**What is iSCSI, and why use it**

iSCSI is a network protocol that allows you to use storage devices (like hard drives or storage servers) located far away as if they were plugged directly into your computer. It sends storage commands over your network instead of using direct cables.

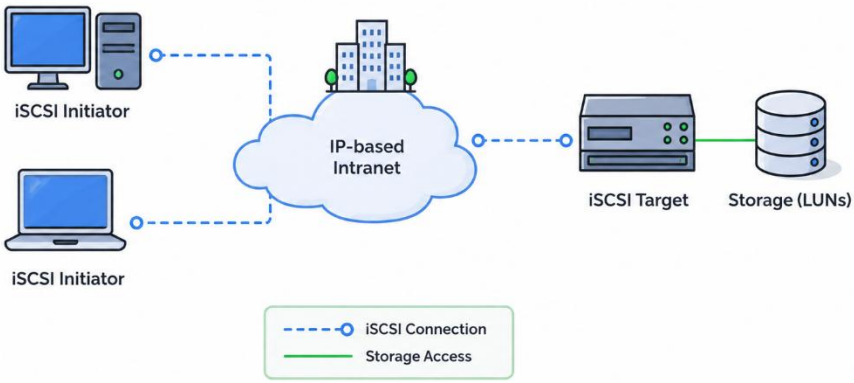
Key Points:

- Transfers data over the internet/network using TCP/IP protocol
- Makes remote storage appear as local storage to your operating system
- Works by encapsulating SCSI commands into network packets for transmission
- Allows multiple servers and computers to share centralized storage

iSCSI solves the problem of accessing storage from far away by making it invisible to your computer. Instead of worrying about network connections, your applications simply see storage as a local device, just like any hard drive you could touch.

**How does iSCSI Work?**

iSCSI works by taking storage commands such as reading and writing data and sending them over a standard TCP/IP network. The server, called the iSCSI Initiator, sends these commands to a remote storage device, called the iSCSI Target. The target processes the requests and returns the data. To the operating system and applications, the remote storage appears exactly like a locally attached SATA, SAS, or NVMe disk, even though it is located elsewhere on the network.




**iSCSI Components: Initiator and Target**

**What is an iSCSI Initiator?**
The iSCSI initiator is the part on your computer that sends storage requests. It's like the person asking for something.

Key Points:

- Sits on the client computer (your machine)
- Can be software (program) or hardware (dedicated chip)
- Software initiator uses your computer's CPU power
- Hardware initiator (HBA) is a special card that does the work
- Sends commands to the storage and receives answers

**What is an iSCSI Target?**

The iSCSI target is the part on the storage server that listens and responds to requests. It's like the person being asked the question.

Key Points:

- Runs on the storage server (far away)
- Listens for incoming storage requests from initiators
- Shares storage resources (called LUNs) with many computers at once
- Reads and writes data based on commands it receives
- Sends data back to whoever asked for it

**Core components and terminology**

Before touching a terminal, learn this vocabulary , every command you'll type refers to one of these concepts.

| Term | What it means |
|---|---|
| Target | The server that owns and exports the storage (the 'disk provider'). Runs the target daemon (LIO / targetcli on Linux). |
| Initiator | The client that consumes the storage exported by a target. Runs open-iscsi on Linux. |
| IQN | iSCSI Qualified Name — a globally unique name identifying an initiator or target, e.g. iqn.2026-08.com.example:server1. |
| Portal | The IP address + TCP port (default 3260) on the target where it listens for initiator connections. |
| Backstore | The actual storage LIO exposes — can be a raw block device, an LVM volume, a file, or a RAM disk. |
| LUN | Logical Unit Number — the numbered storage unit (backed by a backstore) presented to an initiator inside a target. |
| ACL | Access Control List entry on the target — maps a specific initiator IQN to permission to log in and see LUNs. |

**Lab setup used in this guide**

This walkthrough uses two Linux machines.

| Role | Example hostname | Example IP | OS assumed |
|---|---|---|---|
| Target (storage server) | localhost | 192.168.1.64 | RHEL (targetcli) |
| Initiator (client) | client | 192.168.1.2 | RHEL (iscsiadm) |

## Step-by-step: building the iSCSI target

### Install the target service

On RHEL/CentOS/Rocky:
**# dnf or yum install -y targetcli**
**# systemctl enable --now target**

```
root@localhost:~# yum install -y targetcli
Last metadata expiration check: 0:12:46 ago on Tue 04 Aug 2026 08:06:37 AM +0545.
Dependencies resolved.
================================================================================
 Package                    Architecture   Version                Repository      Size
================================================================================
Installing:
 targetcli                  noarch         2.1.58-5.el10          AppStream       90 k
Installing dependencies:
 python3-configshell        noarch         1:1.1.30-9.el10        BaseOS          87 k
 python3-kmod               x86_64         0.9.2-6.el10           BaseOS         114 k
 python3-pyparsing          noarch         3.1.1-7.el10           BaseOS         273 k
 python3-rtslib             noarch         2.1.76-12.el10         AppStream      128 k
 python3-typing-extensions  noarch         4.9.0-6.el10           BaseOS          81 k
 python3-urwid              x86_64         2.5.3-4.el10           BaseOS         1.1 M
 python3-wcwidth            noarch         0.2.6-6.el10           BaseOS          50 k
 target-restore             noarch         2.1.76-12.el10         AppStream       18 k
```

```
root@localhost:~# systemctl enable target --now
root@localhost:~# systemctl status target
● target.service - Restore LIO kernel target configuration
     Loaded: loaded (/usr/lib/systemd/system/target.service; enabled; preset: disabled)
     Active: active (exited) since Tue 2026-08-04 08:20:51 +0545; 3s ago
 Invocation: 39013fedfe2447b5aa949a074b0cbb6b
    Process: 5427 ExecStart=/usr/bin/targetctl restore (code=exited, status=0/SUCCESS)
   Main PID: 5427 (code=exited, status=0/SUCCESS)
   Mem peak: 10.3M
        CPU: 239ms
```

### Open the targetcli shell

targetcli is an interactive shell for configuring LIO (Linux-IO), the in-kernel iSCSI target subsystem. Launch it as root:
*(tab twice for auto-completion and to see available options more easily.)*

### # targetcli

When you run the targetcli command, it opens the iSCSI target management shell. Running the **ls** command inside targetcli shows the current configuration tree, including available backstores and iSCSI targets.

```
root@localhost:~# targetcli
targetcli shell version 2.1.58
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/> ll
Command not found ll
/> ls
o- / ..................................................................... [...]
  o- backstores ......................................................... [...]
  | o- block ..................................................... [Storage Objects: 0]
  | o- fileio .................................................... [Storage Objects: 0]
  | o- pscsi ..................................................... [Storage Objects: 0]
  | o- ramdisk ................................................... [Storage Objects: 0]
  o- iscsi ....................................................... [Targets: 0]
  o- loopback .................................................... [Targets: 0]
```

**Create a backstore**
A backstore is the actual storage that will be exported.
**/>cd backstores**
**/> backstores/create name=angtsering  dev=/dev/nvme0n2**

*Note: - dev=/dev/nvme0n2 specifies which disk you want to share through iSCSI. The device /dev/nvme0n2 is the storage disk that will be used as the backstore and made available to iSCSI clients.*

```
/backstores/block> create name=angtsering dev=/dev/nvme0n2
Created block storage object angtsering using /dev/nvme0n2.
/backstores/block> ls
o- block ..................................................................... [Storage Objects: 1]
  o- angtsering ................................ [/dev/nvme0n2 (10.0GiB) write-thru deactivated]
    o- alua ...................................................................... [ALUA Groups: 1]
      o- default_tg_pt_gp ........................................ [ALUA state: Active/optimized]
/backstores/block>
```

When you create a backstore, it may show deactivated because it is not yet attached to an iSCSI target (LUN).

**Create the iSCSI target and IQN**

Now create the iSCSI target. The target is the storage endpoint that clients will connect to. When you run the command below, a new target with the specified IQN is created.

**/>cd iscsi**
**/> iscsi/ create iqn.2026-07.com.ang:client1**

```
/> cd iscsi
/iscsi>
/          @last      bookmarks  cd         create     delete     exit       get        help
info       ls         pwd        refresh    set        status     version
/iscsi> create iqn.2026-07.com.ang:client1
Created target iqn.2026-07.com.ang:client1.
Created TPG 1.
Global pref auto_add_default_portal=true
Created default portal listening on all IPs (0.0.0.0), port 3260.
```

This command creates a new iSCSI target with the IQN iqn.2026-07.com.ang:client1. It also automatically creates **TPG 1 (Target Portal Group 1)** and opens port 3260 to listen for iSCSI client connections.

```
  o- iscsi ..................................................................... [Targets: 1]
  | o- iqn.2026-07.com.ang:client1 ..................................................... [TPGs: 1]
  |   o- tpg1 ....................................................... [no-gen-acls, no-auth]
  |     o- acls ..................................................................... [ACLs: 0]
  |     o- luns ..................................................................... [LUNs: 0]
  |     o- portals ................................................................ [Portals: 1]
  |       o- 0.0.0.0:3260 ..................................................................... [OK]
  o- loopback .................................................................. [Targets: 0]
```

Above output shows that the iSCSI target **iqn.2026-07.com.ang:client1** has been created successfully. Inside the target, **TPG1 (Target Portal Group 1)** was automatically created, along with folders for **ACLs**, **LUNs**, and **Portals**.

### Create the LUN

After creating the iSCSI target, you move inside iscsi using the cd command. Running ls shows the target structure and the components that will be used to control access, attach storage, and allow network connections.

```
/> cd iscsi/
/iscsi> cd iqn.2026-07.com.ang:client1/
/iscsi/iqn.20...m.ang:client1> ls
o- iqn.2026-07.com.ang:client1 ..................................................... [TPGs: 1]
  o- tpg1 ........................................................... [no-gen-acls, no-auth]
    o- acls ................................................................................ [ACLs: 0]
    o- luns ................................................................................ [LUNs: 0]
    o- portals .......................................................................... [Portals: 1]
      o- 0.0.0.0:3260 ..................................................................... [OK]
/iscsi/iqn.20...m.ang:client1>
```

Now create a LUN (Logical Unit Number) and attach the backstore to the iSCSI target. A LUN is the storage device that iSCSI clients will see and use.

```
/iscsi/iqn.20...:client1/tpg1> cd luns
/iscsi/iqn.20...nt1/tpg1/luns> create
/backstores/block/angtsering  Desktop/                    Documents/
Downloads/                    Music/                      Pictures/
Public/                       Templates/                  Videos/
anaconda-ks.cfg               angt/                       add_mapped_luns=
lun=                          storage_object=
/iscsi/iqn.20...nt1/tpg1/luns> create /b
/backstores/block/angtsering  /bin/                       /boot/
/iscsi/iqn.20...nt1/tpg1/luns> create /backstores/block/angtsering
Created LUN 0.
/iscsi/iqn.20...nt1/tpg1/luns> ls
o- luns ................................................................................ [LUNs: 1]
  o- lun0 ..................................... [block/angtsering (/dev/nvme0n2) (default_tg_pt_gp)]
```

After creating the **LUN**, you create an **ACL (Access Control List)** to allow a specific iSCSI initiator (client) to connect to the target. Without an ACL, the client may not be able to access the shared storage.

```
/iscsi/iqn.20...m.ang:client1> cd tpg1/
/iscsi/iqn.20...:client1/tpg1> cd acls
/iscsi/iqn.20...nt1/tpg1/acls> ls
o- acls ................................................................................ [ACLs: 0]
/iscsi/iqn.20...nt1/tpg1/acls> create iqn.2026-07.con.ang:tsering
Created Node ACL for iqn.2026-07.con.ang:tsering
Created mapped LUN 0.
/iscsi/iqn.20...nt1/tpg1/acls>
```

After creating the LUN and ACL, create a Portal. A portal defines the server's IP address and port that iSCSI clients will use to connect to the iSCSI target. Port 3260 is the default port used by iSCSI for communication between the client (initiator) and the server (target).

```
/iscsi/iqn.20.../tpg1/portals> create ip_address=192.168.1.64 ip_port=3260
Using default IP port 3260
Created network portal 192.168.1.64:3260.
/iscsi/iqn.20.../tpg1/portals> ls
o- portals .............................................................................. [Portals: 1]
  o- 192.168.1.64:3260 ................................................................... [OK]
/iscsi/iqn.20.../tpg1/portals> cd ..
/iscsi/iqn.20...:client1/tpg1> cd ..
/iscsi/iqn.20...m.ang:client1> cd ..
/iscsi> cd ..
/> exit
```

### Open the firewall port

### #firewall-cmd --permanent --add-port=3260/tcp
### #firewall-cmd --reload

```
root@localhost:~# firewall-cmd --permanent --add-port=3260/tcp
success
root@localhost:~# firewall-cmd --reload
success
```

**Checkpoint**
At this point, run 'sudo targetcli ls' to review the full tree:
**backstore → target IQN → TPG → LUN, portal, and ACL** should all be visible and nested correctly before moving to the initiator.

## Step-by-step: building the iSCSI initiator

### Install iscis-initiator-utils

On RHEL/CentOS/Rocky:

### # yum whatprovides iscsi*
### # yum -y install iscsi-initiator-utils

```
root@client:~# yum whatprovides iscsi*
Updating Subscription Management repositories.
Last metadata expiration check: 8:20:08 ago on Tue 04 Aug 2026 09:44:27 AM +0545.
iscsi-initiator-utils-6.2.1.11-0.git4b3e853.el10.x86_64 : iSCSI daemon and utility programs
Repo        : @System
Matched from:
Provide    : iscsi-initiator-utils = 6.2.1.11-0.git4b3e853.el10
Provide    : iscsi-initiator-utils(x86-64) = 6.2.1.11-0.git4b3e853.el10

iscsi-initiator-utils-6.2.1.11-0.git4b3e853.el10.x86_64 : iSCSI daemon and utility programs
Repo        : BaseOS
Matched from:
Provide    : iscsi-initiator-utils = 6.2.1.11-0.git4b3e853.el10
Provide    : iscsi-initiator-utils(x86-64) = 6.2.1.11-0.git4b3e853.el10
```

### Start and enable the iscsid service

```
root@client:~# systemctl enable iscsid --now
root@client:~# systemctl status iscsid
● iscsid.service - Open-iSCSI
     Loaded: loaded (/usr/lib/systemd/system/iscsid.service; enabled; preset: disabled)
     Active: active (running) since Tue 2026-08-04 17:43:02 +0545; 22min ago
 Invocation: 534c8011f10143cdb0b654b57ba6de41
TriggeredBy: ● iscsid.socket
       Docs: man:iscsid(8)
             man:iscsiuio(8)
             man:iscsiadm(8)
   Main PID: 1423 (iscsid)
     Status: "Ready to process requests"
      Tasks: 1 (limit: 10114)
     Memory: 4.4M (peak: 5.1M)
        CPU: 75ms
     CGroup: /system.slice/iscsid.service
             └─1423 /usr/sbin/iscsid -f
```

**Discover the target**
Send a SendTargets discovery request to the target's portal:

**# iscsiadm ---mode discovery --type st -p 192.168.1.64 3260**

```
root@client:~# iscsiadm --mode discovery --type st --portal 192.168.1.64 3260
192.168.1.64:3260,1 iqn.2026-07.com.ang:client1
root@client:~#
```

**Log in to the target**

After configuring the iSCSI target, use the iscsiadm command on the client to log in to the iSCSI server. If the login fails, check the initiator configuration file **etc/iscsi/initiatorname.iscsi** and ensure the client IQN matches the ACL configured on the target.
If the client IQN in /etc/iscsi/initiatorname.iscsi does not match the IQN configured in the target's ACL, the iSCSI login will fail. The server will reject the connection because the client is not authorized.
**# iscsiadm -m node -T iqn.2026-07.com.ang:client -p 192.168.1.64:3260 --login**

```
root@client:~# iscsiadm --mode node -T iqn.2026-07.com.ang:client1 --portal 192.168.1.64 3260 --login
iscsiadm: Could not login to [iface: default, target: iqn.2026-07.com.ang:client1, portal: 192.168.1.64,3260].
iscsiadm: initiator reported error (24 - iSCSI login failed due to authorization failure)
iscsiadm: Could not log into all portals
root@client:~#
```

**Set the initiator's IQN**
Every initiator needs a unique IQN. It lives in /etc/iscsi/initiatorname.iscsi — edit it to match what you registered in the target's ACL:

**# vim /etc/iscsi/initiatorname.iscsi**

```
root@client:~# cd /etc/iscsi/
root@client:/etc/iscsi# ll
total 20
-rw-r--r--. 1 root root    50 Aug  4 09:43 initiatorname.iscsi
-rw-------. 1 root root 15501 May  7  2025 iscsid.conf

root@client:/etc/iscsi# cat initiatorname.iscsi
InitiatorName=iqn.2026-07.con.ang:tsering
root@client:/etc/iscsi#
```

After correcting the initiator IQN and restarting the iscsid service, the client successfully connected to the iSCSI target. The message confirms that the login to the target was successful.

```
root@client:~# systemctl restart iscsid
root@client:~# iscsiadm --mode node -T iqn.2026-07.com.ang:client1 --portal 192.168.1.64 3260 --login
Login to [iface: default, target: iqn.2026-07.com.ang:client1, portal: 192.168.1.64,3260] successful.
root@client:~#
```

## Verifying and Confirm the new block device

## # lsblk

```
root@client:~# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda             8:0    0   10G  0 disk
sr0            11:0    1 10.3G  0 rom  /mnt/ang
nvme0n1       259:0    0   20G  0 disk
├─nvme0n1p1   259:1    0  600M  0 part /boot/efi
├─nvme0n1p2   259:2    0    2G  0 part /boot
└─nvme0n1p3   259:3    0 17.4G  0 part
  ├─rhel-root 253:0    0 15.4G  0 lvm  /
  └─rhel-swap 253:1    0    2G  0 lvm  [SWAP]
```

The new disk typically appears as the next available /dev/sda. Confirm which device with lsblk before formatting — never format the wrong disk.

## Partition, format, and mount

After successfully connecting to the iSCSI target, the shared storage appears as a local disk (for example, /dev/sda). Use fdisk to create a partition on the disk before creating a filesystem and mounting it.

## # fdisk /dev/sda

- m  # Display help menu
- n  # Create new partition
- p  # Print partition table
- w  # Save changes and exit

```
root@client:~# fdisk /dev/sda

Welcome to fdisk (util-linux 2.40.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS (MBR) disklabel with disk identifier 0xb3b4e7b4.

Command (m for help):


Command (m for help): p
Disk /dev/sda: 10 GiB, 10737418240 bytes, 20971520 sectors
Disk model: angtsering
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 1048576 bytes
Disklabel type: dos
Disk identifier: 0xb3b4e7b4

Device     Boot Start      End  Sectors Size Id Type
/dev/sda1        2048 10487807 10485760   5G 83 Linux
root@client:~# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda             8:0    0   10G  0 disk
└─sda1          8:1    0    5G  0 part
sr0            11:0    1 10.3G  0 rom  /mnt/ang
```

*The output confirms that the partition sda1 has been created successfully on the iSCSI disk /dev/sda. The disk is 10 GB in size, and the new partition sda1 uses 5 GB of that space.*

After creating the partition /dev/sda1, you need to format it with a filesystem.Here I will use the XFS filesystem. Formatting prepares the partition so Linux can store files on it.

```
root@client:~# partprobe /dev/sda
root@client:~# mkfs.xfs /dev/sda
sda   sda1
root@client:~# mkfs.xfs /dev/sda1
meta-data=/dev/sda1              isize=512    agcount=4, agsize=327680 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=1
         =                       reflink=1    bigtime=1 inobtcount=1 nrext64=1
         =                       exchange=0   metadir=0
data     =                       bsize=4096   blocks=1310720, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1, parent=0
log      =internal log           bsize=4096   blocks=16384, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
         =                       rgcount=0    rgsize=0 extents
         =                       zoned=0      start=0 reserved=0
```

The command mkfs.xfs /dev/sda1 successfully formatted the partition /dev/sda1 with the XFS filesystem.

```
root@client:~# lsblk -ft
NAME FSTYPE FSVER LABEL                  UUID                                   FSAVAIL FSUSE% MOUNTPOINTS ALIGNMENT MIN
-IO  OPT-IO PHY-SEC LOG-SEC ROTA SCHED       RQ-SIZE   RA WSAME
sda                                                                                                                0
512 1048576     512     512    0 mq-deadline     226 2048    0B
└─sda1
     xfs                                 63849f64-3eeb-41a9-b696-ace995f404be                                      0
```

After formatting the partition /dev/sda1 with the XFS filesystem, create a mount point and mount the partition. This makes the storage accessible through the directory.

```
root@client:~# mkdir /file
root@client:~# mount /dev/sda1 /file
root@client:~# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda             8:0    0   10G  0 disk
└─sda1          8:1    0    5G  0 part /file
sr0            11:0    1 10.3G  0 rom  /mnt/ang
nvme0n1       259:0    0   20G  0 disk
├─nvme0n1p1   259:1    0  600M  0 part /boot/efi
├─nvme0n1p2   259:2    0    2G  0 part /boot
└─nvme0n1p3   259:3    0 17.4G  0 part
  ├─rhel-root 253:0    0 15.4G  0 lvm  /
  └─rhel-swap 253:1    0    2G  0 lvm  [SWAP]
```

**Persist the mount in /etc/fstab**

Always use the _netdev option so the system waits for the network and iSCSI session before mounting at boot:

```
# /etc/fstab
# Created by anaconda on Fri Jun 26 12:30:12 2026
#
# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
#
UUID=7ab1d3f0-3e82-4632-a59d-fb1fa32c6ef0 /                       xfs     defaults        0 0
UUID=2f5d3709-f68f-4d9d-89aa-2b824be6dd22 /boot                   xfs     defaults        0 0
UUID=0D19-8B98          /boot/efi               vfat    umask=0077,shortname=winnt 0 2
UUID=8576c4a9-2835-4fd5-afc7-1b32d0ddadf0 none                    swap    defaults        0 0
/dev/sr0 /mnt/ang iso9660 defaults 0 0

UUID="63849f64-3eeb-41a9-b696-ace995f404be" /file xfs _netdev 0 0
~
~
~
```

After making the mount persistent (usually by adding an entry to /etc/fstab), use the findmnt -x command to check for configuration errors. If no issues are found, Linux displays a success message.

```
root@client:~# findmnt -x
Success, no errors or warnings detected


root@client:~# df -h
Filesystem             Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root   16G  6.1G  9.3G  40% /
devtmpfs               806M     0  806M   0% /dev
tmpfs                  837M   84K  837M   1% /dev/shm
efivarfs               256K   56K  196K  23% /sys/firmware/efi/efivars
tmpfs                  335M  6.9M  328M   3% /run
tmpfs                  1.0M     0  1.0M   0% /run/credentials/systemd-journald.service
/dev/nvme0n1p2         2.0G  436M  1.6G  22% /boot
/dev/nvme0n1p1         599M  9.0M  590M   2% /boot/efi
/dev/sr0                11G   11G     0 100% /mnt/ang
tmpfs                  168M  116K  168M   1% /run/user/0
/dev/sda1              5.0G  130M  4.9G   3% /file
```

## Summary

iSCSI turns a regular Ethernet network into shared block storage. A target exports storage, an initiator connects to it over TCP port 3260, and the storage appears as a local disk. For production environments, security and reliability should be improved using CHAP authentication, network isolation, and multipathing.