NFS

Network File Sharing

Origins and evolution of NFS

- It was introduced by Sun Microsystems in 1984 to serve *diskless* clients. It is now an open standard.
- The first NFS protocol released to the public was version 2 (NFS2). It was slow because it had to commit every write request to disk (instead of the normal write to memory buffers) before responding to the client.
- The second released version (NFS3) solved this problem by enabling asynchronous writes.
- NFS v4 introduced a lot of new features and capabilities like the ability to deal with firewalls and NAT devices, support for Unicode filenames, support for ACL (access control lists) among many others.
- The way NFS is configured is the same regardless of the version used.
- It is strongly recommended that version 3 or 4 is used.

NFS installation and launching

- You can install the necessary modules using yum or apt-get to install all NFS packages. For example `yum –y install nfs*` or `apt-get install nfs*`
- The above commands will install both server and the client components
- The service can be started by issuing `service nfs start` on Red Hat or service `nfs-kernel-server start` on Ubuntu
- NFS is dependent on another service called `rpcbind`. It must be running for the NFS services to function correctly.

Start sharing

- You can share directories or complete filesystems by editing the /etc/exports file
- This file is structured as follows:
  - Directory   host1()   host2()
  - Each host can be followed by a set of options in the round brackets.
  - IP addresses as well as domain names can be used.
  - Multiple hosts or IP addresses can also be specified by using wildcards. For example For example *.linuxadmin.dev will match all hosts in the linuxadmin.dev domain, while 192.168.0.0/24 will match all hosts in the 192.168.0.0 subnet.
  - Comments can be made by preceding them with #

/etc/exports options

- You can fine tune the way you share directories by specifying the options per host(s). Here are some examples:
  - ro: read-only; prevent clients from uploading or changing files in the shared directory.
  - all_squash: the squash option is used to prevent users from accessing files on the shared directory with their original usernames and user ID's. Those are substituted by the user id 65534 and username nfsnobody (nobody in Ubuntu and Debian). This ensures that the user will have limited access.
  - This nfsnobody user can be configured using anonuid=xx and anongid=xx to set the UID and GID for the anonymous user respectively.
  - root_squash: the root user only is treated as nfsnobody while the rest of the users are treated using their own usernames and ID's
  - no_root_squash: allow the root user to access the shared directory are root with UID 0. This option should be carefully considered before setting.
- If you set the usernames and UID's of users to be the same on the client and server, users can have *roaming* home directories if they are shared on a central servers.

Effecting changes

- After you make any changes to the `/etc/exports` file, you needn't restart the whole NFS services to effect those changes, you just need to run the `exportfs` command. For example `exportfs –arv` where `–a` indicates that all entries in the file should be exported, `-r` re-syncs shared entries, it adds new exports and ensures that commented or deleted lines are no longer available. `–v` gives you a verbose output.
- This command must be run as root
- If you restart the machine or restart the NFS services, the command is run automatically.

NFS security concerns

- Because NFS is one of the oldest protocols used for file sharing in Linux, and since – back then – security was not as a big issue as it is nowadays, NFS up to version 3 suffers from serious security vulnerabilities like for example:
  - Although the root user on the remote client machine cannot gain root access on the destination machine by using the root_squash option, the root user on the destination machine can have access to files owned by other users in the shared directory.
  - Users are identified in NFS by their UID rather than by usernames. This means that if user John is bearing the ID of 501 created some files in the shared NFS directory, a user Peter, who coincidentally has the same UID of 501 can and will have access to John's file as if he was the owner.
  - Communication between clients and the server are unencrypted. Any network sniffing tool can reveal the information coming and going from and to the shared directories.
  - Directory mapping is exposed to the clients. For example, if a share is on /opt/myshare, this is the physical location on the server. Such information may be used by hackers to attack the server.

NFSv4 to the rescue

- With the introduction of NFS v4, most of the security issues suffered from by previous versions were considered. For example:
  - Using a Kerberos server you can integrate NFS user-logging with Kerberos-based authentication, where each user will obtain a ticket and use it to access the share.
  - NFSv4 binds the locations of the shared directories to /exports, thus hiding the real physical path on the server.
  - Linux security tools like TCP wrappers, iptables, and SElinux can manage to increase the security of NFS

Configuring iptables to allow NFS traffic

- Because NFS relies on a number of services to operate, the firewall must be configured to allow different TCP and UDP ports to be open.
- The following ports in particular must be opened: (TCP and UDP):
  - 111
  - 2049
  - 20048
- But NFS uses other ports that those to operate, those ports are randomly changed, which makes it impossible for any firewall to control except if they were locked down to specific port numbers. This can be done by editing the `/etc/sysconfig/nfs` file, un-commenting the lines specifying the following lines, giving them specific ports:

```
RQUOTAD_PORT=49001
LOCKD_TCPPORT=49002
LOCKD_UDPPORT=49003
MOUNTD_PORT=49004
STATD_PORT=49005
STATD_OUTGOING_PORT=49006
RDMA_PORT=20049
```

- Then adding rules to the firewall to allow traffic through those ports accordingly.

Using the shares

- You can access an NFS share on a remote file server by mounting it the same way you mount any filesystem using the mount command. For example: `mount RedHat03:/export/share /mnt/share`
- The mount entries can also be added in /etc/fstab file to be mounted automatically on system start. The format is as follows:

```
host:directory    mountpoint    nfs        options        0   0
```

- You can use the `noauto` option in `/etc/fstab` file to disable mounting the share on system boot and forcing it be mounted manually.
- Remote shares can be discovered using the `showmount` command. For example: `showmount -e RedHat03.linuxadmin.dev` will show the available shares and the machines that are allowed to access this share.

Mount options

- Mounting filestems in general can be controlled by several options either when using the mount command or when editing the /etc/fstab file. Those options are useful in many occasions like in the following examples:
  - ro: for mounting the filesystem in read-only mode
  - noauto: do not mount the filesystem on system boot. You will have to manually mount it using a stripped down mount command: just `mount /filesystem`
  - hard: if the NFS server becomes unreachable while a process is actively using it, the process will wait until the server is back up instead of reporting an I/O error. This is used when using data-critical systems like databases. It is enabled by default.
  - soft: if the NFS server goes down while being used by a process, after a defined period of time, the process will report an I/O error.
  - rsize=#: the size of data blocks (in bytes) used when reading data from an NFS server. The default is 1024 but you can set it to a larger value on a fast, reliable network such as LAN.
  - wsize=#: the same as rsize but used when writing data.
  - retry=#: the number of minutes to keep retrying to mount a failed NFS mount point. The default is 10000.
  - bg: when a mount attempt fails or times out, continue to mount the rest of the filesystems in the background. That is, do not wait for this mount operation to complete. This is very useful on a slow or unreliable network; as a server may experience a very long boot time trying to mount NFS shares.

Mounting on demand

- Using the autofs service, you can automatically mount remote filesystems.
- The service can be installed by using yum or apt-get on Red Hat or Debian systems.
- You edit the /etc/auto.master and uncomment the line that reads
  ```
  /net        -hosts
  ```
  which will enable the service
- After start the service you can you can navigate to the /net directory where you will find all mountable NFS shares.
- As soon as you cd into the directory, it is automatically mounted.

LAB: using autofs to create a roaming user account

- Target: Create a user's home directory on a centralized server and export it using NFS. The user can login to any other server, and using the autofs, he can have the same home directory.
- On the server, create the directory that will hold the user's home directory:
  ```
  mkdir /home/nfs/johndoe
  ```
- Create a user and take note of the UID:
  ```
  useradd –d /home/nfs/johndoe  johndoe
  ```
- Export the directory to the local network/domain: in /etc/exports
  ```
  /home/nfs *.linuxadmin.dev(rw)
  ```
- Apply the changes by running exportfs –arv or restart the NFS services
- Add an entry to the /etc/auto.master to identify the mount point:
  ```
  /home/nfs      /etc/auto.johndoe
  ```
- On the client machine, create the file /etc/auto.johndoe containing the following:
  ```
  johndoe          -rw          RedHat03:/home/nfs/johndoe
  ```
- Restart the autofs service
- On the client system, create a user with the same UID as the one you created on the NSF server: useradd –u 501 –d /home/nfs/johndoe johndoe
- Create some files and ensure that those files are available on the NFS server share