iSCSI tricks and techniques

Assumptions

I'm assuming you have built your iSCSI target device using the LVM method in the document Building iSCSI target device. This means that when you want to create/modify/whatever the targets you export, it is simply a matter of manipulating your LVM on the target and, if applicable, adding/deleting entries in /etc/iet/ietd.conf

Terms

See iSCSI Terms for a definition of some of the more “interesting” things here.

Basic Assumptions

In the following, you will see something like “name of target” and “ip:port”. The ip is the IP address (or, if you like, resolvable DNS name), and the port is whatever port your iSCSI target is listening on, generally 3260.

“name of target” is the name of the device exported by the target, generally beginning with the letters 'iqn'. An example would be “iqn.2014-11.net.dailydata.castor:simon0”. You can always get this either by one of the following commands **from the initiator**.

“ *iscsiadm -m discovery -t st -p 10.19.219.2*”

“*iscsiadm -m session*”

The first one (discover) says “Show me all exports on the iSCSI target at 10.19.219.2. the second one (session) says “Show me all exports I know about on this machine.”

Tricks and Techniques

Script to update all targets

For some reason, I'm occasionally a little out of sync between what my target offers and what my initiator knows about. The following script will not *remove* anything, but it will add every target offered by the iscsi target.

Add one or more entries into @servers, and it will scan the targets in question, then compare against what our initiator knows, then add any new entries.

addAlliSCSIS.pl

```
#! /usr/bin/env perl

use strict;
use warnings;

# change following to be a list of 1 or more iSCSI targets to be
queried
my @servers = ( '10.10.10.10','10.10.10.9' );
my %targets;


foreach my $server ( @servers ) {
   print "\n" . '-'x40 . "\nGetting targets on server $server\n" . '-
'x40 . "\n";
   my @list = `iscsiadm -m discovery -t st -p $server`;
   chomp @list;
   # @list contains lines of type
   # 10.19.209.2:3260,1 iqn.2014-11.net.dailydata.castor:simon0
   # split them apart and add them to the hash
   foreach my $entry ( @list ) {
      my ( $portal, $targetName ) = split( ' ', $entry );
      # $portal has some extra info after a comma, so clean it up
      $portal =~ m/^([0-9:.]+)/;
      $portal = $1;
      # some targets return multiple IP's for a given name, so
      # only add them if they are in this IP
      $targets{ $targetName } = $portal if $portal =~ m/^$server/;
      print "$targetName\t$targets{ $targetName }\n";
   }
}

print "\n" . '-'x40 . "\nGetting active sessions\n". '-'x40 . "\n";
# now, get active sessions so we can filter them
my @activeSessions = `iscsiadm -m session`;
chomp @activeSessions;
foreach my $session ( @activeSessions ) {
   $session =~ m/^.*[^0-9:.]([0-9,:.]+).*(iqn\S*)/;
   my ( $portal,$targetName ) = ( $1,$2 );
   print "$portal\t$targetName";
   if ( exists( $targets{$targetName} ) ) {
      print "\tNOT updating\n";
      delete $targets{ $targetName };
   } else {
      print "Needs to be added\n";
   }
}

# check if we have any new entries and bail if not
if ( scalar keys %targets ) {
   # We have new entries, so run them;
   foreach my $targetName ( sort keys %targets ) {
```

```
      my $portal = $targets{$targetName};
      print "Adding $targetName\n";
      `iscsiadm -m node --targetname '$targetName' --portal '$portal' -
-login`;
   }
} else {
   print "No new entries\n";
}
# print `ls /dev/disk/by-path/`;
```

Fixing bolluxed machine

Ok, your initiator is in a really bad shape, or you did something on the target you shouldn't have, and you want to basically reinitialize everything. But, you don't want to shut everything down. This was the procedure I used to do so. I migrated all of my Xen devices onto a different machine, then ran the following on the machine I wanted to reinitialize:

```
/etc/init.d/open-iscsi stop
rm -fR /etc/iscsi/send_targets/* /etc/iscsi/nodes/*
/etc/init.d/open-iscsi start
/media/xen-store/scripts/addAlliSCIS.pl
```

Note: This is on an old Debian Wheezy machine, so start/stop commands are different after that.

The first line stops the initiator. The second line removes all remembered connections from the initiators cache, then we start the initiator again.

At this point, we don't “know” any targets, so we use the script (above) to grab all targets available on the iscsi target. YEAH, we're done.

Viewing targets exported from a target device, from the target device

```
cat /proc/net/iet/volume # Linux
```

Adding new target on target device

Log into your target device. Create a new LV to be exported. The example below creates a 10G partition named 'myserver-disk0'

```
lvcreate -L 10G -n myserver-disk0 iscsi-export-pool
```

Edit /etc/iet/ietd.conf (Linux) or /etc/ctl.conf (BSD) and add the following lines

```
target iqn.2014-11.net.dailydata.castor:myserver-disk0
     Lun 1 Type=fileio,Path=/dev/iscsi-export-pool/myserver-disk0
```

```
    Alias myserver.disk0
```

Restart iet

```
/etc/init.d/iscsitarget restart # Linux
service ctld reload # FreeBSD
```

Adding new target to initiator

After you've added a new target to a target device, you should do the following to get it identified by the initiator. This is assuming your portal is at 10.19.209.2 and your new target is simon0. This is all done on the initiator.

```
# start a new discovery on your target. This will result in a lot of output,
one line per export.
iscsiadm -m discovery -t st -p 10.19.219.2
# change to node mode. I'm not sure this is required, but it doesn't \\ #
hurt. It gives the same information as above
iscsiadm -m node
# Now, actually log in to get access to the export. The first part
# in quotes ("iqn...simon0") directly comes from the second column
# of the previous commends. The second part in quotes
#(10.19.219.2:3260" comes from the first column of the corresponding
# line
iscsiadm -m node --targetname "iqn.2014-11.net.dailydata.castor:simon0" --
portal "10.19.219.2:3260" --login
# devmapper creates an entry in /dev/disk/by-path, so look at it and
# verify you did things correctly. NOTE: may take a couple of seconds
ls /dev/disk/by-path/ | grep simon
```

What do I have on my initiator

In many of these commands, you need to know what targets are currently known by the initiator so you can work on it. The simplest way is to get a listing of everything. Following are two ways of doing that.

```
iscsiadm -m session   # find the name of the target, second column\\ # OR,
the following shows you only the target name
iscsiadm -m session | cut -d',' -f2 | cut -d' ' -f2-
```

Resizing your target

Ok, you messed up and the target is too small. If you are using LVM2 partitions for your exports (good idea), simply take it offline on the initiators, then add space on the target. Now (and I'm not sure this is required), restart iscsi on the target.

However, I generally use a FreeBSD target (with ZFS on it, setting the targets to Volumes). Following

assumes you are using FreeBSD ZFS Volumes for your target, and Devuan (Debian) on your initiator.

Warning: Do not allow anything to access the target during this process. You can probably get by with just shutting down anything which accesses it. Or, for the paranoid amongs us, logout of the target from your initiator, do the work, then log back in.

On FreeBSD target

```
# set the new size on the target (assumed to be /storage/iscsi/target in example(
zfs set volsize=<newsize> storage/iscsi/target
# let ctld know about the change. reload may work instead of restart
service ctld restart
```

On initiator

You now need to rescan the target from the initiators. The first example rescans everything, which may not be what you want.

```
# probably not the best, since it rescans everything.
# see next code block.
iscsiadm -m session --rescan
```

Better to only scan the one you actually changed. This assumes the target has the substring *thing1* in it.

```
# find the Session ID of the target in question.
# NOTE: that you can just do iscsiadm -m session for the whole
# list and manually find the SID (number surrounded by square brackets)
iscsiadm -m session -P 0 | grep thing1 | cut -d'[' -f2 | cut -d']' -f 1
# now, rescan only that SID (that one target)
iscsiadm --mode session --sid=15 --rescan
# use any tool you like to check the new size. I use fdisk -l, which will
# return the size on the first line
fdisk -l /dev/disk/by-path/ip-.....
```

At this point, your size it set correctly (we hope). Now, you need to resize the file system which is different depending on which file system it is. For virtuals, I generally boot from the gparted live CD (https://gparted.org/download.php) to avoid any problems.

Remove a target

Removing a target is easy; simply remove it from the target, restart openiscsi. However, the initiators still have it in their “database” and you must remove it from there also.

To “clean up” after removing a target, you must log out of the target, then clean up the database. Repeat, you must LOG OUT of the target, THEN clean up the database.

```
# first, log out of the session
iscsiadm -m node --target='name of target' --portal "ip:port" --logout
# now, do a remove it from the session
iscsiadm -m node --target='name of target' --portal "ip:port" -o delete
```

What about removing **all** exports from a target, say when you are retiring a complete iSCSI target? That is fairly simple (though you need to log of each individually, I think). If you can shut down the iSCSI initiator, do that, then find the directory/file which contains all the information.

On Debian, it is a directory in /etc/iscsi/send_targets and /etc/iscsi/nodes. Just remove the files/directories. Red Hat appears to use the same thing, but they put the information in /var/lib/iscsi instead of /etc/iscsi.

Once you have removed the directories, you can start up your iSCSI initiator and they will be gone.

You can also do it by logging out of all targets, then issuing some command that I have not been able to figure out. I have to do this soon, so I'll update it after I've done it, but for now, look at the thread at https://groups.google.com/forum/#!topic/open-iscsi/7x28lO6-Rho

LUN's

LUN's always confused me, but I found a very clear explanation at http://ask.metafilter.com/181504/In-an-iSCSI-context-what-do-LUNs-mean. Basically, a LUN is a way to break up a single device into multiple ones. The explanation was, if you have 14 SCSI drive attached to a controller, each one can have the same LUN. However, if each of those “devices” is actually a cage holding 14 more drives, then each cage would have a (different) SCSI address, and the drives inside it would have the same SCSI address as its cage, but different LUNs. Thus, a device in this complex area would be address:lun to identify it.

If you wanted to have several subexports from a single iSCSI export, you would give the iSCSI export a name, then each export inside it would have a different LUN.

In my case, each export I have happens only once, so I can get away with giving each export the same LUN. From what I saw elsewhere, however, vmware (maybe others) do not like this and will barf.

I guess the final answer is, give everything a separate LUN, “just in case”.

Using fileio vs blockio

I found a really great explanation of fileio vs blockio at a discussion on hte iscsitarget development list. You can read it at http://iscsi-enterprise-target.996254.n3.nabble.com/newbie-question-difference-between-fileio-and-blockio-in-IET-td18.html. Basically, the type determines whether the target caches things or not (fileio does, blockio does not), so it depends on what use the export will have. From what I understand, if your target is a file, you should really use fileio no matter what. But, if your target is a partition (lvm, physical partition, physical drive), it will be based on what the use of the partition will be; if you will

have random reads and writes, use blockio. If you will have sequential reads and writes, tend towards fileio.

Also, you should look at how much memory your iSCSI server has. Anything memory over the operating systems requirements are used for caching, so if you have 16G of RAM, go ahead and use fileio; it doesn't hurt anything (well, mostly, I worry about my drbd not being updated until ram is flushed).

The author gave some examples, which I will summarize:

Databases have caching in them already, and they do all sorts of random reads/writes on the target, so in this case, you would want blockio. When a block read request is made, using the cache on the remote machine is really not all that usefull, especially since it is unlikely you will be doing sequential reads from the device.

Virtual Server OS Partition will read a lot of the same places over and over, and the OS itself is generally loaded into memory and stay there. So, fileio is better in this case (not sure I really understand why on this), since the reads will be cached on the iscsi server.

File Server will read large amounts of data (a single file), then write the entire block back out onto the hard drive when the edit is done. So, caching (using fileio) would end up with a faster responce, but then, the file server will likely cache also, so double caching may not make sense, in which case you would want to go with blockio.

The bottom line I read into this is, if your target has a lot of memory, and that memory is generally available, go ahead and use fileio, since it will be cached. However, if your initiator is going to use the space for something that requires fast access to random locations on the target's physical drives, use blockio.

Old Stuff

I'm leaving this here for archival use, but most of it is superceded by the above.

Well, you need some volumes for your initiators to use. This can be just about anything, but I've found LVM to be the most flexible on the target. I take my large storage, turn it into an LVM Physical Volume, build a volume group from it, then create Logical Volumes that I export to the clients. Assume you have a software RAID set /dev/md0, you can do the following:

```
pvcreate /dev/md0
vgcreate iscsi-storage /dev/md0
lvcreate -L 100G -n myExport1 iscsi-storage
```

Now, simply export /dev/iscsi-storage/myExport1 from your iSCSI server and it is available for partitioning and formatting on your target machine(s).

Note: Just because you can connect to the same export from multiple machines doesnt mean you should mount them. Unless you have a file system on an export which knows how to handle writes from several systems, you will end up with corruption. However, in most cases, you can write from one machine and read from the others. I generally set my iSCSI server up to do a couple of NFS exports simply to solve this problem. However, NFS degrades rapidly as the number of machines and

writes increases, so I do not use NFS for things like File Backed Devices (FBD's) for Xen. Instead I export and LVM as follows.

LVM2 -- I do NOT do this anymore. Too many headaches. I create my exports as LV's, but simply send them in-situ as single devices. I'm leaving this here just because.

The main thing my iSCSI server does is provide space for Xen virtual machines. Xen can handle virtual machines which are actually files on an NFS mount, but the performance degrades rapidly. So, instead, I use LVM2.

Aside from a few headaches, LVM it is the most flexbile, and it does not degrade like NFS. However, it requires you manually rescan whenever a change is made, and YOU are responsible for making sure only one machine uses an LVM partition at a time. You also need to rescan the LVM2 metadata on all machines if you make a change. CLVM (Clustered LVM) fixes both of these problems, but I did not know about it when I tested.

Basically, you can avoid these issues by the following two rules:

- Force rescan of LVM on all machines whenever you make an LVM2 change on any single machine. Our practice is to only modify the LVM structure on the iSCSI target, then do a rescan with */etc/init.d/lvm2 start* on all initiators. This is likely only necessary if you are going to move a virtual, but if you're doing that, you have other problems. I have thought about simply running the scan as a cron job daily.
- Keep a record of which machines are responsible for which Logical Volumes. I have had the times when I have brought up the same virtual on two DOM0's simultaneously, which is very bad. There are tools which manage which DOM0 has which DOMU's and avoid that, but I have not investigated; we just do it by hand.

I chose a slightly more complicated setup to increase flexibility and with an eye for mirroring the target using DRBD in the future. The initiator has a RAID set, where the md is a physical volume (pv) for a volume group (vg). I then create a very large Logical Volume (lv) which I turn into a physical volume for a new volume group. This physical volume is what I export. Example:

/dev/md0 – RAID-6 array set as LVM physical volume

iscsi-storage – LVM volume group using /dev/md0 as my physical volume

iscsi-export – Logical Volume which is a member of the volume group iscsi-storage

I then export /dev/iscsi-storage/iscsi-export to the initiators. On one of the devices (either the initiator or the target itself), I take this and turn it into a Physical Volume and create multiple logical volumes from it.

What I gain from this is maintenance simplicity (albeit, a more complex setup on the target). The initiators only care about one iSCSI export from the target; iscsi-export. The images for my DOMU's are all in this, so once that single connection is made, the initiators have (almost) all of the information they need to start virtuals. **Note**: I said almost. I actually have an NFS export from the iSCSI server which contains the configuration files for the Xen virtuals.

Once the iSCSI initiator is up, I simply tell LVM to rescan and it finds all of my exported volumes. I can

also change (extend, reduce, add, remove) the logical volumes on any of the machines (including the target) and simply tell LVM to rescan. The simplest solution to that is to run

```
/etc/init.d/lvm2 start
```

which rescans everything and creates all the nodes you need.

It also means, when I set up DRBD, I only have to mirror iscsi-export, and it will have all of my virtuals in it (well, assuming I have the configurations someplace).

From:
https://kb.unixservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://kb.unixservertech.com/unix/iscsi/iscsi_tricks_and_techniques

Last update: **2026/01/17 03:23**