Open-iSCSI.org Software iSCSI Initiator Configuration Guide

Abstract

This document covers installing and configuring the open-iSCSI.org Software iSCSI initiator under debian or Ubuntu Linux and enabling the MPIO capability

V1.05

August 2010

Dell, Inc.
300 Innovative Way
Suite 301
Nashua, NH 03063

Tel: 603.579.9762
Fax: 603.579.6910

Support: 877.887.7337

REVISION INFORMATION

The following table describes the release history of this document.

Technical Report Revision	Date	Change
1.0	12/2007	Initial Release
1.01	7/2008	Updated for change in MPIO requirement to specify transport layer
1.02	9/2008	Corrected addr . Updated revision to 2.0-869. and 2.6.26+
1.03	9/2008	Builds 2.0-870+ have new discovery options needed.
1.04	4/2010	Updated to match RH doc
1.05	8/2010	Corrected logout command , removed queue_if_no_path

Table of Contents

Revision Information 3

Installation Notes: 6

Obtaining and unpacking the source code: 7

Compiling the source code 8

Discovering Targets: 11

Logging in: 12

- Log into all targets. 12
- Log into an individual target 12

Logging out: 12

- Logging off an individual target 12
- Logging out all targets 12

Checking Session Status: 13

Mapping EQL volume name to /dev/sd(*X*) device name 14

Using CHAP authentication: 15

Persistent Device naming: 16

Setting the default values for all Equallogic devices: 21

Create Filesystem on iSCSI volume: 22

Mounting iSCSI Filesystems at Boot: 23

Configuring Mulitpath Connections: 24

Flowcontrol 26

Enabling Jumbo Frames: 27

Performance tuning options: 28

- #/etc/iscsi/iscsid.conf 28
- #/etc/mulitpath.conf 28
- Linux Read Ahead Value 29
- Changing Linux I/O scheduler 30

Known issues: 31

- Linux ARP behavior: 31
- Linux Netfilter: 31

INSTALLATION NOTES:

Notes:

- Don't use the debian open-iSCSI package that comes with debian etch. At the time of this document, that debian package was old, and was v1.0 build 754. The current Open-iSCSI build as of this writing is 2.0-871.tar.gz http://www.open-iscsi.org
- This guide can be used with Ubuntu v7.x/8.x and debian etch.
- Minimum suggested kernel revision 2.6.18.
- You need 'flex', 'bison', 'gcc' and at the kernel-headers for your kernel in order to compile the open-iSCSI initiator.
- DISCLAIMER: Currently EqualLogic hasn't certified the open-iSCSI initiator. We cannot guarantee its suitability. We suggest that you subscribe to the open-iscsi.org mailing list for technical support. Please contact your EQL sales rep or customer support for the latest information.

OBTAINING AND UNPACKING THE SOURCE CODE:

First download the tarball from the open-iSCSI website; open-iscsi-2.0-871.1.tar.gz.

```
# mkdir /source
# cd /source
# wget http://www.open-iscsi.org/bits/open-iscsi-2.0-871.1.tar.gz
--13:03:26--  http://www.open-iscsi.org/bits/open-iscsi-2.0-87.1.tar.gz
           => `open-iscsi-2.0-871.1.tar.gz'
Resolving www.open-iscsi.org... 209.210.238.66
Connecting to www.open-iscsi.org|209.210.238.66|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 279,453 (273K) [application/x-gzip]

100%[=====================================================>] 279,453      112.50K/s

13:03:29 (112.26 KB/s) - `open-iscsi-2.0-871.1.tar.gz' saved [279453/279453]
```

Now unpack the archive.

```
#tar xvzf open-iscsi-2.0-871.1.tar.gz
open-iscsi-2.0-871.1/
open-iscsi-2.0-871.1/kernel/
open-iscsi-2.0-871.1/kernel/libiscsi.c
open-iscsi-2.0-871.1/kernel/scsi_transport_iscsi.c
open-iscsi-2.0-871.1/kernel/iscsi_tcp.h
open-iscsi-2.0-871.1/kernel/2.6.16-18_compat.patch
open-iscsi-2.0-871.1/kernel/2.6.20-21_compat.patch
open-iscsi-2.0-871.1/kernel/iscsi_tcp.c
....
```

COMPILING THE SOURCE CODE

Change to that directory

#cd <source dir>/open-iscsi-2.0-871.1

There are two edits that we suggest you do before building the initiator.

1. Enable auto connect at startup time. This means that on startup discovered volumes will be connected automatically. (Default is manual)
2. Add a 'killall' line to the startup script to ensure the iSCSI daemon is stopped.

Enable automatic connect in the source directory edit `<source dir>/etc/iscsid.conf`

#vi etc/iscsi.conf

```
#*****************
# Startup settings
#*****************

# To request that the iscsi initd scripts startup a session set to "automatic".
# node.startup = automatic
#
# To manually startup the session set to "manual". The default is manual.
node.startup = manual
```

- Uncomment 'node.startup = automatic'
- Comment out 'node.startup = manual'

So it will look like this when you're done.

```
#*****************
# Startup settings
#*****************

# To request that the iscsi initd scripts startup a session set to "automatic".
node.startup = automatic
#
# To manually startup the session set to "manual". The default is manual.
#node.startup = manual
```

Save the file and exit.

Add the '*killall*' command to the startup script.

#vi etc/initd/initd.debian (this script is used for both Ubuntu and debian)

Find the section labeled 'stop()'

```
stop() {
        stoptargets
        log_daemon_msg "Stopping iSCSI initiator service"
        start-stop-daemon --stop --quiet --pidfile $PIDFILE --exec $DAEMON
        rm -f $PIDFILE
        modprobe -r ib_iser 2>/dev/null
        modprobe -r iscsi_tcp 2>/dev/null
        log_end_msg 0
}
```

Add a 'killall –1 iscsid' after the 'rm –f $PIDFILE' line.

```
        rm -f $PIDFILE
        killall -1 iscsid
```

Save the file and exit.

Now you can build the code.

1. Run 'make clean'

```
# make clean
make -C utils clean
make[1]: Entering directory `/source/open-iscsi-2.0-871.1/utils'
rm -f *.o iscsi-iname
make[1]: Leaving directory `/source/open-iscsi-2.0-871.1/utils'
make -C usr clean
make[1]: Entering directory `/source/open-iscsi-2.0-871.1/usr'
rm -f *.o iscsid iscsiadm iscsistart
make[1]: Leaving directory `/source/open-iscsi-2.0-871.1/usr'
make -C kernel clean
make[1]: Entering directory `/source/open-iscsi-2.0-871.1/kernel'
make -C /lib/modules/2.6.22-14-generic/build M=`pwd` KBUILD_OUTPUT=  V=0
clean
make[2]: Entering directory `/usr/src/linux-headers-2.6.22-14-generic'
make[2]: Leaving directory `/usr/src/linux-headers-2.6.22-14-generic'
rm -f Module.symvers
make[1]: Leaving directory `/source/open-iscsi-2.0-871.1/kernel'
make -C utils/fwparam_ibft clean
make[1]: Entering directory `/source/open-iscsi-2.0-
871.1/utils/fwparam_ibft'
rm -f *.o fwparam_ibft
make[1]: Leaving directory `/source/open-iscsi-2.0-871.1/utils/fwparam_ibft'
```

2. Run ‘make’ when complete you’ll see the following.

```
Compilation complete                Output file
----------------------------------- ---------------
Built iSCSI Open Interface module:  kernel/scsi_transport_iscsi.ko
Built iSCSI library module:         kernel/libiscsi.ko
Built iSCSI over TCP kernel module: kernel/iscsi_tcp.ko
Built iSCSI daemon:                 usr/iscsid
Built management application:       usr/iscsiadm
Built utility:                      utils/fwparam_ibft/fwparam_ibft

Read README file for detailed information.
```

3. Run ‘make install’

```
# /etc/init.d/open-iscsi start
```

Enable the iSCSI service to be started at boot time; the *update-.rc.d* command can be used as follows:

```
# update-rc.d –f open-iscsi defaults
  Adding system startup for /etc/init.d/open-iscsi ...
   /etc/rc0.d/K20open-iscsi -> ../init.d/open-iscsi
   /etc/rc1.d/K20open-iscsi -> ../init.d/open-iscsi
   /etc/rc6.d/K20open-iscsi -> ../init.d/open-iscsi
   /etc/rc2.d/S20open-iscsi -> ../init.d/open-iscsi
   /etc/rc3.d/S20open-iscsi -> ../init.d/open-iscsi
   /etc/rc4.d/S20open-iscsi -> ../init.d/open-iscsi
   /etc/rc5.d/S20open-iscsi -> ../init.d/open-iscsi
```

> *You also need to do the same for the Multipath daemon.*

```
# update-rc.d –f multipath-tools defaults
(If it’s already set you’ll see this message)
System startup links for /etc/init.d/multipath-tools already exist.
```

DISCOVERING TARGETS:

Once you have the iSCSI service running you will use the '*iscsiadm*' utility to discover, login and logout of targets.

To get a list of available targets type:

#iscsiadm –m discovery –t st –p <Group IP address>:3260

Example:

#iscsiadm -m discovery -t st -p 172.23.10.240:3260

```
172.23.10.240:3260,1 iqn.2001-05.com.equallogic:0-8a0906-
83bcb3401-16e0002fd0a46f3d-open-iscsi-test
```

The example shows that the '*open-iscsi-test*' volume has been found. However, it's not yet logged in.

Note: Starting with builds 2.0-870 and greater there's a change in how you add additional discovery addresses to find targets on another Group.

In earlier versions of Open-iSCSI (OiS) you could simply re-run the discovery command.

#iscsiadm -m discovery -t st -p 172.23.10.240:3260

However, with builds 2.0-870 and greater this will now generate an error.

#iscsiadm -m discovery -t st -p 172.23.10.10:3260
iscsiadm: This command will remove the record [iface: iface.eth0, target: iqn.2001-05.com.equallogic:6-8a0900-ffef60402-36c23acd6e745be0-dw-open-iscsi-vol0, portal: 172.23.10.240,3260], but a session is using it. Logout session then rerun command to remove record.

To resolve this you need to add '*-op new*' to the command. This will create a new discovery entry for the new target.

#iscsiadm -m discovery -t st -p 172.23.10.10:3260 -o new

LOGGING IN:

- LOG INTO ALL TARGETS.

```
#iscsiadm –m node –l
```

- LOG INTO AN INDIVIDUAL TARGET.

```
#iscsiadm –m node –l –T <Complete Target Name> –l –p <Group IP address>:3260
```

Example:

```
#iscsiadm –m node –l –T iqn.2001-05.com.equallogic: 83bcb3401-16e0002fd0a46f3d-open-iscsi-test –p <Group IP address>:3260
```

This is useful when using array snapshots.

LOGGING OUT:

- LOGGING OFF AN INDIVIDUAL TARGET.

```
#iscsiadm –m node –u –T iqn.2001-05.com.equallogic:0-8a0906-83bcb3401-16e0002fd0a46f3d-rhel5-test –p <Group IP address>:3260
```

Logging in and out of individual targets is very useful especially when using array snapshots.

- LOGGING OUT ALL TARGETS.

```
#iscsiadm –m node –u
```

CHECKING SESSION STATUS:

To see the connection status run:

```
#iscsiadm -m session -P 0
tcp: [1] 172.23.10.240:3260,1 iqn.2001-05.com.equallogic:6-8a0900-ffef60402-36c23acd6e745be0-dw-open-iscsi-vol0
tcp: [2] 172.23.10.240:3260,1 iqn.2001-05.com.equallogic:6-8a0900-ffef60402-36c23acd6e745be0-dw-open-iscsi-vol0
tcp: [3] 172.23.10.240:3260,1 iqn.2001-05.com.equallogic:6-8a0900-8d1cb3401-c2f0002decc46730-dw-open-iscsi-backup
tcp: [4] 172.23.10.240:3260,1 iqn.2001-05.com.equallogic:6-8a0900-8d1cb3401-c2f0002decc46730-dw-open-iscsi-backup
```

For a more verbose output change to '-P 1'

```
#iscsiadm -m session -P 1
Target: iqn.2001-05.com.equallogic:6-8a0900-ffef60402-36c23acd6e745be0-dw-open-iscsi-vol0
        Current Portal: 172.23.10.246:3260,1
        Persistent Portal: 172.23.10.240:3260,1
                **********
                Interface:
                **********
                Iface Name: iface.eth0
                Iface Transport: tcp
                Iface Initiatorname: iqn.2005-03.org.open-iscsi:2009610cb1b
                Iface IPaddress: 172.23.49.170
                Iface HWaddress: 00:04:23:C7:E1:5A
                Iface Netdev: default
                SID: 1
                iSCSI Connection State: LOGGED IN
                iSCSI Session State: LOGGED_IN
                Internal iscsid Session State: NO CHANGE
        Current Portal: 172.23.10.244:3260,1
        Persistent Portal: 172.23.10.240:3260,1
                **********
                Interface:
                **********
                Iface Name: iface.eth1
                Iface Transport: tcp
                Iface Initiatorname: iqn.2005-03.org.open-iscsi:2009610cb1b
                Iface IPaddress: 172.23.49.171
                Iface HWaddress: 00:04:23:C7:E1:5B
                Iface Netdev: default
                SID: 2
                iSCSI Connection State: LOGGED IN
                iSCSI Session State: LOGGED_IN
                Internal iscsid Session State: NO CHANGE
```

MAPPING EQL VOLUME NAME TO /DEV/SD(*X*) DEVICE NAME

To see what SCSI devices are being used run:

#iscsiadm -m session –P3 | less

Sample output:

```
iscsiadm version 2.0-742
************************************
Session (sid 1) using module tcp:
************************************
TargetName: iqn.2001-05.com.equallogic:6-8a0900-ffef60402-36c23acd6e745be0-dw-open-iscsi-vol0
Portal Group Tag: 1
Network Portal: 172.23.10.240:3260
iSCSI Connection State: LOGGED IN
.

.

.

.


************************
Attached SCSI devices:
************************
Host Number: 2  State: running
scsi2 Channel 00 Id 0 Lun: 0
Attached scsi disk sdb          State: running
```

USING CHAP AUTHENTICATION:

To enable CHAP for ALL targets, edit the `/etc/iscsi/iscsid.conf` file. Find the CHAP setting section and uncomment the following and add in the CHAP username and passwords

```
# *************
# CHAP Settings
# *************

# To enable CHAP authentication set node.session.auth.authmethod
# to CHAP. The default is None.
node.session.auth.authmethod = CHAP

# To set a CHAP username and password for initiator
# authentication by the target(s), uncomment the following lines:
node.session.auth.username = username
node.session.auth.password = password


# To enable CHAP authentication for a discovery session to the target
# set discovery.sendtargets.auth.authmethod to CHAP. The default is None.
discovery.sendtargets.auth.authmethod = CHAP

# To set a discovery session CHAP username and password for the initiator
# authentication by the target(s), uncomment the following lines:
discovery.sendtargets.auth.username = username
discovery.sendtargets.auth.password = password
```

To enable CHAP for a particular target use the `iscsiadm` command to update the settings for that target.

```
#iscsiadm  -m node -T "<Complete Target Name>"  -p <Group IP>:3260 --
op=update --name node.session.auth.authmethod --value=CHAP

#iscsiadm -m node -T "<Complete Target Name>“ -p <Group IP>:3260 --op=update
--name node.session.auth.username --value=<CHAP user>

#iscsiadm -m node -T "<Complete Target Name>“ -p <Group IP>:3260 --op=update
--name node.session.auth.password --value=<password>

#iscsiadm -m node –T "<Target Name>" -p <Group IP>:3260 -l
```

I.e.

```
#iscsiadm -m node –T iqn.2001-05.com.equallogic:0-8a0906-83bcb3401-
16e0002fd0a46f3d-rhel5-test –p 172.23.10.240:3260 –l
```

PERSISTENT DEVICE NAMING:

Devices using the software initiator do not have a persistent naming scheme, and do not guarantee that a device (i.e. */dev/sdc*) will always have the same device node. Adding or removing a disk can change the device order on the next boot up. Persistent Naming describes mechanisms where the system identifies devices without relying on the /dev node, and provides a reference point for it that does not change at reboot.

First you have to comment out the 'Blacklist all devices' section in `/etc/multipath.conf` file

Note:

If the example file, `multipath.conf` is not in `/etc`, copy it from `/usr/share/doc/multipath-tools/multipath.conf.synthetic`

```
#cp /usr/share/doc/multipath-tools/multipath.conf.synthetic
/etc/multipath.conf
```

```
# Blacklist all devices by default. Remove this to enable multipathing
# on the default devices.
blacklist {
        devnode "*"
}
```

```
# By default all devices are blacklisted. Modify this to enable multipathing

# on the default devices.  Most will want to exclude /dev/sda and /dev/sdb,
exceptions would include those booting from SAN and wanting MPIO support.
```

Note: Please review the blacklist settings to make sure they're applicable. Some systems may require that some devices, including boot disks, remain blacklisted. Your OS vendor may be able to provide more specific guidance

So at minimum it should look like this:

```
blacklist {
        devnode "^sd[a]$"
}
(This excludes the first SCSI/SATA disk)
```

Or

```
blacklist {
        devnode "^sd[ab]$"
}
(This excludes the first two disks if you are mirroring your boot drives)
```

To cover other objects typically not needed by MPIO add the following:

```
blacklist {
        wwid SATA_WDC_WD2500YS-18_WD-WCANY4730307 (SAMPLE for local SATA HD)
        devnode "^sd[a]$"
        devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
        devnode "^hd[a-z][[0-9]*]"
        devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"

}
```

More information about blacklisting devices can be found at:

http://kbase.redhat.com/faq/docs/DOC-4042

Then restart the multipathd daemon

#/etc/init.d/multipathd restart

Now check that dev-mapper has configured the volume.

```
#multipath –v2
#multipath -ll
mpath0 (36090a01840b3bc833d6fa4d02f00e016) dm-2 EQLOGIC,100E-00
[size=8.0G][features=0][hwhandler=0]
\_ round-robin 0 [prio=1][active]
 \_ 2:0:0:0 sdb 8:16  [active][ready]
```

The highlighted number is the UUID of the volume. That never changes. You can use that UUID to create a persistent, friendlier name. For example you can name it the same as you called the volume on the EQL array.

Again edit the `/etc/multipath.conf` file.

Uncomment the following section and change the defaults to match your UUID and set a friendly alias name.

Edit the `/etc/multipath.conf` file and uncomment out the following:

```
#multipaths {
#       multipath {
#               wwid                    3600508b4000156d700012000000b0000
#               alias                   yellow
#               path_grouping_policy    multibus
#               path_checker            readsector0
#               path_selector           "round-robin 0"
#               failback                manual
#               rr_weight               priorities
#               no_path_retry           5
#               rr_min_io               100
#       }
#       multipath {
#               wwid                    1DEC_____321816758474
#               alias                   red
#       }
#}
```

Change the number after '*wwid*' to the UUID for your volume. Change the '*alias*' to something more friendly or use the volume name from the array. Change the '*rr_min_io*' to 10.

Here's an example showing how to create persistent names for more than one volume.

```
multipaths {
        multipath {
                wwid                    36090a02830f251891f74744263735281
                alias                   open-iscsi-test
                path_grouping_policy    multibus
                path_checker            readsector0
                path_selector           "round-robin 0"
                failback                manual
                rr_weight               priorities
                no_path_retry           5
                rr_min_io               10
        }
        multipath {
                wwid                    36090a01840b31c74e173a4873200a02f
                alias                   svr-vol
        }
}
```

Save the file, then run:

```
#multipath -v2
#multipath -ll
 open-iscsi-test (36090a02830f251891f74744263735281) dm-1 EQLOGIC,100E-
00
[size=100G][features=1 queue_if_no_path][hwhandler=0]
\_ round-robin 0 [prio=0][active]
 \_ 9:0:0:0 sdc 8:48  [active][ready]
 svr-vol (36090a01840b31c74e173a4873200a02f) dm-0 EQLOGIC,100E-00
[size=10G][features=0][hwhandler=0]
\_ round-robin 0 [prio=0][enabled]
 \_ 6:0:0:0 sdb 8:16  [active][ready]
```

```
#ls -l /dev/mapper
 total 0
crw-rw---- 1 root root  10, 63 2007-11-16 17:15 control
brw-rw---- 1 root disk 254,  1 2007-11-19 15:59 open-iscsi-test
brw-rw---- 1 root disk 254,  0 2007-11-19 15:58 svr-vol
```

> *You now have persistent names to access those volumes.*

Note: It's not required, but if you partition the device, devmapper will create a –`part1` (or p1) MPIO device. You will have to use that or you will get a "device busy" error.

I.e. `/dev/mapper/open-iscsi-test-part1` or `/dev/mapper/open-iscsi-testp1` which represents the partition on that volume.

Use that for device for all filesystem creation and mount commands.

A second partition slice, would be –part2 or –testp2

Now create a filesystem on that device. In this example we're using the EXT3 filesystem. You are free to use any supported filesystem.

Example:

#**mke2fs –j –v /dev/mapper/open-iscsi-test** creates an EXT3 filesystem on that device.

SETTING THE DEFAULT VALUES FOR ALL EQUALLOGIC DEVICES:

This will configure the default parameters for all EQL devices. You can change the parameters afterwards on a volume-by-volume basis in the 'multipaths' section. I.e. you could have a different *rr_min_io* setting for SQL volumes vs. NFS share volumes. Either method works fine. This just makes it easier as you add new volumes, they will automatically get these default settings. You can still set them all individually, if you wish.

Find the "devices" section near the end of the file.

```
} devices {

device {
       vendor "EQLOGIC"
       product "100E-00"
       path_grouping_policy multibus
       getuid_callout "/sbin/scsi_id -g -u -s /block/%n"
       path_checker readsector0
       failback immediate
       no_path_retry            5
       path_selector "round-robin 0"
       rr_min_io 10 <-- See tuning section for more on this setting
       rr_weight priorities
 }
}
```

Then restart the multipathd daemon

```
#service multipathd restart
```

Now check that dev-mapper has configured the volume.

```
#multipath –v2
#multipath -ll
mpath0 (36090a01840b3bc833d6fa4d02f00e016) dm-2 EQLOGIC,100E-00
[size=8.0G][features=0][hwhandler=0]
\_ round-robin 0 [prio=1][active]
 \_ 2:0:0:0 sdb 8:16  [active][ready]
```

The highlighted number is the UUID of the volume. That never changes. You can use that UUID to create a persistent, friendlier name. For example you can name it the same as you called the volume on the EQL array.

CREATE FILESYSTEM ON ISCSI VOLUME:

Now create a filesystem on that device.

Example:

```
#mke2fs –j –v /dev/mapper/open-iscsi-test
```

This creates an EXT3 filesystem on that device.

Any filesystem that your Linux OS supports is available for iSCSI volumes.

Note: It's not required, but if you partition the device, devmapper will create a –part1 (or p1) MPIO device. You will have to use that or you will get a "device busy" error.

I.e. /dev/mapper/open-iscsi-test-part1 or /dev/mapper/open-iscsi-testp1which represents the partition on that volume. Use that for device for all filesystem creation and mount commands.

A second partition slice, would be –part2 or –testp2

Now create a filesystem on that device. In this example we're using the EXT3 filesystem

Example:

```
#mke2fs –j –v /dev/mapper/open-iscsi-test creates an EXT3 filesystem on that device.
```

Disk virtualization products like Logical Volume Manager (LVM), may require modification to the system start up scripts, since the iSCSI service starts after LVM. This means LVM'd iSCSI volumes won't be activated automatically. This is beyond the scope of this document.

MOUNTING ISCSI FILESYSTEMS AT BOOT:

In order to mount a filesystem that exists on an iSCSI Volume connected through the open-iSCSI Software initiator, you need to add a line to the /etc/fstab file. The format of this line is the same as any other device and filesystem with the exception being that you need to specify the *_netdev* mount option, and you want to have the last two numbers set to 0 (first is a dump parameter and the second is the fsck pass).

The *_netdev* option delays the mounting of the filesystem on the device listed until after the network has been started and also ensures that the filesystem is unmounted before stopping the network subsystem at shutdown.

An example of a `/etc/fstab` for a filesystem to be mounted at boot that exists on an iSCSI Volume is as follows:

```
#cat /etc/fstab
LABEL=/1                /                       ext3    defaults        1 1
tmpfs                   /dev/shm                tmpfs   defaults        0 0
devpts                  /dev/pts                devpts  gid=5,mode=620  0 0
sysfs                   /sys                    sysfs   defaults        0 0
proc                    /proc                   proc    defaults        0 0
LABEL=SWAP-sda5         swap                    swap    defaults        0 0
#
## Equallogic iSCSI volumes
#
/dev/mapper/open-iscsi-test   /mnt/test          ext3  _netdev,defaults  0 0
```

Once you've added the `/etc/fstab` entry you can manually mount the volume with:

#mount /mnt/test

Or if you just want to manually mount the volume, i.e. for a snapshot use:

#mount <device name> <mount point>

*Note: The mount point must already exist.

CONFIGURING MULITPATH CONNECTIONS:

To create the multiple logins needed for Linux dev-mapper to work you need to create an '*interface*' file for each GbE interface you wish to use to connect to the array.

Use the following commands to create the interface files for MPIO.

(Select the appropriate Ethernet interfaces you're using.)

#iscsiadm -m iface -I eth0 -o new
New interface eth0 added

Repeat for the other interface, i.e. eth1

#iscsiadm –m iface –I eth1 –o new
New interface eth1 added

Now update the interface name for each port:

#iscsiadm –m iface –I eth0 - -op=update –n iface.net_ifacename –v eth0
eth0 updated
#iscsiadm –m iface –I eth1 - -op=update –n iface.net_ifacename –v eth1
eth1 updated

Here's an example of what the `/var/lib/iscsi/ifaces/eth0` looks like:

iface.iscsi_ifacename = eth0
iface.net_ifacename = eth0
iface.hwaddress = default
iface.transport_name = tcp

If you have already discovered your volumes, you now need to re-discover the target(s).

```
#iscsiadm –m discovery –t st –p <Group IP Addr>:3260
172.23.10.90:3260,1 iqn.2001-05.com.equallogic: 0-8a0906-83bcb3401-
16e0002fd0a46f3d-open-iscsi-test
172.23.10.90:3260,1 iqn.2001-05.com.equallogic: 0-8a0906-83bcb3401-
16e0002fd0a46f3d-open-iscsi-test
```

You should see the volume info for each interface you specified. In this example, two interfaces were defined so we see the volume listed twice.

You'll need to log into the volume.

#iscsiadm –m node –l –T iqn.2001-05.com.equallogic:0-8a0906-8951f2302-815273634274741f-open-iscsi-test –p <Group IP Addr>:3260

#iscsiadm –m session

```
 tcp: [3] 172.23.10.90:3260,1 iqn.2001-05.com.equallogic: 0-8a0906-
83bcb3401-16e0002fd0a46f3d-open-iscsi-test
tcp: [4] 172.23.10.90:3260,1 iqn.2001-05.com.equallogic: 0-8a0906-
83bcb3401-16e0002fd0a46f3d-open-iscsi-test
```

This shows that both adapters have connected to the array.

To verify that the multipathing is correctly configured type:

#multipath –v2
#multipath –ll

```
 open-iscsi-test(36090a02830f251891f74744263735281)dm-1 EQLOGIC,100E-00
[size=100G][features=1 queue_if_no_path][hwhandler=0]
\_ round-robin 0 [prio=0][active]
 \_ 9:0:0:0 sdd 8:48  [active][ready]
 \_ 8:0:0:0 sde 8:64  [active][ready]
svr-vol (36090a01840b31c74e173a4873200a02f) dm-0 EQLOGIC,100E-00
[size=10G][features=0][hwhandler=0]
\_ round-robin 0 [prio=0][enabled]
 \_ 6:0:0:0 sdb 8:16  [active][ready]
 \_ 7:0:0:0 sdc 8:32  [active][ready]
```

In this example you see that there are two paths to each volume.

FLOWCONTROL

Be sure that the network interfaces are configured to use Flow Control and Jumbo Frames if supported. To do this, use the *ethtool* utility on Red Hat.

To check for Flow Control (RX/TX Pause) on your interface, use:

```
# ethtool –a <interface>
```

Example:

```
# ethtool -a eth0
Pause parameters for eth0:
Autonegotiate:  on
RX:             off
TX:             off
```

To set Flow Control to on with `ethtool` use:

```
#ethtool –A <interface> autoneg off [rx|tx] on
```

Example:

```
# ethtool –A eth0 autoneg off rx on tx on
# ethtool -a eth0
Pause parameters for eth0:
Autonegotiate:  off
RX:             on
TX:             on
```

Using `ethtool` will not persistently set this setting. See the manufacturer of the NIC for steps to configure the Flow Control setting for the NIC, or add the `ethtool` command to the end of the `/etc/rc.d/rc.local` file to set Flow Control for rx and tx to on. The majority of GbE NICs will detect and enable flowcontrol properly

ENABLING JUMBO FRAMES:

Note: Not all switches support both Jumbo Frames and Flowcontrol simultaneously. Please check with your switch vendor first before enabling jumbo frames. We also suggest first running without jumbo frames to set a baseline, and then see what if any improvement you get with jumbo frames enabled.

For Jumbo Frames, you can use the *ifconfig* utility to make the change on a running interface. Unfortunately, this change will revert back to the default on a system reboot.

First, show the current setting for the interface in question using the *ifconfig* command using the interface name as an argument:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0E:0C:70:C9:63
          inet addr:172.19.51.160  Bcast:172.19.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

The setting we are interested in is the MTU value. As we can see from the above output, this is currently set to 1500 which is not a Jumbo packet size. To change this, we need to set the mtu again using the ifconfig command as follows:

```
# ifconfig eth0 mtu 9000
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0E:0C:70:C9:63
          inet addr:172.19.51.160  Bcast:172.19.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
```

To make this setting persistent, you need to add the `MTU="<mtu-value>"` parameter to the end of the ifcfg startup scripts for your SAN interfaces. These are found in the `/etc/sysconfig/network-scripts` directory. The naming format of the cfg files for your interfaces is `ifcfg-<interface instance>`.

A sample output of one of these files after adding the MTU line.

```
#cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
address 192.168.1.100
netmask 255.255.255.0
gateway 192.168.1.1
MTU=9000
```

Once the above Network changes have been made, you should reboot the host to verify that they have been made correctly and that the settings are persistent.

PERFORMANCE TUNING OPTIONS:

- #/ETC/ISCSI/ISCSID.CONF

```
# cat /etc/iscsi/iscsid.conf | grep -v "#"
node.startup = automatic
node.session.timeo.replacement_timeout = 120
node.conn[0].timeo.login_timeout = 15
node.conn[0].timeo.logout_timeout = 15
node.conn[0].timeo.noop_out_interval = 5
node.conn[0].timeo.noop_out_timeout = 5
node.session.err_timeo.abort_timeout = 15
node.session.err_timeo.lu_reset_timeout = 20
node.session.initial_login_retry_max = 4
node.session.cmds_max = 1024 < --- changed from the default of 128
node.session.queue_depth = 128 < --- default here was 32
node.session.iscsi.InitialR2T = No
node.session.iscsi.ImmediateData = Yes
node.session.iscsi.FirstBurstLength = 262144
node.session.iscsi.MaxBurstLength = 16776192
node.conn[0].iscsi.MaxRecvDataSegmentLength = 131072
discovery.sendtargets.iscsi.MaxRecvDataSegmentLength = 32768
node.session.iscsi.FastAbort = Yes
```

Note: This is the default template, if you have already discovered targets you will need to either update the individual configuration files (/var/lib/iscsi/nodes/xxx) or rediscover your targets.

- #/ETC/MULITPATH.CONF

rr_min_io

In `/etc/multipath.conf`, the `rr_min_io` parameter sets how many IOs go down a path before switching to another path. A lower number, 10-20, tend to work better in SQL environments. Larger numbers work better for more sequential loads. (I.e. 100 to 512). Larger values, 200+ require that you increase the max commands and queue depth parameters. See "`iscsid.conf`" info above.

When you change the minimum IO setting you must either restart the multipath service or run `#multipath -v2.` To find the optimal setting for your environment will require some experimentation.

- LINUX READ AHEAD VALUE

By default Linux requests the next 256 sectors when doing a read. In a very sequential environment, increasing this value can improve read performance.

You can set the read-ahead on an sd device by using the "*blockdev*" command. This tells the SCSI layer to read X sectors ahead. This is only valuable with sequential I/O-type applications, and can cause performance problems with high random I/O. Under sequential I/O. the performance gain was observed to be in the 10%-20% range.

Syntax: **`blockdev –setra X <device name>`**

i.e.

#**`blockdev –setra 4096 /dev/sda or /dev/mapper/mpath1`**

(Note: 4096 is just an example value. You will have to do testing to determine the optimal value for your system). The OS will read-ahead X pages, and throughput will be higher.

To make the blockdev change effective every time you boot, add the following to `/etc/rc.d/rc.local`.

```
/sbin/blockdev –setra 4096 <device name>
/sbin/blockdev –setra 4096 <device name>
```

etc…

You can also put a setting in /etc/sysctl.conf which will set the read-ahead on boot:
/sys/bus/scsi/drivers/sd/[DEVICEID]/block/queue/read_ahead_kb

Note: You can also use the persistent device names vs. the /dev/sd? device names.

This information came from this external web site, and has not been rigorously verified by EqualLogic:

http://www.3ware.com/kb/article.aspx?id=11050

To check the existing read ahead setting use:

#**`blockdev –getra <device name>`**

- CHANGING LINUX I/O SCHEDULER

Linux offers different kernel I/O schedulers. In Redhat the default is “CFQ” (Completely Fair Queuing) However, the Open-iSCSI group reports that sometimes using the “NOOP” scheduler works better in iSCSI server environments.

This website provides information on selecting different schedulers.

http://www.redhat.com/magazine/008jun05/features/schedulers/

Here's a small excerpt:

The Linux kernel, the core of the operating system, is responsible for controlling disk access by using kernel I/O scheduling. The I/O schedulers provided in Red Hat Enterprise Linux 4, embedded in the 2.6 kernel, have advanced the I/O capabilities of Linux significantly. With Red Hat Enterprise Linux 4, applications can now optimize the kernel I/O at boot time, by selecting one of four different I/O schedulers to accommodate different I/O usage patterns:

*** Completely Fair Queuing —elevator=cfq (default)**

*** Deadline —elevator=deadline** ** this is intended for Real Time applications **

*** NOOP —elevator=noop**

*** Anticipatory —elevator=as** **this is intended for desktop environments**

You can change the scheduler on the fly to see whether the NOOP scheduler is better for your environment. This setting is not persistent.

```
#echo noop > /sys/block/${DEVICE}/queue/scheduler.
```

Change ‘noop’ to ‘cfq’ to return it to the default setting.

KNOWN ISSUES:

- LINUX ARP BEHAVIOR:

If multiple interfaces are on the same subnet, Linux's ARP behavior will result in a reset (RST) being sent to the array from the client. The following changes need to be made to `/etc/sysctl.conf` to work around this behavior:

```
net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_announce=2
```

- LINUX NETFILTER:

Per: https://bugzilla.redhat.com/show_bug.cgi?id=493226, it appears that netfilter will mark packets as being invalid under heavy load.

To work around this bug, the following needs to be added to

`/etc/sysctl.conf:`

```
net.ipv4.netfilter.ip_conntrack_tcp_be_liberal=1
```