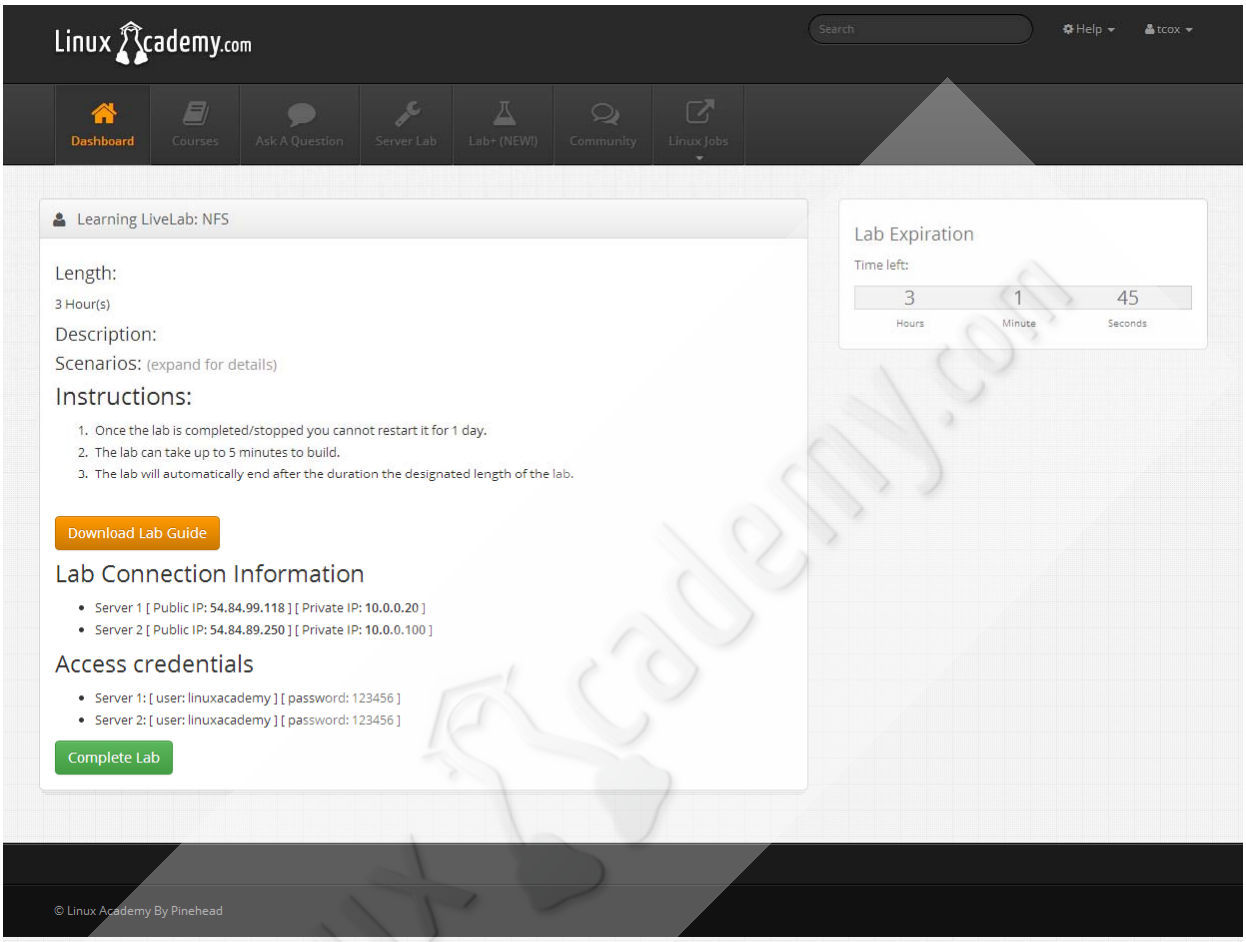


[Linux Academy Live Labs+ - Start Screen]

A few things to note before you start this process:

- When you launch the lab from this screen, it may take up to FIVE MINUTES for your servers to be deployed and be available for your use.
- Do not leave your desktop and come back, once the servers are launched, you will only have TWO HOURS to complete this lab from start to finish. After that point, the servers time out and are deleted permanently. Any and all work that you have done will then be lost.
- You can only use this lab ONCE PER DAY. If you try to use it more than that after completing the lab or the servers timing out, the screen will tell you when it will be available to you again.
- Other than those descriptions, you may retry any of the Labs+ labs as many times as you wish as long as you are a subscriber.

Once you have clicked on the 'Start Lab' button that you see above, a process will launch on our servers that will deploy the two servers we will use in our lab for testing. After a few minutes of processing (and you will see a status message that says "Creating Lab… Please Wait"), you should see a screen that looks like this one:



[Linux Academy Live Labs+ - Start Screen]

You will see all the information you need to access your servers from another system. Specifically, you need:

- The two server public IP addresses
- Access credentials for both

One thing to note is that, in addition to the two IPs that you see above, each server will have another IP assigned to it in the 10.0.0.x subnet. This is a private IP address and will not route outside of your private server pool. Server 1 will have a dynamic private address in the 10.0.0.x subnet while Server 2 will have a static private address of 10.0.0.100.

In our setup, Server 1 will function as an NFS client that we will use to test our shared folders. Server 2 will function as our primary NFS server for the purposes of this lab.

NFS Configuration Types

Although this lab focuses on the most common NFS server configuration (single NFS Server with multiple clients), there are a couple of different ways you can deploy NFS:

- Single Server
  - This is the most common scenario where you will deploy a single server that allows one or more individual clients or networks to have access to one or more folders that can be mounted locally.
- Simple Failover
  - A failover server is simply an identically configured NFS server that can be quickly be activated in the event of the failure of a primary server. It's file shares are often synchronized via CRON and rsync in order to keep the content current. The most common configuration is a secondary network interface that has an identical IP as the primary server but is not active.
- Clustered
  - This is typically where one or more additional nodes exist in a clustered environment. All file systems are kept in sync using a synchronization daemon (like GlusterFS) or via CRON and rsync scripts that run periodically. The cluster shares a common "floating" IP so that if any node fails, the client does not drop connection to the share.

NFS Server – Preparation

The most important thing to remember about your NFS server to be is that it MUST have at least one static IP address that we can bind NFS service to. You may have any number of network interfaces and associated IP addresses as long as at least one is static.

Next, be sure that the hostname (short and fully qualified) exist as an entry in your local hosts file. Using our setup, the private server IP of 10.0.0.100 is going to be our NFS server. On that server, make sure your local /etc/hosts file looks something like this:

```
127.0.0.1          localhost

# The following lines are desirable for IPv6 capable hosts
::1                ip6-localhost ip6-loopback
fe00::0            ip6-localnet

10.0.0.100         ip-10-0-0-100.linuxacademy.lab ip-10-0-0-100
```

The IPv6 information does not have to be there for our purposes since we will not be binding or using IPv6. The line with our local server IP is setting our host file appropriately using the short name and a fully qualified domain name.

Once that is complete, you can test by pinging the short and FQDN of your local host and be sure you get a response with the appropriate IP. If you get any errors regarding "Unknown Host", be sure your IP, hostname and FQDN are saved properly in the /etc/hosts file as listed above.

NFS Server – Installation

Strictly in terms of pure installation steps, there really is only one. You can have everything you need pulled down and installed from the standard Ubuntu repositories by executing the following command:

```
linuxacademy@ip-10-0-0-100:~$ sudo apt-get install ~
nfs-kernel-server rpcbind nfs-common
```

This installs the full NFS server daemon that we need to make our exports (our shares) available as well as the common NFS client library in addition to rpcbind utility (which is a program number to universal address converter, basically it tells other networked machines at what location to find a service). You will notice that once the installation is complete, the NFS daemon will tell you that it is not starting because we do not have any exports configured, this is normal for now.

NFS Server – Service Configuration

Now we need to make some configuration file updates in order to set our service up to be visible on the network. First, we need to create a new file for our "rpcbind" utility. Since we do not need to customize this service for NFS, we just need to explicitly call out that we are not passing any options to the daemon. Using your favorite editor, create a file called "/etc/default/rpcbind" and add the following:

```
OPTIONS=""
```

Once we have that file created and saved, we need to call out the network address (single) or network range (multiple) that can use our service. Again, using your favorite editor, edit the file called "/etc/hosts.allow" and add the following:

```
portmap: 10.0.0.
```

What this does is allow any host on our private network of 10.0.0.x to connect to and request network services from our server. If you like, you can specifically add the full network address of

your second lab server (the client) in order to restrict the connection further. Since we are in a completely private network, we can just allow anyone on the network to connect.

Next up, we need to edit another configuration file called "/etc/default/nfs-common". We are simply going to indicated that we will be using "idmapd" for our NFS configuration. This is required for NFS Version 4. It works in conjunction with "rpcbind" in the name and address resolution that NFS now uses by translating user and group IDs to names (and vice versa). We simply need to add the following to whatever is already in the file:

```
NEED_IDMAPD=YES
```

Once this file is saved, we need to finish up the server configuration setup by editing "/etc/idmapd.conf". This file will complete our configuration by telling NFS where to find several pieces of information about our system.

```
[General]

Verbosity = 0
Pipefs-Directory = /run/rpc_pipefs
# set your own domain here, if id differs from FQDN minus hostname
# Domain = localdomain

[Mapping]

Nobody-User = nobody
Nobody-Group = nogroup
```

Let's define our network shares next (called our "exports").

NFS Server – Export Configuration

Now we have completed our server installation and configured NFS so that our service can start. If you actually try to start the service now, you will get the same message as when we installed the server daemon, there are no exports defined so no need for it to run.

We are going to fix that by creating a folder called "/exports" like so:

```
linuxacademy@ip-10-0-0-100:~$ sudo mkdir /exports
```

Now we need to copy a file so we can check for it on our client after we mount the share. Let's take some of our user bash files and create a text file::

```
linuxacademy@ip-10-0-0-100:~$ cat .bash* > bash.txt && cp bash.txt~
/exports
```

Finally, let's edit our actual export file so that we can start the NFS daemon and make the share available to our client. Edit the file called "/etc/exports" and add the following line to the end:

```
/exports ~
10.0.0.0/255.255.255.0(rw,no_root_squash,no_subtree_check,crossmnt,fsid=
0)
```

This simple file does a number of things. It defines the base directory of our share (in our case the previously created directory we called "/exports"), it provides read and write access to anyone on our allowed client network, it gives remote root/admin users full control over local root owned files (no_root_squash), don't worry about the exported directory being an entire file system (no_subtree_check), allow subdirectories of the exported folder to be seen as subfolders (crossmnt) and finally assume that the volume share exported is a regular file system and not another share or special device (fsid=0).

Having said all that, you could simply use the "(rw,sync)" options and the share would work and most things would behave correctly. However, we are setting up our server using the best practices and most client flexibility. These options will prevent headaches with permissions or strange behavior under certain use cases. For a very comprehensive discussion of these parameters and other options, you can feel free to see the *exports* man pages.

We now save this file and can start our NFS server. Execute the following two commands to start the services needed:

```
linuxacademy@ip-10-0-0-100:~$ sudo /etc/init.d/nfs-kernel-server ~
start
 * Exporting directories for NFS kernel daemon…       [OK]
 * Start NFS kernel daemon                            [OK]

linuxacademy@ip-10-0-0-100:~$ sudo /etc/init.d/rpcbind start
```

The "rpcbind" daemon should already be running after our installation earlier, but we need to be sure. It neither confirms nor denies its status unless you issue the "status" option to the daemon directly to see if it is running.

We now have a fully configured NFS server running on the IP of 10.0.0.100 and serving all files and directories in the directory called "/exports". Let's configure our client and mount the directory.

Configure Client and Verify

We need to install a couple of things on our client so that we can mount the exported shares on our system. Complete the following steps:

```
linuxacademy@ip-10-0-0-162:~$ sudo apt-get install nfs-common ~
rpcbind
```

There is no other client configuration necessary, in fact if we want, we can mount the remote export manually right now by creating a directory to mount it to and executing the mount command:

```
linuxacademy@ip-10-0-0-162:~$ mkdir share
linuxacademy@ip-10-0-0-162:~$ mount.nfs4 10.0.0.100:/ share
```

Voila! If you have done everything right, we can now change to the local "/home/youruser/share" directory and we should see our remotely created and shared "bash.txt" file from earlier!

We can make this permanent (so that the share mounts on reboot automatically) by adding the following line to our "/etc/fstab" file:

```
10.0.0.100:/     /home/youruser/share     nfs4     rw 0 0
```

That's all there is to it. There is a lot you can do with NFS and, like Samba, it works across platforms for Windows, Mac OSX and Linux (and any other *nix you might be using). Version 4 is more stable, better performing and more secure than previous versions while being backward compatible. NFS has been around a long time and that is its main advantage.

While not all older systems will support Samba, almost all will support NFS. Finally, creating large datastores for backup or hosting of virtual files (for example, you can use a large file system via NFS in VMWare's ESXi server to store all your virtual servers on and have a single place to back them up to).

In the following Appendices, you will find the full sample server and client configuration files that we used during the course of our lab. If you copy them EXACTLY and simply substitute the private IP address of your NFS server and client where appropriate, your server will work flawlessly. However, don't let our configuration be the end of your experimenting. Feel free to experiment with additional entries and then validate your configuration.

Thanks for joining us on this ride and feel free to ask any questions you may have. Good luck!

Appendix A – rpcbind, hosts.allow and nfs-common Examples

```
// /etc/rpcbind
OPTIONS=""

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

// /etc/hosts.allow
portmap: 10.0.0.

-----------------------------------------------------------------
// /etc/nfs-common
# Do you want to start the statd daemon? It is not needed for
NFSv4.
NEED_STATD=

# Options for rpc.statd.
# For more information, see rpc.statd(8) or
# http://wiki.debian.org/SecuringNFS
STATDOPTS=

# Do you want to start the gssd daemon? It is required for
# Kerberos mounts.
NEED_GSSD=

# Configure IDMAPD
NEED_IDMAPD=YES
```

Appendix B – idmapd.conf Example

```
[General]

Verbosity = 0
Pipefs-Directory = /run/rpc_pipefs
# set your own domain here, if id differs from FQDN minus
# hostname
# Domain = localdomain

[Mapping]

Nobody-User = nobody
Nobody-Group = nogroup
```

Appendix C – exports Example

```
# /etc/exports: the access control list for filesystems which
# may be exported to NFS clients.  See exports(5).

# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
# hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4
# gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/exports
192.168.1.0/255.255.255.0(rw,no_root_squash,no_subtree_check,cro
ssmnt,fsid=0)
```

NOTE: The export line (the last line in the configuration file) must be on a single line, this line is simply overflowing the length of the line available in this document.

Appendix D – Client /etc/fstab Example

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to
# name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options> <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=ad68db37-d3bf-4314-896d-a54e51371ab7 / ext4 errors=remount-
ro 0 1

# mounting our example NFS export on our client machine
10.0.0.100:/  /home/youruser/share  nfs4  rw  0  0
```