ECE590 Enterprise Storage Architecture Lab #2: NAS, SAN, and Filesystems

Updates: 2018-10-04, Total rewrite of the iSCSI stuff ☹. 2018-10-07, Added missing link.

Now that we understand RAID, let's use those drives as actual storage.

Directions:

- This assignment will be completed in your groups. However, **every member of the group must be fully aware of every part of the assignment**. Further, while you can discuss concepts with other groups, actual steps and answers should not be shared between groups.
- The assignment will ask for short answers or screenshots; this material should be collected in a file called `ece590-group<NUM>-lab2.pdf`, where <NUM> is your group number, and submitted via Sakai. *Word documents will not be accepted*. Anything you need to include in this document is highlighted in cyan.

1 Prepare your resources[1]

For this assignment, you'll need your server to be configured with a 3-disk software RAID5 with a hot spare, just like in Lab 1 section 3.1[2]. Do this now, and show mdadm command used.

We will again need to set up a filesystem. However, later in this assignment, we'll want to export our RAID device over SAN to a Windows machine, so unlike last time, so we'll need to do a few more steps. Before, we laid our filesystem down directly on the entire block device. This is fine in UNIX, but the more common practice (and one required by Windows) is to have a *partition table*. There are two variants of partition table, the classic MBR-style and the more modern GPT type. We'll use an MBR type for simplicity.

A simple tool to manage MBR partition tables is **`fdisk`**. Research a bit on using this tool (or, if you prefer, one of its more modern siblings, like **`cfdisk`**), and use it to create a single partition on **`/dev/md0`** spanning the entire device. Set the type of this partition to 0x07 (NTFS type). Paste screenshots or console logs of how you did this.

With the partition created, you should now see a new device, **`/dev/md0p1`**, which represents the partition. It's time to put a file system on it, and because we're going to have Windows access this block device, let's pick a filesystem that our Windows client will be able to natively understand: NTFS. Use

[1] Updated 2018-10-04: Rewrote large parts of this section. *If you've done it previously, I'm afraid you'll need to redo it.* Sorry.
[2] Updated 2018-10-04: Reference was incorrect previously.

mkfs.ntfs to prepare a filesystem on **/dev/md0p1**, then mount this new filesystem at the directory **/x** (creating the directory if needed). Paste screenshots or console logs of this process.

Place a few files into this filesystem which you'll recognize later. Paste screenshots or console logs showing the files created for later reference.

Later on, you'll also need a **Windows client** (Windows 7, 8, or 10) for which you have administrator access. If your personal computer is Windows-based, you can use it. If you run Mac/Linux on your personal computer, you can either set up a Windows VM in a free hypervisor, such as VMware Player or VirtualBox, or reserve a plain Windows 10 in the Duke Virtual Computing Manager (VCM). The Windows client will need network access to the storage controller.

2 Deploy a CIFS NAS

On the server, install the samba server package, which serves CIFS shares compatible with Microsoft Windows clients. Configure it to share our filesystem in /x as a share called "x". There are a lot of tutorials on this, so I leave the details to you.

To test it, login to your **Windows client**. In Windows Explorer, navigate to "\\<*IP_ADDRESS_OF_STORAGE_CONTROLLER*>\x". The files you placed there previously should be visible. Verify this, then in Windows, add a few more files.

Provide screenshots or a terminal log of these steps.

Show evidence that you can access the files created in step 1 and note what new files you create.

IN ADDITION, provide the relevant portion of your smb.conf file and a screenshot of Windows Explorer showing your storage controller NAS share.

3 Create an iSCSI SAN[3]

Now, instead of a CIFS NAS, we're going to instead deploy an iSCSI SAN. However, we cannot serve the filesystem **/x** and the block device **/dev/md0** at the same time, since **/x** resides on **/dev/md0p1**. If we did, both the Linux storage controller and the Windows SAN client would be making changes to **/dev/md0**, with neither aware of the other. This would lead to file system corruption and inconsistency.

Therefore, begin by stopping the samba service and using umount to unmount **/x**. At this point, neither **/dev/md0** nor **/dev/md0p1** should appear in your mount list:

```
root@xub1404dt:/ # mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
```

[3] Updated 2018-10-04: This section was based on an iSCSI target package for Ubuntu 16.04. It has been rewritten to use the '**tgt**' package deployed with Ubuntu 18.04. Credit to this guide for some of the commands/examples.

```
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
 ...
systemd on /sys/fs/cgroup/systemd type cgroup
(rw,noexec,nosuid,nodev,none,name=systemd)
```
(no /dev/md0 in this list)

The procedure below will get the iSCSI target software installed and configured.

3.1 Install 'tgt'

Do the following procedure as root.

First, let's update/upgrade our existing packages:

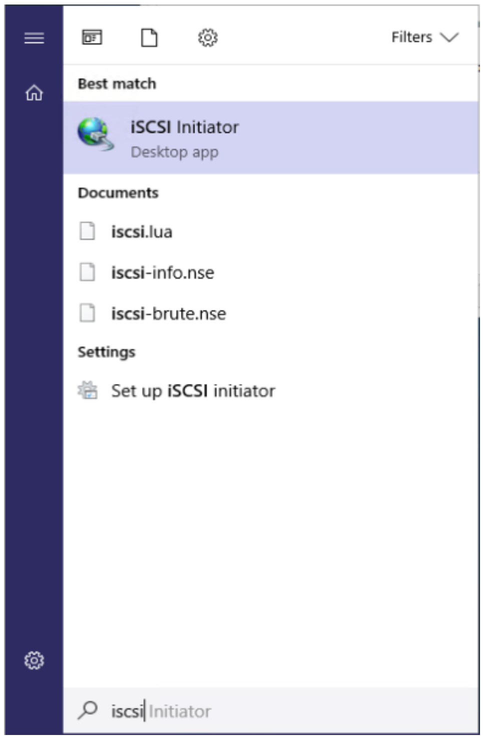
```
# apt update
# apt upgrade
```

Next, install the **`tgt`** package (an iSCSI target)

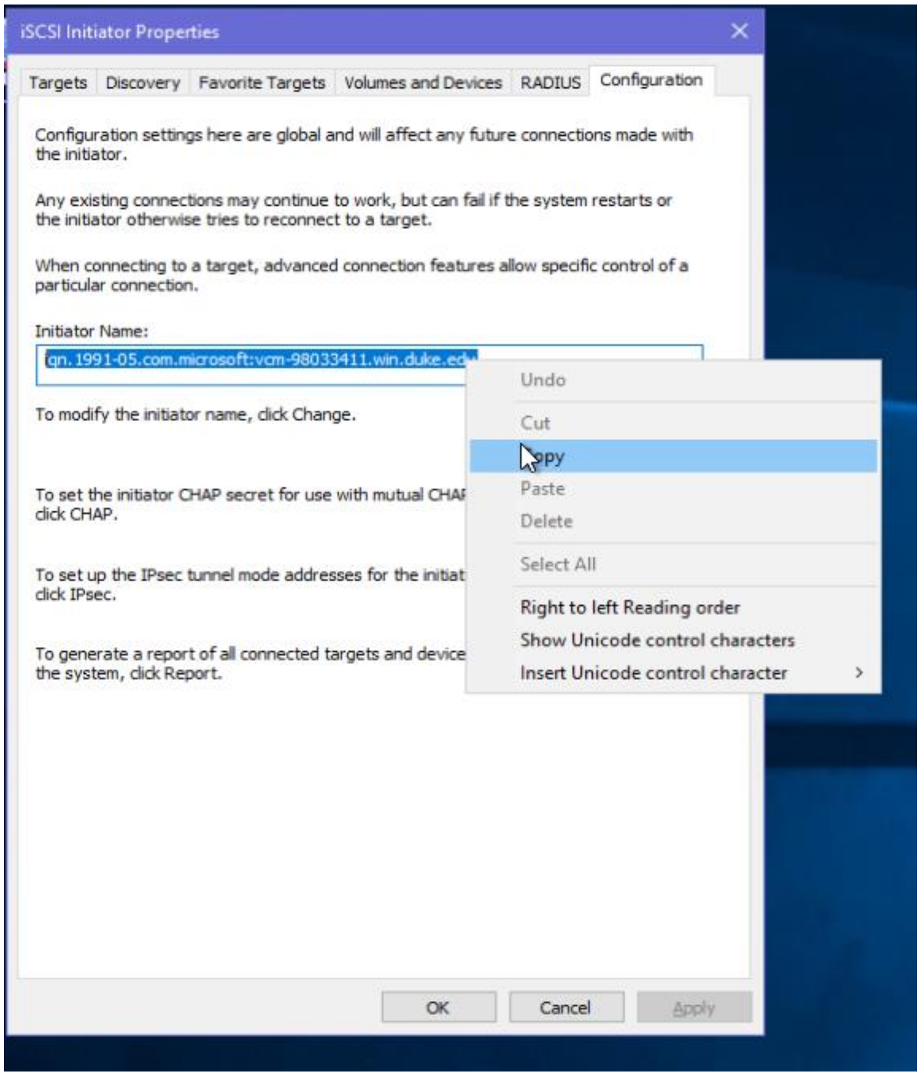
```
# apt -y install tgt
```

3.2 Note the initiator IQN

Via RDP, connect to your Windows VM. In the start menu, search for "iscsi" and launch the "iSCSI Initiator":



In the "Configuration" tab, make note of the initiator IQN. We could change it to something else if we wanted to, but the default is fine. From now on, we'll refer to this value as the *initiator IQN*.


iSCSI Initiator Properties
Targets
Discovery
Favorite Targets
Volumes and Devices
RADIUS
Configuration
Undo
Cut
Copy
Paste
Delete
Select All
Right to left Reading order
Show Unicode control characters
Insert Unicode control character
OK
Cancel
Apply

Configuration settings here are global and will affect any future connections made with the initiator.

Any existing connections may continue to work, but can fail if the system restarts or the initiator otherwise tries to reconnect to a target.

When connecting to a target, advanced connection features allow specific control of a particular connection.

Initiator Name:
iqn.1991-05.com.microsoft:vcm-98033411.win.duke.edu

To modify the initiator name, click Change.

To set the initiator CHAP secret for use with mutual CHAP, click CHAP.

To set up the IPsec tunnel mode addresses for the initiator, click IPsec.

To generate a report of all connected targets and devices on the system, click Report.

Note the initiator IQN in your write-up.

3.3 Prepare target IQN

Let's make up a target IQN for the storage server. Recall the format of an IQN:

```
    iqn.(year)-(month).(reversed-domain-name):(arbitrary-string)
```

Where the year and month refer to when the domain was registered. Create an IQN for Duke, noting that duke.edu was registered in June 1986. For the arbitrary string, use your NetID. Note your target IQN in your write-up.

3.4 Set up a LUN export

The **tgt** package keeps its LUN export records in **/etc/tgt/conf.d/**. Create a file called "**mylun.conf**" in this directory. For the content, you can find full documentation on the format here[4], but the following should be sufficient:

```
<target <TARGET-IQN>>
    backing-store /dev/md0
    initiator-name <INITIATOR-IQN>
</target>
```

Restart the **tgt** daemon:

```
# systemctl restart tgt
```

Display a list of targets to confirm the export worked; the output should look similar to this example:

```
# tgtadm --mode target --op show
```

Include this output in your write-up.

[4] Updated 2018-10-07: Added hyperlink.

3.5 Attach the LUN

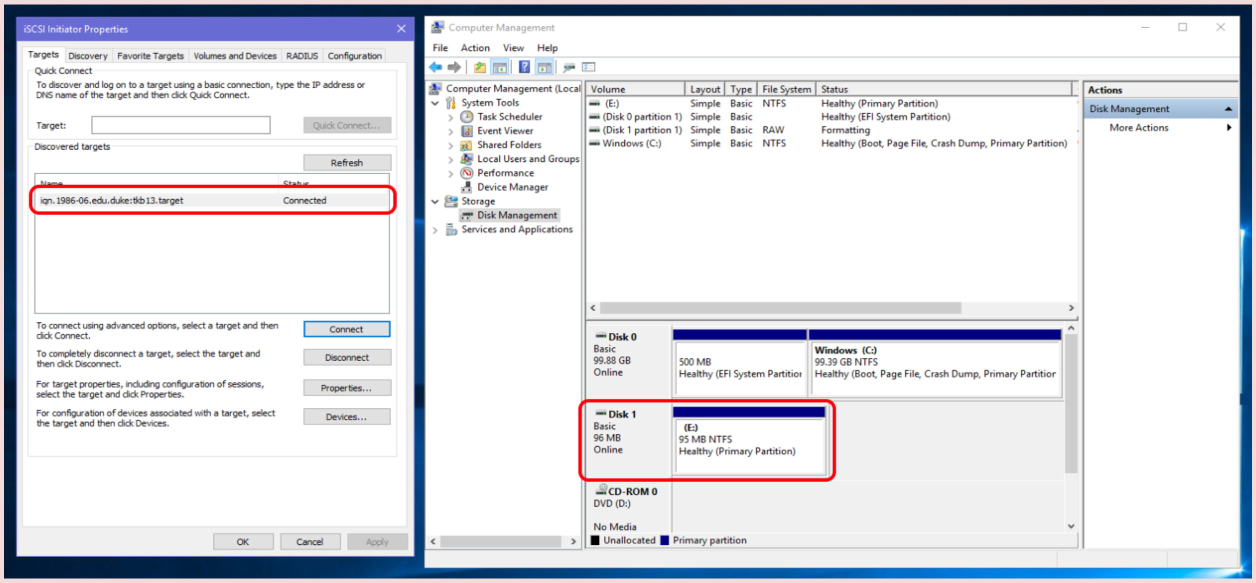
On the Windows VM's iSCSI Initiator controls, in the "Targets" tab, put in the IP address of your storage server into the "Quick connect" target box and hit "Quick connect…". It should identify the target IQN and connect.

You should now be able to open Explorer, go to "This PC", and find a new drive attached. When you view the new drive, you should see the content you put there previously. So we have access to the same data as with the NAS, but now Windows is the one doing the file system logic, so the data shows up as its own drive, and Windows is sending simple read-block/write-block requests to the SAN target.

TROUBLESHOOTING AT THIS STEP

If you get a prompt to format the drive, you may have erred in one of the steps above, as Windows doesn't recognize either the partition table or filesystem.

If no drive appears and you get no prompt, you may wish to check if the block device is attached at all. For this, type "computer management" into the start menu and launch it, then use the "Disk Management" component of this tool. Below is a screenshot showing an attached target in the iSCSI Initiator view next to the block device itself showing in the Disk Management view:



To discover and log on to a target using a basic connection, type the IP address or DNS name of the target and then click Quick Connect.

Target:

Name	Status
iqn.1986-06.edu.duke:tkb13.target	Connected

To connect using advanced options, select a target and then click Connect.

To completely disconnect a target, select the target and then click Disconnect.

For target properties, including configuration of sessions, select the target and click Properties.

For configuration of devices associated with a target, select the target and then click Devices.

Volume	Layout	Type	File System	Status
(E:)	Simple	Basic	NTFS	Healthy (Primary Partition)
(Disk 0 partition 1)	Simple	Basic		Healthy (EFI System Partition)
(Disk 1 partition 1)	Simple	Basic	RAW	Formatting
Windows (C:)	Simple	Basic	NTFS	Healthy (Boot, Page File, Crash Dump, Primary Partition)

Disk 0
Basic
99.88 GB
Online

500 MB
Healthy (EFI System Partitior

Windows (C:)
99.39 GB NTFS
Healthy (Boot, Page File, Crash Dump, Primary Partitior

Disk 1
Basic
96 MB
Online

(E:)
95 MB NTFS
Healthy (Primary Partition)

CD-ROM 0
DVD (D:)

No Media

■ Unallocated ■ Primary partition

If the target isn't shown on the left, you've failed to connect to the storage server. If you are connected but the block device doesn't appear on the right, the LUN isn't properly attached (export config issue?). If the device exists but says something like "Unallocated" instead of "Healthy", you have a problem with your partition table and/or file system.

Provide screenshots of the Windows GUI steps performed, and evidence that you can access files created both on the Linux server in step 1 as well as files created from Windows via CIFS in step 2.

4 Thin provisioning

NOTE: Each group member should do the "File sizing and sparse files" questions from the individual homework before proceeding.

Thin provisioning is a technique to take advantage of underutilization of storage objects (NAS shares or SAN LUNs). It allows the storage administrator to give out more *logical* storage than the available *physical* storage, banking on the notion that most volumes will be underutilized and that more physical storage can be added as actual utilization increases over time.

We're going to use some basic UNIX concepts to deploy thinly-provisioned iSCSI SAN LUNs.

4.1 Tear down existing SAN[5]

On the Windows client, disconnect from the iSCSI target.

On the Linux server, stop the **`tgt`** service ("**`systemctl stop tgt`**"). Remove the existing LUN export file (you should just move it out of the **`/etc/tgt/conf.d/`** directory so you still have it around as a template for step 4.4).

4.2 Make new container file system

Reformat your RAID device as an ext4 volume. (We could technically reuse the existing NTFS filesystem, but Linux NTFS support isn't great, and it's better to start with a clean slate anyway.)

Mount the filesystem to /x like you did in section 2.

4.3 Create virtual LUN backing files

Within this filesystem, make three 1TB sparse files called `lun0`, `lun1`, and `lun2`. We will expose these files as virtual LUNs – files used to represent a logical block device are often called *backing files*.

Check the apparent and allocated size for these files.

4.4 Export and attach the virtual LUNs[6]

Create a new LUN export file based on the earlier LUN export file, except there should be three "backing-store" lines set to each of the files we created. Include your new LUN definition file in your write-up.

[5] Updated 2018-10-04 as part of the iSCSI rewrite.
[6] Updated 2018-10-04 as part of the iSCSI rewrite.

Re-start the `tgt` service, and on the Windows machine, re-attach to the target. The OS should discover the three LUNs.

4.5 Prepare the LUNs

Unlike in section 3, we didn't prepare a filesystem ahead of time, so the drives won't appear in Explorer. Instead, find them in the disk management interface (described in the troubleshooting of section 3.5). Format each one with an NTFS filesystem. As you do so, be sure to choose "quick format", else Windows will try to write zeroes to the whole thing, then it will no longer be a sparse file. Also, set the filesystem labels to LUN0, LUN1, and LUN2 to help you keep track of which is which. When done, include a screenshot of the Disk management interface showing the three LUNs attached, formatted, and ready.

4.6 Understanding allocation and oversubscription

On the Linux server, check apparent and allocated size for these files.

The allocated size will have increased -- why?

How big are the LUNs as far as Windows can tell?

How much **total** *logical* capacity is the storage server providing to the Windows machine via iSCSI?

How much **total** *physical* capacity does the storage server RAID filesystem have?

What is the rate of oversubscription (logical_bytes_advertised / physical_bytes_available)?

At this point, make note of:

- The free space of the LUN in Windows.
- The allocated and apparent sizes of the LUN file on the server.
- The free space of the underlying storage on the server.

4.7 Add stuff and check the result

On the Windows machine, copy a few megabytes of data into one of the LUNs. After the copy, on the Linux server, monitor the allocated size of the backing file[7]. It may increase for some time after the actual GUI copy appears to finish -- this is due to OS buffering and write-back caching.

When the copy is done and the size change stabilizes, compare to the measurements you made before:

- The free space of the LUN in Windows.
- The allocated and apparent sizes of the LUN file on the server.
- The free space of the underlying storage on the server.

How have these numbers changed?

4.8 Delete stuff and check the result

Delete the data you put on the LUN (hold shift when you do so to force an actual deletion instead of going to the recycle bin; if it does go to the recycle bin, empty the recycle bin before proceeding). Again, check these measurements:

- The free space of the LUN in Windows.
- The allocated and apparent sizes of the LUN file on the server.
- The free space of the underlying storage on the server.

How have these numbers changed?

You should find that Windows says free space went up, but on the Linux server, apparent file size and underlying storage free space are unchanged. This is because the iSCSI initiator has no way to indicate "it's safe to de-allocate these blocks", as block devices don't have a concept of free versus filled. (One exception to this is the TRIM command commonly used on SSDs to inform the drive of de-allocated space, but that is not enabled on a SAN LUN.)

4.9 Add too much stuff and break it

You've probably realized by now that it is not possible to actually fill all three LUNs to their advertised size, as the server doesn't actually have that much capacity. Let's find out what happens when we try. On the Windows host, add stuff to LUN 0, periodically checking the file system free space on the Linux server's RAID and the LUN's allocated size on the server.

You'll eventually fill the physical storage while the LUN appears to have plenty of space. What happens in Windows? Would you characterize the OS as "healthy" right now?

[7] Note: you can use the `watch` command to view the output of a command repeatedly.

Check /var/log/syslog for messages. Hopefully the iSCSI target daemon is reporting errors -- note them in your write-up.

Why is running out of space in this manner different than running out of space on a traditional disk? (Hint: could the Windows OS have predicted this would happen?)

As a storage administrator, you should avoid this outcome; what pro-active steps could you take to avoid this while still getting the benefits of thin provisioning?

4.10 Tear it all down[8]

If the Windows machine is just a scratch VM, just delete it. Otherwise, shutdown Windows, stop the **`tgt`** daemon on the Linux server (so Windows doesn't see it on reboot), boot Windows, and detach from the iSCSI target.

On the Linux server, ensure the **`tgt`** daemon is stopped, unmount the RAID device, and remove the LUN definition file from **`/etc/tgt/conf.d`**.

[8] Updated 2018-10-04 as part of the iSCSI rewrite.