NFS

(Redirected from Nfs)

From Wikipedia:

> Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network in a manner similar to how local storage is accessed.

Related articles

NFS/Troubleshooting

Note:

- NFS is not encrypted. Tunnel NFS through an encrypted protocol like Kerberos or (secure) VPN when dealing with sensitive data.
- Unlike Samba, NFS does not have any user authentication by default, client access is restricted by their IP-address/hostname.
- NFS expects the user and/or user group IDs are the same on both the client and server. Enable NFSv4 idmapping or overrule the UID/GID manually by using `anonuid` / `anongid` together with `all_squash` in `/etc/exports`.
- NFS does not support POSIX ACLs.

1 Installation

Both client and server only require the installation of the **nfs-utils (https://archlinux.org/packages/?name=nfs-utils)** package.

It is **highly** recommended to use a time synchronization daemon to keep client/server clocks in sync. Without accurate clocks on all nodes, NFS can introduce unwanted delays.

2 Configuration

2.1 Server

Global configuration options are set in `/etc/nfs.conf`. Users of simple configurations should not need to edit this file.

The NFS server needs a list of directories to share, in the form of exports (see **exports(5) (https://man.archlinux.org/man/exports.5)** for details) which one must define in `/etc/exports` or `/etc/exports.d/*.exports`. These shares are relative to the so-called NFS root. A good security practice is to define a NFS root in a discrete directory tree which will keep users limited to that mount point. Bind mounts are used to link the share mount point to the actual directory elsewhere on the filesystem.

Consider this following example wherein:

1. The NFS root is `/srv/nfs`.
2. The export is `/srv/nfs/music` via a bind mount to the actual target `/mnt/music`.

```
# mkdir -p /srv/nfs/music /mnt/music
# mount --bind /mnt/music /srv/nfs/music
```

Note: ZFS filesystems require special handling of bindmounts, see ZFS#Bind mount.

To make the bind mount persistent across reboots, add it to fstab:

/etc/fstab

```
/mnt/music /srv/nfs/music  none   bind   0   0
```

Add directories to be shared and limit them to a range of addresses via a CIDR or hostname(s) of client machines that will be allowed to mount them in `/etc/exports`, e.g.:

/etc/exports

```
/srv/nfs        192.168.1.0/24(rw,sync,crossmnt,fsid=0)
/srv/nfs/music  192.168.1.0/24(rw,sync)
/srv/nfs/home   192.168.1.0/24(rw,sync,nohide)
/srv/nfs/public 192.168.1.0/24(ro,all_squash,insecure) desktop(rw,sync,all_squash,anonuid=99,anongid=99)
# map to user/group - in this case nobody
```

Note: When using NFSv4, the nfs root directory is specified by the entry denoted by `fsid=0`, other directories must be below it. The `rootdir` option in the `/etc/nfs.conf` file has no effect on this.

Tip:

- The `crossmnt` option makes it possible for clients to access **all** filesystems mounted on a filesystem marked with `crossmnt` and clients will not be required to mount every child export separately. Note this may not be desirable if a child is shared with a different range of addresses.
- Instead of `crossmnt`, one can also use the `nohide` option on child exports so that they can be automatically mounted when a client mounts the root export. Being different from `crossmnt`, `nohide` still respects address ranges of child exports.
- The `insecure` option allows clients to connect from ports above 1023. (Presumably only the root user can use low-numbered ports, so blocking other ports by default creates a superficial barrier to access. In practice neither omitting nor including the `insecure` option provides any meaningful improvement or detriment to security.)
- Use an asterisk (`*`) to allow access from any interface.

It should be noted that modifying `/etc/exports` while the server is running will require a re-export for changes to take effect:

```
# exportfs -arv
```

To view the current loaded exports state in more detail, use:

```
# exportfs -v
```

For more information about all available options see **exports(5) (https://man.archlinux.org/man/exports.5)**.

Tip: ip2cidr (https://ip2cidr.com/) is a tool to convert IP address ranges to correctly structured CIDR specifications.

Note: If the target export is a tmpfs filesystem, the `fsid=1` option is required.

2.1.1 Starting the server

- To run a server using protocol version 3, start and enable `nfs-server.service`.
- To run a server using protocol version 4, start and enable `nfsv4-server.service`.

Users of protocol version 4 exports will probably want to mask at a minimum both `rpcbind.service` and `rpcbind.socket` to prevent superfluous services from running. See FS#76453 (https://bugs.archlinux.org/task/76453). Additionally, consider masking `nfs-server.service` which pulled in for some reason as well.

Note: If exporting ZFS shares, also start/enable `zfs-share.service`. Without this, ZFS shares will no longer be exported after a reboot. See ZFS#NFS.

2.1.2 Restricting NFS to interfaces/IPs

By default, starting `nfs-server.service` will listen for connections on all network interfaces, regardless of `/etc/exports`. This can be changed by defining which IPs and/or hostnames to listen on.

/etc/nfs.conf

```
[nfsd]
host=192.168.1.123
# Alternatively, use the hostname.
# host=myhostname
```

Restart `nfs-server.service` to apply the changes immediately.

2.1.3 Firewall configuration

To enable access through a firewall, TCP and UDP ports `111`, `2049`, and `20048` may need to be opened when using the default configuration; use `rpcinfo -p` to examine the exact ports in use on the server:

```
$ rpcinfo -p | grep nfs
```

```
100003    3   tcp   2049  nfs
100003    4   tcp   2049  nfs
100227    3   tcp   2049  nfs_acl
```

When using NFSv4, make sure TCP port `2049` is open. No other port opening should be required:

/etc/iptables/iptables.rules

```
-A INPUT -p tcp -m tcp --dport 2049 -j ACCEPT
```

When using an older NFS version, make sure other ports are open:

```
# iptables -A INPUT -p tcp -m tcp --dport 111 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 2049 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 20048 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 111 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 2049 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 20048 -j ACCEPT
```

To have this configuration load on every system start, edit `/etc/iptables/iptables.rules` to include the following lines:

```
/etc/iptables/iptables.rules
```

```
-A INPUT -p tcp -m tcp --dport 111 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 2049 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 20048 -j ACCEPT
-A INPUT -p udp -m udp --dport 111 -j ACCEPT
-A INPUT -p udp -m udp --dport 2049 -j ACCEPT
-A INPUT -p udp -m udp --dport 20048 -j ACCEPT
```

The previous commands can be saved by executing:

```
# iptables-save > /etc/iptables/iptables.rules
```

Warning: This command will **override** the current iptables start configuration with the current iptables configuration!

If using NFSv3 and the above listed static ports for `rpc.statd` and `lockd` the following ports may also need to be added to the configuration:

```
/etc/iptables/iptables.rules
```

```
-A INPUT -p tcp -m tcp --dport 32765 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 32803 -j ACCEPT
-A INPUT -p udp -m udp --dport 32765 -j ACCEPT
-A INPUT -p udp -m udp --dport 32803 -j ACCEPT
```

To apply changes, Restart `iptables.service`.

2.1.4 Enabling NFSv4 idmapping

Note:

- NFSv4 idmapping does not work with the default `sec=sys` mount option. [1] (https://web.archive.org/web/20220602190451/https://dfusion.com.au/wiki/tiki-index.php?page=Why+NFSv4+UID+mapping+breaks+with+AUTH_UNIX)
- NFSv4 idmapping needs to be enabled on **both** the client and server.
- Another option is to make sure the user and group IDs (UID and GID) match on both the client and server.
- Enabling/starting `nfs-idmapd.service` should **not** be needed as it has been replaced with a new id mapper:

```
# dmesg | grep id_resolver
```

```
[ 3238.356001] NFS: Registering the id_resolver key type
[ 3238.356009] Key type id_resolver registered
```

The NFSv4 protocol represents the local system's UID and GID values on the wire as strings of the form `user@domain`. The process of translating from UID to string and string to UID is referred to as *ID mapping*. See **nfsidmap(8) (https://man.archlinux.org/man/nfsidmap.8)** for details.

Even though idmapd may be running, it may not be fully enabled. If `/sys/module/nfs/parameters/nfs4_disable_idmapping` or `/sys/module/nfsd/parameters/nfs4_disable_idmapping` returns `Y` on a client/server, enable it by:

Note: The kernel modules `nfs4` and `nfsd` need to be loaded (respectively) for the following paths to be available.

On the client:

```
# echo N > /sys/module/nfs/parameters/nfs4_disable_idmapping
```

On the server:

```
# echo N > /sys/module/nfsd/parameters/nfs4_disable_idmapping
```

Set as **module option** to make this change permanent, i.e.:

```
/etc/modprobe.d/nfsd.conf
```

```
options nfs nfs4_disable_idmapping=0
options nfsd nfs4_disable_idmapping=0
```

To fully use *idmapping*, make sure the domain is configured in `/etc/idmapd.conf` on **both** the server and the client:

```
/etc/idmapd.conf
```

```
# The following should be set to the local NFSv4 domain name
# The default is the host's DNS domain name.
Domain = domain.tld
```

See **[2] (https://unix.stackexchange.com/a/464950)** for details.

2.2 Client

Users intending to use NFS4 with **Kerberos** need to **start** and **enable** `nfs-client.target`.

2.2.1 Manual mounting

For NFSv3 use this command to show the server's exported file systems:

```
$ showmount -e servername
```

For NFSv4 mount the root NFS directory and look around for available mounts:

```
# mount servername:/ /mountpoint/on/client
```

Then mount omitting the server's NFS export root:

```
# mount -t nfs -o vers=4 servername:/music /mountpoint/on/client
```

If mount fails try including the server's export root (required for Debian/RHEL/SLES, some distributions need `-t nfs4` instead of `-t nfs`):

```
# mount -t nfs -o vers=4 servername:/srv/nfs/music /mountpoint/on/client
```

Note: `servername` needs to be replaced with a valid hostname (not just IP address). Otherwise mounting of remote share will hang.

2.2.2 Mount using /etc/fstab

Using **fstab** is useful for a server which is always on, and the NFS shares are available whenever the client boots up. Edit `/etc/fstab` file, and add an appropriate line reflecting the setup. Again, the server's NFS export root is omitted.

/etc/fstab

```
servername:/music   /mountpoint/on/client   nfs   defaults,timeo=900,retrans=5,_netdev  0 0
```

Note: Consult **nfs(5) (https://man.archlinux.org/man/nfs.5)** and **mount(8) (https://man.archlinux.org/man/mount.8)** for more mount options.

Some additional mount options to consider:

rsize and wsize
: The `rsize` value is the number of bytes used when reading from the server. The `wsize` value is the number of bytes used when writing to the server. By default, if these options are not specified, the client and server negotiate the largest values they can both support (see **nfs(5) (https://man.archlinux.org/man/nfs.5)** for details). After changing these values, it is recommended to test the performance (see **#Performance tuning**).

soft or hard
: Determines the recovery behaviour of the NFS client after an NFS request times out. If neither option is specified (or if the `hard` option is specified), NFS requests are retried indefinitely. If the `soft` option is specified, then the NFS client fails an NFS request after *retrans* retransmissions have been sent, causing the NFS client to return an error to the calling application.

Warning: A so-called `soft` timeout can cause silent data corruption in certain cases. As such, use the `soft` option only when client responsiveness is more important than data integrity. Using NFS over

TCP or increasing the value of the `retrans` option may mitigate some of the risks of using the `soft` option.

timeo
: The `timeo` value is the amount of time, in tenths of a second, to wait before resending a transmission after an RPC timeout. The default value for NFS over TCP is 600 (60 seconds). After the first timeout, the timeout value is doubled for each retry for a maximum of 60 seconds or until a major timeout occurs. If connecting to a slow server or over a busy network, better stability can be achieved by increasing this timeout value.

retrans
: The number of times the NFS client retries a request before it attempts further recovery action. If the `retrans` option is not specified, the NFS client tries each request three times. The NFS client generates a "server not responding" message after *retrans* retries, then attempts further recovery (depending on whether the hard mount option is in effect).

_netdev
: The `_netdev` option tells the system to wait until the network is up before trying to mount the share - **systemd** assumes this for NFS.

Note: Setting the sixth field (`fs_passno`) to a nonzero value may lead to unexpected behaviour, e.g. hangs when the systemd automount waits for a check which will never happen.

2.2.3 Mount using /etc/fstab with systemd

Another method is using the **x-systemd.automount** option which mounts the filesystem upon access:

/etc/fstab

```
servername:/home   /mountpoint/on/client  nfs  _netdev,noauto,x-systemd.automount,x-systemd.mount-timeout
=10,timeo=14,x-systemd.idle-timeout=1min 0 0
```

To make systemd aware of the changes to fstab, **reload** systemd and restart `remote-fs.target` **[3] (https://bbs.archlinux.org/viewtopic.php?pid=1515377#p1515377)**.

Tip:

- The `noauto` mount option will not mount the NFS share until it is accessed: use `auto` for it to be available immediately.
  If experiencing any issues with the mount failing due to the network not being up/available, **enable** `NetworkManager-wait-online.service`. It will ensure that `network.target` has all the links available prior to being active.
- The `users` mount option would allow user mounts, but be aware it implies further options as `noexec` for example.
- The `x-systemd.idle-timeout=1min` option will unmount the NFS share automatically after 1 minute of non-use. Good for laptops which might suddenly disconnect from the network.
- If shutdown/reboot holds too long because of NFS, **enable** `NetworkManager-wait-online.service` to ensure that NetworkManager is not exited before the NFS volumes are unmounted.
- Do not add the `x-systemd.requires=network-online.target` mount option as this can lead to ordering cycles within systemd **[4] (https://github.com/systemd/systemd-stabl**

e/issues/69). systemd adds the `network-online.target` dependency to the unit for `_netdev` mount automatically.
- Using the `nocto` option may improve performance for read-only mounts, but should be used only if the data on the server changes only occasionally.

2.2.4 As systemd unit

Create a new `.mount` file inside `/etc/systemd/system`, e.g. `mnt-home.mount`. See **systemd.mount(5) (https://man.archlinux.org/man/systemd.mount.5)** for details.

Note: Make sure the filename corresponds to the mountpoint you want to use. E.g. the unit name `mnt-home.mount` can only be used if you are going to mount the share under `/mnt/home`. Otherwise the following error might occur:
`systemd[1]: mnt-home.mount: Where= setting does not match unit name. Refusing.`
. If the mountpoint contains non-ASCII characters, use systemd-escape).

`What=` path to share

`Where=` path to mount the share

`Options=` share mounting options

Note:

- Network mount units automatically acquire `After` dependencies on `remote-fs-pre.target`, `network.target` and `network-online.target`, and gain a `Before` dependency on `remote-fs.target` unless `nofail` mount option is set. Towards the latter a `Wants` unit is added as well.
- Append `noauto` to `Options` preventing automatically mount during boot (unless it is pulled in by some other unit).
- If you want to use a hostname for the server you want to share (instead of an IP address), add `nss-lookup.target` to `After`. This might avoid mount errors at boot time that do not arise when testing the unit.

`/etc/systemd/system/mnt-home.mount`

```
[Unit]
Description=Mount home at boot

[Mount]
What=172.16.24.192:/home
Where=/mnt/home
Options=vers=4
Type=nfs
TimeoutSec=30

[Install]
WantedBy=multi-user.target
```

Tip: In case of an unreachable system, append `ForceUnmount=true` to `[Mount]`, allowing the export to be (force-)unmounted.

To use `mnt-home.mount`, start the unit and enable it to run on system boot.

2.2.4.1 automount

To automatically mount a share, one may use the following automount unit:

/etc/systemd/system/mnt-home.automount

```
[Unit]
Description=Automount home

[Automount]
Where=/mnt/home

[Install]
WantedBy=multi-user.target
```

Disable/stop the `mnt-home.mount` unit, and enable/start `mnt-home.automount` to automount the share when the mount path is being accessed.

Tip: Append `TimeoutIdleSec` to enable auto unmount. See **`systemd.automount(5)` (https://man.archlinux.org/man/systemd.automount.5)** for details.

2.2.5 Mount using autofs

Using autofs is useful when multiple machines want to connect via NFS; they could both be clients as well as servers. The reason this method is preferable over the earlier one is that if the server is switched off, the client will not throw errors about being unable to find NFS shares. See **autofs#NFS network mounts** for details.

3 Tips and tricks

3.1 Performance tuning

When using NFS on a network with a significant number of clients one may increase the default NFS threads from *8* to *16* or even a higher, depending on the server/network requirements:

/etc/nfs.conf

```
[nfsd]
threads=16
```

It may be necessary to tune the `rsize` and `wsize` mount options to meet the requirements of the network configuration.

In recent linux kernels (>2.6.18) the size of I/O operations allowed by the NFS server (default max block size) varies depending on RAM size, with a maximum of 1M (1048576 bytes), the max block size of the server will be used even if nfs clients requires bigger `rsize` and `wsize`. See **https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/5.8_technical_notes/known_issues-kernel** It is possible to

change the default max block size allowed by the server by writing to the `/proc/fs/nfsd/max_block_size` before starting *nfsd*. For example, the following command restores the previous default iosize of 32k:

```
# echo 32768 > /proc/fs/nfsd/max_block_size
```

Note: This is mainly useful for 32-bit servers when dealing with the large numbers of nfsd threads. Lowering the `max_block_size` may decrease NFS performance on modern hardware.

To make the change permanent, create a **systemd-tmpfile**:

```
/etc/tmpfiles.d/nfsd-block-size.conf
```

```
w /proc/fs/nfsd/max_block_size - - - - 32768
```

To mount with the increased `rsize` and `wsize` mount options:

```
# mount -t nfs -o rsize=32768,wsize=32768,vers=4 servername:/srv/nfs/music /mountpoint/on/client
```

Furthermore, despite the violation of NFS protocol, setting `async` instead of `sync` or `sync,no_wdelay` may potentially achieve a significant performance gain especially on spinning disks. Configure exports with this option and then execute `exportfs -arv` to apply.

```
/etc/exports
```

```
/srv/nfs        192.168.1.0/24(rw,async,crossmnt,fsid=0)
/srv/nfs/music  192.168.1.0/24(rw,async)
```

Warning: Using `async` comes with a risk of possible data loss or corruption if the server crashes or restarts uncleanly.

3.2 Automatic mount handling

This trick is useful for NFS-shares on a **wireless** network and/or on a network that may be unreliable. If the NFS host becomes unreachable, the NFS share will be unmounted to hopefully prevent system hangs when using the `hard` mount option **[5] (https://bbs.archlinux.org/viewtopic.php?pid=1260240#p1260240)**.

Make sure that the NFS mount points are correctly indicated in **fstab**:

```
/etc/fstab
```

```
lithium:/mnt/data           /mnt/data           nfs noauto 0 0
lithium:/var/cache/pacman   /var/cache/pacman   nfs noauto 0 0
```

Note:

- Use hostnames in **fstab** for this to work, not IP addresses.
- In order to mount NFS shares with non-root users the `users` option has to be added.

- The `noauto` mount option tells **systemd** to not automatically **mount** the shares at boot, otherwise this may cause the boot process to stall.

Create the `auto_share` script that will be used by **cron** or **systemd/Timers** to use ICMP ping to check if the NFS host is reachable:

`/usr/local/bin/auto_share`

```
#!/bin/bash

function net_umount {
  umount -l -f $1 &>/dev/null
}

function net_mount {
  mountpoint -q $1 || mount $1
}

NET_MOUNTS=$(sed -e '/^.*#/d' -e '/^.*:/!d' -e 's/\t/ /g' /etc/fstab | tr -s " ")$'\n'b

printf %s "$NET_MOUNTS" | while IFS= read -r line
do
  SERVER=$(echo $line | cut -f1 -d":")
  MOUNT_POINT=$(echo $line | cut -f2 -d" ")

  # Check if server already tested
  if [[ "${server_ok[@]}" =~ "${SERVER}" ]]; then
    # The server is up, make sure the share are mounted
    net_mount $MOUNT_POINT
  elif [[ "${server_notok[@]}" =~ "${SERVER}" ]]; then
    # The server could not be reached, unmount the share
    net_umount $MOUNT_POINT
  else
    # Check if the server is reachable
    ping -c 1 "${SERVER}" &>/dev/null

    if [ $? -ne 0 ]; then
      server_notok[${#server_notok[@]}]=$SERVER
      # The server could not be reached, unmount the share
      net_umount $MOUNT_POINT
    else
      server_ok[${#server_ok[@]}]=$SERVER
      # The server is up, make sure the share are mounted
      net_mount $MOUNT_POINT
    fi
  fi
done
```

Note: Test using a TCP probe instead of ICMP ping (default is tcp port 2049 in NFS4) then replace the line:

```
# Check if the server is reachable
ping -c 1 "${SERVER}" &>/dev/null
```

with:

```
# Check if the server is reachable
timeout 1 bash -c ": < /dev/tcp/${SERVER}/2049"
```

in the `auto_share` script above.

Make sure the script is **executable**.

Next check configure the script to run every X, in the examples below this is every minute.

3.2.1 Cron

```
# crontab -e
```

```
* * * * * /usr/local/bin/auto_share
```

3.2.2 systemd/Timers

```
/etc/systemd/system/auto_share.timer
```

```
[Unit]
Description=Automount NFS shares every minute

[Timer]
OnCalendar=*-*-* *:*:00

[Install]
WantedBy=timers.target
```

```
/etc/systemd/system/auto_share.service
```

```
[Unit]
Description=Automount NFS shares
After=syslog.target network.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/auto_share

[Install]
WantedBy=multi-user.target
```

Finally, **enable** and **start** `auto_share.timer`.

3.2.3 Using a NetworkManager dispatcher

NetworkManager can also be configured to run a script on network status change.

The easiest method for mount shares on network status change is to symlink the `auto_share` script:

```
# ln -s /usr/local/bin/auto_share /etc/NetworkManager/dispatcher.d/30-nfs.sh
```

However, in that particular case unmounting will happen only after the network connection has already been disabled, which is unclean and may result in effects like freezing of KDE Plasma applets.

The following script safely unmounts the NFS shares before the relevant network connection is disabled by listening for the `down`, `pre-down` and `vpn-pre-down` events, make sure the script is **executable**:

```
/etc/NetworkManager/dispatcher.d/30-nfs.sh
```

```
#!/bin/sh

# Find the connection UUID with "nmcli con show" in terminal.
# All NetworkManager connection types are supported: wireless, VPN, wired...
```

```
WANTED_CON_UUID="CHANGE-ME-NOW-9c7eff15-010a-4b1c-a786-9b4efa218ba9"

if [ "$CONNECTION_UUID" = "$WANTED_CON_UUID" ]; then

    # Script parameter $1: network interface name, not used
    # Script parameter $2: dispatched event

    case "$2" in
        "up")
            mount -a -t nfs4,nfs
            ;;
        "down"|"pre-down"|"vpn-pre-down")
            umount -l -a -t nfs4,nfs -f >/dev/null
            ;;
    esac
fi
```

Note: This script ignores mounts with the `noauto` option, remove this mount option or use `auto` to allow the dispatcher to manage these mounts.

Create a symlink inside `/etc/NetworkManager/dispatcher.d/pre-down` to catch the `pre-down` events:

```
# ln -s /etc/NetworkManager/dispatcher.d/30-nfs.sh /etc/NetworkManager/dispatcher.d/pre-down.d/30-nfs.sh
```

3.3 TLS encryption

NFS traffic can be encrypted using TLS as of Linux 6.5 using the `xprtsec=tls` mount option. To begin, install the **ktls-utils (https://aur.archlinux.org/packages/ktls-utils/)** [AUR] package on the client and server, and follow the below configuration steps for each.

3.3.1 Server

Create a private key and obtain a certificate containing your server's DNS name (see **Transport Layer Security** for more detail). These files do not need to be added to the system's trust store.

Note: Using a self-signed certificate that has also been encrypted is currently not supported and will result in a mount failure.

Edit `/etc/tlshd.conf` to use these files, using your own values for `x509.certificate` and `x509.private_key`

/etc/tlshd.conf

```
[authenticate.server]
x509.certificate= /etc/nfsd-certificate.pem
x509.private_key= /etc/nfsd-private-key.pem
```

Now **start** and **enable** `tlshd.service`.

3.3.2 Client

Add the server's TLS certificate generated in the previous step to the system's trust store (see **Transport Layer Security** for more detail).

Start and enable `tlshd.service`.

Now you should be able to mount the server using the server's DNS name:

```
# mount -o xprtsec=tls servername.domain:/ /mountpoint/on/client
```

Checking journalctl on the client should show that the TLS handshake was successful:

```
$ journalctl -b -u tlshd.service
```

```
Sep 28 11:14:46 client tlshd[227]: Built from ktls-utils 0.10 on Sep 26 2023 14:24:03
Sep 28 11:15:37 client tlshd[571]: Handshake with servername.domain (192.168.122.100) was successful
```

4 Troubleshooting

There is a dedicated article NFS/Troubleshooting.

5 See also

- See also Avahi, a Zeroconf implementation which allows automatic discovery of NFS shares.
- HOWTO: Diskless network boot NFS root
- Microsoft Services for Unix NFS Client info (https://web.archive.org/web/20201111215940/https://docs.microsoft.com/en-us/archive/blogs/msdn/sfu/all-well-almost-about-client-for-nfs-configuration-and-performance/)
- NFS on Snow Leopard (https://web.archive.org/web/20151212160906/https://blogs.oracle.com/jag/entry/nfs_on_snow_leopard)
- http://chschneider.eu/linux/server/nfs.shtml
- How to do Linux NFS Performance Tuning and Optimization (https://www.slashroot.in/how-do-linux-nfs-performance-tuning-and-optimization)
- Linux: Tune NFS Performance (https://www.cyberciti.biz/faq/linux-unix-tuning-nfs-server-client-performance/)