Chapter 1

Configuring iSCSI Target and Initiator

1.1   Understanding iSCSI Target and Initiator

SCSI (Small Computer System Interface) [read as *scuzzy*] is an alternative to ATA (a.k.a. IDE) Hard drives, which most consumer computers stick to. While SCSI drives provide significantly more throughput for certain scenarios, IDE suffices for most home computer usage. However, in case of servers, SCSI proves to be a much better alternative, since they provide more reliability and data transfer speed (much higher than ATA), owing to the fact that data transfer occurs in full-duplex mode (i.e., data can be read and written at the same time at full speeds). They also boast higher speeds (such as 15,000 RPM) as compared to ATA speeds (7200 RPM). Another reason servers tend to use SCSI (or related technologies, such as Serially Attached SCSI or SAS) is that the protocol makes it easy to *daisy-chain* several SCSI devices to the same controller, several times that of IDE devices. In fact, in the pre-USB era, SCSI was the go-to common interface for connecting peripherals or even devices such as printers.

Traditional SCSI devices use a long cable and a SCSI **Command Descriptor Block (CDB)** command to interact with the SCSI devices. In case of iSCSI, the same CDBs are used, but they're transmitted over IP packets over a network, instead of the cable. Thus, the SCSI devices are emulated by using a storage backend and presenting them on the network using iSCSI targets. SCSI targets are typically storage devices, while the hosts they're connected to are the initiators. Thus, this technology enables us to share PVs or LVs on the network, represented by iSCSI targets.

A **Storage Area Network (SAN)** is a network that provides access to a consolidated, block level data storage. *Block devices* provide a buffered data storage method, where data is transferred from the kernel buffer to the physical device. Also, data can be read and written in entire blocks. SANs thus present devices such as disk arrays as locally attached storage to servers. **Fiber Channel** or **FC** is a high speed network technology developed to enable fast data transfers between servers and SANs. Ethernet structures utilizing iSCSI technology can be as fast as their FC structure counterparts, thus making the technology enterprise ready for SAN creation.

1.1.1   iSCSI Operation

In the case of iSCSI storage, we have the SAN, on which runs a *iSCSI target* which can provide access to the storage backend. For any server that needs to access the files hosted

by the SAN, it needs to run an **iSCSI initiator**, which performs a discovery operation first. During this, the SAN tells it about the iSCSI devices it has to offer. Once this is complete, the iSCSI initiator can login to the devices.

1.1.2 iSCSI Components

Both the iSCSI targets and the storage backends need to be set up for the SAN to operate. The storage backend can be an entire disk, a dedicated partition, a logical volume or even a file! The servers, running the iSCSI initiators, will see the iSCSI targets as new storage devices after successfully logging in to them. This can be verified by viewing the output of the `/proc/partitions` file. A tool called *lsscsi* can also alternatively used, although it is not installed by default.

1.1.3 Basic iSCSI Terminology

Terms	Description
IQN	iSCSI Qualified Name - an unique name assigned to each iSCSI target and initiator, used to identify them.
Initiator	The iSCSI client that is identified by its IQN.
Target	The service on the iSCSI server that gives access to the storage backend.
ACLs	Access Control Lists that are based on the node's IQNs.
Portal	Also known as **nodes**, this is the combination of the IP address and the port that are used by both targets and initiators to establish connections.
discovery	The process through which an indicator finds the available targets that are configured for a given portal.
LUNs	The *Logical Unit Number* is a number used to identify the logical unit (i.e., block devices shared through the target) being addressed by the iSCSI Controller.
login	The act which gives an initiator the relevant LUNs.
TPG	The *Target Portal Group* is a collection of IP Addresses and TCP ports to which a particular iSCSI Target will listen.

So, there can be more than one portals per server, and more than one targets per portal.

1.1.4 After connecting an initiator to an iSCSI Target

The new block devices thus accessed will appear as local devices (/dev/sdb, /dev/sdc etc.) Note that if a LUN is available and used by multiple servers, multiple devices can access the LUN post connection, i.e., multiple servers can use the disk at the same time. This is a bit dangerous, since it requires using clustering, for providing multiple servers to use the storage. Otherwise for a file system like XFS or Ext4, two servers writing to the same file can cause data loss.

To avoid this, shared file systems such as GFS2 can be used. In GFS2, the file system cache is shared among all the nodes. Thus, all nodes writing to the file system know what all the other nodes are doing.

1.2   Setting up an iSCSI Target

The iSCSI target works with several storage backend devices on the SAN. These storage devices can be anything that can be used for PVs when using traditional LVM. All these devices are put together in a volume group, which is then subdivided into several LVs. These LVs are each assigned a LUN.
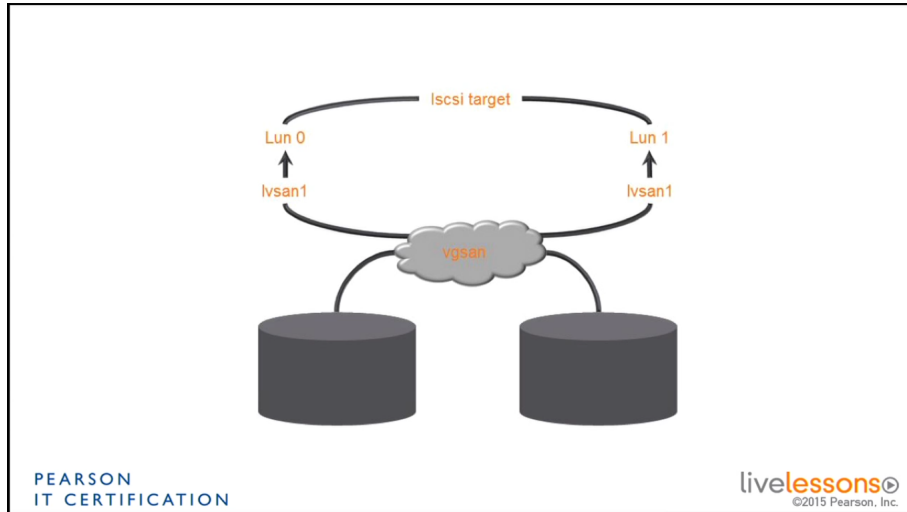

Figure 1.1: iSCSI Target Setup

These LUNs are presented using the iSCSI targets. Thus, the iSCSI configuration is created on top of a traditional LVM configuration.

1.2.1   Creating the LVM

Let us consider we have an empty disk of 1GB on which we want to build the iSCSI configuration. This can be verified using:

```
# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda             8:0      0   20G  0 disk
 ├─sda1         8:1      0    1G  0 part /boot
 └─sda2         8:2      0   19G  0 part
  ├─centos-root 253:0    0   17G  0 lvm  /
  └─centos-swap 253:1    0    2G  0 lvm  [SWAP]
sdb             8:16     0    1G  0 disk
sr0             11:0     1  8.1G  0 rom
```

We can directly create the VG `vgsan` on it, using:

```
# vgcreate vgSAN /dev/sdb
Physical volume "/dev/sdb" successfully created.
Volume group "vgSAN" successfully created
# pvs
PV         VG     Fmt  Attr PSize    PFree
/dev/sda2  centos lvm2 a--  <19.00g       0
/dev/sdb   vgSAN  lvm2 a--  1020.00m 1020.00m
# vgs
```

3

```
 9   VG      #PV #LV #SN Attr   VSize     VFree
10   centos   1   2   0 wz--n- <19.00g        0
11   vgSAN    1   0   0 wz--n- 1020.00m 1020.00m
```

The output of the `pvs` and `vgs` commands show that the PV `/dev/sdb` is now a part of *vgSAN*, which has a free space of 1020MB. Now we create two LVs *lvSAN1* and *lvSAN2* on the VG, using:

```
 1   # lvcreate -L 500M -n lvSAN1 vgSAN
 2   Logical volume "lvSAN1" created.
 3   # lvcreate -l 100%FREE -n lvSAN2 vgSAN
 4   Logical volume "lvSAN2" created.
 5   # lvs
 6   LV     VG     Attr       LSize   Pool Origin ... Convert
 7   root   centos -wi-ao---- <17.00g
 8   swap   centos -wi-ao----   2.00g
 9   lvSAN1 vgSAN  -wi-a-----  500.00m
10   lvSAN2 vgSAN  -wi-a-----  520.00m
```

Now, our LVM setup is complete, and we can proceed with the iSCSI setup. For this, first of all we need to install the iSCSI software, called **targetcli**. The targetcli utility is a relatively new one capable of managing multiple types of storage devices.

1.2.2   Creating the iSCSI configuration using targetcli

We start the utility using:

```
 1   # targetcli
 2   Warning: Could not load preferences file /root/.targetcli/prefs.bin.
 3   targetcli shell version 2.1.fb46
 4   Copyright 2011-2013 by Datera, Inc and others.
 5   For help on commands, type 'help'.
 6
 7   />
```

This interface can be navigated using the same commands as the bash shell. Using the `cd` command produces the output:

```
 1   /> ls
 2   o- / ...................................................................... [...]
 3     o- backstores ........................................................... [...]
 4     | o- block .............................................. [Storage Objects: 0]
 5     | o- fileio ............................................. [Storage Objects: 0]
 6     | o- pscsi .............................................. [Storage Objects: 0]
 7     | o- ramdisk ............................................ [Storage Objects: 0]
 8     o- iscsi ....................................................... [Targets: 0]
 9     o- loopback .................................................... [Targets: 0]
10   />
```

The *backstores* part allow us to work with the different storage devices. To enter backstores, we simply enter `cd` command, and select it from the menu. This will change the prompt to `/backstores>`. Here, we can see it contains the *block*, the *fileio*, the *pSCSI* and the *ramdisk* devices. Their significance is explained below:

4

Types	Description
Block	Refers to any block device that we want to share using iSCSI. This includes all traditional disks, partitions and even LVMs.
fileio	Refers to a file that can be used as a storage source. This refers to a big file created using a tool such as dd.
pscsi	Physical SCSI - a SCSI pass-through backstore is created for such devices.
ramdisk	RAM storage, wiped with every reboot, and is thus a **very** bad idea.

Now, since all our LVs are block devices (by their very nature), we have to create our LUNs inside the block category. This we can do using:

```
1  /backstores> block/ create block1 /dev/vgSAN/lvSAN1
2  Created block storage object block1 using /dev/vgSAN/lvSAN1.
```

The command instructs the targetcli utility to enter the block category, and create a block device called *block1* from the /dev/vgSAN/lvSAN1 device. We can create another block device for the partition and a 1G custom file device using:

```
1  /backstores> block/ create block2 /dev/vgSAN/lvSAN2
2  Created block storage object block2 using /dev/vgSAN/lvSAN2.
3  /backstores> fileio/ create file1 /root/diskFile1 1G
4  Created fileio file1 with size 1073741824
```

When creating a file, we can merely specify the size (1GB) and the name & location (/root/diskFile1) to have the targetcli utility create the file for us, instead of copying from /dev/zero to a file using dd. All the different devices thus added can be seen with:

```
1  /backstores> ls
2  o- backstores ...................................................... [...]
3    o- block ............................................ [Storage Objects: 2]
4    | o- block1 ............ [/dev/vgSAN/lvSAN1 (500.0MiB) write-thru deactivated]
5    | | o- alua ............................................. [ALUA Groups: 1]
6    | |   o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
7    | o- block2 ............ [/dev/vgSAN/lvSAN2 (520.0MiB) write-thru deactivated]
8    |   o- alua ............................................. [ALUA Groups: 1]
9    |     o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
10   o- fileio .......................................... [Storage Objects: 1]
11   | o- file1 ................. [/root/diskFile1 (1.0GiB) write-back deactivated]
12   |   o- alua ............................................. [ALUA Groups: 1]
13   |     o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
14   o- pscsi ........................................... [Storage Objects: 0]
15   o- ramdisk ......................................... [Storage Objects: 0]
```

Now that the block devices are ready, we can go to the /iscsi environment and prepare the iSCSI targets. Initially, there will be no targets:

```
1  /backstores> cd /iscsi
2  /iscsi> ls
3  o- iscsi ...................................................... [Targets: 0]
```

1.2.3 Target Creation

Now, we create a target that provides access to the backing storage devices called *block1*, *block2* and *file1*. This can be done using the create command, followed by an IQN. IQNs

are typically created using a naming format:

```
iqn.<yearOfCreation>-<monthOfCreation>.<reverseDomainName>:<targetName>
```

Thus, ours will be named: `iqn.2018-01.local.somuvmnet:target1`. This can be done with:

```
1  /iscsi> create iqn.2018-01.local.somuvmnet:target1
2  Created target iqn.2018-01.local.somuvmnet:target1.
3  Created TPG 1.
4  Global pref auto_add_default_portal=true
5  Created default portal listening on all IPs (0.0.0.0), port 3260.
```

Thus, both a target and a TPG are created at the same time. The target thus created can be viewed with:

```
1  /iscsi> ls
2  o- iscsi ........................................................ [Targets: 1]
3    o- iqn.2018-01.local.somuvmnet:target1 ............................ [TPGs: 1]
4      o- tpg1 .......................................... [no-gen-acls, no-auth]
5        o- acls .................................................... [ACLs: 0]
6        o- luns .................................................... [LUNs: 0]
7        o- portals ............................................... [Portals: 1]
8          o- 0.0.0.0:3260 ............................................... [OK]
```

1.2.4 TPG Configuration

Within the target is a TPG (Target portal group), which represents the entire configuration of the target. This includes all the ACLs, the LUNs and the portals related to the target.

ACLs

Next, we need to create the ACLs for our target. For this, we need to cd into the ACL environment of our target using (Note that tab-autocompletion works for this tool):

```
1  /iscsi> cd iqn.2018-01.local.somuvmnet:target1/tpg1/acls
2  /iscsi/iqn.20...et1/tpg1/acls>
```

We create the ACL node using:

```
1  /iscsi/iqn.20...et1/tpg1/acls> create iqn.2018-01.local.somuvmnet:vmdeux
2  Created Node ACL for iqn.2018-01.local.somuvmnet:vmdeux
```

Note that the identifier provided to create the node ACL is the IQN that has been set on the second server. The structure now looks like:

```
1  /iscsi/iqn.20...et1/tpg1/acls> cd /iscsi
2  /iscsi> ls
3    o- iscsi ...................................................... [Targets: 1]
4      o- iqn.2018-01.local.somuvmnet:target1 .......................... [TPGs: 1]
5        o- tpg1 ........................................ [no-gen-acls, no-auth]
6          o- acls .................................................. [ACLs: 1]
7          | o- iqn.2018-01.local.somuvmnet:vmdeux ............... [Mapped LUNs: 0]
8          o- luns .................................................. [LUNs: 0]
9          o- portals ............................................. [Portals: 1]
10           o- 0.0.0.0:3260 ............................................. [OK]
```

LUNs

Now, inside the *tpg1* node, we create a LUN by using:

```
1  /iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/block/block1
2  Created LUN 0.
3  Created LUN 0->0 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
```

Now, we can repeat the command a couple of times to create the LUNs for *block2* and *file1* as well:

```
1  /iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/block/block2
2  Created LUN 1.
3  Created LUN 1->1 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
4  /iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/fileio/file1
5  Created LUN 2.
6  Created LUN 2->2 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
```

The contents of *tpg1* should now look like:

```
1  /iscsi/iqn.20...:target1/tpg1> ls
2  o- tpg1 ............................................. [no-gen-acls, no-auth]
3    o- acls ...................................................... [ACLs: 1]
4    | o- iqn.2018-01.local.somuvmnet:vmdeux ..................... [Mapped LUNs: 3]
5    |   o- mapped_lun0 ................................. [lun0 block/block1 (rw)]
6    |   o- mapped_lun1 ................................. [lun1 block/block2 (rw)]
7    |   o- mapped_lun2 ................................. [lun2 fileio/file1 (rw)]
8    o- luns ...................................................... [LUNs: 3]
9    | o- lun0 .............. [block/block1 (/dev/vgSAN/lvSAN1) (default_tg_pt_gp)]
10   | o- lun1 .............. [block/block2 (/dev/vgSAN/lvSAN2) (default_tg_pt_gp)]
11   | o- lun2 ............... [fileio/file1 (/root/diskFile1) (default_tg_pt_gp)]
12   o- portals ................................................. [Portals: 1]
13     o- 0.0.0.0:3260 ................................................... [OK]
```

We can see that not only have the LUNs been created, but they've been assigned to the ACL as well! Thus, it becomes critical to create ACLs before the LUNs because the default behaviour of `targetcli` is to automatically assign any LUN that's been created to the ACLs in the TPG. Now, we have to create a portal.

Portals

We can create portal which will bear the IP address of the server on which our SAN will advertise the LUNs for this particular target. We do this by:

```
1  iscsi/iqn.20...:target1/tpg1> portals/ create 90.0.16.27
2  Using default IP port 3260
```

The complete configuration of the iSCSI setup can be viewed with:

```
1  /iscsi/iqn.20...:target1/tpg1> cd
2  /> ls
3  o- / ................................................................. [...]
4    o- backstores ..................................................... [...]
5    | o- block ........................................... [Storage Objects: 2]
6    | | o- block1 ............ [/dev/vgSAN/lvSAN1 (500.0MiB) write-thru activated]
```

```
 7    | | | o- alua ............................................. [ALUA Groups: 1]
 8    | | |   o- default_tg_pt_gp .................... [ALUA state: Active/optimized]
 9    | | o- block2 ............ [/dev/vgSAN/lvSAN2 (520.0MiB) write-thru activated]
10    | |   o- alua ............................................ [ALUA Groups: 1]
11    | |     o- default_tg_pt_gp .................. [ALUA state: Active/optimized]
12    | o- fileio ......................................... [Storage Objects: 1]
13    | | o- file1 ................. [/root/diskFile1 (1.0GiB) write-back activated]
14    | |   o- alua ............................................ [ALUA Groups: 1]
15    | |     o- default_tg_pt_gp .................. [ALUA state: Active/optimized]
16    | o- pscsi .......................................... [Storage Objects: 0]
17    | o- ramdisk ........................................ [Storage Objects: 0]
18    o- iscsi ...................................................... [Targets: 1]
19    | o- iqn.2018-01.local.somuvmnet:target1 .......................... [TPGs: 1]
20    |   o- tpg1 ............................................ [no-gen-acls, no-auth]
21    |     o- acls ................................................... [ACLs: 1]
22    |     | o- iqn.2018-01.local.somuvmnet:vmdeux ............... [Mapped LUNs: 3]
23    |     |   o- mapped_lun0 ........................... [lun0 block/block1 (rw)]
24    |     |   o- mapped_lun1 ........................... [lun1 block/block2 (rw)]
25    |     |   o- mapped_lun2 ........................... [lun2 fileio/file1 (rw)]
26    |     o- luns ...................................................... [LUNs: 3]
27    |     | o- lun0 ....... [block/block1 (/dev/vgSAN/lvSAN1) (default_tg_pt_gp)]
28    |     | o- lun1 ....... [block/block2 (/dev/vgSAN/lvSAN2) (default_tg_pt_gp)]
29    |     | o- lun2 ......... [fileio/file1 (/root/diskFile1) (default_tg_pt_gp)]
30    |     o- portals ............................................... [Portals: 1]
31    |       o- 0.0.0.0:3260 ................................................ [OK]
32    o- loopback ................................................... [Targets: 0]
```

1.2.5 Adding a rule to the firewall

Now, we need to allow the TCP connections through port 3260 to use for SAN, using:

```
1    # firewall-cmd --add-port=3260/tcp --permanent
2    success
3    # firewall-cmd --reload
4    success
```

1.2.6 Starting target.service

Even though **targetcli** saves the present configuration to disk, a service called *target.service*
must be enabled to ensure that the saved configuration is loaded each time after reboots.
This is done with:

```
1    # systemctl start target
2    # systemctl enable target
3    Created symlink from /etc/systemd/system/multi-user.target.wants/target.service
↪    to /usr/lib/systemd/system/target.service.
4    # systemctl status target
5    ● target.service - Restore LIO kernel target configuration
6    Loaded: loaded (/usr/lib/systemd/system/target.service; enabled; vendor preset:
↪    disabled)
7    Active: active (exited) since Tue 2018-01-02 16:28:20 IST; 25s ago
8    Main PID: 4291 (code=exited, status=0/SUCCESS)
9
10   Jan 02 16:28:19 vmprime.somuvmnet.local systemd[1]: Starting Restore LIO kern...
11   Jan 02 16:28:20 vmprime.somuvmnet.local systemd[1]: Started Restore LIO kerne...
```

1.3   Connecting the iSCSI Initiator to an iSCSI SAN

1.4   Verifying the iSCSI Connection