

**XINNOR**

**Xinnor xiRAID Opus User Manual**

# Contents

<b>Xinnor xiRAID Opus 1.3.0 User Guide.....</b>	<b>5</b>
Document Revision History.....	5
Introduction.....	5
About xiRAID Opus.....	5
Intended Audience.....	6
Acronyms and Definitions.....	7
xiRAID Opus Components.....	9
xiRAID Opus Structure.....	10
Backend devices.....	10
xiRAID Opus CLI.....	11
Installing xiRAID Opus.....	12
System Requirements.....	12
Required Libraries.....	14
Installation Procedure.....	15
System Configuration Best Practices.....	24
Testing the Installation.....	25
Troubleshooting.....	26
Uninstalling xiRAID Opus.....	28
Updating xiRAID Opus.....	29
License Management.....	30
License Installation.....	32
Show License State.....	33
Merging Licenses.....	34
Delete License.....	35
Device Manager (DM).....	35
Discover Devices.....	37
Device State Machine.....	38
Device BDEVs.....	41
Managing Local Devices.....	45

Managing Network Devices.....	48
Working with RAIDs.....	56
Supported RAID levels.....	56
Managing RAIDs.....	57
Operational Modes.....	71
Managing Partitions.....	73
Creating Partitions.....	74
Viewing Partitions.....	74
Deleting Partitions.....	75
Configuring Vhost Targets.....	75
Creating a Vhost Target.....	76
Viewing Vhost Targets.....	76
Deleting a Vhost Target.....	78
Working with NVMe over Fabrics Targets.....	78
Configuring an NVMe-oF Target.....	78
Configuring Asymmetric Namespace Access (ANA).....	90
Configuring NVMe-oF Host Access.....	92
xiRAID Opus Configuration Options.....	93
Memory Configuration.....	93
CPU Mask.....	94
Logging Management.....	95
Advanced Configuration Options.....	96
Command & gRPC Reference.....	98
bdev.....	99
network.....	111
nvmf.....	115
raid.....	145
config.....	160
device-manager (dm).....	163
license.....	178
this.....	181

vhost.....	183
events.....	186

# Xinnor xiRAID Opus 1.3.0 User Guide

Instructions, practical guides, and tips for administering Xinnor xiRAID Opus 1.3.0

## Document Revision History

<b>xiRAID Opus Ver- sion</b>	<b>Documentation Up- date Date</b>	<b>Description of Changes</b>
1.0.0	11-APR-2024	Initial release
1.1.0	18-SEP-2024	Updated document for compatibility with xiRAID Opus 1.1.0.
1.2.0	08-SEP-2025	Updated document for compatibility with xiRAID Opus 1.2.0.
1.3.0	16-FEB-2025	Updated document for compatibility with xiRAID Opus 1.3.0.

## Introduction

### About xiRAID Opus

xiRAID OPUS: NVMe Composer with high-speed Data Protection.

xiRAID OPUS is a Linux user-space software solution that unifies local and network-attached NVMe drives into a high-performance, energy-efficient storage platform. It maximizes speed and reliability for demanding applications while minimizing hardware overhead, lowering power costs, and simplifying infrastructure management.

xiRAID OPUS extends data protection functionality with:

- Integrated network storage for seamless scaling
- Native NVMe-oF initiator and target support across drives, volumes, and RAID arrays
- Built-in VHOST virtualization
- End-to-end QoS controls for predictable performance in shared environments

At its foundation, xiRAID OPUS employs a Linux user-space datapath engine, bypassing the kernel I/O stack with polling-mode to cut latency and eliminate OS dependencies. This not only improves performance but also ensures frictionless Linux distribution updates—no kernel tuning or compatibility headaches required.

Supporting both x86 and ARM64 platforms, including advanced DPUs like NVIDIA BlueField, xiRAID OPUS can offload storage management from the host CPU, freeing valuable compute cycles for applications.

Its polling-mode architecture delivers unbeatable performance:

- 1M IOPS and 100 GB/s per CPU core with ultra-low latency
- Linear scaling across cores and nodes
- Consistently high performance in both normal and degraded/rebuild modes

This unique design unlocks the full potential of today's PCIe 5.0 NVMe drives and provides a future-ready path to PCIe 6.0 and 7.0, where traditional CPU locks and kernel overhead would otherwise become crippling bottlenecks.

With its gRPC API, xiRAID OPUS integrates seamlessly into automated, modular environments—whether in private and public clouds, HPC systems, or AI infrastructures.

## Intended Audience

This guide is intended for administrators and users of block storage devices based on the Xinnor xiRAID Opus software. It provides details on xiRAID Opus operation and instructions on how to configure and manage objects in xiRAID Opus storage.

## Acronyms and Definitions

API	Application programming interface.
BDEV	Any logical block storage item managed by the xiRAID block layer. This includes local physical devices, network devices, and xiRAID Opus RAIDs. BDEVs can be stacked and form multi-level topology.
Chunk	The unit in which data is spread over RAID drives. One chunk of continuous data is written to a certain drive, the next chunk is written to another drive.
CLI	Command Line Interface xiRAID Opus utility for xiRAID engine management.
Device, physical device	A device directly attached via PCI or network, either physically or virtually through a hypervisor. The device can represent a single drive or multiple drives storing user and RAID control data. An example of such a device is a PCIe-connected NVMe.
DMA	A device directly attached via PCI or network, either physically or virtually through a hypervisor. The device can represent a single drive or multiple drives storing user data. An example of such a device is a PCIe-connected NVMe drive.
Drive	A storage unit that can be used to store or access data.
Driver	A driver is a software component that facilitates communication between an operating system or application and a specific hardware device, such as a drive. It translates high-level instructions from software into the low-level commands needed for the hardware to function properly, ensuring seamless interaction between the two.
gRPC	Google Remote Procedure Calls protocol is a framework based on the proto3 specification.
IO	Input/output refers to the process of transferring data to and from storage devices, such as drives or network targets.

**(continued)**

IOMMU group	An IOMMU group is the smallest set of devices that can be considered isolated from the IOMMU's perspective. Devices from the same IOMMU group should not be shared between different users and host drivers.
InfiniBand	A computer networking communications standard used for data interconnection.
Metadata	Persisted RAID control parameters located at the RAID backing drives. the metadata is not available for client application and is used for the RAID operation and data recovery.
NIC	The Network Interface Controller enables computers to communicate over a computer network.
Namespace	A quantity of non-volatile memory that can be formatted into logical blocks. This definition is virtually identical to the SCSI concept of a logical unit. A logical block device that is part of physical NVMe device.
OS	Operating System.
RAID	RAID (Redundant Array of Independent Disks) is a data storage technology that combines multiple physical disk drives into one or more logical units to improve performance, data redundancy, or both.
Server	The machine where xiRAID software is running.
SSD	Solid State Drive.
Subsystem	The NVM subsystem is a collection of physical fabric interfaces (ports), one or more controllers, one or more namespaces and an interface between the controller(s) and non-volatile memory storage medium.
Target	The endpoint that waits for initiators' commands and provides required input/output data transfers. A part of the xiRAID software.
UUID	Globally unique 128-bit identifier. Unique identifier of a disk, logical drive, RAID or a configuration component.

**(continued)**

VirtIO transport	An abstraction layer in the hypervisor to expose a software block devices to the guest as if they are physical devices using a specific transport method
VM socket	The character device used to connect a block storage by using VirtIO protocol.

## xiRAID Opus Components

The storage system consists of the following components:

- **Physical storage devices (SSD)** inserted into a server or connected via network.
- **Backend drive drivers** that facilitate the flow of IO data between drives and the data path engine. xiRAID uses OS drivers for communication with a device at PCIe or network controller level. It uses user-space NVMe or network NVMe-RDMA drivers to handle the main drive communication IO logic.
- **Device Manager** component responsible for connecting, disconnecting, and monitoring physical storage devices and their drivers within the data path engine and OS.
- **BDEV subsystem** that represents unified logical drive API for different drive types such as a backend drive, RAID, or another device within a layered block device topology.
- **RAID engine** based on the fast Xinnor xiRAID technology. The RAID engine implements various RAID features such as RAID structure initialization, general IO operations, degraded mode IO, drive replacement, and drive data

reconstruction (including Xinnor's partial reconstruction algorithms). Future releases will also support hot spare drives.

- **Frontend target subsystem** that connects xiRAID devices or other BDEVs to a client IO engine. The current release includes support for VirtIO targets, as well as NVMe-TCP and NVMe-RDMA target types.
- **Configuration database** that stores and tracks the configuration of the storage system and its components. It allows the system to automatically recover and restart if there's a failure in the engine, operating system, or server.
- **Management logic** for configuring and monitoring all storage components.

## xiRAID Opus Structure

The xiRAID storage engine is implemented as a single binary that runs as a service. This engine exposes a gRPC management API. An additional binary implements a CLI to interact with the gRPC API as a management command-line interface. The CLI utility is a thin client that does not contain any logic. Therefore, another client (such as a GUI, customer CLI, or plugin) can connect directly to the gRPC API. Any engine command available in the CLI tool can be executed by a direct gRPC call.

The CLI uses the network layer to send commands to and receive responses from the data path engine via gRPC. This means that the CLI and xiRAID engine can operate on different network nodes, and the same CLI can manage multiple engines across various network nodes. To specify the destination engine for a command, the CLI tool needs to be provided with the network address (hostname/IP and port number). By default, if the engine and CLI are installed on the same node, the localhost and pre-configured port number are used.

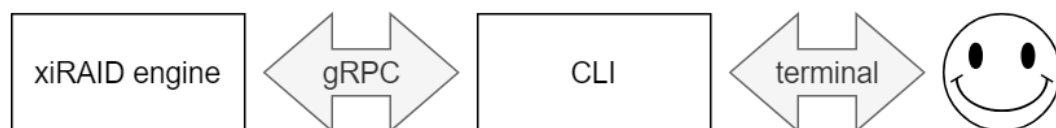


Figure 1: xiRAID Opus Components

## Backend devices

The xiRAID engine is a high-performance software RAID system, leveraging multiple components to optimize performance. One key component is the internal user-space high-performance hardware drivers, facilitating efficient IO operations with local

or network-connected devices. By bypassing the OS drivers, Opus works directly with the hardware storage devices. To use a device with the storage engine, it must be detached from the operating system and attached to the xiRAID engine. Once attached, the device is managed exclusively by Opus and is no longer accessible in the host OS. Upon restart, Opus automatically reattaches previously connected storage devices.

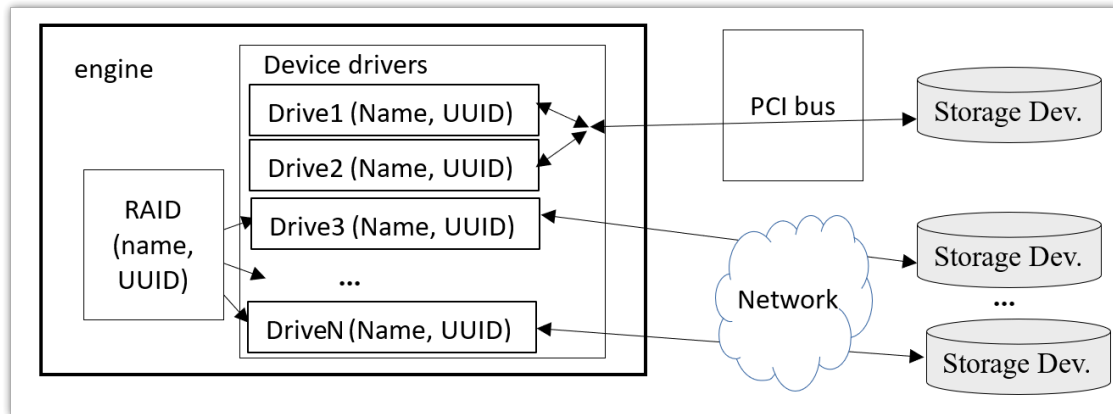


Figure 2: Relationship Between RAID, Drives, and Storage Devices

## xiRAID Opus CLI

This document uses CLI command examples to explain storage management operations, the overall approach, and common use cases. It does not provide detailed descriptions of each command and its options. For a comprehensive description of all commands and options, please refer to the xiRAID Opus Command Reference.

You can obtain a brief command description by using the self-documented CLI with the `-h` flag.

```
xnr_cli --help
```

```
xnr_cli <command> -h
```

The CLI implements shortcuts for commands and command options. For example, `-h` is a shortcut for `--help`.

Some commands and options are hidden. The `--honest` (or `-H`) flag enables help for hidden commands and options. They are experimental and not designed for production usage. The hidden commands can be used for tests or prototyping. The hidden commands syntax may change in future releases or commands may be removed from the CLI.

```
xnr_cli --honest --help
```

```
xnr_cli -H <command> -h
```

If a CLI command option consumes a list of items the list can be input as comma separated values or as repeated options. For example, these two commands are identical:

```
xnr_cli bdev zero --names  
43E0A004TLZJn1,43F0A00ATLZJn1,43T0A002TLZJn1
```

```
xnr_cli bdev zero --names 43E0A004TLZJn1 --names 43F0A00ATLZJn1  
--names 43T0A002TLZJn1
```

## Installing xiRAID Opus

xiRAID Opus is distributed as a package containing the installation script (`install.sh`) and the archive with files required for installation (`xiraid.tgz`). You can check the full list of installed components by referring to the `filelist.txt` in the installation directory (by default, `/opt/xiraid/filelist.txt`).

Installation commands presented in this section are run only with superuser privileges. Please log in as an administrator or root to run the installation commands.

## System Requirements

### Hardware Requirements

The following CPUs are supported by xiRAID Opus:

- 64-bit Intel processors
- 64-bit AMD processors with AVX2 support

16 CPU cores are recommended for xiRAID Opus.

It is recommended to allocate at least 32 GiB of memory to xiRAID Opus.

xiRAID Opus only supports drives that have Power Loss Protection (PLP). Using drives without PLP may result in data corruption or loss if power is unexpectedly interrupted. It is important to note that many consumer SSDs do not include this feature. Below is a list of popular consumer SSDs that do not have PLP:

- Intel SSD 545s Series
- WD Green SSD
- Crucial P2 NVMe SSD
- WD SN700 NVMe SSD
- Intel 670p NVMe SSD
- Samsung 980 NVMe SSD
- Kingston SATA SSDs

## Supported Operating System Distributions

Since the product is delivered as pre-compiled Linux OS binaries, it can run on any x64 Linux OS. Supported operating systems and platform architectures are divided into the following tiers based on the amount of testing performed at Xinnor:

- **Tier 1:** Functional and data integrity testing cycles were performed for these distributions and platform architectures.
- **Tier 2:** Smoke tests were performed for the specified distributions and platform architectures. Functional and integrity testing can be run on request before the solution is installed in the production environment. Support for xiRAID Opus can be provided only upon successful completion of testing.
- **Tier 3:** No testing or build has been done, xiRAID Opus may or may not work on these distributions and platform architectures. Support may be available on a case-by-case basis upon request.

Architecture	Distribution	Tier
x86_64	Ubuntu 24.04 LTS RHEL, Rocky Linux, Alma Linux 9.5, 9.6, 10, 10.1	1
x86_64	Ubuntu 22.04 LTS	2

RHEL, Rocky Linux, Alma Linux 9.3 5.14.0-362

Proxmox 8.3 pve-6.8

Proxmox 8.2 pve-6.8

---

x86_64	Any Linux	3
ARM	Linux	3

---

## Secure Boot

To ensure that xiRAID Opus operates correctly, Secure Boot must be disabled.

## Required Libraries

xiRAID Opus requires the following packages to be pre-installed on your system, with the specified versions or higher:

- GLIBC\_2.34 (libstdc++.so.6, libc.so.6)
- openssl3 (libssl.so.3)
- crypto (libcrypto.so.3)
- libnl-3 (libnl-route-3.so, libnl-3.so)
- libgcc\_s.so.1
- linux-vdso.so.1
- libnuma.so.1
- libibverbs.so.1
- librdmacm.so.1
- libuuid.so.1
- libm.so.6

To install the required libraries on Ubuntu and Proxmox, run the following command:

```
sudo apt install -y \  
    bzip2 \  
    libstdc++6 \  
    libc6 \  
    openssl \  
    libnl-3-200 \  
    libgcc-s1 \  
    libnuma1 \  
    libibverbs1 \  
    librdmacm1 \  
    uuid-runtime \  
    libaio1 \  
    libc6 \  
    libssl3 \  
    libssl-dev
```

To install the required libraries on RHEL and RHEL-based systems, run the following command:

```
sudo dnf install \  
    glibc.x86_64 \  
    libstdc++.x86_64 \  
    openssl-libs.x86_64 \  
    libnl3.x86_64 \  
    libgcc.x86_64 \  
    numactl-libs.x86_64 \  
    libibverbs.x86_64 \  
    librdmacm.x86_64 \  
    libuuid.x86_64 \  
    glibc.x86_64
```

## Installation Procedure

The commands described in this chapter require superuser privileges. Please log in as an administrator to execute them. By doing so, the script will be able to create autorun files and access the destination directory for file installation. If you do not need the autorun feature and have access to the destination directory then the installation script can be run as a regular user, however xiRAID Opus must be run with the superuser privileges in any case.

Follow the steps below to install xiRAID Opus:

1. Begin by downloading the xiRAID Opus package from the [Xinnor website](#). To download the latest version, run the following command:

```
wget https://pkg.xinnor.io/repository/Repository/opus/ver1.3.0/ver1.3.0-d.tgz
```

2. Once the package is downloaded, unpack it using the following command:

```
tar -xvf ver1.3.0-d.tgz
```

3. After unpacking the archive, navigate to the extracted directory:

```
cd ver1.3.0-d/
```

4. Once inside the installation directory, you can proceed with running the installation script (`install.sh`). The script supports two installation modes: **Non-interactive** and **Interactive**.

## Non-interactive Mode

In non-interactive mode, you must specify required installation parameters directly when running the script. The two required parameters are:

- `--hugemem`: Amount of hugepage memory to allocate (in MB)
- `-m`: CPU mask, indicating which CPU cores to allocate

A detailed description of the `--hugemem` and `-m` parameters, along with the additional optional parameters, is provided in the **Installation Parameters** section below. If any parameters are not explicitly provided, the script will use default values. By default, the xiRAID files are installed to the `/opt/xiraid/` directory. xiRAID Opus is registered as a system service and marked to autorun.

Once you have determined the appropriate values for `--hugemem` and `-m`, you can run the installation script as follows. The example below allocates 32 GB of hugepage memory and assigns the first 16 CPU cores:

```
sudo ./install.sh -m [0-15] --hugemem 32768
```

If the installation is successful, the output from the command above will contain the following information:

```
.....
#####
Created symlink /etc/systemd/system/default.target.wants/
xnr_xiraid.service → /etc/systemd/system/xnr_xiraid.service.
The systemd service is enabled.
#####
Path to the files:
filelist.txt: /opt/xiraid/filelist.txt
uninstall.sh: /opt/xiraid/uninstall.sh
Installation completed successfully.

What is installed:
- See 'filelist.txt' in the destination directory for a list of
  installed files.

Uninstallation:
- To remove the package but keep your RAID configurations:
  Run: uninstall.sh
- To remove both the package and all RAID configurations (Warning:
  this action is irreversible):
  Run: uninstall.sh --force

Service Management:
- Autorun is configured as a systemd service, and the service has
  been started automatically.
- To see current status of the xiraid service run:
  sudo systemctl status xnr_xiraid
- To restart or stop the service, use:
  sudo systemctl restart xnr_xiraid
  sudo systemctl stop xnr_xiraid

Configuration:
- Current CPU mask: [0-3]
- Hugepages configuration: 8192
- If your system does not have sufficient resources, change the
  configuration using:
  sudo /opt/xiraid/bin/xnr_xiraid.1.3.0 --xnr-hugemem
[val1,val2,...] -m CPUMASK --xnr-nostart
- For help on parameters, run:
  sudo /opt/xiraid/bin/xnr_xiraid.1.3.0 -h
```

## Installation Parameters

Parameter	Description
<code>--hugemem</code>	<p>Specifies the amount of hugepage memory to allocate, in MB, for each NUMA node. The values are written to the configuration file after the installation is completed. Ensure that the specified amount of memory is available on your system.</p> <p>The parameter expects a comma-separated list of memory values for each NUMA node index, in the format:</p> <pre>32768,-2,16384,0</pre> <ul style="list-style-type: none"><li>• Positive values set the amount of memory (in MB) to allocate for the corresponding NUMA node.</li><li>• Negative values are specified to keep the memory configuration of the specified NUMA node unchanged.</li><li>• Zero values are used to set the amount of memory for the specified NUMA node to zero.</li></ul> <p>The following example will allocate 32 GB of hugepage memory to NUMA node 0, leave NUMA node 1 unchanged, allocate 16 GB to NUMA node 2, and set the amount of memory for NUMA node 3 to zero.</p> <pre>--hugemem 32768,-2,16384,0</pre> <p>If you know the number of I/O operations a single CPU core can handle simultaneously, you may use the following formula to determine how much <code>hugemem</code> is required for your configuration. Note that this formula only applies to configurations where xiRAID Opus runs on a single NUMA node.</p> <pre>hugemem = (IO per core) * (number of allocated CPUs) * 500,000 / 0.6</pre>

**-m** Select which of the CPU cores can be used by the application (the CPU mask parameter). This mask will be written to the configuration after installation is completed. CPU core mask can specify any CPUs at any NUMA node. It is recommended to use same NUMA node for selected CPU cores. The mask can be specified in one of the following ways:

- 0x1f07
- 0x1F07
- [0,1,2,8-12]
- [0, 1, 2, 8, 9, 10, 11, 12]

In the examples above, the application is configured to use the following CPU cores: 0, 1, 2, 8, 9, 10, 11, and 12.

<b>--no-xiraid</b>	Do not install the engine binary (the RAID engine). Only CLI tools will be installed. This is useful if you want to install the CLI on a separate workstation. The default value is to install the engine.
<b>--no-cli</b>	Do not install the CLI binary (the CLI tools). Only the RAID engine will be installed. The default value is to install the CLI.
<b>--no-link</b>	Do not create additional symlinks. If set, this flag prevents symlink creation described in the "-l" flag section. By default, this flag is not set.
<b>--no-autorun</b>	Do not create <i>systemd</i> files to automatically run the engine at system startup. If you want to set up the engine startup manually, please use this flag. If the autorun feature is enabled, the <i>.service</i> file will be generated in the <i>/etc/systemd/system/</i> directory. This will enable the automatic startup of your RAID engine upon reboot. The default value is to have autorun set up.
<b>-o</b>	The output directory where the xiraid directory, containing all the installed files, will be created. By default, it is set to <i>/opt</i> .
<b>-l</b>	The directory where the symlinks to the installed binaries will be created. It is useful if you want to add symlinks to a

directory that is included in the PATH variable. The default value is `"/usr/bin"`.

## Considerations for Installing on Systems with Multiple NUMA Nodes

When installing the software on a system with multiple NUMA (Non-Uniform Memory Access) nodes, there are several important factors to ensure optimal performance:

### 1. Allocate CPU Cores and Hugepages from a Single NUMA Node

For optimal performance, both the CPU cores and the hugepage memory (hugepages) should be allocated from the same NUMA node. Allocating resources across different NUMA nodes can lead to performance degradation due to cross-node memory accesses, which are slower than accessing local memory.

### 2. Memory Allocation Limit per NUMA Node

It is recommended that no more than half of the available memory from a single NUMA node be allocated to xiRAID Opus. Allocating more than half could lead to memory pressure on the NUMA node, which can negatively affect the system's overall performance and stability.

### 3. Ensure Drives and Network Interfaces are on the Same NUMA Node

To further optimize performance, ensure that all drives and network interface cards (NICs) used by xiRAID Opus are located on the same NUMA node as the allocated CPU cores and hugepage memory. If these devices are located on a different NUMA node, communication between them and the CPU will incur higher latency, which can affect the software's overall throughput and efficiency.

### 4. NUMA Node Selection for Performance Optimization

Before running the installation script, carefully plan which NUMA node to use for allocating CPU, memory, drives, and NICs. Tools such as `numactrl` or `lscpu` can help you identify NUMA node layouts and ensure that resources are appropriately matched for maximum performance.

## Interactive Installation Mode

In interactive mode, the script prompts you to input each installation option manually. This mode is suitable if you prefer to be guided through the setup.

To start the script in interactive mode, run:

```
sudo ./install.sh -i
```

In steps 1-6, press enter Enter to accept the default value.

1. The first prompt asks you to specify the installation directory. By default, the software is installed in the `/opt` directory.
2. Next, specify whether to create symlinks to the installed binaries. By default, symlinks are created.
3. You will then be prompted to enter the directory where the symlinks should be created. This is useful if you want to place them in a directory that is included in the `PATH` variable. The default value is `/usr/bin`.
4. Specify whether to install the CLI tool used to manage the software. You can skip this step if you prefer to use the gRPC API or plan to manage the software from another system and do not need the CLI tool locally.
5. Specify whether to install the product binaries. Select "no" if this system is only being used to manage another instance of the software using the CLI tool.
6. Indicate whether the software should start automatically on system boot. If enabled, it will be configured as a systemd service.
7. Specify the amount of hugepage memory to allocate, in MB, for each NUMA node. The values are written to the configuration file after the installation is completed. Ensure that the specified amount of memory is available on your system.

The parameter expects a comma-separated list of memory values for each NUMA node index, in the format:

```
32768,-2,16384,0
```

8. Specify which of the CPU cores can be used by the application (the CPU mask parameter). This mask will be written to the configuration after installation is completed. CPU core mask can specify any CPUs at any NUMA node. It is recommended to use same NUMA node for selected CPU cores. The mask can be specified in one of the following ways:

- 0x1f07
- 0x1F07
- [0,1,2,8-12]
- [0, 1, 2, 8, 9, 10, 11, 12]

In the examples above, the application is configured to use the following CPU cores: 0, 1, 2, 8, 9, 10, 11, and 12.

```
Enter interactive setup
Enter destination directory. The "xiraid" directory will be created there [/opt]:
/opt
Create symlink to binaries? yes/no [y]:
y
Enter directory to create symlink. (Directory from $PATH) [/usr/bin]:
/usr/bin
Install CLI tool? yes/no [y]:
y
Install product binaries? yes/no [y]:
y
Enable autostart application as systemd service [y]:
y
Enter comma separated values in MB for each NUMA node to allocate HUGE PAGES, use zero to skip the node []: 8192
8192
Enter cores to use for data processing, core mask (like 0xF) or core list of '[' embraced (like [0,1,10-14]) []: [0-3]
[0-3]
```

If the installation is successful, the output from the command above will contain the following information:

```
.....
#####
Created symlink /etc/systemd/system/default.target.wants/
xnr_xiraid.service → /etc/systemd/system/xnr_xiraid.service.
The systemd service is enabled.
#####
Path to the files:
filelist.txt: /opt/xiraid/filelist.txt
uninstall.sh: /opt/xiraid/uninstall.sh
Installation completed successfully.

What is installed:
- See 'filelist.txt' in the destination directory for a list of
  installed files.

Uninstallation:
- To remove the package but keep your RAID configurations:
  Run: uninstall.sh
- To remove both the package and all RAID configurations (Warning:
  this action is irreversible):
  Run: uninstall.sh --force

Service Management:
- Autorun is configured as a systemd service, and the service has
  been started automatically.
- To see current status of the xiraid service run:
  sudo systemctl status xnr_xiraid
- To restart or stop the service, use:
  sudo systemctl restart xnr_xiraid
  sudo systemctl stop xnr_xiraid

Configuration:
- Current CPU mask: [0-3]
- Hugepages configuration: 8192
- If your system does not have sufficient resources, change the
  configuration using:
  sudo /opt/xiraid/bin/xnr_xiraid.1.3.0 --xnr-hugemem
[val1,val2,...] -m CPUMASK --xnr-nostart
- For help on parameters, run:
  sudo /opt/xiraid/bin/xnr_xiraid.1.3.0 -h
```

# System Configuration Best Practices

## NVMe Drives

- It is highly recommended to reformat NVMe namespaces to a 4k block size.
- All drives intended for use in the same RAID array must be identical in model, firmware version (preferably the latest), and block size.

## BIOS Settings

- Disable C-States for optimal CPU performance.
- Switch the CPU operation mode to *Performance* (the default is typically *Balanced*).
- Enable Hyper-Threading (HT/SMT) to achieve better performance.

## Kernel Configuration

- To disable power-saving states on Intel processors, add `intel_idle.max_cstate=0` to the Grub kernel command line. This ensures the CPU does not enter any idle power-saving mode.
- Set the polling mode for NVMe devices:

```
echo "options nvme poll_queues=4" >> /etc/modprobe.d/nvme.conf
```

After applying these kernel parameters, rebuild the InitRAMFS:

- For RHEL-derivative distributions, run:

```
dracut -f
```

- For Debian-derivative distributions, run:

```
update-initramfs -u -k all
```

## Recommended System Settings

- Use the `cpupower frequency-set` command to set the CPU frequency governor to *Performance*, ensuring the CPU operates at its highest frequency. This improves performance but may lead to increased power consumption and higher temperatures.

```
cpupower frequency-set -g performance
```

To verify, use the command:

```
echo "Current CPU frequency governor: $(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor)"
```

- Use the `tuned-adm` command to set the system profile to *accelerator-performance* for mostly sequential workloads, or *latency-performance* for workloads requiring low latency.

To set the system profile to *accelerator-performance*, run:

```
tuned-adm profile accelerator-performance
```

To set the system profile to *latency-performance*, run:

```
tuned-adm profile latency-performance
```

## spdk\_top

The `spdk_top` tool provides real-time insights into CPU core usage by SPDK lightweight threads and pollers, helping you assess the efficiency of your xiRAID Opus instance. The `spdk_top` tool is one of the tools that comes pre-installed with xiRAID Opus. For more detailed information, refer to the [SPDK documentation](#).

## Testing the Installation

Check the engine status using the command:

```
systemctl status xnr_xiraid
```

If the output indicates that the engine is loaded and active (running), the installation has completed successfully.

```
● xnr_xiraid.service - XiRAID
   Loaded: loaded (/etc/systemd/system/xnr_xiraid.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-08-22 07:00:21 UTC; 2 days ago
     Main PID: 166585 (reactor_0)
        Tasks: 21 (limit: 19026)
       Memory: 23.0M
          CPU: 1w 4d 23h 28min 52.565s
      CGroup: /system.slice/xnr_xiraid.service
             └─166585 /opt/xiraid/bin/xnr_xiraid.1.2.0
```

If the engine is up and running, check that the CLI is able to connect to the engine and the engine version is correct by using the command:

```
xnr_cli config get --name version
```

## Troubleshooting

### 1. Error while dialing

```
Error while dialing: dial tcp <port_address>: connect:
connection refused.
```

- Reason 1: The appropriate port is not open and inaccessible for the CLI (by default it is 8000).
- Reason 2: The xiRAID Opus configuration has been changed. Restart the service.

### 2. Failed to unlink lock

```
*ERROR* Failed to unlink lock fd for core x, errno: 2
```

Reason: The CPU mask parameter has not been configured properly before starting the server. You can reconfigure it as described below.

### 3. xnr\_plt\_set\_hugemem error

If you specify an incorrect hugemem size—one that is larger than what can be allocated on the system—the product cannot start properly. In case of incorrect hugemem the following error is displayed in the syslog:

```
xnr_plt_set_hugemem: ERROR: Node 0, hugepages 2048kB requested
450000, allocated 4578
```

If the product is installed with a wrong CPU mask or hugemem parameters and cannot start as a result you can change these parameters by calling the xiRAID Opus binary with another CPU mask and hugemem values. The operation must be run with superuser (root) rights. --xnr-nostart flag should be used. Here is an example to update the CPU mask to 0xFFFF and hugemem size to 32768 megabytes:

```
# <path_to_installed_binaries>/xnr_xiraid.1.3.0 -m 0xFFFF
--xnr-hugemem 32768 --xnr-nostart
```

or if you allowed symlink creation on the installation step:

```
# xnr_xiraid -m 0xFFFF --xnr-hugemem 32768 --xnr-nostart
```

#### 4. CONFIG file not found

The warning appears during xiRAID Opus installation: `*WARNING*: CONFIG: file ../../module.cfg not found`

Reason: The hugemem parameter has not been configured properly before starting the server. You can reconfigure it as described in the previous step.

#### 5. Other issues

To investigate and resolve other issues refer to the logs and use `grep` to search for ERROR/WARNING messages. These logs can be located in the `/var/log/syslog` file by searching for the keyword "spdk" in lowercase.

### If You Need Further Assistance

If you encounter an issue that you cannot resolve using this manual, please contact the xiRAID Opus support team for assistance.

Before submitting a support request, it is recommended that you run the `collect_dbg_info.sh` script. This script gathers diagnostic information about your system and xiRAID Opus configuration, creating an archive file that helps the support team troubleshoot your issue more efficiently.

To run the diagnostic script:

1. Open a terminal and navigate to the xiRAID Opus installation directory. By default, the installation directory is `/opt/xiraid`. Navigate into the `scripts` folder inside the installation directory:

```
cd /opt/xiraid/scripts
```

2. Run the script using the following command:

```
sudo ./collect_dbg_info.sh
```

The script will collect relevant system logs and configuration details, and generate a `.tgz` archive in the same directory. Attach the generated archive file to your support request when contacting the support team.

## Uninstalling xiRAID Opus

The process of uninstalling xiRAID Opus involves disabling system processes and removing all installed files. To do so, log in as root and use the `uninstall.sh` script located in the target directory (by default - `/opt/xiraid/uninstall.sh`).

To uninstall the product using the `uninstall.sh` script, run:

```
<install_dir>/uninstall.sh
```

The output of this command may include 'failed to remove' messages because by default, the `uninstall.sh` script does not delete the directory containing configuration files.

If xiRAID is uninstalled but configuration files are retained, xiRAID can be reinstalled to the same directory. In this scenario, the engine configuration and RAID configuration are preserved and will be reapplied to the engine during its next run. This allows you to restore any RAID configurations you previously created.

If you do not need to save the configuration files, run:

```
/opt/xiraid/uninstall.sh --force
```

To uninstall the product manually, disable the systemd service:

```
systemctl stop xnr_xiraid
```

After that, remove all files listed in the `filelist.txt` (located in the installation directory) using the command:

```
rm -rf /opt/xiraid/bin/xnr_conf
```

## Updating xiRAID Opus

Before updating xiRAID Opus:

- Make sure your system meets the [System Requirements](#).
- Stop all input/output operations involving block devices managed by xiRAID Opus.
- If you have configured Vhost targets in xiRAID Opus, stop all VMs connected to those Vhost targets for the duration of the update.
- Back up the configuration files:



```
sudo cp -r /opt/xiraid/bin/xnr_conf/ ~/xnr_conf
```

- It is recommended to back up all data before running the update.
- Downgrading to an earlier version is not supported.

To update xiRAID Opus, download the latest version of the `.tgz` installation package from the [Xinnor website](#), then:

1. Once the package is downloaded, unpack it using the following command:

```
tar -xvf <package_name>.tgz
```

2. After unpacking the archive, navigate to the extracted directory:

```
cd <package_name>/
```

3. Once inside the installation directory, run the following command to perform the update:

```
sudo ./install.sh --upgrade
```

4. Confirm that you want to upgrade xiRAID Opus.

```
The product will be upgraded
Installed version : 1.2.0
Package version  : 1.3.0-pre
Do you want to continue? [n]: y|
```

If the update was successful, you will see the following message:

```
Do you want to continue? [n]: y
y
.....
Created symlink /etc/systemd/system/default.target.wants/xnr_xiraid.service → /etc/systemd/system/xnr_xiraid.service.
The systemd service is enabled.
#####
Path to the files:
filelist.txt: /opt/xiraid/filelist.txt
uninstall.sh: /opt/xiraid/uninstall.sh
Installation completed successfully.
```

Check the engine status using the command:

```
systemctl status xnr_xiraid
```

If the output indicates that the engine is *loaded* and *active (running)*, the update has completed successfully.

Check that the CLI can connect to the engine and the engine version is correct by using the command:

```
xnr_cli config get --name version
```

## License Management

xiRAID Opus requires a license. The license determines access to the following features:

- **Number of Devices** – Specifies the total number of devices that can be attached to the Device Manager. These devices may be used in a RAID and/or exposed over the network.
- **Support (yes/no)** – Determines whether users have access to Xinnor support and the ability to install xiRAID Opus updates.
- **Engineering Mode (yes/no)** – Grants access to certain restricted features. This mode is primarily intended for the support team and is not recommended for general customer use.

## Trial License

xiRAID Opus comes with a 7-day trial license that allows you to manage up to 4 drives. To see the expiration date of the trial license and how many drives are currently in use, run the following command:

```
xnr_cli license show -o json
```

The output of the command contains the following information:

```
{
  "hwkey": "C0A9EB3C13788900",
  "keys": [],
  "features": [
    {
      "name": "DEVICES",
      "created": "2025-07-02",
      "expired": "2025-07-09",
      "status": "TRIAL",
      "counts": {
        "licensed": 4,
        "in_use": 0
      }
    }
  ]
}
```

To see which drives are currently managed by xiRAID Opus, run:

```
xnr_cli dm show -f managed_by_us=true
```

If your trial license has expired, it is possible to reapply the license and reset the expiration date by restarting the engine:

```
systemctl stop xnr_xiraid
systemctl start xnr_xiraid
```

## License Installation

To use xiRAID Opus in a production environment, you need a proper license.

When you request a license, you need to provide the hardware key (`hwkey`) of the machine where you plan to install xiRAID Opus. Your hwkey can be obtained from the output of the `xnr_cli license show` command:

```
xnr_cli license show -o json
{
  "hwkey": "C0A9EB3C13788900",
  "keys": [],
  "features": [
    {
      "name": "DEVICES",
      "created": "2025-07-07",
      "expired": "2025-07-14",
      "status": "TRIAL",
      "counts": {
        "licensed": 4,
        "in_use": 4
      }
    }
  ]
}
```

Licenses are distributed as text files and include the following data:

```
hwkey: C0A9EB3C13788900
license_key: <license_key>
version: 2
created: 2025-07-07
disks: 10
    expire: 2025-07-12
support:
    expire: 2025-07-12
```

To install a license, use the `xnr_cli license add` command. As an option for the command, specify either the path to the license file or the hardware key:

- `xnr_cli license add --path ./license.txt`
- `xnr_cli license add --key <license_key>`

To verify that the license was installed correctly, run the `xnr_cli license show` command.

## Show License State

To get the list of available licenses, use the command:

```
xnr_cli license show
```

Below is an example of a valid license:

HWKEY C0A9EB3C13788900	
FEATURE	INFO
devices	created: 2025-08-25 expired: 2025-09-30 status: VALID licensed: 10 in_use: 0
support	created: 2025-08-25 expired: 2025-09-30 status: VALID

The `in_use` parameter in the devices section show how many devices are currently attached to the Device Manager. To see which drives are currently managed by xiRAID Opus, run:

```
xnr_cli dm show -f managed_by_us=true
```

## Merging Licenses

If several features (like drives and support) are licensed by the same license, then the expiration date is applied to all of them. However, the Support feature has its own expiration date.

Multiple licenses can be applied to one system, with each license stored separately in the system configuration file. During system startup, all installed licenses are applied in the same order in which they were originally installed. When multiple licenses are applied to one system, the features covered by these licenses are combined according to the merging rules:

1. If the license includes 0 drives, the number of available drives remains the same as the value specified in the previously applied license.
2. If the current license in use is a trial and another license is applied, the number of supported drives will be determined by the new license (if it is not 0).
3. If the number of drives in the new license is greater than in the current (non-trial) AND the expiration date of the new license is later than the expiration date of the current license, then the new drive configuration is applied. Otherwise, the number of drives value of the new license is ignored.
4. If a new drive value is applied by a license, the new number of drives **OVERRIDES** the old value, but does not add to it.
5. If the Support feature is enabled in a new license and the license's support expiration date is later than the current license's support expiration date, or if support was not previously licensed for the system, then the engine's support expiration date is extended to match the new license's support expiration date.
6. If the same license is applied twice then second applying is ignored and nothing is changed.

## Delete License

To delete all installed licenses, use the command:

```
xnr_cli license remove --force
```

If licenses are deleted, the system reverts to the trial mode.

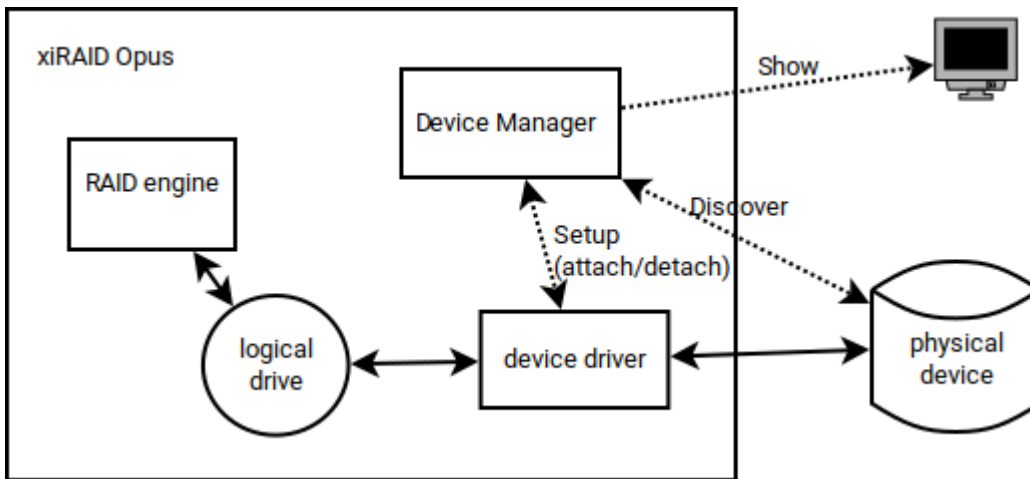
If you have deleted licenses but have backup copy of license files, you can reinstall these licenses.

## Device Manager (DM)

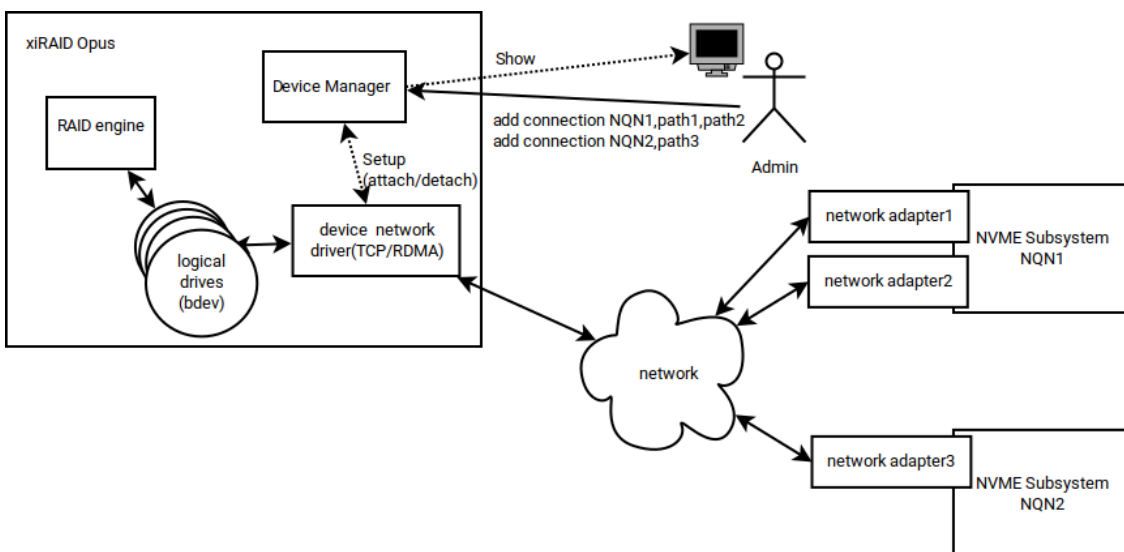
In the xiRAID Opus topology, any RAID is built from logical drives. Logical drives become available for the RAID create operation only after devices have been attached to xiRAID Opus. The attach and detach operations are managed by the device manager (DM).

The DM provides the following major functions:

1. Discover available PCIe NVMe devices and show their list. The device manager cannot discover network devices.
2. Attach physical or network devices to xiRAID Opus so that logical drives are ready to be used in RAID.
3. Provide a list of logical drives available in xiRAID Opus and show which physical device contains which logical drives.
4. Detach logical drives so that the corresponding physical or network devices can be used in another application.
5. Reattach previously attached devices after a restart of xiRAID Opus or the host OS.



Device Manager with attached physical devices



Device Manager with attached network devices

**Note about virtual devices**

Virtual devices provided by virtual machine hypervisor can be attached by the Device Manager if xiRAID engine is running on the virtual machine. Virtual devices IO is unstable as well and can be used in experimental mode only.

## Discover Devices

The DM discovers available devices on the host OS upon start of the xiRAID engine and periodically throughout the engine's lifecycle. Each discovered device is associated with a unique ID. The devices managed by the DM are a subset of all the physical devices available in the host OS. DM does not report devices of unsupported types. During the discovery phase, the DM only detects the availability of devices and does not have information about their content and other parameters.

To get the list of all discovered devices, use the following command:

```
xnr_cli device-manager show
```

To get detailed information about specific devices, specify a list of comma-separated drive identifiers.

```
xnr_cli device-manager show --id <device_id1>,<device_id2>
```

DEVICE ID	STATUS	PARAMETERS	DRIVES
nvme.c0a9eb3c13788900.0	ATCH	address 0000:06:0a.0 vendor VirtIO dev_id 0x1001	name: nvme.c0a9eb3c13788900.0 uuid: aaa1cb53-2a9c-5c1d-97c7-f66ffbf2fb2f0
nvme.c0a9eb3c13788900.1	OS	address 0000:06:0d.0 vendor VirtIO dev_id 0x1001	/dev/vdb

The output of the `show` command is presented in either table or JSON format depending on the `--output` option. Each line in the table or item in the JSON array represents a particular physical device and provides the following information:

- The unique ID of the device assigned by the DM. The ID is used to reference the device in DM commands.
- The device parameters such as PCI address, vendor name and hardware model identifier
- The device status: OS, US, ATCH, or LOST
- If the device is attached, name, UUID and serial number of the logical drive(s) hosted by this device
- If a drive of the device is used by a RAID, the RAID name and UUID

To attach one or several devices to xiRAID Opus, use the command:

```
xnr_cli device-manager nvme attach --ids <device_id1>,<device_id2>
```

You may also use `dm` in the place of `device-manager`.

```
xnr_cli dm nvme attach --ids <device_id1>,<device_id2>
```

## Blacklisting Devices

To prevent users from accidentally attaching devices that must not be attached to the device manager, add these drives to the blacklist. Blacklisted devices cannot be attached to xiRAID Opus.

To add a device to the blacklist, you must know its identifier. To view device identifiers, use the `xnr_cli dm show` command. Once you've obtained the identifier of the device to be added to the blacklist, execute the following command (replace the identifier in the example with the identifier of your device):

```
xnr_cli dm bl add -i nvme.c0a9eb3c13788900.2
```

It is also possible to add multiple device to the blacklist simultaneously:

```
xnr_cli dm bl add -i nvme.c0a9eb3c13788900.2,nvme.c0a9eb3c13788900.3
```

To verify that a device has successfully been added to the blacklist, execute the `xnr_cli dm show` command. Devices in the blacklist will have their status set to `BLACKLISTED`.

## Device State Machine

The device status reports the state in the device lifecycle state machine.

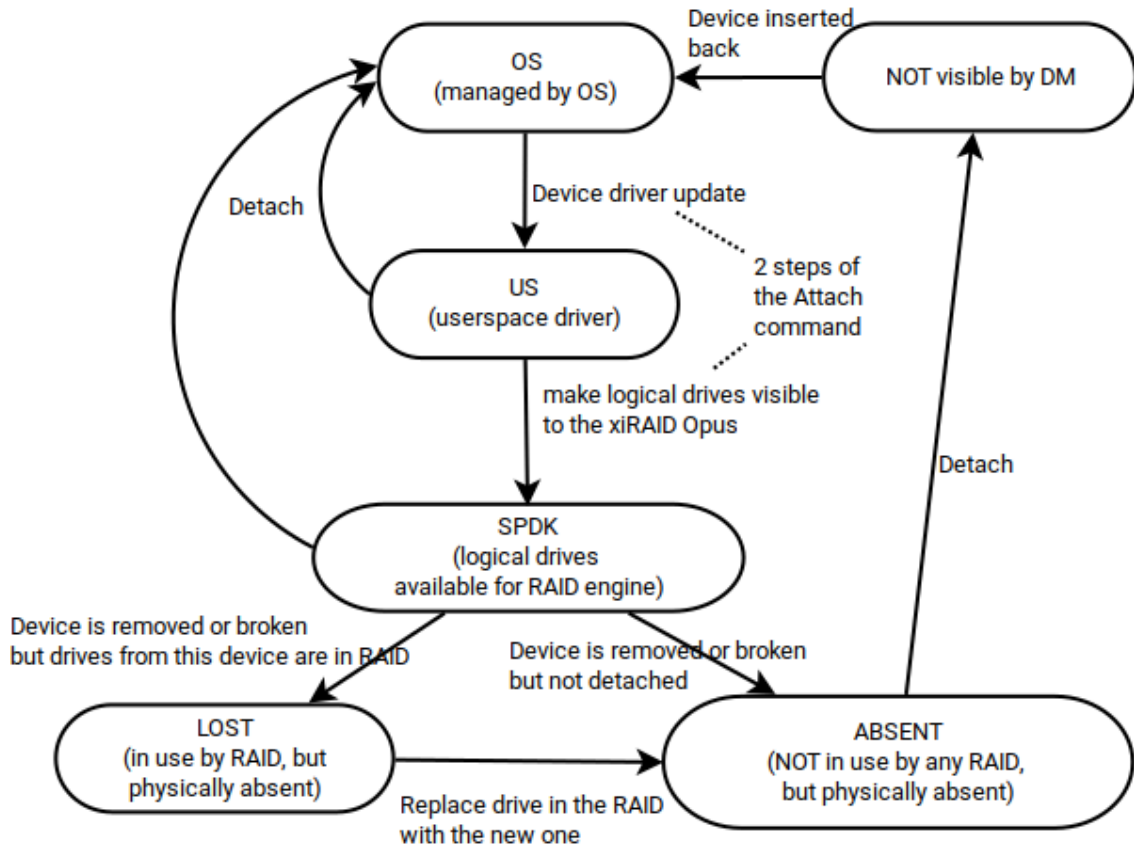


Figure 4: Device life cycle state machine

The possible states are:

- **OS** – The device is managed by host OS. The OS drivers are used for IO to the device’s drives. The device cannot be used by a RAID or other xiRAID Opus modules.
- **USpace** – The device driver has been updated, and the host OS does not manage it. However, the logical drives located on this device are not attached as BDEVs and are still not available as RAID backing drives. This is an intermediate state between fully attached and detached cases. Usually the state means that the device is detached but the xiRAID engine could not restore OS drivers. Attach or Detach command should be used to move the device to ATCH or OS state respectively. When a device is attached to the Device Manager, devices in the same IOMMU group are moved from the OS state to the USpace state. If the device is detached and no other devices from the IOMMU group are attached to

the Device Manager, all devices that were previously moved to the USpace state will be returned to the OS state.

- **ATCH** – The device driver has been updated, and the logical drives located on this device are attached as BDEVs. They can be used as backing drives for a RAID.
- **LOST** – The logical drive located on the device is used in a RAID configuration, and the physical device has been removed or is malfunctioning. The corresponding RAID state is reported as Degraded. To recover the drives and reconnect them back to the RAID the physical device has to be inserted back, the device should be reattached by DM Attach command, then the drive should be inserted to the corresponded RAID logical position by the “raid replace” command.
- **ABSENT** – Similar to LOST, but the drives of the device are not used in any RAID. The DM automatically stops to manage the device. Absent devices are not reported by the DM show command.
- **NOT VISIBLE** – The device does not exist or cannot be managed by the DM. Such devices are not reported by the DM show command.

## Device Scanner

xiRAID Opus can auto-attach PCIe devices that are hot-plugged into the system. This behavior applies to PCIe devices that were previously attached to the Device Manager.

When a PCIe device is attached to the Device Manager via the `xnr_cli device-manager nvme attach` command, xiRAID Opus stores information about this device and assigns it an identifier in the Opus configuration database. On every startup, xiRAID Opus then scans the PCIe bus and checks the list of available devices against the list of devices in the configuration database. Available devices managed by the Device Manager are automatically attached to xiRAID Opus. Unavailable devices are marked as *absent*. xiRAID Opus checks every hot-plugged PCIe device and compares it against the list of devices in the configuration database. If one of the unavailable devices is hot-plugged into the system at some point, it will be attached to xiRAID Opus.

When a PCIe device is detached from the Device Manager via the `xnr_cli device-manager nvme detach` command, xiRAID Opus removes information about this device from the configuration database. This device will not be automatically attached to xiRAID Opus on system startup.

## Device BDEVs

Internally any logical drive is represented as BDEV. Therefore, drives of an attached device are reported in scope of BDEVs list. To get the list of all available devices attached to xiRAID Opus, use the command:

```
xnr_cli bdev show
```

The output of the command includes information about all physical and network devices attached to xiRAID Opus, xiRAID Opus RAIDs, and partitions. For each BDEV in xiRAID Opus, the following information is provided:

```
{
  "id": {
    "name": "nvme.622f976b98657af3.2p1",
    "uuid": "1485aadd-3fc1-451e-8467-895d5cf161fc"
  },
  "product_name": "GPT Disk",
  "block_size": 512,
  "num_blocks": "2097152",
  "assigned_rate_limits": {
    "rw_ios_per_sec": "0",
    "rw_mbytes_per_sec": "0",
    "r_mbytes_per_sec": "0",
    "w_mbytes_per_sec": "0"
  },
  "supported_io_types": {
    "read": true,
    "write": true,
    "unmap": true,
    "flush": false,
    "reset": true,
    "nvme_admin": false,
    "nvme_io": false,
    "write_zeroes": true,
    "compare": false,
    "compare_and_write": false,
    "abort": false
  }
}
```

- id
  - name - This is the identifier name of the BDEV partition. In the example above, it is the first partition (`p1`) on a drive identified by the unique string `nvme.622f976b98657af3.2.`
  - uuid - The Universally Unique Identifier (UUID) is a globally unique identifier assigned to the BDEV.
- product\_name - The product name of this device. For NVMe namespaces, this is the product name of the NVMe drive. For xiRAID Opus RAIDs, this is the name of the RAID.
- block\_size - The block size (in bytes) of the device. This is the smallest unit of storage that the device can read or write in one operation.
- num\_blocks - This represents the total number of blocks on the BDEV.
- assigned\_rate\_limits - These are the rate limits for input/output operations. See [Managing BDEV IOPS and Throughput Limits](#) for more details.
- supported\_io\_types - These are the types of I/O operations supported by the block device:

- read - Represents the ability of the BDEV to read data from a specified location.
- write - Represents the ability of the BDEV to write data to a specified location.
- unmap - Refers to the ability to deallocate or remove a block of data, typically to free up space.
- flush - Refers to the operation of ensuring that all data written to the BDEV is physically persisted.
- reset - Represents the ability to reset the state of the BDEV, potentially clearing errors or reinitializing it.
- nvme\_admin - Refers to administrative commands related to NVMe devices, used for device management and configuration.
- nvme\_io - Refers to Input/Output operations specific to NVMe devices, handling the actual data transfer to and from the device.
- write\_zeroes - Refers to an operation that writes zeroed data to a block of memory or storage, often used for space reclamation.
- compare - Represents the ability to compare two blocks of data to check for equality.
- compare\_and\_write - Refers to an operation that first compares two blocks of data and, if they are different, writes new data to the block.
- abort - Refers to the ability to cancel or stop an ongoing operation on the BDEV.

## Managing BDEV IOPS and Throughput Limits

In some cases, you may want to set BDEV IOPS and/or throughput limits. This may be necessary during testing or development or to prevent resource exhaustion. To set limits for a BDEV, use the `xnr_cli bdev qos` command. Note that either the name (`--name`) of the BDEV must be provided as an option. To identify the BDEV name or UUID, use the `xnr_cli bdev show` command.

Use the `bdev qos set` command to set the IOPS limit for a BDEV. In the following example, the IOPS limit is set to 40,000.

```
xnr_cli bdev qos set -n d74fb9ce-efae-484f-ac2f-2a3b13e19e1f --rwi 40
```

Use the command below to set the read and write throughput limits. In the following example, the limit for read and write operations is set to 150 MB/s.

```
xnr_cli bdev qos set -n d74fb9ce-efae-484f-ac2f-2a3b13e19e1f
--rw_mbps 150
```

Note that it is also possible to set different limits for read and write operations. In the example below, the limit for read operations is set to 200 MB/s and the limit for write operations is set to 150 MB/s.

```
xnr_cli bdev qos set -n d74fb9ce-efae-484f-ac2f-2a3b13e19e1f
--read_mbps 200 --write_mbps 150
```

To view existing BDEV limits, use the `xnr_cli bdev show` command. To clear limits, use the `xnr_cli bdev qos clear` command as shown below.

```
#Clear the IOPS limit:
xnr_cli bdev qos clear -n d74fb9ce-efae-484f-ac2f-2a3b13e19e1f --rwi
#Clear the throughput limit for read and writer operations:
xnr_cli bdev qos clear -n d74fb9ce-efae-484f-ac2f-2a3b13e19e1f
--read_mbps --write_mbps
```

## Managing Local Devices

### Attaching Local Devices

To use a device's drive(s) in xiRAID engine logic the device has to be attached first. The attach operation replaces the device drivers and exposes drives hosted by the device to the xiRAID engine. After attachment, the device is no longer managed by the host OS. The drives can be used in scope of new or previously configured RAID. The DM keeps devices configuration and all attached devices are reattached automatically in case of xiRAID engine restart.

Drives of the attached device are configured as BDEVs. Technically any such BDEV can be used out of a RAID scope. For example, the BDEV can be exposed through vhost target. However, such configuration is not designed for production and is not persisted over the xiRAID engine restart.

To attach the device(s) use the command:

```
xnr_cli device-manager nvme attach --ids device_id,[device_id,...]
```

where `device_id` is an ID reported by `xnr_cli device-manager show`

If a device is being used by another host OS application (for example, if it contains partitions mounted to the operating system) the attach command will fail and an appropriate message will be displayed. User can force the attachment of such device, but this can lead to data corruption caused by changing the driver on the device used by the other application:

```
xnr_cli device-manager nvme attach --force --ids device_id
```

## Detaching Local Devices

If the device is not needed anymore it can be detached, returned to the OS and used by other applications. The detach operation is waiting while in-progress IO to the device drives is not completed, then removes the device drive's BDEV, then the DM restores OS device driver and return the device and its logical drives to the host OS. After detach the logical drives located on the device are no longer available to any RAID.

In some cases described in the “Important Notice on device driver management” section below, the device driver is not changed back to OS and stay in US state for some time.

To detach a device, use the command:

```
xnr_cli device-manager nvme detach --ids device_id
```

## Reattaching Local Devices

However, if a device is physically removed and reinserted it must be reattached manually to continue using the device's drives in RAIDs.

On host OS reboot, the Device Manager restores all device drivers to their states prior to the reboot. So, all devices are reattached automatically and corresponding drives become visible as BDEVs. Users do not need to attach devices again after each reboot/restart if the devices were attached before.

If a device is physically removed from the system its state is changed to LOST. If the same device or another device is reinserted into the same physical slot, the device is discovered by OS but is not reattached to xiRAID Opus automatically. Therefore

such removed and reinserted device must be reattached by the `device-manager nvme attach` command.

## Important Notice on PCIe-connected NVMe Device Management

The xiRAID Opus user space device driver operates on top of one of the standard OS drivers: `uio_pci_generic` and `vfio-pci`. By default, another device driver is typically used by the OS. This is the reason to change the device driver during attachment.

The `vfio-pci` driver is preferable and used if both `uio_pci_generic` and `vfio-pci` drivers are available on the host OS. However, the `vfio-pci` driver has an important limitation – the driver can only be changed for all devices in the same IOMMU group simultaneously. Therefore, if a user attaches one of the devices in the IOMMU group to the xiRAID Opus, the device driver for all other devices in that group will be changed as well. See the picture below for details:

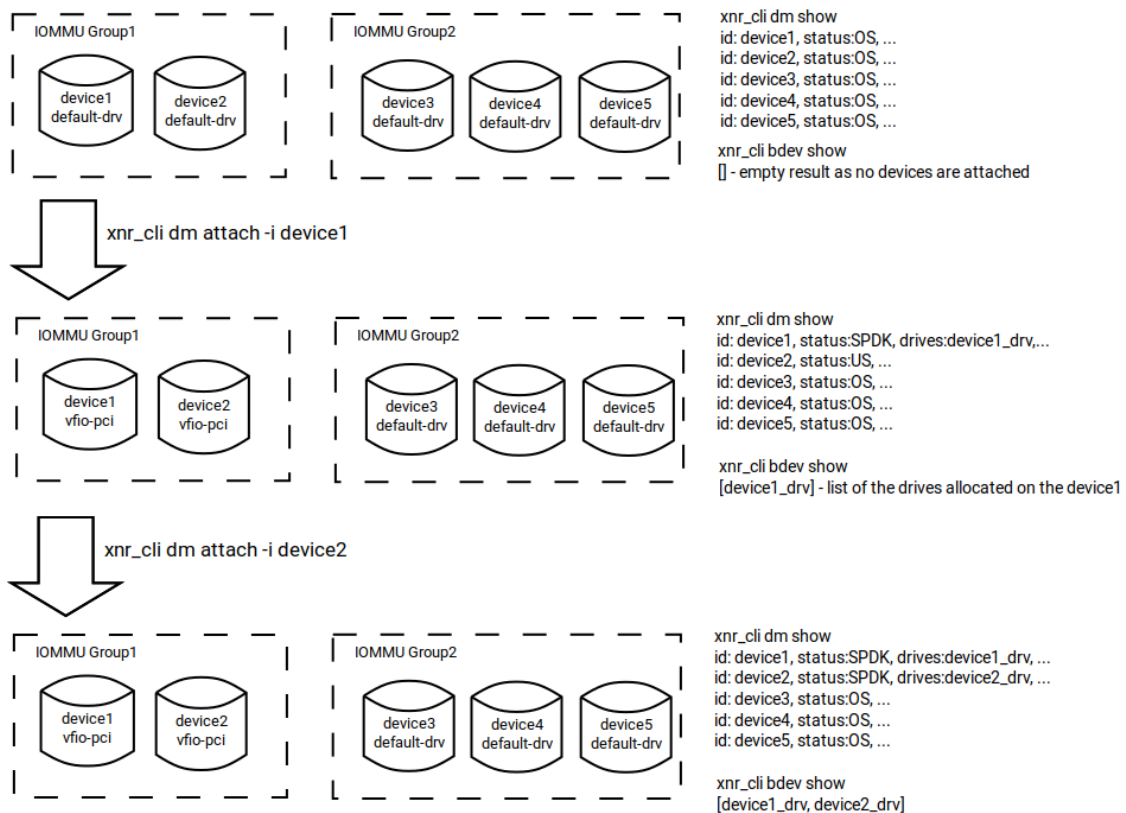


Figure 5. Attach devices located in the same IOMMU group

Here, after attach command is executed for device1 the driver is updated for both device1 and device2 as they are in the same IOMMU group. However, only the drive(s) allocated on device1 can be accessed by xiRAID Opus. Device1 has ATCH status, while the status of the device2 is US. As soon as the attach is executed for device2, drive(s)

allocated on the device2 also become available for the xiRAID Opus. Device2 driver is not updated again as it was already updated on previous attach step.

The same algorithm in reverse order works for the detach operation. If a user has several devices in the same IOMMU group then the driver will be updated to OS driver only after all devices in that group are detached. See the picture below for details:

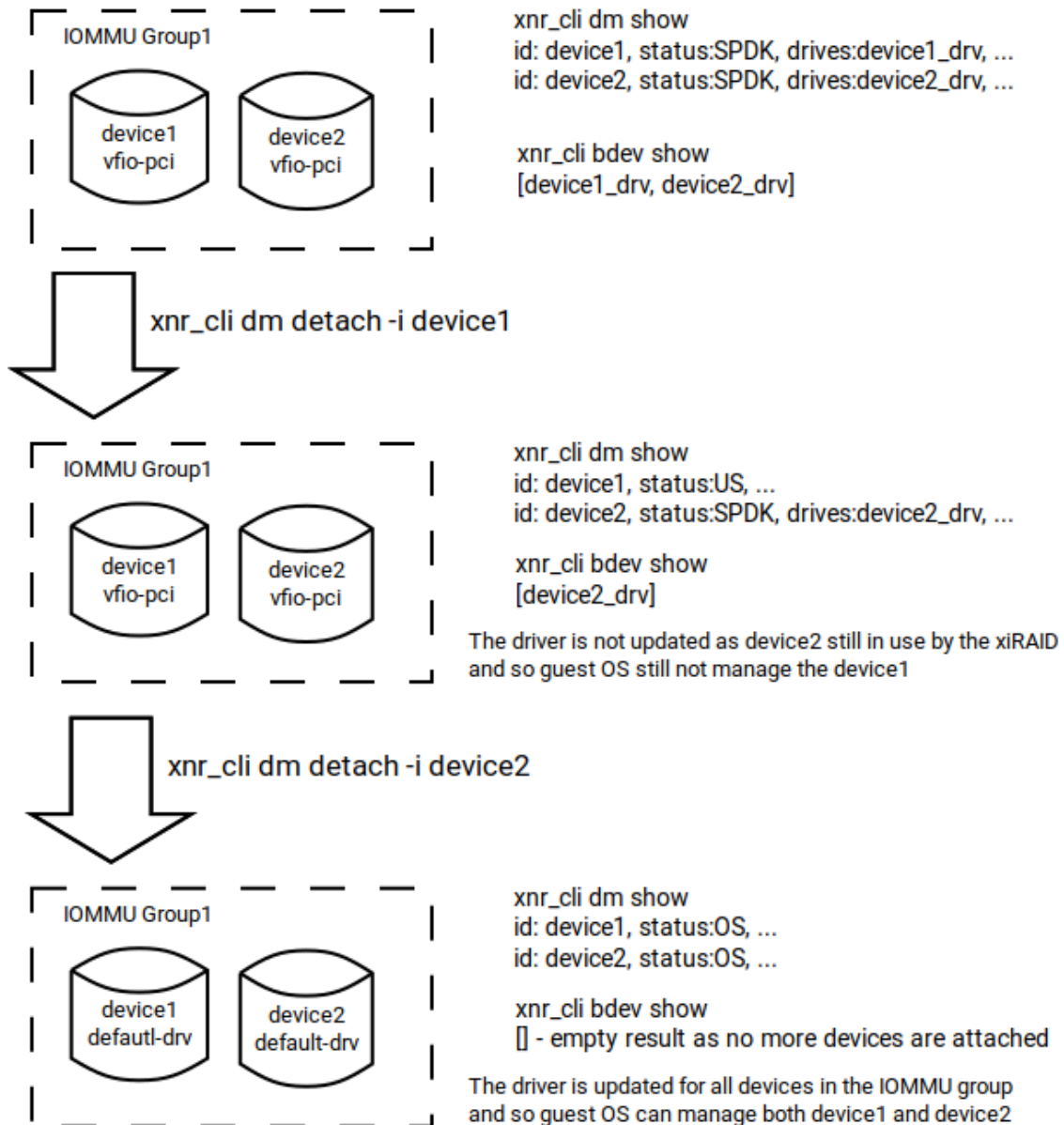


Figure 6: Detach devices located in the same IOMMU group

The allocation of devices along IOMMU groups, the number of groups and groups sizes are hardware dependent and can't be managed by the DM.

## Managing Network Devices

## Adding Network Devices

This section describes how to connect NVMe-oF devices to xiRAID Opus. To connect an NVMe-oF device to xiRAID Opus, you must know the NQN of the NVMe subsystem or namespace to connect to. You can attach either entire subsystems or individual namespaces to xiRAID Opus. Once you've attached a subsystem to xiRAID Opus, its namespaces are available as BDEVs to the system. As with other BDEVs, BDEVs based on namespaces of an NVMe subsystem can be used to create RAIDs.

If your NVMe-oF target is configured to expose NVMe-oF subsystems over multiple network paths, you can specify these paths simultaneously when you attach the system to xiRAID Opus. Additionally, it is required to configure the multipath policy for such subsystems.

To connect an NVMe-oF device to xiRAID Opus, run the following command:

```
xnr_cli dm net add-conn -q <NQN> -t <transport type> -a <path> --mp  
<multipath mode>
```



You can use `dm` in the place of `device-manager` in `xnr_cli`.

where:

- `<NQN>` - the NQN of the remote storage target for which to add a network connection. If this target has not been configured in xiRAID Opus, it will be automatically added with the provided path.
- `<transport type>` - the transport type of this connection: `TCP` or `RDMA`.
- `<path>` - the path to the remote storage target in the `<addrFam>/<address>/<svcId>` format, where:

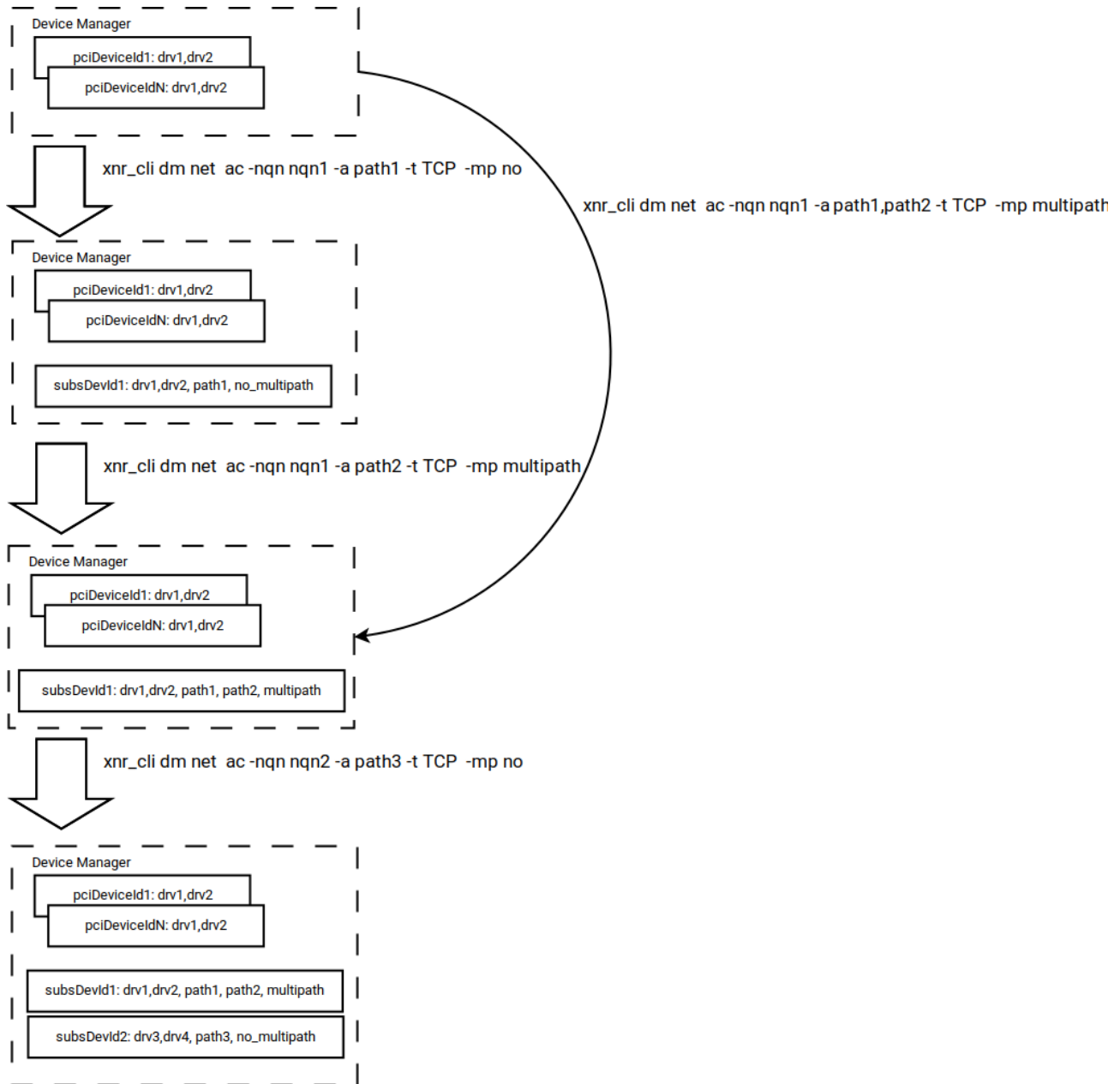
- `addrFam` - NVMe-oF transport address family: `ipv4`, `ipv6`, `ib`, or `fc`. This value can be omitted if you use `ipv4` or `ipv6`.
- `address` - NVMe-oF transport address (IP address).
- `svcId` - Transport service ID (port number).
- `<multipath mode>` - the multipath mode to use. Refer to [Multipath Configuration](#) for detailed documentation.
  - `disable` or `no` - In this mode, multipath is disabled. Only one path can be added for a single remote storage target.
  - `failover` or `fo` - Multipath operates in the failover mode, where only one path can be used at a time.
  - `multipath` or `mp` - In this mode, multiple paths can be active simultaneously.

### Command examples:

```
xnr_cli dm net ac -nqn nqn.xxx -t TCP -a ipv6/abcd::08:08:aa/4420
  -mp disable
xnr_cli dm net ac -nqn nqn.xxx -t TCP -a 1.2.3.4/4420 -mp disable
xnr_cli dm net ac -nqn nqn.yyy -t TCP -a ipv4/4.5.6.7/4420 -a
  ipv4/6.7.8.9/4420 -mp multipath
```

For more details, see the [device-manager network add-conn \(ac\)](#) page in the Command Reference.

### Example



The image above illustrates a workflow with the steps necessary to configure multiple network devices. Additionally, there is a scheme depicting the current state of the device manager before and after every step. The steps in the image are described below:

1. Adding a connection to a new network device.
2. Adding another connection to that device.
3. Adding a connection to another network device.

Note that the first two steps can be combined by specifying multiple paths to a network device in one command.

## Multipath Configuration



NVMe multipathing must be enabled on the host for this setup to function correctly. For instructions on enabling NVMe multipathing, please refer to [this article](#).

xiRAID Opus supports three multipath modes:

- **Active-Active** - In this multipath configuration, multiple paths are used to process I/O requests simultaneously. Additionally, users can configure how IO requests are distributed between paths. There are two policies available:
  - `rr` - With this selector, I/O requests are distributed in a round-robin fashion across all available paths. It is also possible to set a minimum number of I/O operations that should be routed to the current I/O path before switching to another path.
  - `mq` - With this selector, I/O requests are routed to the path with the lowest number of pending I/O requests.
- **Active-Passive** - In this multipath configuration, multiple paths are active at the same time, but only one path is used to process I/O requests at any given time. If ANA states are not configured on the target system, the first available path in the enumeration order is used as the active path.
- **Failover** - Multipath operates in the failover mode, where only one path can be used at a time. The other paths are only activated during switchovers. This results in higher failover latency compared to the multipath mode with an active-passive policy.

### Multipath behavior in environments with and without ANA:

	<b>ANA optimized/opti- mized</b>	<b>ANA optimized/non- optimized</b>	<b>Without ANA</b>
Active-Active	Active-Active (Round-Robin with <code>max_ queue_depth</code> or <code>min_ io</code> )	Uses optimized paths. If there is one opti- mized path and ten non-optimized paths, only the optimized path will be active.	Round-Robin with <code>max_queue_depth</code> or <code>min_io</code>

Active-Passive	Uses the first enumerated path.	Active for optimized paths, passive for non-optimized paths.	Uses the first enumerated path.
Failover	Only one path is active, the others are offline. ANA is not used.	Only one path is active, the others are offline. ANA is not used.	Only one path is active, the others are offline. ANA is not used

### Limitations:

- It is not possible to change the timeout settings to switch between active-passive modes; the initial settings will be used.
- There is no output showing path status (active/passive); only ANA-related information can be viewed.
- Path priority cannot be selected without using ANA.
- It is not possible to change the default path modification modes in the event of failback (automatic/manual mode for optimized path return).

For details about the commands required to configure multipath, see the [device-manager network add-conn \(ac\)](#) page in the Command Reference.

## Deleting Network Devices

This section describes how to remove an NVMe-oF subsystem from xiRAID Opus. You can either remove one of the configured network paths to an NVMe-oF subsystem, or you can remove all configured network paths at the same time.

To remove a connection to an NVMe-oF subsystem, run the following command:

```
xnr_cli device-manager network rem-conn -q <NQN> -a <address>
```

where:

- `<NQN>` - the NQN of the NVMe-oF subsystem to remove a connection to.
- `<address>` - the path to the NVMe-oF subsystem to remove a connection to.

If you delete the only path to a network device, the network device itself will also be removed from xiRAID Opus.

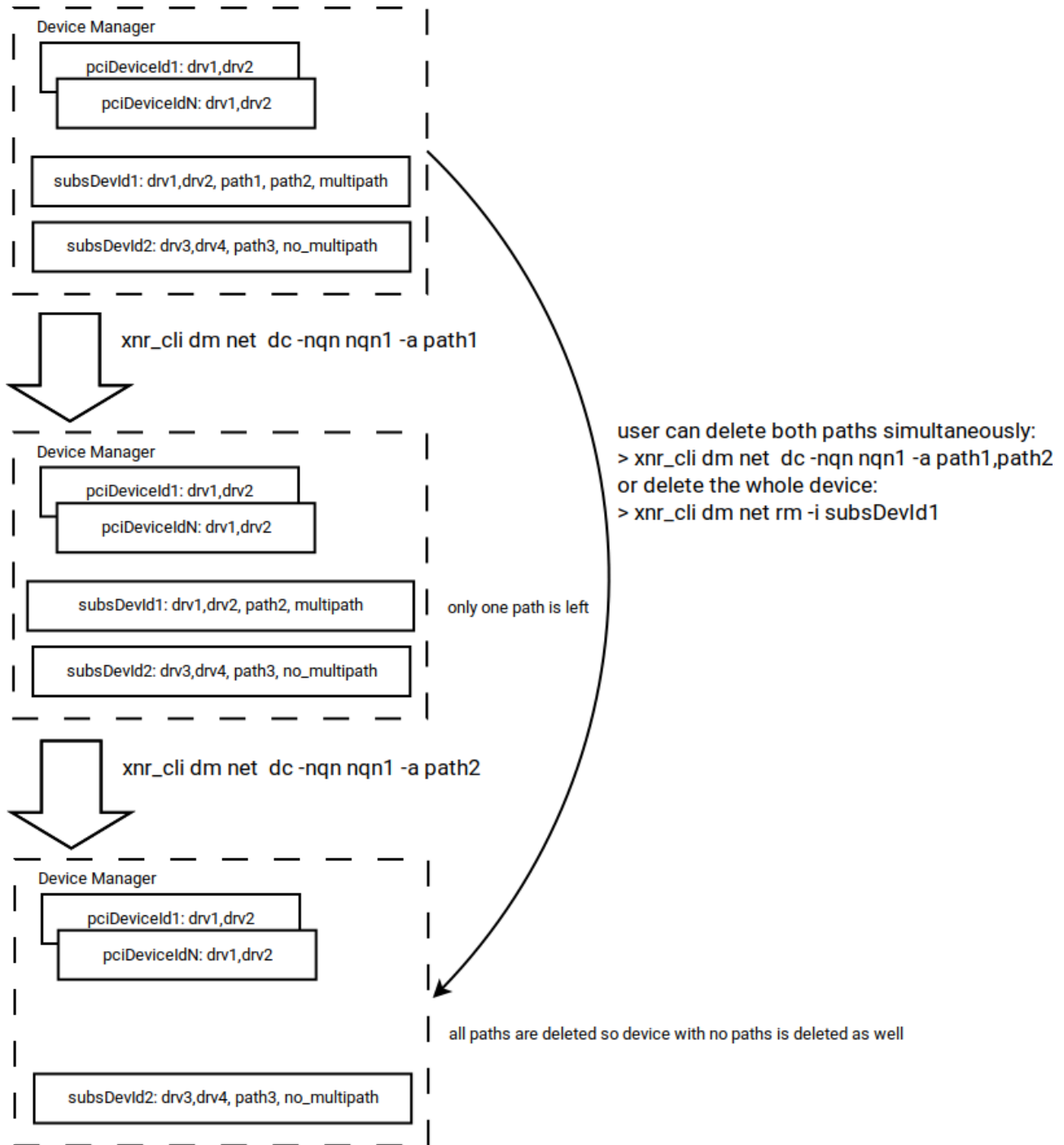
For more details about removing connections to NVMe-oF subsystems, see the [device-manager network rem-conn \(dc\)](#) page in the Command Reference.

To remove an NVMe-oF subsystem and all paths to this subsystem, run the following command:

```
xnr_cli device-manager network remove-dev -i <identifier>
```

where `<identifier>` is the ID of the device to remove. IDs of connected NVMe-oF subsystems can be obtained from the output of the `xnr_cli dm show` command. For more details, see the [device-manager network remove-dev \(rm\)](#) page in the Command Reference.

### Example



The image above illustrates a workflow with the steps necessary to remove connections to a network device. Additionally, there is a scheme depicting the current state of the device manager before and after every step. The steps in the image are described below:

1. Removing a single connection to a device without deleting the device itself from the device manager.
2. Removing another connection to the same device. Since there are no other connections to this device, it is removed from the device manager.

Alternatively, the user can either remove both connections by specifying their paths in one command or delete the whole device without first deleting its network connections.

## Working with RAIDs

RAID (Redundant Array of Independent Disks) is a way of combining storage devices to ensure data integrity and high performance. There are several methods of combining hard drives called RAID levels. Each level has its pros and cons and offers a different balance of performance, data protection, and storage efficiency.

### Supported RAID levels

xiRAID Opus enables you to create RAIDs of levels 0, 1, 5, 6, 7, 10, 50, 60 and 70.

**RAID 0** – interleaving data blocks without mirroring (replication of data across drives) or parity. The data blocks are distributed across several drives. The distribution results parallel IO mode that provides high performance. Due to the lack of redundancy, RAID 0 doesn't provide data reliability – the failure of one drive in RAID leads to the data corruption. RAID 0 requires at least 2 drives.

**RAID 1** – mirroring without parity or striping. The data is mirrored on all drives of the RAID, and the RAID size can only be of the smallest drive size. Random read performance of a RAID 1 may equal up to the sum of each member's performance, while the write performance remains at the level of a single slowest drive. RAID 1 requires at least 2 drives.

**RAID 5** – interleaving blocks with distributed parity. RAID 5 requires at least three drives. RAID 5 sustains the complete failure of one drive and provides a minimal degree of reliability.

**RAID 6** – interleaving blocks with double parity distribution. RAID 6 requires at least four drives. RAID 6 can sustain the complete failure of two drives. Redundant parity information provides additional time to restore redundancy without loss of information.

**RAID 7** – interleaving blocks with triple parity distribution. RAID 7 requires at least four drives. It is RAID 6 analog, but has a higher degree of reliability: three checksums are calculated using different algorithms, the capacity of three drives is allocated for checksums. Thus, the RAID 7 can sustain the complete failure of three drives.

## Nested RAIDs

The architecture of RAID levels 10, 50, 60, and 70 represents RAID 0 which components are RAID 1, 5, 6 and 7 respectively instead of separate drives. These levels can be described as grouped or nested RAID levels. The nested level **group size** parameter defines how many drives are used in each of RAID 5, 6 or 7 components. RAID 10 always use group size equal to 2. the drive number in a nested RAID must be a multiple of the group size.

**RAID 10** – striped set from a series of mirrored drives that is RAID 0, which components are RAID 1 instead of separate drives. Each RAID 1 mirror of RAID 10 always consists of two drives, the minimum number of RAID 1 in a stripe array is 2. Thus, in RAID 10 the minimum number of drives is 4. Data integrity is maintained in case of failure of half drives; the irreversible RAID destruction occurs when two drives of one mirrored pair already fail.

**RAID 50** – RAID 0 striping combination across multiple RAID 5. With such a combination, RAID 50 may show better performance with reduced latency. The group size is at least 3 drives. Recoverable from 1 drive failure in each group.

**RAID 60** – RAID 0 striping combination across multiple RAID 6. RAID 60 is the equivalent of RAID 50 with a higher level of fault tolerance. The group size is at least 4 drives. Recoverable from 2 failures in each group.

**RAID 70** – RAID 0 striping combination across multiple RAID 7. RAID 70 is the equivalent of RAID 60 with a higher level of fault tolerance. The group size is at least 4 drives. Recoverable from 3 failures in each group.

## Managing RAIDs

The section describes a RAID management actions and the RAID state transitions initiated by a command or some event. xiRAID engine supports many RAIDs in parallel. The RAIDs and their life cycles and states are independent from each other.

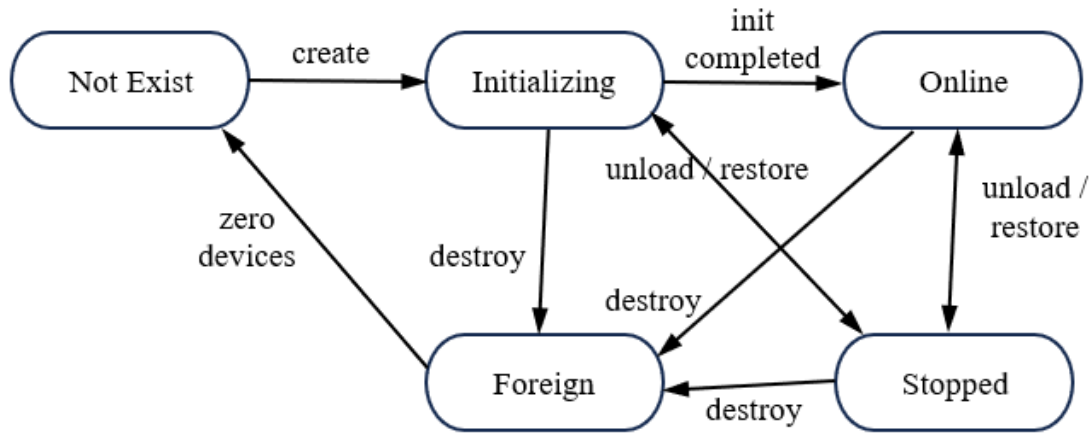


Figure 7: RAID lifecycle

After the RAID is created, it enters the initializing phase. During this phase, the drives are initialized, checksum and parity bits are built. During the initializing phase the RAID is operational, but performance may not be optimal. Once the initialization process is complete, the RAID transitions to a fully operational state.

When one or more drives fail, the RAID system automatically switches to a degraded state. To restore it, the failed drive needs to be replaced. After replacing the drive, the user should start the process of reconstructing data on the new drive. During this rebuild process, performance and redundancy remain degraded until the rebuild is completed.

A RAID can be unloaded into the stopped state, where it still exists but does not perform IO and is not shown in the list of RAIDs. Drive replacement is not possible in this state.

A RAID can be destroyed and go into the foreign state from any other state.

## Create

The RAID lifecycle begins from the “raid create” command. Before creating the RAID you have to make sure that the corresponding devices are attached or attach them by the Device Manager operation if needed.

Drives must be cleared before they can be used in a RAID. If the first 4 kilobytes of a drive contain any data, the drive cannot be used in a new RAID. This prevents the user from using a drive that may contain important data in a RAID. For example, a drive used in an unloaded RAID. If drives contain data, they can be cleared by using the following command:

```
xnr_cli bdev zero --names
  43E0A004TLZJn1,43F0A00ATLZJn1,43T0A002TLZJn1,
  43U0A01BTLZJn1,43U0A01CTLZJn1,43U0A01JTLZJn1
```

BDEV names can be obtained from the output of the `device-manager show` command.

To create a RAID use the command:

```
xnr_cli raid create --name <name> --level 5 --chunk-size 64
  --block-size 4096
  --bdevs 43E0A004TLZJn1,43F0A00ATLZJn1,43T0A002TLZJn1,
  43U0A01BTLZJn1,43U0A01CTLZJn1,43U0A01JTLZJn1
```

**level** can be one of 0, 1, 5, 6, 7, 10, 50, 60, 70.

**block-size** can be 4096 or 512 bytes. It defines the RAID device minimal IO size. If underlined drives (bdevs) use 4096 block size the RAID block size 512 is not allowed.

**chunk-size** specifies each drive data chunk size in Kbytes in scope of a single RAID stripe. For example, RAID 5 on 8 drives with chunk size 64 has full stripe size equal to  $64 * 7 = 448$  Kbytes. Chunk size should be selected based on client IO data pattern and backend device firmware logic to optimize IO performance and latency.

**uuid** is an optional parameter. It is recommended to provide UUID to identify the RAID in big installations. If UUID is not provided it is autogenerated during RAID create operation and is reported by "raid show" operation. The UUID can be used together or instead of "name" parameter in any RAID related operation to identify the RAID. "name" is useful for manual input while UUID is better for scripting.

Use the following command to display information about RAIDs managed by xiRAID Opus:

```
xnr_cli raid show
```

To configure rate limits for input/output operations on the created RAID volume, see [Managing BDEV IOPS and Throughput Limits](#).

## Unload

If the RAID is not needed at the moment, you can unload it by using the following command:

```
xnr_cli raid unload --name <name>
```

The unload operation switches the RAID to the Offline state, starts to reject new incoming IO requests, waits for outstanding IO completion, gracefully interrupts internal services such as initialization and reconstruction, clears the auto-start flag and stops the RAID. The Stopped RAID is not reported by the "raid show" command. Although, it is configured (persisted in the configuration database), it is not automatically restarted if the engine or the server is restarted.

## Restore

If the RAID is Stopped, it can be loaded back to Operational by the command:

```
xnr_cli raid restore --name <name>
```

The restore operation reads the RAID configuration from persisted database, lookup the RAID drives, checks the drives' metadata, connects as many drives as possible, attempts to bring the RAID to the Online state and sets auto-start flag.

If successful, user's IO is enabled and initialization or resync continue. Otherwise the RAID remains Offline until missing drives are attached using the "raid replace" command. Once the number of connected drives is sufficient for RAID operation, the RAID is switched to the 'Online' state and starts supporting user IO.

Sufficient drives number means set of available drives at appropriate logical positions to allow RAID logic read and write data considering parity areas. For example, RAID 60 can be restored if fewer than 2 drives are unavailable in each of the RAID group.

During the raid restoration, the initialization process is restarted automatically, if needed. In case of emergency RAID stop (engine or the storage hardware powered down or restarted unexpectedly) the RAID is marked as possibly corrupted and the initialization process is restarted automatically to fix possible "write holes" at next RAID restore or automatic RAID restart. This process is referred to as "resync".

## Destroy

The RAID can be stopped and removed from the engine runtime and configuration database using the command:

```
xnr_cli raid destroy --name <name>
```

The destroy operation unloads the RAID if it is running (this step is similar to the unload operation), and then removes the RAID from the persisted configuration database. Destroyed RAID is considered as foreign by the xiRAID Opus engine.

Destroying the RAID does not remove user data or RAID metadata from the RAID drives.

If the destroyed RAID drives are going to be used in a new RAID, the drives metadata area should be cleaned by the `bdev zero` command or by any other way that clears the first 4 Kbytes of the drives.

## Import

The RAID import procedure may be required after moving drives from one system to another after a system failure, as part of a server hardware upgrade, or for a different reason. If you've moved drives from one system to another, it's necessary to attach these drives to the device manager before you can import the RAID. For information on how to attach drives to the drive manger, refer to [this document](#).

To import a RAID into xiRAID Opus, use the `raid import` command.

```
xnr_cli raid import --bdevs <bdev1,bdev2>
```

1. Use the `--bdevs` option to specify a comma-separated list of BDEVs that were used in the old RAID.
2. Optionally, specify the name (`--name`) of the old RAID. Note that the name or UUID must match the name of the old RAID exactly.
3. If a RAID with the same name as the old RAID already exists, use the `--rename` option to specify a new name for this RAID.
4. If you don't have all of the drives that were used in the old RAID, use the `--allow_incomplete` flag. Note that if the number of drives is not sufficient to rebuild the RAID, the RAID will not be imported.

To start the reconstruction process after importing the RAID, **replace** the missing drives and start the RAID reconstruction process.

## Moving RAIDs to Another Server

There are several situations where it may be necessary to move RAID drives from one system to another. Depending on the circumstances, the steps involved can vary. This topic outlines two common scenarios: planned transfers and emergency transfers.

## Planned RAID Drive Transfer

To move a RAID from one system to another when the original system is still operational, follow these steps:

1. **Unload** the RAID to switch the RAID to the offline state, stop accepting new IO requests, wait for pending IO to complete, and then shut down the RAID. This command does not delete any data on the drives.
2. If necessary, **uninstall xiRAID Opus**.
3. Remove the RAID drives from the current system and insert them into another system with xiRAID Opus. Make sure to install a license on the new system before proceeding to the next step.
4. **Attach** the RAID drives to the device manager.
5. **Import** the RAID.
6. On the old system, **Destroy** the RAID.
7. On the old system, **Detach RAID drives** from the device manager.

## Emergency RAID Drive Transfer

If a hardware failure occurred on the current system, you can restore your RAID on a different system, provided that at least some drives are functioning and there are enough working drives to rebuild the RAID.

1. Remove the RAID drives from the old system and insert them into another system with xiRAID Opus. Make sure to install a license on the new system before proceeding to the next step.
2. **Attach** the RAID drives to the device manager.
3. **Import** the RAID.
4. If necessary, **replace the failed drives** in the RAID.

## RAID Import Limitations

- Only one RAID can be imported at a time. If BDEV identifiers from different RAIDs are specified in a single `xnr_cli raid import` command, only the RAID containing the first BDEV in the list will be imported.
- BDEV IOPS and throughput limits, Vhost, and NVMe-oF configurations cannot be restored.
- If you've imported a RAID without some of the drives that were used in the original RAID, xiRAID Opus will be able to start the reconstruction process as long as the number of drives is sufficient to rebuild the RAID. If you later attach the missing drives, it will have no effect on the reconstruction process.

## Drive Failure and Degraded Mode

A RAID (except for RAID 0) supports user IO if one or more drives have failed or been disconnected. As soon as a drive cannot be used for IO, its state is changed to 'Offline' and the RAID state is changed to 'Degraded'. If too many drives fail and the RAID cannot support IO, its state is changed to 'Offline'. When the RAID goes 'Offline', the engine logic immediately interrupts user IO to minimize potential data damage. The RAID engine ensures data consistency except for IO packets being processed when the RAID goes 'Offline'. These IO requests are marked as failed to the client, indicating that the data was not written. The corresponding data areas on drives may contain "old" data, "new" data or a mix of old and new data if the IO packet size is larger than a single RAID chunk size and spans multiple drives.

RAID stripes impacted by failed I/O data requests may suffer from damaged parity consistency, especially if multiple drive failures occur during a write operation. In such cases, some stripe chunks may be updated while others, including the parity area, remain outdated or inconsistent. Additionally, if a system or RAID engine failure occurs during a write request, it may result in a write hole, a form of silent data inconsistency that can remain undetected.

If the RAID is marked as 'Degraded', follow these steps:

1. Check if one the drives in the RAID has failed. If a drive has failed, replace it with a new one of the same or larger size.
2. If a drive is replaced by a new one, clear it by using the `xnr_cli bdev zero --bdev <bdev_name>` command.
3. Replace the failed drive by using the command `xnr_cli raid replace --name <name> --position 1 --bdev 0000:06:0b.0n1`. The drive state reported by the `raid show` command should change to Online.
4. If RAID is Offline, unload the RAID and restore it using the commands:

```
xnr_cli raid unload --name <name>
xnr_cli raid restore --name <name>
```

5. Start the RAID reconstruction to recover data from temporary disconnected drives. The reconstruction process also fills new drives with actual data. Use the command:

```
xnr_cli raid recon start --name <name>
```

If there is no IO in progress and drives are disconnected and the same drives are reinserted back, the RAID can switch from a Degraded (or Degraded and Offline) state back to Online without the need for reconstruction.

## Replace Drive

The replace drive operation substitutes the current drive in the RAID logical position with another drive or inserts a drive to the logical position if there is no Online drive at that position.

To replace a drive in a RAID array, it's important to first **attach** the drive to xiRAID Opus and **clear the metadata** on the drive.

To replace a drive in a RAID, use the command:

```
xnr_cli raid replace --name <name> --position 1 --bdev
0000:06:0b.0n1
```

where "position" is the drive number reported by the "raid show" command and "bdev" is the drive name reported by the "device-manager show" command.

The drive replacement logic initiated by the “raid replace” command involves the following steps:

1. If an Online drive is at the specified position, the drive is marked as Offline.
2. Wait for any outstanding IO operations to the drive to be completed.
3. Close the BDEV corresponding to the RAID logical drive at the specified position.
4. Open the BDEV specified by the 'bdev' option of the command.
5. Connect the specified BDEV as the RAID logical device at the specified position.
6. Read the connected drive metadata to determine if the drive is brand new, a previously used drive at the correct position, a foreign RAID drive, or a known drive inserted at an incorrect position.
7. Update the connected drive metadata to synchronize it with the latest RAID metadata.
8. Enable the drive for user IO
9. Mark the drive as Online or Degraded depending on the data state

A new drive is always marked as Degraded. A previously installed drive can be marked as Online if there has been no IO activity between the removal and reinsertion events (no data has changed on the drive), and marked as Degraded otherwise.

If the drive specified by “bdev” option cannot be opened or is foreign or inserted incorrectly, the replace operation will fail with the corresponding message. The previous drive will not be reinserted if the operation fails

For debugging and experimental purposes, it is possible to replace a drive with nothing, effectively removing a logical RAID drive without replacing it. Logically the operation is equal to the physical drive removal but without actual removing and detaching the physical device. However, this operation is useless in a production environment. It can be run by the following command

```
xnr_cli raid replace --name <name> --position 1 --bdev null
```

The replace operation is needed in the following cases:

- A drive fails and the corresponding device is damaged and must be replaced by another physical device reinserted in the same or a different physical slot.
- A device is unstable and needs to be proactively replaced by another physical device in the same or a different physical slot.
- A device or several devices are disconnected due to hardware failure, such as a cable disconnect or drives controller failure. In this case, the disconnected drives should be reconnected (replaced with same drives used before) after the hardware problem is fixed.
- A device or several devices need to be migrated to another physical slot(s) due to a controller failure or another reason. This may involve changing the PCIe address and device name. In this case, drives should be reconnected (replaced with the drive used before in the same positions).

#### Drive replace recommendations:

- If a device changes its PCIe address and the drive name is changed, the drive UUID has to be used to identify which drive should be reconnected to which RAID logical position. The UUID is persisted in the RAID configuration and in the drive metadata (misc. area at the drive). The UUID is reported by the “device-manager show” and by the “raid show” commands to establish a connection between a physical device’s drives (reported by device manager) and logical RAID drives (reported by the “raid show” command).
- If the connection is lost for any reason and you forget the corresponding logical slot, you can try reinserting the drive to any Offline position. The replace logic checks the UUID and blocks inserting the drive to incorrect slot (except when inserting a new drive with zeroed metadata area). Therefore, you can safely try to reinsert the drive into each Offline position one by one until the operation is successful.

## Device Roaming

The logical position of a RAID drive is not rigidly tied to its physical device location (slot). For example, if the RAID consists of 8 drives you can swap drives 1 and 2 between physical slots without affecting RAID’s logical drive positions. Additionally, all or some drives may change their physical location (PCIe address) due to hardware reconfiguration or PCIe controller replacement. It is also possible to insert a new drive into a different physical slot than the one being replaced in the RAID array.

There are two ways to change a drive's physical location. For an operational RAID the roaming process can be executed without stopping IO if the total number of simultaneously removed and reinserted drives is less than or equal to the allowed amount by the RAID level. Follow these steps:

1. Remove the drive from the slot. It is recommended to detach the drive using the device manager before removal.
2. Insert the drive into another slot and attach it using the device manager.
3. Reinsert the drives into the same logical RAID position using the "raid replace" command.
4. Run reconstruction if the reinserted drive and the RAID are marked as need reconstruction (need\_recon).
5. Wait for reconstruction to complete, and then repeat the steps to move other drives.

If there is a chance to stop IO the safer way to migrate drives to another slots is:

1. Unload the RAID.
2. Detach drives from the device manager.
3. Move devices between physical slots as needed.
4. Attach devices using the new slots' PCIe addresses.
5. Restore the RAID. The RAID should reconnect all drives automatically, with no need for replacement or reconstruction.

## RAID Service Operations

xiRAID Opus automatically handles operations such as initialization, reconstruction, or resynchronization when necessary. Users typically do not need to start or stop these maintenance operations manually, although it is possible to do so.

To start RAID maintenance, users can run the `xnr_cli raid service start` command:

```
xnr_cli raid service start -n <raid_name>
```

Note that it is not possible to manually specify which RAID maintenance operation (Initialization, Reconstruction, or Resynchronization) to perform, nor can you control the order in which these operations occur. The system automatically detects the required operations and executes them in the correct sequence.

However, it is possible to manually initialize a RAID resynchronization. This may be necessary if you suspect data consistency or redundancy across the drives in a RAID array has been compromised (e.g., after a power loss), or as part of preventative maintenance. To start a RAID resync, run the following command:

```
xnr_cli raid service force-resync -n <raid_name>
```

To stop all maintenance operations, run the `xnr_cli raid service stop` command:

```
xnr_cli raid service stop -n <raid_name>
```

Below are descriptions of the initialization, reconstruction, and resynchronization processes.

## Initialization

The raid create operation adds the RAID configuration to the persisted database and sets auto-start option to the RAID. The RAID is available for user's IO as soon as the create operation completes. If the engine is restarted or the node is rebooted the RAID is automatically restored to the operational mode (if the required number of RAID drives is attached and available). Auto-starting the RAID flow is similar to the RAID restore operation.

After the RAID is created, the engine starts initializing drives. During the initialization process, RAID parity checksums are calculated and written to the parity area of the drives. The RAID supports user's IO while initialization is in progress. If IO goes to uninitialized areas, the parity is being calculated on-demand. Once initialization is completed, the RAID transitions to a fully operational state.

The initialization process impacts IO performance. Without full RAID initialization, consistency checks like silent data corruption detection can't be performed, as uninitialized areas may appear corrupted. Uninitialized areas do not impact the consistency of already written data or the reconstruction process. It is recommended to wait until initialization is finished before using the RAID for user's IO.

For testing purposes, it is possible to force a pseudo-completion of the initialization process. Performing this operation is not recommended, as it can result in an inability to recover data and may lead to data loss or corruption.

To force a pseudo-completion of the initialization process, run the following command:

```
xnr_cli raid service init-force-finish -n <raid_name> --force
```

## Reconstruction

If a RAID backend drive fails and is replaced by a new drive or the drive is temporarily disconnected and reconnected back after some time the drive data may become invalid and will be reconstructed from other drives data using RAID recovery algorithms.

A RAID data area is logically divided to slices. If a drive failed or was removed, the xiRAID Opus logic remembers what slices are modified by user IO. If the same drive returns to the RAID, this information allows us to identify which slices of the drive are impacted and need reconstruction, as well as which slices of data remain unchanged. If some slices are impacted only the partial recovery algorithm is used for reconstruction. The partial reconstruction logic significantly reduces reconstruction time. This is why it's important to attempt to reconnect a temporarily disconnected drive instead of replacing it with a new one. In case of replacing the drive with a new one all slices are marked as impacted and full reconstruction logic is used.

Many RAID levels (such as 6 and 7 or group levels 50, 60, 70) operate as if more than one backend drive is disconnected. If disconnected drives are reconnected (or replaced by new drives), the reconstruction can start and recover replaced drives while other drives are still disconnected.

At RAID levels 6 and 7 if several drives are removed and reinserted back one by one some slices of data can be impacted at two or three reconnected drives and other slices of data can be impacted at one drive only. In such case the reconstruction logic recovers the most impacted slices first and less impacted slices next to minimize risk of losing user data.

The reconstruction process can be temporarily stopped, which can be useful in reducing the impact of reconstruction IO flow on the user's IO performance and latency. However, it is not recommended because it increases the risk of data loss, and the user's IO performance and latency are not optimal if the RAID is degraded.

For testing purposes, it is possible to force a pseudo-completion of the reconstruction process. Performing this operation is not recommended, as it can result in an inability to recover data and may lead to data loss or corruption.

To force a pseudo-completion of the reconstruction process, run the following command:

```
xnr_cli raid service recon-force-finish -n <raid_name> --force
```

## Resync

A RAID consists of a set of stripes. Each stripe has several data chunks and one or several parity chunks. When writing data to the stripe, the RAID engine updates some data chunks and recalculates the parity chunks resulting in several IOs to the RAID backing device. So, write operations are not atomic. If a write operation is interrupted by xiRAID engine crash, a driver failure, the server OS or hardware restart or a power down, the stripe can be logically broken and the parity chunks may not correspond to the data chunks. This scenario is known as a “write hole”.

When multiple threads are running IO operations with some IO depth, many stripes can be impacted by such interruptions. Interrupted IO operations can either be completed with an error or remain incomplete. The RAID logic has no mechanism to recover data from the “old” or “new” state and the client should perform data recovery if an IO operation is completed with an error or times out.

The RAID logic is designed to recover broken stripes' parity to restore the stripe to operational state, preparing for potential drive failures and reconstruction of the broken and replaced drive.

xiRAID Opus engine detects unexpected RAID down events and marks such RAIDs as dirty at next restart. The “resync” logic is implemented to locate and repair all broken stripes by recovering their parity. If a RAID is 'dirty,' the initialization restarts automatically during the next RAID restart, unless it is reconstructing or 'Degraded'. The “resync” logic is designed as an additional initialization cycle and the RAID state reports as Initializing if the “resync” is in progress.

The resync logic can be disabled or re-enabled by setting `resync_enabled` configuration parameter to `False` or `True` respectively:

```
xnr_cli config set --name resync_enabled --value False
```

Current resync mode can be displayed using the command

```
xnr_cli config get --name resync_enabled
```

If resync is disabled then a possible dirty RAID event will be lost. In this case the RAID can get broken stipes and this cannot be detected later.

## Operational Modes

A RAID operates in different modes indicated by a set of states. Some states are interrelated, while others can be controlled independently. Changes may occur due to user actions, hardware failures, or internal logic events like initialization completion. The RAID drives have their own states. This drive state is a logical state in scope of the RAID functionality. For example, a drive can be connected to the engine and visible as a BDEV but be offline in terms of the RAID backing drive.

The `raid show` command reports the RAID states, level, size, and other useful information. It also reports the RAID backend drives and their parameters. The show output can be formatted as `--output json` or as three table formats: `table`, `compact`, `tiny`. Below is the `raid show` command output example:

```
$ xnr_cli raid show
```

RAID	PARAMETERS	BDEVs	
test	size: 9.81 GiB level: 1 chunk size: 64 KB block size: 4096 B num blocks: 2572288  service IO priority: 50%  init progress: 1% reconstruct progress: 0%  UUID: f5f9245c-93c3-4ff1-a677-d375f7f825ad	0 nvme.c0a9eb3c13788900.2 online 8de743c3-9735-57c5-ade2-12da13d739e7	type: VirtioBlk Disk size: 10.00 GiB block size: 512 B blocks: 20971520
		1 nvme.c0a9eb3c13788900.3 online 7c503789-827e-5c56-8047-f65fb745e5bc	type: VirtioBlk Disk size: 10.00 GiB block size: 512 B blocks: 20971520

RAID operational modes may also depend on the **states of the BDEV** used in the RAID, as well as the **RAID status**.

## RAID states

A RAID in xiRAID Opus may be in one of the following states:

1. Online - The RAID operates normally and serves user IO.
2. Offline - If the RAID is not online the Offline state is reported. While the RAID is Offline some IO operation to its drives is possible to read or update metadata or service areas while user's IO is blocked.
3. Initializing (initing) - The parity data for the stripes is being initialized. This initialization process runs once after the RAID is created or in the event of an emergency stop and stripes resync.
4. Need Initialization (need\_init) - The initialization of stripes is incomplete, and the initialization process is not running.
5. Initialized - Stripes parity areas are initialized.
6. Degraded - One or more RAID backing drives have failed. The RAID may or may not support user IO (it can be Online or Offline) depending on number of failed drives.
7. Need Reconstruction (need\_recon) - A drive has failed and has been replaced with the same (reinserted and reconnected) or another drive. Data reconstruction for the drive is required.
8. Reconstructing - The RAID reconstruction is in progress.
9. Unrecoverable - If multiple drives fail and become inaccessible (cannot be reinserted or reconnected), the RAID stripes may lack sufficient parity information to support IO and recover lost data. In such cases, manual recovery is necessary, and there is a risk of data loss or corruption.

In addition to states listed above and reported by the `raid show` command, the RAID can be in the following states:

1. Not Exist - The RAID with this name, UUID and user data does not exist (was never created).
2. Stopped - The RAID IO is stopped, and the RAID configuration object is removed from the engine runtime. User data is retained on the RAID drives, and the RAID configuration is stored in the engine configuration database. User data is consistent in case of a normal RAID shutdown but can be inconsistent in case of an emergency shutdown.
3. Foreign - The RAID is stopped, and the RAID configuration is removed from the configuration database. Alternatively, the RAID is created by another RAID

engine (compatible with xiRAID or incompatible with xiRAID). Such a RAID cannot run using the engine, but it retains user data. Future releases will be able to import foreign RAID if its format is compatible.

If the RAID is running and can support IO it is labeled as **Operational** in the document. If the RAID cannot support IO, it is named as **Failed**. The **Operational** and **Failed** states are not reported by the API as they are considered integral states. Typically, a RAID is considered Operational if its state is Online. However, the RAID can be labeled as Failed if it is Online but Unlicensed.

## BDEV Logical States

BDEV useds in RAIDs may be in one of the following states:

1. Online - The device is available and operating normally.
2. Offline - The drive is currently not online and not participating in user IO. The offline device can be connected and made available for low-level IO. While the RAID metadata can be read from or written to the offline drive, it is not recognized as being logically ready for the RAID user's IO.
3. Need Reconstruction (need\_recon) - The drive data area is partially or fully incorrect and needs reconstruction from other drive data and parity. The drive can still participate in user IO operations when it requires reconstruction.
4. Reconstructing - The drive data is being reconstructed. The drive can participate in user IO while reconstructing.

## Managing Partitions

In xiRAID Opus, partitions may be used to logically divide an individual BDEV or a RAID for better management and organization. Each partition can function as an individual storage unit with its own file system.

In scenarios where xiRAID Opus is installed on a hypervisor, partitions can be exposed to virtual machines through the VirtIO transport mechanism. This enables administrators to expose partitions from a xiRAID Opus RAID to the virtual machines on the hypervisor, allocating only the necessary storage to each virtual machine. This approach is much more cost-effective than creating a separate RAID for each virtual machine. For instructions on how to expose partitions through the VirtIO transport mechanism, refer to [Configuring Vhost Targets](#).

Partitions can also be exposed as NVMe-oF targets, providing a way to access and manage data remotely. This may be useful when quick access to large volumes of data is required over the network. For instructions on how to expose partitions over NVMe-oF, refer to [Working with NVMe over Fabrics Targets](#).

## Creating Partitions

Partitions can be created from any BDEV managed by xiRAID Opus. This includes local physical devices, network devices, and xiRAID OpusRAIDs.

Partitions are created by using the `bdev partition add` command. To create a partition, you must first obtain either the name or UUID of the BDEV to divide into partitions. This information can be found in the output of the `bdev show` command. To create a partition, use the `bdev partition add` command:

```
xnr_cli bdev part add -n <bdev_name> -p <partition_size>
```

where:

- `bdev_name` is the name of the BDEV from which partitions will be created.
- `partition_size` is the size of the partition in MiB.

Note that it's possible to create multiple partitions from one BDEV in a single command:

```
xnr_cli bdev part add -n <bdev_name> -p 10000,10000,20000
```

To configure rate limits for input/output operations on the created partition, see [Managing BDEV IOPS and Throughput Limits](#).

## Viewing Partitions

Partitions may be viewed either in the output of the `bdev show` or `bdev partition show` command.

The `bdev show` command displays the list of all BDEVs managed by the device manager. This list includes local physical devices, network devices, xiRAID OpusRAIDs, and partitions. Partition names are usually derived from the BDEV name, with the partition number appended at the end. For example, in the name

`nvme.622f976b98657af3.2p1`, `nvme.622f976b98657af3.2` represents the BDEV, and `p1` indicates the first partition on that device.

To display all partitions created on a specific BDEV, use the `bdev partition show` command:

```
xnr_cli bdev partition show -n <bdev_name>
```

## Deleting Partitions

To remove a specified number of partitions from a BDEV, use the `bdev partition remove` command:

```
xnr_cli bdev partition remove --name <bdev_name> --count  
<number_of_partitions>
```

where:

- `name` the name of the BDEV from which partitions will be removed.
- `count` indicates the number of partitions to remove, starting from the end of the device. For example, if the BDEV contains three partitions and you specify `--count 1`, the last partition will be deleted, leaving the remaining two intact.

To remove all partitions from a BDEV, use the following command:

```
xnr_cli bdev partition remove --name <bdev_name> --all
```

## Configuring Vhost Targets

Any BDEV attached to the device manager in xiRAID Opus can be exposed to a client over the VirtIO transport mechanism. After a BDEV is exposed, a vhost socket is created on the FS in the `<install_dir>/bin/xnr_conf/sock` directory. This vhost socket enables the transfer of data between virtual machines and the BDEV. A vhost target can be created on top of any BDEV and is automatically recreated at engine restart if the corresponding BDEV is operational.

## Creating a Vhost Target

Creating a vhost target is an easy way to expose a xiRAID Opus BDEV (a single device, a RAID, or a partition) on a hypervisor to its virtual machines. To create a vhost target, use the **vhost create** command:

```
xnr_cli vhost create -n <vhost_name> -d <bdev> -c <socket file name>
-m <cpu_mask>
```

where:

- `-n/--name` is the name for this vhost target.
- `-d/--bdev` is the name of the BDEV to expose through the VirtIO transport mechanism. Any BDEV managed by the device manager may be exposed. This includes local physical devices, network devices, xiRAID OpusRAIDs, and partitions.
- `-c/--ctrl` is name for the socket file. For example, `vhost.1.socket`.
- `-m/--cpumask` defines which CPU cores will be used to process IO requests for this vhost target. `--cpumask` must be a subset of the xiRAID Opus CPU mask. Use the `xnr_cli config get --name cpu_mask` command to check the xiRAID Opus CPU mask. Different vhosts can use different, the same or intersecting masks. See [Installing xiRAID Opus](#) for details on how to set the `cpumask` parameter.



To destroy a RAID exposed as a vhost target, first power off the virtual machine that is using this vhost target.

## Viewing Vhost Targets

To view details about all configured vhost targets, use the **vhost show** command:

```
xnr_cli vhost show
```

Example output:

VHOST	PARAMETERS
vhost.1 1173eb85-22e8-40a3-a50e-9b0d0d780d08 started	bdev: testp1 8f612676-f417-4704-8e43-6840635d8d24 socket: /opt/xiraid/bin/xnr_conf/sock/vhost.1 mask: 0xf cores: (0-3)
vhost.2 258fe3d5-7440-4ac3-9adc-f83f673c5b60 started	bdev: testp2 4e6799b0-8ed1-456c-b185-51296f0d6a3a socket: /opt/xiraid/bin/xnr_conf/sock/vhost.2 mask: 0xf cores: (0-3)

To view all available information about a vhost target, run the `vhost show` command with the `-o json` option. Additionally, you can use the `--name [-n]` option to display information for a specific vhost target only.

```
xnr_cli vhost show -o json -c vhost.1
```

The output of this command contains the following information:

```
{
  "vhosts": [
    {
      "id": {
        "name": "vhost.1",
        "uuid": "1173eb85-22e8-40a3-a50e-9b0d0d780d08"
      },
      "operating_state": "OPER_STARTED",
      "cpu_mask": "0xf",
      "virtio_type": "BLOCK",
      "block_specific": {
        "bdev_id": {
          "name": "testp1",
          "uuid": "8f612676-f417-4704-8e43-6840635d8d24"
        }
      },
      "params": {
        "socket": "/opt/xiraid/bin/xnr_conf/sock/vhost.1"
      }
    }
  ]
}
```

- `id.name` is the name of the vhost target.
- `id.uuid` is the UUID of the vhost target.
- `operating_state` is the current state of the vhost target.
- `cpu_mask` defines which CPU cores are used to process IO requests for this vhost target.
- `virtio_type` indicates the type of this target.
- `block_specific.bdev_id.name` is the name of the BDEV exposed as a vhost target.
- `block_specific.bdev_id.uuid` is the UUID of the BDEV exposed as a vhost target.
- `params.socket` is the socket name to be used by virtual machines to connect the BDEV.

## Deleting a Vhost Target

To delete a vhost target, use the `vhost destroy` command:

```
xnr_cli vhost destroy --name <target_name>
```

Consider disconnecting virtual machines from this vhost target before deleting it.

## Working with NVMe over Fabrics Targets

### Configuring an NVMe-oF Target

#### NVMe-oF Overview

NVM Express (NVMe) is a storage access and transport protocol designed for flash and solid-state drives.

NVMe over Fabrics (NVMe-oF) extends the NVMe interface to support remote access over a network. It uses either the Message or Message/Memory transport models to enable this, and supports various transport protocols. Configuring NVMe-oF targets in xiRAID Opus allows users to expose any BDEV over the network.

Below are some important NVMe-oF terms that will be used in this section:

- The **Target** is the NVMe-oF storage system that provides access to NVMe namespaces. It responds to requests from Clients and exposes the storage resources over a fabric.
- The **Client**, also known as the Host or Initiator, is the system that connects to the Target to access storage over the network.
- A **Subsystem** is a logical grouping of NVMe components, including one or more namespaces and controllers. It organizes and manages access to storage resources, presenting them to Clients.
- A **Namespace** is a block of non-volatile memory (like SSD storage) within a subsystem. It is the actual data storage unit that Clients read from or write to. In xiRAID Opus, it can be a single device, a RAID, or a partition.
- A **Transport** in xiRAID Opus defines the transport protocol used to transfer data between Clients and Targets. Currently, TCP and RDMA transports are supported.
- An **NVMe-oF Port** is a network endpoint used by either the Client or the Target to communicate over the selected transport.
- Asymmetric Namespace Access (**ANA**) is a method used by NVMe-oF targets to set path priorities by assigning namespaces to specific groups and defining states for those groups on different ports.
- An **ANA group** is essentially a collection of namespaces that share the same access properties.
- An **ANA state** determines the priority of a given ANA group on a specific NVMe-oF port.

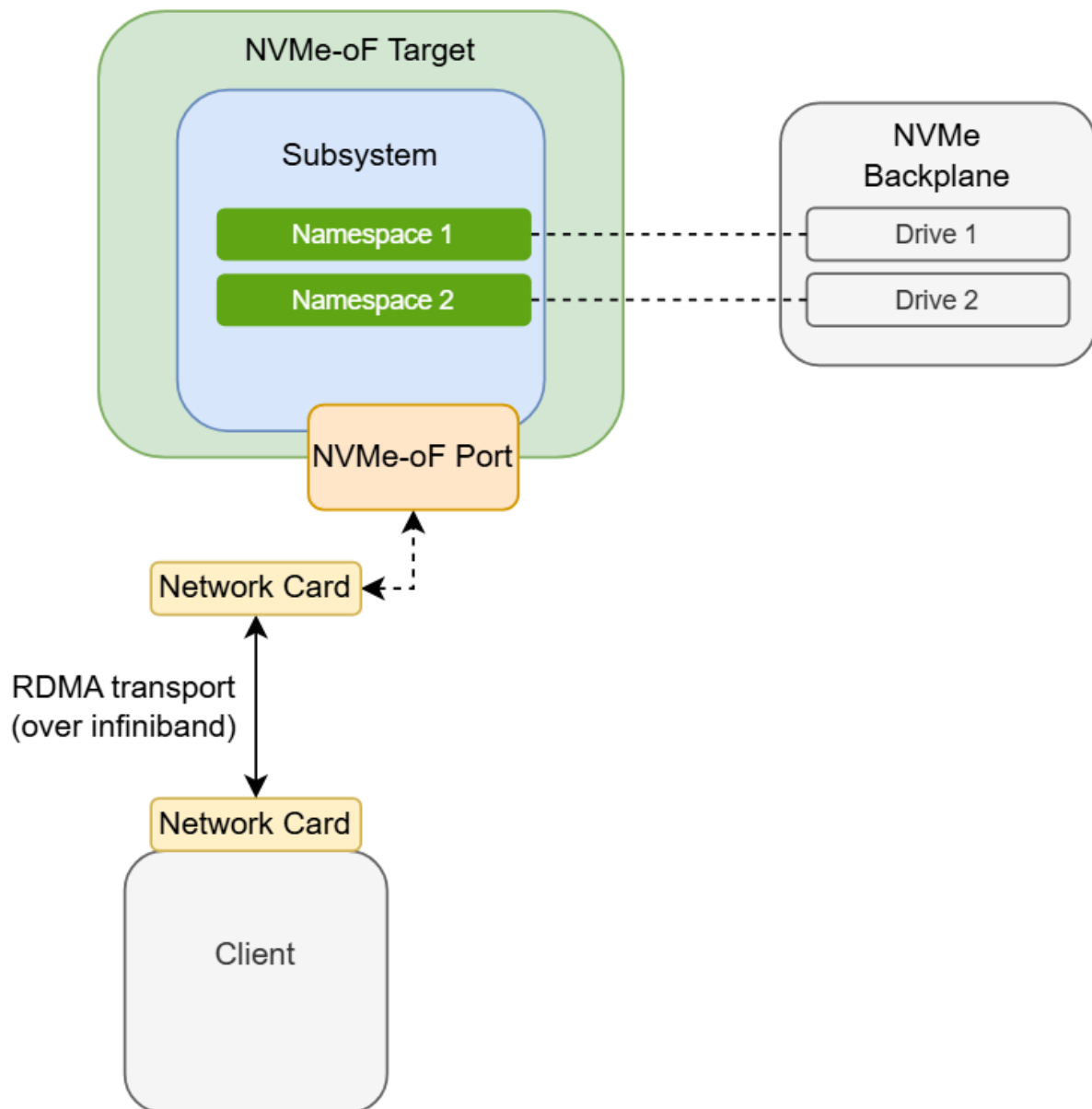
xiRAID Opus can be used both as a Target and a Client. The following section describes how to configure an NVMe-oF Target using xiRAID Opus. For instructions on how to connect to an NVMe-oF Target using xiRAID Opus, refer to [Managing Network Devices](#).

## Configuration Overview

The process of configuring an NVMe-oF Target using xiRAID Opus involves the following steps:

1. **Creating a Transport:** The first step is to create a transport (either TCP or RDMA) that facilitates communication between the NVMe target and the client system.
2. **Creating a Subsystem:** Once the transport is established, a subsystem is created. This acts as a container for the namespaces and links to the network address and port for connection.
3. **Configuring a Network Address:** Before connecting the target to the client, a network address (IP address) must be configured for communication.
4. **Creating an NVMe-oF Port:** A port is created to link the transport, network address, and subsystem together, enabling the system to accept connections.
5. **Creating Namespaces:** After setting up the transport, subsystem, and port, namespaces are created by attaching drives to the system and associating them with the created subsystem. These namespaces are essential for data storage and access.
6. **Configuring a Host:** Add the system you will use to connect to this target to the known list to ensure it can access the exposed subsystem. Alternatively, set the host-access mode for this subsystem to public to allow all hosts to connect to it.
7. **Establishing a Connection Between Client and Target:** The final step involves connecting the client to the target system. At this stage, the system is ready for communication and data transfer.

A diagram describing the resulting configuration of the setup is provided below.



## Prerequisites

To configure NVMe-oF (NVMe over Fabrics) targets and ensure proper communication between the target and client, the following prerequisites are essential:

### 1. Firewall Configuration

- **On the Target:** The firewall must be configured to allow incoming traffic on port 4420, which is the default port used for NVMe-oF communication.
- **On the Client:** The firewall must also be configured to allow outgoing traffic to port 4420 on the target. This ensures the client can connect to the target and access the NVMe namespaces.
- Both systems (target and client) must allow traffic on this port in both directions to ensure that the NVMe-oF protocol functions properly.

## 2. MTU (Maximum Transmission Unit) Configuration

- **Consistent MTU Values:** The Maximum Transmission Unit (MTU) must be set to the same value on both the target and the client systems. This is crucial for ensuring that network packets are transmitted correctly between the two systems without fragmentation.
- **Adjustable MTU Size:** While there is no fixed limit on the MTU value, it must be explicitly set to the same value on both systems to ensure compatibility and optimal performance.
- **Impact of MTU Settings:** A mismatch in MTU sizes between the client and the target could lead to inefficient data transmission or loss of communication, so both devices must have matching MTU configurations.

## Creating a Transport

The first step in configuring an NVMe-oF setup is to create a transport. xiRAID Opus currently supports TCP and RDMA transports.



Each transport type can have only one instance. This means you cannot create more than one TCP or RDMA transport, but you can have one TCP transport and one RDMA transport.

To create a TCP transport, run the following command:

```
xnr_cli nvme transport create -n transport0 -t TCP -i 512KB
```

The `max_io_size` parameter defines the maximum supported I/O operation size. It is specified using the `-i` flag and should be configured based on the geometry and stripe size of the RAID array(s) exposed over the transport. The value may be specified in bytes or using size suffixes such as `KB` (kilobytes) or `MB` (megabytes).



For example, in a RAID 5 configuration with 5 disks and a stripe size of 128 KiB, the `max_io_size` parameter should be set to 512 KiB. Since RAID 5 uses a parity stripe along with data stripes, the number of data stripes is typically 4 (out of 5 disks). To accommodate this, the `max_io_size` should be set to 128 KiB × 4 (the number of data stripes), which equals 512 KiB.

If multiple RAID arrays are exposed over the same transport, the `max_io_size` should be configured to accommodate the largest I/O size requirement across all arrays. Ensure that the parameter value is large enough to support the largest RAID configuration in use.

For details about other available options, refer to [transport \(tr\)](#).

To verify that the transport was created successfully, run:

```
xnr_cli nvme transport show
```

Below is an example of the expected output:

TRANSPORT	PARAMETERS
<pre>transport0 fac0f11f-bcf8-42d4-a6c8-3563644645a6 started type: TCP</pre>	<pre>queue depth: 128 max io size: 524288  num bufs:      511 reserved bufs: 95  zcopy: true abort timeout (sec): 1</pre>

## Creating a Subsystem

The next step is to create a subsystem that will act as a container for namespaces created in the following sections and will be linked to a network address and port that will be used to establish connections between the target and client.

To create an NVMe-oF subsystem, run the following command:

```
xnr_cli nvme subsystem create -n subsystem0 -q
nqn.2025-04.io.xinnor:cnode1
```



By default, the subsystem's host-access mode is set to *limited*, which means only hosts on the subsystem's allowed hosts list can connect. To allow all hosts to connect, add the `--access public` option to the command. For instructions on how to configure host-access, refer to [Configuring NVMe-oF Host Access](#).

In the command above, `nqn.2025-04.io.xinnor:cnode1` is the NQN to use for this subsystem. To specify an NVMe-oF NQN, follow this format: `nqn.yyyy-mm.domain_name:unique_string`. The format starts with the string `nqn`, followed by a date code in the `yyyy-mm` format, which reflects a time when the naming authority owned the domain name. After another period, include the reverse domain name of the authority creating the NQN. Finally, append a colon and a unique string assigned by the domain owner.

For example, `nqn.2014-08.com.example:nvm:nvm-subsystem-sn-d78432` or `nqn.2014-08.com.example:nvme.host.sys.xyz` are valid NQNs. The naming authority is responsible for ensuring that the NQN is globally unique. For more details, refer to the NVM Express Base Specification, Chapter 4.5, "NVMe Qualified Names."

Additionally, you can specify the serial number, model number, and the maximum number of spaces for this subsystem. For more details, run `xnr_cli nvme subsystem create -h`.

To verify that the subsystem was created successfully, run:

```
xnr_cli nvme subsystem show
```

Below is an example of the expected output:

SUBSYSTEM	PARAMETERS	NODES
subsystem0 9ff709a2-1489-470e-b23c-dfafa1fcb581  nqn.2025-04.io.xinnor:cnodel  state: <b>started</b> serial: 48BA3A48C19B3CF6 model: XNR bdev Controller	security mode: SM_LIMITED max namespaces: 32  ANA group ids: (none)	node1 b2d1bae1-500d-4b50-a792-e1e6b1b95e8b min ctrl id: 1 max ctrl id: 65519

## Configuring a Network Address

Before you can create an NVMe-oF port, a network address must first be configured. This involves specifying the system's IP address and, optionally, the IP family (IPv4, IPv6, or IB).

To configure a network address, run the following command:

```
xnr_cli network address add -n address0 -a 172.16.133.245
```

In this example, `172.16.133.245` represents the IP address of the system. For details about other available options, refer to [address add](#).

At this stage, you will not be able to connect to the NVMe-oF target yet. However, you can verify that the network address was configured correctly by running the command:

```
xnr_cli network address show
```

Below is an example of the expected output:

ID	ADDRESS
address0 e4ab39b6-2024-4560-8859-c72f8ce4b348	Family: AFAM_IPV4 Address: 172.16.133.245

## Creating an NVMe-oF Port

To create an NVMe-oF port, run the following command:

```
xnr_cli nvme port add -n port0 -t transport0 -a address0 -s  
subsystem0
```

Where `transport0`, `address0`, and `subsystem0` are the names of the transport, address, and subsystem created in the previous steps. By default, port 4420 will be used for this NVMe-oF port. To use a different port, use the `--svcid/-i` option. For details about other available options for this command, refer to [port \(p\)](#).

If your network card has multiple ports, you can create additional NVMe-oF ports and use the same transport and subsystem as for the first port. In this case, network addresses must be configured for additional NVMe-oF ports.

To verify that the port was created successfully, run:

```
xnr_cli nvme port show
```

Below is an example of the expected output:

PORTS	INFO	ANA GROUPS
port0 4420 OPER_STARTED ANA_OPTIMIZED	uuid: 863e63ce-a5a8-4135-ace4-32790df40f47  subsystem: subsystem0 9ff709a2-1489-470e-b23c-dfafa1fcb581 nqn.2025-04.io.xinnor:cnode1  transport: transport0 fac0f11f-bcf8-42d4-a6c8-3563644645a6 TCP  interface: address0 e4ab39b6-2024-4560-8859-c72f8ce4b348 172.16.133.245	no groups

## Creating Namespaces

In xiRAID Opus, a namespace can be any BDEV (a single device, a RAID, or a partition). To get a list of available BDEVS, run the following command:

```
xnr_cli bdev show
```

To create a namespace, run the following command:

```
xnr_cli nvme namespace add -n namespace0 -s subsystem0 -d  
nvme.a51e40d836541dc6.0
```

Note that for the `-d/--bdev` parameter, you must specify the name or UUID of the BDEV.

To verify that the namespace was created successfully, run the following command:

```
xnr_cli nvme namespace show
```

Below is an example of the expected output:

NAMESPACE	PARAMETERS
namespace0 f0f68895-ea90-4a8a-afb8-9133fc43faa3  <b>started</b> nsid: 1 ANA group id: 1 nquid: e2bfd141e0365ee784665857fa4fa440	backing bdev: nvme.48ba3a48c19b3cf6.0 e2bfd141-e036-5ee7-8466-5857fa4fa440  subsystem: subsystem0 9ff709a2-1489-470e-b23c-dfafa1fcb581 nqn.2025-04.io.xinnor:cnodel

In this example, the **nsid** for this namespace is set to **1**. If the **nsid** is not explicitly specified in the `xnr_cli nvme namespace add` command, it will default to the next available ordinal number. For example, the second namespace will have an **nsid** of **2**.

Other options, such as `ana_group_id`, `nguid`, and `eui64`, can be specified explicitly by using the corresponding command options. For more details, run `xnr_cli nvme namespace add -h`.

To create another namespace and add it to the same subsystem, run:

```
xnr_cli nvme namespace add -n namespace1 -s subsystem0 -d
nvme.a51e40d836541dc6.1
```

Run `xnr_cli nvme namespace show` again to verify that it was created successfully.

## Configuring a Host (Initiator)

Currently, the target does not allow any hosts to connect. By default, the host-access mode for subsystems is set to *limited*, meaning that only hosts on the subsystem's allowed hosts list can connect. The list is currently empty, so you cannot establish a connection. You can either add your host to the xiRAID Opus configuration and include it in the subsystem's allowed host list, or change the host-access mode for the subsystem to *public*.

## Configuring a Host and Adding It to the Allowed Hosts List

To add your host to the xiRAID Opus configuration, run the following command:

```
xnr_cli nvme host add -n host0 --nqn
nqn.2014-08.org.nvmexpress:uuid:3dd2228b-b1c2-4fc2-b5c2-3929ce45e57c
```

In this command, `--nqn` is the NQN of your host. To find the NQN of your host, run the command `nvme show-hostnqn`. If you are using xiRAID Opus as the initiator, run the command `xnr_cli config get -n default_host_nqn`.

Next, to add this host to the subsystem's allowed hosts list, run the following command:

```
xnr_cli nvme subsystem access add -n subsystem0 -d host0
```

## Changing the Subsystem's Host-Access Mode to Public

To allow any host to connect to the subsystem, run this command:

```
xnr_cli nvme subsystem access public -n subsystem0
```

For additional information on configuring NVMe-oF host access, refer to [this topic](#).

## Establishing a Connection Between Host and Target

Now that we've created a transport, subsystem, NVMe-oF port and two namespaces, we can establish a connection between the target and the host.



You can establish a connection without creating namespaces first, or configure the namespaces beforehand. The order of these two operations is not important.

The target is configured to expect connections, so now you only need to connect to it from the client. For example, to connect to the target on a Linux machine, run the following command:

```
sudo nvme connect -t tcp -n nqn.2025-04.io.xinnor:cnodel -a
172.16.133.245
```

To verify a successful connection, run the following command on the host:

```
nvme list -v
```

The output should display information about the target, such as:

```
NVM Express Subsystems

Subsystem          Subsystem-NQN
-----
-----
nvme-subsys0      nqn.2025-04.io.xinnor:cnodel
                                     nvme0

NVM Express Controllers

Device  SN              MN
FR      TxPort Address      Subsystem  Namespaces
-----
-----
nvme0   A51E40D836541DC6  XNR bdev Controller
      24.05   tcp   traddr=172.16.133.147 trsvcid=4420 nvme-subsys0
      nvme0c0n1, nvme0c0n2

NVM Express Namespaces

Device          Generic      NSID      Usage          Format
-----
-----
nvme0n1        ng0n1          1          10.74 GB / 10.74 GB
      512 B + 0 B   nvme0
nvme0n2        ng0n2          2          10.74 GB / 10.74 GB
      512 B + 0 B   nvme0
```

You can also verify the namespaces by checking the output of the `lsblk` command to ensure they are now available to the system.

Note that the configuration made with the `nvme connect` command is not persistent by default. When you use `nvme connect`, it temporarily establishes a connection to the NVMe-oF target for the current session. Once the system is rebooted or the connection is manually removed, it will be lost. To ensure the connection persists across reboots, it is necessary to configure it in a more permanent manner, such as by adding the relevant settings to the system's configuration files.

## Configuring Asymmetric Namespace Access (ANA)

Asymmetric Namespace Access (ANA) is a method used by NVMe-oF targets to set path priorities by assigning namespaces to specific groups and defining states for those groups on different ports. An ANA group is essentially a collection of namespaces that share the same access properties. The ANA state determines the priority of a given ANA group on a specific port. For each ANA group, each NVMe-oF port can be configured with one of the following states:

- **Optimized:** A preferred path that will be used when available.
- **Non-Optimized:** A path that will only be used if the optimized path is unavailable.
- **Inaccessible:** A path that exists but cannot pass any I/O, typically used in high-availability failover situations where traffic should not flow through the backup path.

By default, all namespaces are added to the same ANA group and assigned the **Optimized** state on every NVMe-oF port.

To assign a namespace to a specific ANA group, you must specify the group identifier during the namespace creation. It is currently not possible to change the ANA group of an existing namespace.

In the command below, the `-a` (or `--ana-grp-id`) parameter is used to assign the `namespace0` namespace to the first ANA group.

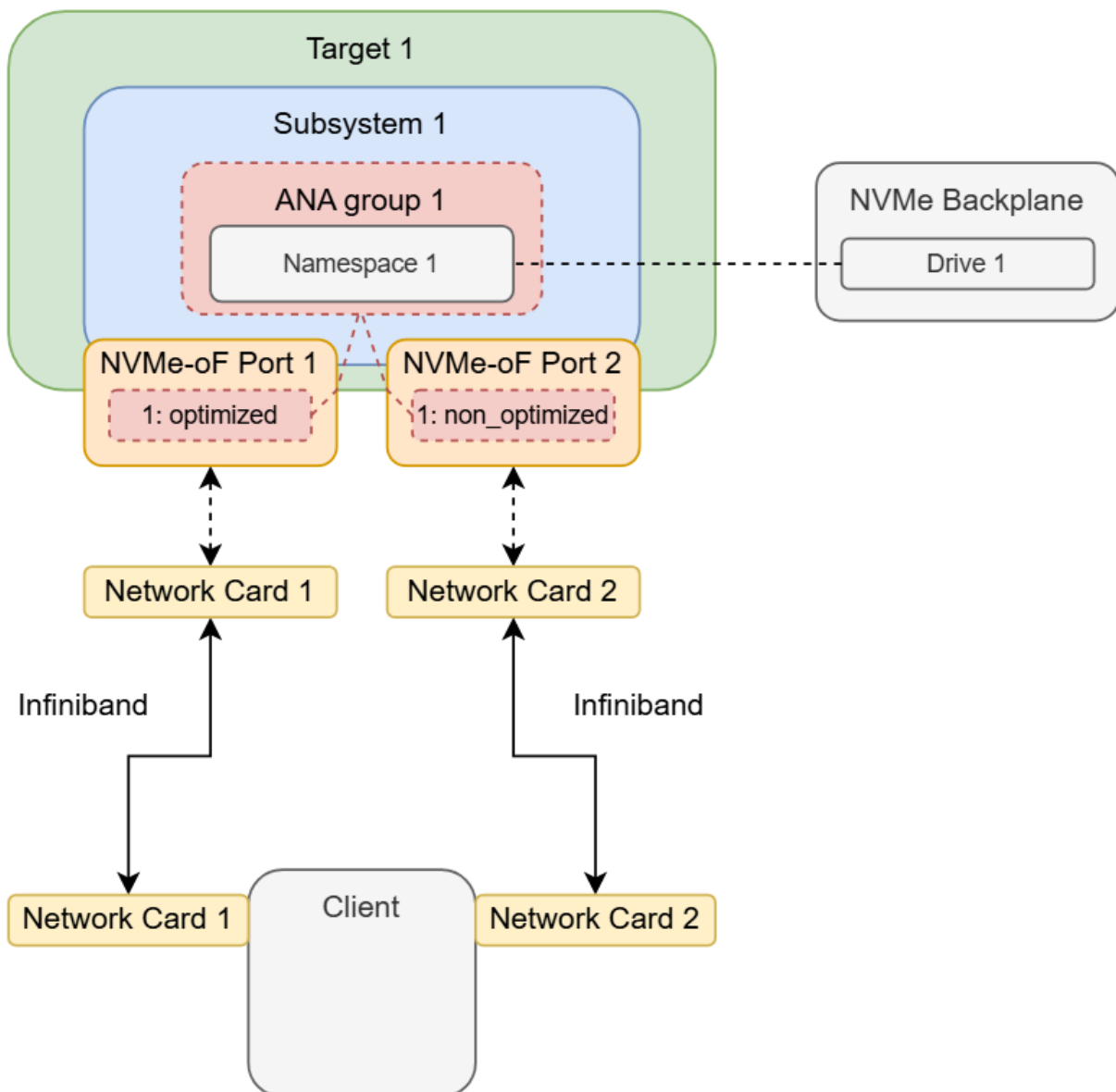
```
xnr_cli nvmf namespace add -n namespace0 -s subsystem0 -d <Device  
Name or UUID> -a 1
```

To update the ANA state of an ANA group, run the following command:

```
xnr_cli nvmf ana set -p port2 -g 1 -s non_optimized
```

This command sets the state of the first ANA group on the `port2` NVMe-oF port to **Non-Optimized**. Note that this command sets the ANA state for this group on a single port. This step may have to be repeated on other NVMe-oF ports, if applicable.

Below is a diagram illustrating a setup in which a single namespace is made available to a client over multiple NVMe-oF ports. The namespace belongs to the first ANA group. On the first NVMe-oF port, the ANA state of this group is set to **Optimized**, while on the second port, it is set to **Non-Optimized**. With this configuration, the client will use the first path to connect to the target. If the first path becomes unavailable, the client will switch to the second (**Non-Optimized**) path.



## Configuring NVMe-oF Host Access

You can control which hosts (initiators) can connect to the subsystem exposed over NVMe-oF. By default, the subsystem's host-access mode is set to *limited*, which means only hosts on the subsystem's allowed hosts list can connect. This topic explains how to:

- Configure hosts in xiRAID Opus
- Update the subsystem's allowed hosts list
- Change the host-access mode for subsystems

## Managing Hosts in xiRAID Opus

Before adding a host to a subsystem's allowed hosts list, the host must first be added to the xiRAID Opus configuration.

To add a host to the configuration, run the following command:

```
xnr_cli nvme host add -n host0 --nqn
nqn.2014-08.org.nvmexpress:uuid:3dd2228b-b1c2-4fc2-b5c2-3929ce45e57c
```

In this command, `--nqn` is the NQN of your host. To find the NQN of your host, run the command `nvme show-hostnqn`. If you are using xiRAID Opus as the initiator, run the command `xnr_cli config get -n default_host_nqn`.

To remove a host from the configuration:

```
xnr_cli nvme host remove -n host0
```

To show the configured hosts:

```
xnr_cli nvme host show
```

## Configuring Subsystem Allowed Hosts List

Once a host is added to the xiRAID Opus configuration, you can add it to the subsystem's allowed hosts list (if the subsystem's host-access mode is set to *limited*).

To add a host to the subsystem's allowed hosts list, run the following command:

```
xnr_cli nvme subsystem access add -n subsystem0 -d host0
```

To remove a host from the subsystem's allowed hosts list:

```
xnr_cli nvme subs access remove -n subsystem0 -d host0
```

To show hosts in the subsystem's allowed hosts list:

```
xnr_cli nvme subsystem access show -n subsystem0
```

## Changing Subsystem Host-Access Mode

You can change the host-access mode to control who can connect to the subsystem. By default, the host-access mode is set to *limited*, meaning that only hosts explicitly listed in the subsystem's allowed hosts list can connect.

If you want to allow any host to connect to the subsystem, you can change the host-access mode to *public* by running:

```
xnr_cli nvme subsystem access public -n subsystem0
```

If you want to restrict access so that only allowed hosts can connect, ensure the host-access mode is set to *limited* by running:

```
xnr_cli nvme subsystem access limited -n subsystem0
```

# xiRAID Opus Configuration Options

## Memory Configuration

The xiRAID Opus uses huge memory pages. The amount of huge memory needs to be specified during the product configuration at installation. You can check the size of the configured huge memory, using the command:

```
$ xnr_cli config get --name hugemem_2mb
```

After the initial configuration of the engine, this parameter can be updated using the following command:

```
$ xnr_cli config set --name hugemem_2mb --value <new_hugemem_value>
```

where `new_hugemem_value` is the comma separated list of hugepage memory sizes in megabytes to be allocated to the corresponding NUMA nodes.

- Positive values set the amount of memory (in MB) to allocate for the corresponding NUMA node.
- Negative values are specified to keep the memory configuration of the specified NUMA node unchanged.
- Zero values are used to set the amount of memory for the specified NUMA node to zero.

The current xiRAID release does not support memory domains. For optimal performance it is recommended to allocate the memory at one NUMA node only and the node number should correspond to the CPU mask where the engine is running. The xiRAID service must be restarted to apply the new memory configuration.

The following example will allocate 32 GB of hugepage memory to NUMA node 0, leave NUMA node 1 unchanged, allocate 16 GB to NUMA node 2, and set the amount of memory for NUMA node 3 to zero.

```
$ xnr_cli config set --name hugemem_2mb --value 32768,-2,16384,0
```

There is no limitation on the hugepage size except for the size of the physical memory on your hardware. It is recommended to set it to at least 32 GB. If you pass too big of a value for the hugepage, the following error will be in the syslog and the actual hugepage parameter will not be updated:

```
xnr_plt_set_hugemem: ERROR: Node 0, hugepages 2048kB requested  
450000, allocated 4578
```

## CPU Mask

xiRAID allows to check and change the CPU mask in runtime. To check the configured CPU mask use the command:

```
$ xnr_cli config get --name cpu_mask
```

This parameter can be updated by the following command. The xiRAID service must be restarted to apply the new CPU mask value:

```
$ xnr_cli config set --name cpu_mask --value 0x1f07
```

CPU core mask can specify any CPUs at any NUMA node. It is recommended to use same NUMA node for selected CPU cores. The mask can be specified in one of the following ways:

- 0x1f07
- 0x1F07
- [0,1,2,8-12]
- [0, 1, 2, 8, 9, 10, 11, 12]

In the examples above, the application is configured to use the following CPU cores: 0, 1, 2, 8, 9, 10, 11, and 12.

## Logging Management

xiRAID Opus Logging parameters include `log_flags`, `log_level` and `log_print_level`.

The `log_level` and `log_print_level` parameters can be set to ERROR, WARNING, NOTICE, INFO. INFO log level requires enabling corresponding component `log_flag`.

The `log_level` controls logging to the syslog, `log_print_level` controls logging to the stdout. The `log_print_level` is used only when the xiRAID engine is running as a background process and is not used in normal running mode where stdout is not available. The syslog can be accessed and printed using `journalctl` or a similar OS command.

The `log_flags` enable or disable INFO logging for different components. To see the list of all components, check the output of the `xnr_xiraid -h` command in the -L flag help message. Here is the command output:

```
-L, --logflag <flag>    enable log flag (all, accel, accel_ioat,
aio, app_config, app_rpc, bdev, bdev_concat, bdev_ftl, bdev_malloc,
bdev_null, bdev_nvme, bdev_raid, bdev_raid0, bdev_raid1, blob,
blob_esnap, blob_rw, blobfs, blobfs_bdev, blobfs_bdev_rpc,
blobfs_rw, ftl_core, ftl_init, gpt_parse, ioat, iscsi, json_util,
log, log_rpc, lvol, lvol_rpc, nbd, notify_rpc, nvme, nvme_vfio,
nvmf, nvmf_tcp, opal, rdma, reactor, rpc, rpc_client, scsi, sock,
sock_posix, thread, trace, vbdev_delay, vbdev_gpt, vbdev_lvol,
vbdev_opal, vbdev_passthru, vbdev_split, vbdev_zone_block,
vfio_pci, vfio_user, vhost, vhost_blk, vhost_blk_data, vhost_ring,
vhost_rpc, vhost_scsi, vhost_scsi_data, vhost_scsi_queue, virtio,
virtio_blk, virtio_dev, virtio_pci, virtio_user, virtio_vfio_user,
vmd, xnr, xnr_dev, xnr_raid_config)
```

## Advanced Configuration Options

xiRAID Opus system configurations can be managed using the “config” command. To get a list of all the available parameters use the command:

```
xnr_cli config get
```

Here is the command output example:

```
version : 1.3.0, build: 10910, tag: ver1.3.0-d, date:
  2026-02-09T12:41:09+00:00
commit_base : fc46836dcfefc83cf8f1bc4f46604750c3035160
commit_spdk : 42cd57263c541c0a39940c058e9d4ff5cc298b54
node_uuid : b2d1bae1-500d-4b50-a792-e1e6b1b95e8b
default_host_nqn :
  nqn.2014-08.org.nvmexpress:uuid:b2d1bae1-500d-4b50-a792-e1e6b1b95e8b
cpu_mask : 0xf
hugemem_2mb : 8192
skip_drive_fresh_check : false
resync_enabled : true
disable_iommu_reset_by_removal : false
log_flags :
log_level : NOTICE
log_print_level : NOTICE
engineering_mode : false
config_format_version : 3
```

Some parameters are read-only (for example `commit_base` and `version`). Other parameters can be updated by the command:

```
$ xnr_cli config set --name Name --value NewValue
```

If you try to update a read-only parameter, you will get the following error message:

```
$ xnr_cli config set --name engineering_mode --value true
2024-00-00T18:06:09.726Z          ERROR   Fail to update the
configuration: cannot change the parameter
```

The current release supports the following parameters:

- **version** (read only) - The xiRAID Opus version.
- **commit\_base** (read only) - Management engine git hash.
- **commit\_spdk** (read only) - Data path engine git hash.
- **engineering\_mode** - Always False in normal operation mode. Support engineer can enable the mode for test and recovery.
- **log\_level** - Log level for the syslog journal
- **log\_print\_level** - Log level for the console output if the engine running as foreground process
- **log\_flags** - List of enabled log message classes for the INFO log level
- **resync\_enabled** - Enable or disable the resync functionality for all RAIDs
- **cpu\_mask** - CPU mask of cores used by the data path engine for processing IO
- **hugemem\_2mb** - The amount of huge memory in megabytes per NUMA node used by the data path engine for internal buffers

## Command & gRPC Reference

This section outlines the two primary methods for interacting with the software: through the gRPC API or via the `xnr_cli` tool. The CLI tool acts as a wrapper for the gRPC API, providing the same functionality but with a user-friendly interface that may be more suitable for manual management. For automation and programmatic integration, the gRPC API is the preferred method. This reference will cover both the available commands in the CLI tool and the functions exposed through the gRPC API, along with examples demonstrating how to interact with the gRPC API using the `grpcurl` tool.

### Using `xnr_cli`

You can obtain a brief command description by using the self-documented CLI with the `-h` flag.

```
xnr_cli --help
```

```
xnr_cli <command> -h
```

The CLI implements shortcuts for commands and command options. For example, `-h` is a shortcut for `--help`.

Some commands and options are hidden. The `--honest` (or `-H`) flag enables help for hidden commands and options. They are experimental and not designed for production use. The hidden commands can be used for testing or prototyping. The hidden commands syntax may be changed in future releases or the commands may be removed from the CLI.

```
xnr_cli --honest --help
```

```
xnr_cli -H <command> -h
```

For some commands, it is possible to specify multiple values for a single command parameter. There are two ways to do this:

- Pass the values as comma-separated values:

```
xnr_cli bdev zero --names
43E0A004TLZJn1,43F0A00ATLZJn1,43T0A002TLZJn1
```

- Specify the parameter multiple times, each time with a different value:

```
xnr_cli bdev zero --names 43E0A004TLZJn1 --bdevs 43F0A00ATLZJn1
--bdevs 43T0A002TLZJn1
```

Some command options are values separated by the `/` symbol:

describe name/uuid

```
-a ipv6/abcd::08:08:aa/4420
```

```
-a 1.2.3.4/4420
```

```
-a ipv4/4.5.6.7/4420
```

## bdev

Operations with BDEVs.

```
xnr_cli bdev
```

Subcommands for the `bdev` command:

<code>bdev malloc create</code>	Create a ramdisk BDEV.
---------------------------------	------------------------

<code>bdev malloc destroy</code>	Destroy a ramdisk BDEV.
----------------------------------	-------------------------

bdev show	Display information about BDEVs managed by xiRAID Opus.
bdev zero	Clear the metadata of the specified BDEVs.
bdev partition add	Add new BDEV partitions.
bdev partition remove	Remove BDEV partitions.
bdev partition show	Display BDEV partitions.
bdev qos set	Set BDEV IOPS and throughput limits.
bdev qos clear	Clear BDEV IOPS and throughput limits.

To view the available functions for interacting with BDEVs through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Bdev
```

## bdev malloc (experimental)

Use the following command to manage malloc BDEVs in xiRAID Opus.



Ramdisk BDEVs are non-persistent. When the system is turned off or restarted, the data in RAM is lost. This functionality should be used for testing purposes only.

```
xnr_cli bdev malloc
```

The following subcommands are available for the `malloc` command:

create	Create a ramdisk BDEV. Ramdisks are created using memory allocated to xiRAID Opus during the installation. If necessary, update the <code>hugemem_2mb</code> parameter to allocate more memory to xiRAID Opus. See the <a href="#">config</a> page for more details.
destroy	Destroy a ramdisk BDEV.

## bdev malloc create

Use the following command to create a malloc BDEV:

```
xnr_cli bdev malloc create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.MallocCreate
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The only required option is `--num-blocks`.

<code>-n</code>	Optionally, specify the name of this BDEV. By default,
<code>--name</code>	the system sets the name to <code>MallocN</code> , where <code>N</code> is the next
<code>"id"</code>	available ordinal number (e.g., <code>Malloc0</code> , <code>Malloc1</code> , <code>Malloc2</code> , etc.).
<code>-b</code>	Optionally, specify the block size in bytes. The block size
<code>--block-size</code>	must be a multiple of 512. The default value is 4096.
<code>"block_size"</code>	
<code>-c</code>	Specify the number of blocks to use for this ramdisk.
<code>--nb</code>	
<code>--num-blocks</code>	
<code>"num_blocks"</code>	

`xnr_cli` example:

```
xnr_cli bdev malloc create --num-blocks 10 -n newmalloc
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"num_blocks":"10","id":  
[{"name":"newmalloc"}]}' \  
localhost:8000 xnr.Bdev.MallocCreate
```

## bdev malloc destroy

Use the following command to destroy a malloc BDEV:

```
xnr_cli bdev malloc destroy
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.MallocDestroy
```

The following option is required for this command. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the BDEV to destroy. Use the <code>xnr_cli</code>
<code>--names</code>	<code>bdev show</code> command to get a list of BDEVs.
<code>"ids"</code>	

`xnr_cli` example:

```
xnr_cli bdev malloc destroy -n Malloc0,Malloc1
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"ids":[{"name":"Malloc0"},
{"name":"Malloc1"}]}' \
localhost:8000 xnr.Bdev.MallocDestroy
```

## bdev show

Use the following command to display information about BDEVs managed by xiRAID Opus.

```
xnr_cli bdev show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the name(s) of the BDEV for which to display information. If this option is not specified, <code>xnr_cli</code>
<code>--names</code>	<code>bdev show</code> displays information about all BDEVs managed
<code>"ids"</code>	by xiRAID Opus.

`xnr_cli` example:

```
xnr_cli bdev show -n newmalloc,newmalloc1
```

grpcurl example:

```
grpcurl -plaintext -d '{"ids":[{"name":"newmalloc"},  
{"name":"newmalloc1"}]}' \  
localhost:8000 xnr.Bdev.Show
```

## bdev zero

Use the following command to clear the metadata of the specified BDEVs. Clearing the metadata of a BDEV is often required before it can be added to a RAID. If a BDEV contains data, xiRAID Opus will return an error if you attempt to add the BDEV to a RAID before clearing its metadata.

Note that `bdev zero` will return an error message if the specified BDEV is used in a RAID or has partitions.

```
xnr_cli bdev zero
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.Clean
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the names of the BDEV(s) for which to clear meta-
<code>--names</code>	data.
<code>"ids"</code>	

`xnr_cli` examples:

```
xnr_cli bdev zero --names  
43E0A004TLZJn1,43F0A00ATLZJn1,43T0A002TLZJn1
```

```
xnr_cli bdev zero --names 43E0A004TLZJn1 --names 43F0A00ATLZJn1  
--names 43T0A002TLZJn1
```

```
xnr_cli bdev zero --names 43E0A004TLZJn1
```

grpcurl examples:

```
grpcurl -plaintext \  
-d '{"ids":[{"name":"nvme.232f618fee65a548.4"}]}' \  
localhost:8000 xnr.Bdev.Clean
```

```
grpcurl -plaintext \  
-d '{"ids":[{"name":"nvme.232f618fee65a548.3"},  
{"name":"nvme.232f618fee65a548.4"}]}' \  
localhost:8000 xnr.Bdev.Clean
```

## bdev partition (part)

Use the following command to manage BDEV partitions in xiRAID Opus.

```
xnr_cli bdev partition
```

You may also use `part` in the place of `partition`:

```
xnr_cli bdev part
```

The following subcommands are available for the `partition` command:

add	Add a BDEV partition.
remove	Remote a BDEV partition.
show	List partitions of a BDEV.

## bdev partition add

Use the following command to add new BDEV partitions:

```
xnr_cli bdev partition add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.PartAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) and partition size (`-p`, `--parts`) options are required.

<code>-n</code>	Specify the name of the BDEV to divide into partitions in the <code>name/uuid</code> format, where:
<code>--name</code>	
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the BDEV.</li> <li>• <code>uuid</code> is the UUID of the BDEV.</li> </ul>
<p>Either <code>name</code> or <code>uuid</code> is required. Examples:</p> <pre>-n test -n de416ea2-0b1e-40e9-9d15-3224d1152730 -n test/de416ea2-0b1e-40e9-9d15-3224d1152730</pre>	
<code>-k</code>	Specify the alignment boundary for a partition start LBA in kilobytes.
<code>--align-kb</code>	
<code>"align_kb"</code>	
<code>--force</code>	Pass this flag to recover the partition table if it's broken.
<code>"force"</code>	This operation may lead to data loss if there are no partitions on this BDEV.
<code>-p</code>	Specify the size and, optionally, UUID for new BDEV partition(s) in the <code>s/u</code> format, where:
<code>--parts</code>	
<code>"partitions"</code>	<ul style="list-style-type: none"> <li>• <code>s</code> - the size of the partition in MB.</li> <li>• <code>u</code> - the UUID to set for this partition.</li> </ul>

Examples:

```
-p 1024
-p 1024/d74fb9ce-efae-484f-ac2f-2a3b13e19e1f
-p 1024,2048
```

xnr\_cli example:

```
xnr_cli bdev part add -n test -p 1000,1000
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}], "partitions": [{"size_mb": 1000}, {"size_mb": 1000}]}' \
localhost:8000 xnr.Bdev.PartAdd
```

## bdev partition remove

Use the following command to remove partitions of a BDEV:

```
xnr_cli bdev partition remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.PartRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required.

<code>-n</code>	Specify the name of the BDEV to remove in the <code>name/uuid</code> format, where:
<code>--name</code>	
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the BDEV.</li> <li>• <code>uuid</code> is the UUID of the BDEV.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-c</code>	Specify the number of partitions to remove. To remove all partitions, use the <code>--all</code> option described below.
<code>--count</code>	
"count"	
<code>-a</code>	Pass this flag to remove all partitions.
<code>--all</code>	
"all"	
<code>--force</code>	Pass this flag to wipe the partition table if it's broken. This operation may lead to data loss if there are no partitions on this BDEV.
"force"	

`xnr_cli` example:

```
xnr_cli bdev part remove -n test -a
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}], "all":"true"}' \
localhost:8000 xnr.Bdev.PartRemove
```

## bdev partition show

Use the following command to display partitions of a BDEV:

```
xnr_cli bdev partition show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.PartShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required.

<code>-n</code>	Specify the name of the BDEV in the <code>name/uuid</code> format,
<code>--name</code>	where:
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the BDEV.</li> <li>• <code>uuid</code> is the UUID of the BDEV.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-s</code>	Optionally, pass this flag to read data from the secondary
<code>--force</code>	partition table.
"force_secondary"	

`xnr_cli` example:

```
xnr_cli bdev part show -n test
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}]}' localhost:8000
xnr.Bdev.PartShow
```

## bdev qos

Use the following command to manage BDEV IOPS and throughput limits..

```
xnr_cli bdev qos
```

The following subcommands are available for the `qos` command:

<code>set</code>	Set IOPS or throughput limits.
<code>clear</code>	Clear IOPS or throughput limits.

## bdev qos set

Use the following command to set BDEV IOPS and throughput limits:

```
xnr_cli bdev qos set
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.QosSet
```

To view the current limits of a BDEV, use the `xnr_cli bdev show` command.

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required.

<code>-n</code>	Specify the name of the BDEV in the <code>name/uuid</code> format,
<code>--name</code>	where:
"id"	<ul style="list-style-type: none"><li><code>name</code> is the name of the BDEV.</li><li><code>uuid</code> is the UUID of the BDEV.</li></ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`--rwi` Specify the IOPS limit for this BDEV in thousands. For example, to limit the number of IOPS to 40,000:

`--rw_kiops`

`"rw_kiops"`

```
--rwi 40
```

`--rwm` Specify the MB/s throughput limit for read and write operations.

`--rw_mbps`

`"rw_mbps"`

`--rm` Specify the MB/s throughput limit for read operations.

`--read_mbps`

`"r_mbps"`

`--wm` Specify the MB/s throughput limit for write operations.

`--write_mbps`

`"w_mbps"`

xnr\_cli example:

```
xnr_cli bdev qos set -n newmalloc --rwi 40 --rwm 100
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"newmalloc"}],
  "rw_kiops":"40", "rw_mbps":"100"}' \
localhost:8000 xnr.Bdev.QosSet
```

## bdev qos clear

Use the following command to clear BDEV IOPS and throughput limits:

```
xnr_cli bdev qos clear
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Bdev.QosClear
```

To view the current limits of a BDEV, use the `xnr_cli bdev show` command.

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required.

<code>-n</code>	Specify the name of the BDEV in the <code>name/uuid</code> format,
<code>--name</code>	where:
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the BDEV.</li> <li>• <code>uuid</code> is the UUID of the BDEV.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>--rwi</code>	Pass this flag to clear the IOPS limit for this BDEV
--------------------	--

<code>--rw_kiops</code>	
-------------------------	--

"rw_kiops"	
------------	--

<code>--rwm</code>	Pass this flag to clear the throughput limit for read and write operations.
--------------------	---

<code>--rw_mbps</code>	
------------------------	--

"rw_mbps"	
-----------	--

<code>--rm</code>	Pass this flag to clear the throughput limit for read operations.
-------------------	---

<code>--read_mbps</code>	
--------------------------	--

"r_mbps"	
----------	--

<code>--wm</code>	Pass this flag to clear the throughput limit for write operations.
-------------------	--

<code>--write_mbps</code>	
---------------------------	--

"w_mbps"	
----------	--

`xnr_cli` example:

```
xnr_cli bdev qos clear -n newmalloc --rwi --rwm
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"newmalloc"}],  
  "rw_kiops":"true", "rw_mbps":"true"}' \  
localhost:8000 xnr.Bdev.QosClear
```

## network

This section describes the `network` command for managing network configurations for NVMe-oF ports.

```
xnr_cli network
```

The following subcommands are available for the `network` command:

`address`                      Manage network address configurations.

To view the available functions for managing network configurations through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Network
```

## address

Use the following command to manage network address configurations for NVMe-oF ports:

```
xnr_cli network address
```

The following subcommands are available for the `network` command:

`add`                          Add a network address configuration.

---

`remove`                      Remove a network address configurations.

---

`show`                        Show configured network address configurations.

## address add

Use the following command to add a network address configuration.

```
xnr_cli network address add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Network.AddressAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the network address configuration in the <code>name/uuid</code> format, where:
<code>--name</code>	
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the network address configuration.</li> <li>• <code>uuid</code> is the UUID of the network address configuration. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-a</code>	Specify the IP address to use in the <code>addressFamily/address</code> format, where:
<code>--address</code>	
<code>"address"</code>	<ul style="list-style-type: none"> <li>• <code>addressFamily</code> is the address family (<code>ipv4</code>, <code>ipv6</code>, or <code>ib</code>). In <code>xnr_cli</code>, the <code>addressFamily</code> is an optional field and is only required for InfiniBand (<code>ib</code>) addresses.</li> <li>• <code>address</code> is the actual IP address.</li> </ul>

`xnr_cli` examples:

```
xnr_cli network address add -n address0 -a 172.16.133.128
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"address0"}], "address":{"addr":"172.16.133.128", "family": 1}}' \
localhost:8000 xnr.Network.AddressAdd
```

`xnr.AddressFamily` values:

- 1 - IPv4
- 2 - IPv6
- 3 - InfiniBand

## address remove

Use the following command to remove a network address configuration.

```
xnr_cli network address remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Network.AddressRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the network address configuration in the <code>name/uuid</code> format, where:
<code>--name</code>	
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the network address configuration.</li> <li>• <code>uuid</code> is the UUID of the network address configuration. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli network address remove -n address0
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"address0"}]}' \
localhost:8000 xnr.Network.AddressRemove
```

## address show

Use the following command to show network address configurations.

```
xnr_cli network address show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Network.AddressRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the names of the network address configurations in the <code>name/uuid</code> format, where:
<code>--names</code>	
<code>"ids"</code>	<ul style="list-style-type: none"><li>• <code>name</code> is the name of the network address configuration.</li><li>• <code>uuid</code> is the UUID of the network address configuration.</li></ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli network address show
xnr_cli network address show -n address0,address1
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.Network.AddressShow
grpcurl -plaintext -d '{"ids":[{"name":"address0"},
{"name":"address1"}]}' \
localhost:8000 xnr.Network.AddressShow
```

## nvmf

This section describes the `nvmf` command for managing NVMe-oF targets..

```
xnr_cli nvmf
```

The following subcommands are available for the `nvmf` command:

transport	Manage NVMe-oF transports.
subsystem	Manage NVMe-oF subsystems.
namespace	Manage NVMe-oF namespaces.
port	Manage NVMe-oF ports.
ana	Manage ANA groups.

To view the available functions for managing network configurations through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Nvmf
```

## transport (tr)

Use the following command to manage NVMe-oF transports:

```
xnr_cli nvmf transport
```

You may also use `tr` in the place of `transport`:

```
xnr_cli nvmf tr
```

The following subcommands are available for the `transport` command:

create	Create an NVMe-oF transport.
delete-config	Delete an NVMe-oF transport configuration.
show	Show NVMe-oF transport configurations.

## transport create

Use the following command to create an NVMe-oF transport.

```
xnr_cli nvmf transport create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.TransportCreate
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF transport in the <code>name/</code>
<code>--name</code>	<code>uuid</code> format, where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF transport.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF transport. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-t</code>	Specify the transport type: <code>TCP</code> or <code>RDMA</code> .
<code>--type</code>	
<code>"type"</code>	
<code>-d</code>	Specify the max number of outstanding I/O per queue. By default, the value is set to 128.
<code>--queue-depth</code>	
<code>"queue_depth"</code>	
<code>-q</code>	Specify the max number of qpairs per controller, including one qpair used for admin queue. By default, the value is set to 128.
<code>--ctrl-qpairs</code>	
<code>"ctrl_qpairs"</code>	
<code>-c</code>	Specify the max number of in-capsule data size. By default, the value is set to 4096.
<code>--in-capsule-size</code>	
<code>"in_capsule_size"</code>	

<code>-i</code>	Specify the max I/O size in bytes or using size suffixes such as <code>KB</code> (kilobytes) or <code>MB</code> (megabytes).
<code>--max-io-size</code>	In a RAID 5 configuration with 5 disks and a stripe size of 128 KiB, the <code>max_io_size</code> parameter should be set to 512 KiB. Since RAID 5 uses a parity stripe along with data stripes, the number of data stripes is typically 4 (out of 5 disks). To accommodate this, the <code>max_io_size</code> should be set to 128 KiB × 4 (the number of data stripes), which equals 512 KiB.
<code>"max_io_size"</code>	If multiple RAID arrays are exposed over the same transport, the <code>max_io_size</code> should be configured to accommodate the largest I/O size requirement across all arrays. Ensure that the parameter value is large enough to support the largest RAID configuration in use.
<code>-a</code>	Specify the max number of admin commands per admin queue. By default, the value is set to 128.
<code>--admin-depth</code>	
<code>"admin_depth"</code>	
<code>-b</code>	Specify the number of data buffers available to the transport. By default, the value is set to 511 for TCP transports and 255 for RDMA transports.
<code>--num-buffers</code>	
<code>"num_bufs"</code>	
<code>-r</code>	Specify the number of buffers reserved for each poll group. By default, the number is determined automatically.
<code>--reserved-buffers</code>	
<code>"reserved_bufs"</code>	
<code>--disable-zcopy</code>	Pass this flag to disable zero-copy operations. By default, zero-copy operations are enabled if they are supported.
<code>"disable_zcopy"</code>	Note: In the gRPC API, this is a boolean value and should be specified as <code>"disable_zcopy": "true"</code> to disable zero-copy operations.
<code>-x</code>	Specify the abort execution timeout in seconds. By default, the value is set to 1.

--abort-timeout

---

"abort\_timeout\_sec"

---

-p Specify the polling interval of the acceptor for incoming connections (usec). By default, the value is set to 10000.

--acceptor-poll-rate

---

"acceptor\_poll\_usec"

xnr\_cli examples:

```
xnr_cli nvmf transport create -n transport0 -t TCP
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"transport0"}], "type":2}' \
localhost:8000 xnr.Nvmf.TransportCreate
grpcurl -plaintext -d '{"id":[{"name":"transport-rdma"}], "type":1}' \
localhost:8000 xnr.Nvmf.TransportCreate
```

## transport delete-config

Use the following command to delete an NVMe-oF transport.

```
xnr_cli nvmf transport delete-config
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.TransportDeleteConfig
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

-n Specify the name of the NVMe-oF transport in the `name/uuid` format, where:

---

--name

---

"id"

- `name` is the name of the NVMe-oF transport.
- `uuid` is the UUID of the NVMe-oF transport. If a UUID is not specified, it will be automatically generated by the system.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli nvmf tr delete-config -n transport0
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"transport0"}]}' \
localhost:8000 xnr.Nvmf.TransportDeleteConfig
```

## transport show

Use the following command to show NVMe-oF transports.

```
xnr_cli nvmf transport show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.TransportShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the names of the NVMe-oF transports
<code>--names</code>	in the <code>name/uuid</code> format, where:
<code>"ids"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF transport.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF transport.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

xnr\_cli examples:

```
xnr_cli nvme transport show
xnr_cli nvme transport show -n transport0,transport1
```

grpcurl example:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.TransportShow
grpcurl -plaintext -d '{"ids":[{"name":"transport0"},
{"name":"transport1"}]}' \
localhost:8000 xnr.Nvme.TransportShow
```

## subsystem (subs)

Use the following command to manage NVMe-oF subsystems:

```
xnr_cli nvme subsystem
```

You may also use `subs` in the place of `subsystem`:

```
xnr_cli nvme subs
```

The following subcommands are available for the `subsystem` command:

create	Create an NVMe-oF subsystem.
destroy	Delete an NVMe-oF subsystem.
show	Show NVMe-oF subsystems.

## subsystem create

Use the following command to create an NVMe-oF subsystem.

```
xnr_cli nvme subsystem create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemCreate
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF subsystem in the <code>name/</code>
<code>--name</code>	<code>uuid</code> format, where:
<code>"fid.id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF subsystem.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF subsystem. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the subsystem.
<code>--nqn</code>	
<code>"fid.nqn"</code>	
<code>-s</code>	Specify the serial number. By default, the serial number is generated automatically and follows the following naming convention: <code>XNR000000000000001</code> .
<code>--serial-number</code>	
<code>"serial_number"</code>	
<code>-m</code>	Specify the model number. By default, the model number is generated automatically: <code>XNR bdev Controller</code> .
<code>--model-number</code>	
<code>"model_number"</code>	
<code>-a</code>	Specify the host-access mode to control who can connect to the subsystem:
<code>--access</code>	
<code>"security_mode"</code>	

- `public` (1 for gRPC) - Allow any host to connect to the subsystem.
- `limited` (2 for gRPC) - Allow only hosts from the subsystem's allowed hosts list to connect.

---

<code>-M</code>	Specify the maximum number of namespaces allowed for this subsystem. By default, the value is set to 32.
<code>--max-namespaces</code>	
<code>"max_namespaces"</code>	

xnr\_cli examples:

```
xnr_cli nvme subsystem create -n subsystem0 -q
nqn.2025-04.io.xinnor:cnodel
```

grpcurl example:

```
grpcurl -plaintext -d '{"fid": {"id": {"name": "subsystem0"}, "nqn":
  "nqn.2025-04.io.xinnor:cnodel"}}' \
localhost:8000 xnr.Nvme.SubsystemCreate
```

## subsystem destroy

Use the following command to destroy an NVMe-oF subsystem.

```
xnr_cli nvme subsystem destroy
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemDestroy
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

Specify either the name of the subsystem, its NQN, or both.

<code>-n</code>	Specify the name of the NVMe-oF subsystem in the <code>name/</code>
<code>--name</code>	<code>uuid</code> format, where:
<code>"fid.id"</code>	

- `name` is the name of the NVMe-oF subsystem.
- `uuid` is the UUID of the NVMe-oF subsystem. If a UUID is not specified, it will be automatically generated by the system.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

---

`-q` Specify the NQN of the subsystem.

---

`--nqn`

---

"fid.nqn"

xnr\_cli examples:

```
xnr_cli nvme subsystem destroy -n subsystem0 -q
nqn.2025-04.io.xinnor:cnode1
```

grpcurl example:

```
grpcurl -plaintext -d '{"fid": {"id": {"name": "subsystem0"}, "nqn":
  "nqn.2025-04.io.xinnor:cnode1"}}' \
localhost:8000 xnr.Nvme.SubsystemDestroy
```

## subsystem show

Use the following command to display NVMe-oF subsystems.

```
xnr_cli nvme subsystem show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

Optionally, provide either the names of the subsystems or their NQNs.

<code>-n</code>	Specify the name of the NVMe-oF subsystem in the <code>name/</code>
<code>--names</code>	<code>uuid</code> format, where:
<code>"fids.id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF subsystem.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF subsystem. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the subsystem.
<code>--nqns</code>	
<code>"fid.nqn"</code>	

`xnr_cli` examples:

```
xnr_cli nvme subsystem show
xnr_cli nvme subsystem show -n subsystem0,subsystem1
xnr_cli nvme subsystem show -q
nqn.2025-04.io.xinnor:cnode2,nqn.2025-04.io.xinnor:cnode1,nqn.2025-04.io.xinnor:cnode2
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemShow
grpcurl -plaintext -d '{"fids": [{"id": { "name": "subsystem0" } }, {"id": { "name": "subsystem1" } }]} ' \
localhost:8000 xnr.Nvme.SubsystemShow
```

## subsystem access

Use the following command to configure subsystem allowed hosts list and to change subsystem host-access mode.

```
xnr_cli nvme subsystem access
```

The following subcommands are available for the `access` command:

public	Change the host-access mode of a subsystem to <i>public</i> .
limited	Change the host-access mode of a subsystem to <i>limited</i> .
add	Add a host to a subsystem's allowed hosts list.
remove	Remove a host from a subsystem's allowed hosts list.
show	Show hosts in a subsystem's allowed hosts list.

## public

Use the following command to change a subsystem's host-access mode to *public*:

```
xnr_cli nvme subsystem access public
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemAccessMode
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the subsystem in the <code>name/uuid</code> format, where:
<code>--name</code>	
<code>"fids.id"</code>	<ul style="list-style-type: none"> <li><code>name</code> is the name of the subsystem.</li> <li><code>uuid</code> is the UUID of the subsystem.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the subsystem.
<code>--NQN</code>	
<code>"fids.nqn"</code>	
<code>--force</code>	If the subsystem's host access list is not empty, you must specify the force flag to change the subsystem's host-access mode to public.
<code>"force"</code>	

xnr\_cli examples:

```
xnr_cli nvme subsystem access public -n subsystem0
```

grpcurl example:

```
grpcurl -plaintext -d '{"fids": {"id": {"name":  
  "subsystem0"}}}, "mode": "1"}' \  
localhost:8000 xnr.Nvme.SubsystemAccessMode
```

## limited

Use the following command to change a subsystem's host-access mode to *limited*:

```
xnr_cli nvme subsystem access limited
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemAccessMode
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the subsystem in the <code>name/uuid</code> format, where:
<code>--name</code>	
<code>"fid.id"</code>	<ul style="list-style-type: none"> <li><code>name</code> is the name of the subsystem.</li> <li><code>uuid</code> is the UUID of the subsystem.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test  
-n de416ea2-0b1e-40e9-9d15-3224d1152730  
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the subsystem.
<code>--NQN</code>	
<code>"fid.nqn"</code>	

<code>--force</code>	If the subsystem's host access list is not empty, you must specify the force flag to change the subsystem's host-access mode to public.
<code>"force"</code>	

xnr\_cli examples:

```
xnr_cli nvme subsystem access limited -n subsystem0
```

grpcurl example:

```
grpcurl -plaintext -d '{"fids": {"id": {"name": "subsystem0"}}, "mode": "2"}' \
localhost:8000 xnr.Nvme.SubsystemAccessMode
```

## add

Use the following command to add a host to the subsystem's allowed hosts list.

```
xnr_cli nvme subsystem access add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemAccessAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name(s) of the subsystem(s) in the <code>name/uuid</code> format, where:
<code>--names</code>	

<code>"fids.id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the subsystem.</li> <li>• <code>uuid</code> is the UUID of the subsystem.</li> </ul>
------------------------	--

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN(s) of the subsystem(s).
<code>--nqns</code>	

"fids.nqn"

---

**-d** Specify the name(s) of the host(s) in the `name/uuid` format,  
**--hosts** where:

---

"hosts"

- `name` is the name of the subsystem.
- `uuid` is the UUID of the subsystem.

xnr\_cli examples:

```
xnr_cli nvme subsystem access add -n subsystem0 -d host0
```

grpcurl example:

```
grpcurl -plaintext -d '{"fids": {"id": {"name": "subsystem0"}}, "hosts": [{"name": "host0"}]}' \
localhost:8000 xnr.Nvme.SubsystemAccessAdd
```

## remove

Use the following command to add a host to the subsystem's allowed hosts list.

```
xnr_cli nvme subsystem access remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemAccessRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

---

**-n** Specify the name(s) of the subsystem(s) in the `name/uuid`  
**--names** format, where:

---

"fids.id"

- `name` is the name of the subsystem.
- `uuid` is the UUID of the subsystem.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN(s) of the subsystem(s).
<code>--nqns</code>	
<code>"fids.nqn"</code>	
<code>-d</code>	Specify the name(s) of the host(s) in the <code>name/uuid</code> format,
<code>--hosts</code>	where:
<code>"hosts"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the subsystem.</li> <li>• <code>uuid</code> is the UUID of the subsystem.</li> </ul>

xnr\_cli examples:

```
xnr_cli nvme subsystem access remove -n subsystem0 -d host0
```

grpcurl example:

```
grpcurl -plaintext -d '{"fids": {"id": {"name": "subsystem0"}}, "hosts": [{"name": "host0"}]}' \
localhost:8000 xnr.Nvme.SubsystemAccessRemove
```

## show

Use the following command to add a host to the subsystem's allowed hosts list.

```
xnr_cli nvme subsystem access show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.SubsystemAccessShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name(s) of the subsystem(s) in the <code>name/uuid</code>
<code>--names</code>	format, where:
<code>"fids.id"</code>	

- `name` is the name of the subsystem.
- `uuid` is the UUID of the subsystem.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN(s) of the subsystem(s).
<code>--nqns</code>	
<code>"fids.nqn"</code>	
<code>-o</code>	Specify the output format: <code>table</code> or <code>json</code> . By default, the
<code>--output</code>	<code>table</code> format is used.

This option is relevant only for `xnr_cli`.

`xnr_cli` examples:

```
xnr_cli nvme subsystem access show
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"fids": {"id": {"name": "subsystem0"}}}' \
localhost:8000 xnr.Nvme.SubsystemAccessShow
```

## namespace (ns)

Use the following command to manage NVMe-oF namespaces:

```
xnr_cli nvme namespace
```

You may also use `ns` in the place of `namespace`:

```
xnr_cli nvme ns
```

The following subcommands are available for the `namespace` command:

<code>add</code>	Add a namespace to a subsystem.
<code>remove</code>	Remove a namespace from a subsystem.
<code>show</code>	Show namespaces.

## namespace add

Use the following command to create an NVMe-oF namespace.

```
xnr_cli nvme namespace add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.NamespaceAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF namespace in the <code>name/uuid</code> format, where:
<code>--name</code>	
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF namespace.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF namespace. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-s</code>	Specify the name of the NVMe-oF subsystem to add this namespace to in the <code>name/uuid</code> format, where:
<code>--subsystem</code>	
"subsys.name"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF namespace.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF namespace. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

<code>-q</code>	Specify the NQN of the subsystem. The NQN may be specified instead of the subsystem name/UUID.
<code>--nqn</code>	
"subsys.nqn"	

<code>-d</code>	Specify the name of the BDEV to use as a namespace in the <code>name/uuid</code> format, where:
<code>--bdev</code>	
<code>"bdev"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF namespace.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF namespace. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>
<code>-g</code>	Specify the namespace globally unique identifier. If the NGUID is not specified, it will be generated by xiRAID Opus.
<code>--nguid</code>	
<code>"nguid"</code>	
<code>-e</code>	Specify the namespace EUI-64 identifier. If the identifier is not specified, it will be generated by xiRAID Opus.
<code>--eui64</code>	
<code>"eui64"</code>	
<code>-a</code>	Specify the ANA group to add this namespace to. By default, all namespaces are added to the same ANA group.
<code>--ana-grp-id</code>	Note that the ANA group cannot be changed after the namespace creation.
<code>"ana_group_id"</code>	

xnr\_cli examples:

```
xnr_cli nvmf namespace add -n namespace0 -s subsystem0 -d
nvme.c0a9eb3c13788900.0
```

grpcurl example:

```
grpcurl -plaintext -d '{
  "id": { "name": "namespace0" },
  "subsys": { "id": { "name": "subsystem0" } },
  "bdev": { "name": "nvme.c0a9eb3c13788900.0" }
}' localhost:8000 xnr.Nvmf.NamespaceAdd
```

## namespace remove

Use the following command to remove an NVMe-oF namespace.

```
xnr_cli nvmf namespace remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.NamespaceRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF namespace in the <code>name/</code>
<code>--name</code>	<code>uuid</code> format, where:
<code>"id"</code>	<ul style="list-style-type: none"> <li><code>name</code> is the name of the NVMe-oF namespace.</li> <li><code>uuid</code> is the UUID of the NVMe-oF namespace.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli nvme namespace remove -n port0
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"namespace0"}]}' \
localhost:8000 xnr.Nvmf.NamespaceRemove
```

## namespace show

Use the following command to show NVMe-oF namespaces.

```
xnr_cli nvme namespace show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.NamespaceShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the names of the NVMe-oF namespaces in the <code>name/uuid</code> format, where:
<code>--names</code>	
<code>"ids"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF namespace.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF namespace.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli nvmf namespace show
xnr_cli nvmf namespace show -n namespace0,namespace1
```

grpcurl example:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.NamespaceShow
grpcurl -plaintext -d '{"ids":[{"name":"namespace0"},
{"name":"namespace1"}]}' \
localhost:8000 xnr.Nvmf.NamespaceShow
```

## port (p)

Use the following command to manage NVMe-oF ports:

```
xnr_cli nvmf port
```

You may also use `p` in the place of `port`:

```
xnr_cli nvmf p
```

The following subcommands are available for the `port` command:

<code>add</code>	Create an NVMe-oF port.
<code>remove</code>	Delete an NVMe-oF port.
<code>show</code>	Show NVMe-oF ports.

## port add

Use the following command to create an NVMe-oF port.

```
xnr_cli nvmf port create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.PortAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF port in the <code>name/uuid</code> format, where:
<code>--name</code>	
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF port.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF port. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the subsystem. Either the NQN of the subsystem or its name/UUID must be specified.
<code>--nqn</code>	

"subsys.nqn"

<code>-s</code>	Specify the name of the NVMe-oF subsystem in the <code>name/uuid</code> format, where:
<code>--subsystem</code>	

"subsys.id"

- `name` is the name of the NVMe-oF port.
- `uuid` is the UUID of the NVMe-oF port.

Either `name` or `uuid` is required.

<code>-a</code>	Specify the name of the network address configuration in the <code>name/uuid</code> format, where:
-----------------	--

<code>--address</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF port.</li> </ul>
<code>"address"</code>	<ul style="list-style-type: none"> <li>• <code>uuid</code> is the UUID of the NVMe-oF port.</li> </ul>

Either `name` or `uuid` is required.

---

<code>-t</code>	Specify the name(s) of the transport(s) in the <code>name/uuid</code>
<code>--transports</code>	format, where:
<code>"transports"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF port.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF port.</li> </ul>

Either `name` or `uuid` is required.

---

<code>-i</code>	Specify the NVMe-oF service ID (network port) to use. By
<code>--svcid</code>	default, the value is set to 4420.
<code>"svc_id"</code>	

---

<code>-d</code>	Specify the default ANA state for ANA groups on this port:
<code>--default-ana</code>	<code>optimized</code> , <code>non_optimized</code> , or <code>inaccessible</code> . By default, the
<code>"ana_state"</code>	value is set to <code>optimized</code> .

In the gRPC API, this field is an enum:

- `1` - Optimized
- `2` - Non-optimized
- `3` - Inaccessible

xnr\_cli examples:

```
xnr_cli nvme port add -n port0 -t transport0 -a address0 -s
subsystem0
```

grpcurl example:

```
grpcurl -plaintext -d '{
  "id": { "name": "port0" },
  "transports": [{ "name": "transport0" }],
  "address": { "name": "address0" },
  "subsys": { "id": { "name": "subsystem0" } }
}' localhost:8000 xnr.Nvmf.PortAdd
```

## port remove

Use the following command to remove an NVMe-oF port.

```
xnr_cli nvmf port remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.PortRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the NVMe-oF port in the <code>name/uuid</code> format, where:
<code>--name</code>	
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF port.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF port.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli nvmf port remove -n port0
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"port0"}]}' \
localhost:8000 xnr.Nvmf.PortRemove
```

## port show

Use the following command to show NVMe-oF ports.

```
xnr_cli nvmf port show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.PortShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the names of the NVMe-oF ports in the
<code>--names</code>	<code>name/uuid</code> format, where:
<code>"ids"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the NVMe-oF port.</li> <li>• <code>uuid</code> is the UUID of the NVMe-oF port.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli nvmf port show
xnr_cli nvmf port show -n port0,port1
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.PortShow
grpcurl -plaintext -d '{"ids":[{"name":"port0"}, {"name":"port1"}]}'
\
localhost:8000 xnr.Nvmf.PortShow
```

## ana (a)

Use the following command to manage Asymmetric Namespace Access:

```
xnr_cli nvme ana
```

You may also use `a` in the place of `ana`:

```
xnr_cli nvme a
```

The following subcommands are available for the `ana` command:

<code>set</code>	Set the state of an ANA group.
<code>remove</code>	Remove a subsystem from an ANA group.

## ana set

Use the following command to update the state of an ANA group on a specific NVMe-oF port.

```
xnr_cli nvme ana set
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvme.PortSetAna
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-p</code>	Specify the name of the NVMe-oF port in the <code>name/uuid</code> format, where:
<code>--port</code>	
"port"	<ul style="list-style-type: none"> <li><code>name</code> is the name of the NVMe-oF port.</li> <li><code>uuid</code> is the UUID of the NVMe-oF port.</li> </ul>

Either `name` or `uuid` is required.

<code>-g</code>	Specify the identifier of the ANA group to update.
<code>--group</code>	
"group_id"	
<code>-s</code>	Specify the new state for this ANA group:
<code>--state</code>	

"state"

- `optimized` (1 in the gRPC API)
- `non_optimized` (2 in the gRPC API)
- `inaccessible` (3 in the gRPC API)

xnr\_cli examples:

```
xnr_cli nvmmf ana set -p port0 -g 1 -s non_optimized
```

grpcurl example:

```
grpcurl -plaintext -d '{"port": { "name": "port0" }, "group_id":
  "1", "state": 2}' \
localhost:8000 xnr.Nvmmf.PortSetAna
```

## ana remove

Use the following command to remove an ANA group assigned to a subsystem. There should not be any namespaces assigned to this group.

```
xnr_cli nvmmf ana remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmmf.SubsystemRemoveAna
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-q</code>	Specify the NQN of the subsystem. Either the NQN of the subsystem or its name/UUID must be specified.
<code>--nqn</code>	
<code>"fid.nqn"</code>	
<code>-s</code>	Specify the name of the NVMe-oF subsystem in the <code>name/</code>
<code>--subsystem</code>	<code>uuid</code> format, where:
<code>"fid.id"</code>	

- `name` is the name of the NVMe-oF port.
- `uuid` is the UUID of the NVMe-oF port.

Either `name` or `uuid` is required.

---

<code>-g</code>	Specify the identifier of the ANA group from which to re-
<code>--group</code>	move the subsystem.
<code>"group_id"</code>	

---

xnr\_cli examples:

```
xnr_cli nvmf ana remove -s subsystem0 -g 1
```

grpcurl example:

```
grpcurl -plaintext -d '{"fid": {"id": {"name": "subsystem0"}},
  "group_id": "1"}' \
localhost:8000 xnr.Nvmf.SubsystemRemoveAna
```

## host

Use the following command to manage hosts in xiRAID Opus. Adding a host to the configuration does not grant it access to any subsystems. Once a host is added to the xiRAID Opus configuration, you can add it to the subsystem's allowed hosts list by using the `subsystem access add` command.

```
xnr_cli nvmf host
```

The following subcommands are available for the `host` command:

<code>add</code>	Add a host to the configuration.
<code>remove</code>	Remove a host from the configuration.
<code>show</code>	Show configured hosts.

## add

Use the following command to add a host to the config.

```
xnr_cli nvmf host add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.HostAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the host in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"fid.id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the host.</li> <li>• <code>uuid</code> is the UUID of the host. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the host. To find the NQN of your host,
<code>--NQN</code>	run the command <code>nvme show-hostnqn</code> . If you are using xi-
<code>"fid.nqn"</code>	RAID Opus as the initiator, run the command <code>xnr_cli config get -n default_host_nqn</code> .

`xnr_cli` examples:

```
xnr_cli nvme host add -n host0 \
--nqn
nqn.2014-08.org.nvmexpress:uuid:3dd2228b-b1c2-4fc2-b5c2-3929ce45e57c
```

`grpcurl` example:

```
grpcurl -plaintext \
-d '{"fid": {"id": {"name": "host0"}, "nqn":
  "nqn.2014-08.org.nvmexpress:uuid:3dd2228b-b1c2-4fc2-b5c2-3929ce45e57c"} }' \
localhost:8000 xnr.Nvmf.HostAdd
```

## remove

Use the following command to remove a host from the config.

```
xnr_cli nvmf host remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.HostRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the host in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"fid.id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the host.</li> <li>• <code>uuid</code> is the UUID of the host.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-q</code>	Specify the NQN of the host.
<code>--NQN</code>	
<code>"fid.nqn"</code>	

`xnr_cli` examples:

```
xnr_cli nvmf host remove -n host0
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"fid": {"id": {"name": "host0"}}}' \
localhost:8000 xnr.Nvmf.HostRemove
```

## show

Use the following command to show configured hosts.

```
xnr_cli nvmf host show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.HostShow
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the names of the network address configurations in the <code>name/uuid</code> format, where:
<code>--names</code>	
"ids"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the network address configuration.</li> <li>• <code>uuid</code> is the UUID of the network address configuration.</li> </ul>
	Either <code>name</code> or <code>uuid</code> is required. Examples:
	<pre>-n test -n de416ea2-0b1e-40e9-9d15-3224d1152730 -n test/de416ea2-0b1e-40e9-9d15-3224d1152730</pre>
<code>-q</code>	Specify the NQN(s) of the host(s) to show.
<code>--nqns</code>	This option is relevant only for <code>xnr_cli</code> .
<code>-o</code>	Specify the output format: <code>table</code> or <code>json</code> . By default, the
<code>--output</code>	<code>table</code> format is used.
	This option is relevant only for <code>xnr_cli</code> .

`xnr_cli` examples:

```
xnr_cli nvmf host show
xnr_cli nvmf host show -n host0
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.Nvmf.HostShow
```

## raid

This section describes the `raid` command for managing RAIDs in xiRAID Opus.

```
xnr_cli raid
```

The following subcommands are available for the `raid` command:

<code>create</code>	Create a new RAID.
<code>destroy</code>	Delete the RAID without possibility to restore the RAID and data on it.
<code>restore</code>	Restore the RAID from the drive metadata.
<code>unload</code>	Remove (unload) the RAID with possibility to restore the RAID and save data on it.
<code>replace</code>	Replace or remove the drive from the RAID.
<code>show</code>	Show info about one or multiple RAIDs.
<code>import</code>	Import a RAID that was previously destroyed.
<code>service start</code>	Start the reconstruction (if necessary) and initialization processes.
<code>service stop</code>	Stop the reconstruction and initialization processes.
<code>service force-resync</code>	Initialize the resynchronization process.
<code>service init-force-finish</code>	Forcefully terminate the RAID initialization process and mark the RAID as initialized. Please note that data corruption may occur.
<code>service recon-force-finish</code>	Forcefully terminate the RAID reconstruction process, put the RAID back into the Online state and remove the <code>need_recon</code> (Need Reconstruction) flag. Please note that data corruption may occur.

To view the available functions for managing RAIDs through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Raid
```

## raid create

Use the following command to create a new RAID. New RAIDs are immediately available for IO operations.

```
xnr_cli raid create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Create
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID. If a UUID is not specified, it will be automatically generated by the system.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-l</code>	Specify the RAID level for this RAID. The following levels
<code>--level</code>	are supported: 0, 1, 5, 6, 7, 10, 50, 60, 70.
"level"	
<code>-c</code>	Specify the chunk size in kilobytes. A chunk represents
<code>--chunk-size</code>	the amount of data that is written to each drive within a
"chunk_size"	single stripe. For example, the size of a single stripe in a RAID 5 with 8 drives will be equal to <i>chunk-size multiplied</i>

by 7. If the chunk size is set to 48 KB, the stripe size will be 448 KB (64 multiplied by 7).

<code>-d</code>	Specify a comma-separated list of BDEVs to use in this RAID.
<code>--bdevs</code>	
<code>"bdevs"</code>	
<code>-g</code>	For RAID levels 50, 60, and 70, specify the number of drives in one RAID group.
<code>--group-size</code>	
<code>"group_size"</code>	
<code>-b</code>	Optionally, specify the RAID block size in bytes. By default, this value is 4096 bytes. The RAID block size can be set to 512 or 4096 bytes. Note that if the block size of the underlying devices is 4096 bytes, the RAID block size cannot be set to 512 bytes.
<code>--block-size</code>	
<code>"block_size"</code>	

xnr\_cli examples:

```
xnr_cli raid create -n test -l 1 -d
  nvme.8a46f40d82bc9c3e.0,nvme.8a46f40d82bc9c3e.1
```

grpcurl example:

```
grpcurl -plaintext \
  -d '{
    "id": [{"name": "test"}],
    "level": "1",
    "bdevs": [
      {"name": "nvme.8a46f40d82bc9c3e.0"},
      {"name": "nvme.8a46f40d82bc9c3e.1"}
    ]
  }' \
localhost:8000 xnr.Raid.Create
```

## raid destroy

Use the following command to destroy a RAID. This command does not delete data on drives in the RAID. Destroyed RAID can be restored by using the `xnr_cli raid import` command.

```
xnr_cli raid destroy
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Destroy
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"><li>• <code>name</code> is the name of the RAID.</li><li>• <code>uuid</code> is the UUID of the RAID.</li></ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli raid destroy -n test
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}]}' localhost:8000
xnr.Raid.Destroy
```

## raid restore

Use the following command to restore a RAID that was previously unloaded:

```
xnr_cli raid restore
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Restore
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-a</code>	Specify this flag to restore all RAIDs that can be restored.
<code>--all</code>	
<code>"all"</code>	

`xnr_cli` examples:

```
xnr_cli raid restore -n test
xnr_cli raid restore --all
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}]}' localhost:8000
xnr.Raid.Restore
grpcurl -plaintext -d '{"all":"true"}' localhost:8000
xnr.Raid.Restore
```

## raid unload

Use the following command to unload (temporarily stop) a RAID. Once you've unloaded a RAID, it is no longer available for IO operations. The RAID status changes to offline, and the autostart flag is set to false.



If you plan to restore this RAID later, it is essential that you save its name or UUID. Unloaded RAID's are not shown in the output of the `xnr_cli raid show` command, so you will not be able to retrieve the name or UUID after unloading.

```
xnr_cli raid unload
```



Unloaded RAID's can be restored by using the `raid restore` command.

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Unload
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

`-n`  


---

`--name`  


---

Specify the name of the RAID in the `name/uuid` format, where:

- `name` is the name of the RAID.
- `uuid` is the UUID of the RAID.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`-a`  


---

`--all`  


---

Specify this flag to unload all RAID's in the system.

`xnr_cli` examples:

```
xnr_cli raid unload -n test
xnr_cli raid unload --all
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}]}' localhost:8000
xnr.Raid.Unload
grpcurl -plaintext -d '{"all":"true"}' localhost:8000
xnr.Raid.Unload
```

## raid replace

Use the following command to replace a drive in a RAID. This command can also be used to remove a drive from the RAID and mark it as missing.

```
xnr_cli raid replace
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Replace
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:

"id"	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>
------	--

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-p</code>	Specify the drive position in the RAID. Use the <code>raid show</code>
<code>--number</code>	command to determine the position of the drive.

"position"	
------------	--

<code>-d</code>	Specify a BDEV to replace the drive with. To remove the
<code>--bdev</code>	drive and mark it as missing, pass <code>null</code> as the value for
"bdev"	this option:

```
-d null
```

xnr\_cli examples:

```
xnr_cli raid replace -n test -p 0 -d nvme.8a46f40d82bc9c3e.2
```

grpcurl example:

```
grpcurl -plaintext \  
  -d '{  
    "id": [{"name": "test"}],  
    "position": "0",  
    "bdev": [  
      {"name": "nvme.8a46f40d82bc9c3e.2"}  
    ]  
  }' \  
localhost:8000 xnr.Raid.Replace
```

## raid show

Use the following command to display information about RAIDs managed by xiRAID Opus.

```
xnr_cli raid show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

To display information about a specific RAID, specify its name (`-n`, `--name`).

<code>-n</code>	Specify the name(s) of the RAID in the <code>name/uuid</code> format,
<code>--names</code>	where:

<code>"ids"</code>	<ul style="list-style-type: none"><li>• <code>name</code> is the name of the RAID.</li><li>• <code>uuid</code> is the UUID of the RAID.</li></ul>
--------------------	---

Either `name` or `uuid` is required. Examples:

```
-n test
-n test1,test2
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

---

**-U**  
**--units** Specify the unit of measurement: `b`, `kb`, `mb`, `gb`, or `tb`. By default, `gb` is used. This option cannot be used if the output format is set to `json`.

This option is relevant only for `xnr_cli`.

---

**-o**  
**--output** Specify the output format: `compact`, `table`, `json`, `tiny`. By default, the `table` format is used.

This option is relevant only for `xnr_cli`.

`xnr_cli` examples:

```
xnr_cli raid show -n test
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"ids":[{"name":"test"}]}' localhost:8000
xnr.Raid.Show
```

## raid import

Use the following command to import a RAID that was previously destroyed.

```
xnr_cli raid import
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Import
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The following options are available for this command. The `-d (--bdevs)` option is required for this command, while the other options are optional and can be skipped.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>-r</code>	Specify a new name for this RAID. This may be helpful if
<code>--rename</code>	the system already has a RAID with the same name as
<code>"new_name"</code>	that of the destroyed RAID.
<code>--allow_incomplete</code>	Use this flag to import the RAID even without some of the
<code>"allow_incomplete"</code>	drives that were used in the original RAID. Note that if
	the number of drives is not sufficient to rebuild the RAID,
	the RAID will not be imported. To start the reconstruction
	process after importing the RAID, <b>replace</b> the missing dri-
	ves and start the RAID reconstruction process.
<code>-d</code>	Specify a comma-separated list of BDEVs that were used
<code>--bdevs</code>	in the previously destroyed RAID.
<code>"bdevs"</code>	

xnr\_cli examples:

```
xnr_cli raid import -n test -r test1 -d
nvme.8a46f40d82bc9c3e.0,nvme.8a46f40d82bc9c3e.1
```

grpcurl example:

```

grpcurl -plaintext \
  -d '{
    "id": [{"name": "test"}],
    "new_name": "test1",
    "bdevs": [
      {"name": "nvme.8a46f40d82bc9c3e.0"},
      {"name": "nvme.8a46f40d82bc9c3e.1"}
    ]
  }' \
localhost:8000 xnr.Raid.Import

```

## raid service

This section describes the `raid service` command for managing the RAID initialization, resynchronization and reconstruction processes in xiRAID Opus.

```
xnr_cli raid service
```

The following subcommands are available for the `raid service` command:

service start	Start the reconstruction (if necessary) and initialization processes.
service stop	Stop the reconstruction and initialization processes.
service force-resync	Initialize the resynchronization process.
service init-force-finish	Forcefully terminate the RAID initialization process and mark the RAID as initialized. Please note that data corruption may occur.
service recon-force-finish	Forcefully terminate the RAID reconstruction process, put the RAID back into the Online state and remove the <code>need_recon</code> (Need Reconstruction) flag. Please note that data corruption may occur.

To view the available functions for managing the RAID initialization, resynchronization and reconstruction processes through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe .xnr.RaidCommandRequest
```

## start

Use the following command to start the reconstruction (if necessary) and initialization processes.

```
xnr_cli raid service start
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Command
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"><li>• <code>name</code> is the name of the RAID.</li><li>• <code>uuid</code> is the UUID of the RAID.</li></ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli raid service start -n test
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}],
  "cmd":"SERVICE_START"}' \
localhost:8000 xnr.Raid.Command
```

## stop

Use the following command to stop the reconstruction and initialization processes.

```
xnr_cli raid service stop
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Command
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

`xnr_cli` examples:

```
xnr_cli raid service stop -n test
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}],
  "cmd":"SERVICE_STOP"}' \
localhost:8000 xnr.Raid.Command
```

## force-resync

Use the following command to initialize the resynchronization process.

```
xnr_cli raid service force-resync
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Command
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

xnr\_cli examples:

```
xnr_cli raid service force-resync -n test
```

grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}],
  "cmd":"FORCE_RESYNC"}' \
localhost:8000 xnr.Raid.Command
```

## init-force-finish

Use the following command to forcefully terminate the RAID initialization process and mark the RAID as initialized. Please note that data corruption may occur.

```
xnr_cli raid service init-force-finish
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Command
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required. Additionally, the `--force` flag must be specified to terminate the initialization process.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:

- "id"
- `name` is the name of the RAID.
  - `uuid` is the UUID of the RAID.

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

---

`--force` Specify this flag to forcefully terminate the initialization process. If this flag is not specified, the initialization process will not be terminated.

This option is relevant only for `xnr_cli`.

`xnr_cli` examples:

```
xnr_cli raid service init-force-finish -n test --force
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}],
  "cmd":"INIT_FORCE_FINISH"}' \
localhost:8000 xnr.Raid.Command
```

## recon-force-finish

Use the following command to forcefully terminate the RAID reconstruction process, put the RAID back into the Online state and remove the `need_recon` (Need Reconstruction) flag. Please note that data corruption may occur.

```
xnr_cli raid service recon-force-finish
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Raid.Command
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

The name (`-n`, `--name`) option is required. Additionally, the `--force` flag must be specified to terminate the reconstruction process.

<code>-n</code>	Specify the name of the RAID in the <code>name/uuid</code> format,
<code>--name</code>	where:
<code>"id"</code>	<ul style="list-style-type: none"> <li>• <code>name</code> is the name of the RAID.</li> <li>• <code>uuid</code> is the UUID of the RAID.</li> </ul>

Either `name` or `uuid` is required. Examples:

```
-n test
-n de416ea2-0b1e-40e9-9d15-3224d1152730
-n test/de416ea2-0b1e-40e9-9d15-3224d1152730
```

<code>--force</code>	Specify this flag to forcefully terminate the reconstruction process. If this flag is not specified, the reconstruction process will not be terminated.
----------------------	---

This option is relevant only for `xnr_cli`.

`xnr_cli` examples:

```
xnr_cli raid service recon-force-finish -n test --force
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"id":[{"name":"test"}],
  "cmd":"RECON_FORCE_FINISH"}' \
localhost:8000 xnr.Raid.Command
```

## config

This section describes the `config` command for managing the RAID engine configuration.

```
xnr_cli config
```

The following subcommands are available for the `config` command:

<code>set</code>	Update one of the RAID engine configuration parameters
<code>get</code>	Display the current RAID engine configuration

To view the available functions for managing the RAID engine configuration through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Config
```

## config set

Use the following command to update one of the RAID engine configuration parameters.

```
xnr_cli config set
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Config.Set
```



Some of the system parameters are read-only and cannot be modified. If you attempt to modify the value of a read-only parameter, the system will return an error.

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

`-n` Specify the name of the system parameter to update.

`--name`

`"name"`

`-v` Specify the new value for this parameter.

`--value`

`"value"`

`xnr_cli` example:

```
xnr_cli config set -n resync_enabled -v false
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"name":"resync_enabled", "value":"false"}' \
localhost:8000 xnr.Config.Set
```

## config get

Use the following command to display the current RAID engine configuration.

```
xnr_cli config get
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Config.Get
```

Below is an example of the expected output:

```
version : 1.3.0, build: 10910, tag: ver1.3.0-d, date:
  2026-02-09T12:41:09+00:00
commit_base : fc46836dcfefc83cf8f1bc4f46604750c3035160
commit_spdk : 42cd57263c541c0a39940c058e9d4ff5cc298b54
node_uuid : b2d1bae1-500d-4b50-a792-e1e6b1b95e8b
default_host_nqn :
  nqn.2014-08.org.nvmexpress:uuid:b2d1bae1-500d-4b50-a792-e1e6b1b95e8b
cpu_mask : 0xf
hugemem_2mb : 8192
skip_drive_fresh_check : false
resync_enabled : true
disable_iommu_reset_by_removal : false
log_flags :
log_level : NOTICE
log_print_level : NOTICE
engineering_mode : false
config_format_version : 3
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Optionally, specify the name of the parameter to display
<code>--name</code>	the current value for. For example, <code>--name version</code> . If this
<code>"name"</code>	option is not specified, all system parameters are displayed.

`xnr_cli` example:

```
xnr_cli config get --name version
```

grpcurl example:

```
grpcurl -plaintext -d '{"name":"version"}' localhost:8000  
xnr.Config.Get
```

## device-manager (dm)

This section describes the device-manager command for managing devices in xiRAID Opus.

```
xnr_cli device-manager
```

You may also use `dm` in the place of `device-manager`.

```
xnr_cli dm
```

The following subcommands are available for the `device-manager` command:

show	Show devices that can be managed by xiRAID Opus
nvme attach	Attach a device to xiRAID Opus.
nvme detach	Detach a device from xiRAID Opus.
nvme reset-driver	Reset drivers for a local NVMe device.
network add-conn	Add a network connection to a remote storage target.
network rem-conn	Delete a path to a network devic.
network remove-dev	Remove a network device and all of its paths.
network set-mp-policy	Configure the multipath policy for remote storage targets with multiple network connections.
blacklist add	Add a device to the blacklist to prevent administrators from attaching this device to xiRAID Opus.
blacklist remove	Remove a device from the blacklist

To view the available functions for managing RAIDs through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Device
```

## device-manager show

Use the following command to show devices that can be managed by xiRAID Opus.

```
xnr_cli device-manager show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify a list of comma-separated drive identifiers for which to display detailed information.
<code>--id</code>	
"ids"	
<code>-o</code>	Specify the output format: <code>table</code> , <code>json</code> , or <code>raw</code> . By default, the <code>table</code> format is used.
<code>--output</code>	
	This option is relevant only for <code>xnr_cli</code> .
<code>-f</code>	Specify a comma-separated list of filters to use. The following filters are available:
<code>--filter</code>	
"filters"	<ul style="list-style-type: none"> <li>• <code>managed_by_us</code> - whether the device is managed by xiRAID Opus (<code>true</code>, <code>false</code>, <code>1</code>, <code>0</code>, <code>TRUE</code>, <code>FALSE</code>)</li> <li>• <code>dev_absent</code> - whether the device is available on the system (<code>true</code>, <code>false</code>, <code>1</code>, <code>0</code>, <code>TRUE</code>, <code>FALSE</code>)</li> <li>• <code>dev_type</code> - device type (<code>virtio</code>, <code>nvme</code>, or <code>network</code>)</li> <li>• <code>transport</code> - transport type used to connect to the device (<code>tcp</code>, <code>rdma</code>, or <code>pcie</code>)</li> <li>• <code>nqn</code> - NQN of the device</li> <li>• <code>ip</code> - IP address of the device</li> <li>• <code>raid_usage</code> - whether the device attach to the Device Managed is used in a RAID:</li> </ul>

- `any_drive_unused` - match devices that are not used in a RAID (this filter is meant to be used with another filter)
- `all_drives_in_use` - display all devices used in RAIDs
- `no_drives_in_use` - match devices that are not used in RAIDs
- `any_drive_in_use` - match devices that are used in a RAID (this filter is meant to be used with another filter)
- `blacklisted` - whether the device is blacklisted (`true`, `false`, `1`, `0`, `TRUE`, `FALSE`)

When multiple filters are used together, the logical AND is applied:

```
Shows all devices on RDMA transport that are
not used in any RAID
-f transport=rdma,raid_usage=no_drives_in_use
```

When multiple values are used within a single filter, the logical OR is applied:

```
Shows all devices on RDMA or TCP transport
-f transport=rdma,transport=tcp
```

xnr\_cli examples:

```
xnr_cli dm show -i nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
xnr_cli dm show -f managed_by_us=true,dev_type=nvme
```

grpcurl example:

```
grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ]
}' \
localhost:8000 xnr.Device.Show
```

## device-manager nvme

Use the following command to manage local NVMe devices in xiRAID Opus:

```
xnr_cli device-manager nvme
```

You may also use `net` in the place of `network` and `dm` in the place of `device-manager`:

```
xnr_cli dm nvme
```

The following subcommands are available for the `network` command:

attach	Attach a local NVMe device.
detach	Detach a local NVMe device.
reset-driver (res)	Reset drivers for a local NVMe device.

## device-manager nvme attach

Use the following command to attach a local NVMe device to xiRAID Opus.

```
xnr_cli device-manager nvme attach
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NvmePciAttach
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify a comma-separated list of identifiers of drives to attach to xiRAID Opus.
<code>--ids</code>	
<code>"ids"</code>	
<code>-f</code>	Specify this flag to forcefully attach drives to xiRAID Opus.
<code>--force</code>	This flag may be necessary if the device-manager tool reports that the device is used by the operating system or a third-party application, but you are confident that the device is not actually in use.
<code>"force"</code>	

Devices in the **blacklist** cannot be attached even if the `--force` flag is specified.

xnr\_cli examples:

```
xnr_cli dm nvme attach --ids
  nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
xnr_cli dm nvme attach --ids
  nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2 --force
```

grpcurl example:

```
grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ]
}' \
localhost:8000 xnr.Device.NvmePciAttach

grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ],
  "force":"true"
}' \
localhost:8000 xnr.Device.NvmePciAttach
```

## device-manager nvme detach

Use the following command to detach a local NVMe device from xiRAID Opus. Detached devices will become available to the operating system. These devices will be configured to use drivers that had been used before they were attached to xiRAID Opus.

```
xnr_cli device-manager nvme detach
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NvmePciDetach
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify a comma-separated list of identifiers of drives to detach from xiRAID Opus.
<code>--ids</code>	
<code>"ids"</code>	

`xnr_cli` examples:

```
xnr_cli dm nvme detach --ids
  nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
```

`grpcurl` example:

```
grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ]
}' \
localhost:8000 xnr.Device.NvmePciDetach
```

## device-manager nvme reset-driver

Use the following command to detach a device from xiRAID Opus and reset its drivers to the OS default.

```
xnr_cli device-manager nvme reset-driver
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NvmePciDriverReset
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify a comma-separated list of identifiers of devices to
<code>--ids</code>	reset.
<code>"ids"</code>	

xnr\_cli examples:

```
xnr_cli dm nvme reset-driver --ids
nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
```

grpcurl example:

```
grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ]
}' \
localhost:8000 xnr.Device.NvmePciDriverReset
```

## device-manager network (net)

Use the following command to manage network devices in xiRAID Opus:

```
xnr_cli device-manager network
```

You may also use `net` in the place of `network` and `dm` in the place of `device-manager`:

```
xnr_cli dm net
```

The following subcommands are available for the `network` command:

<code>add-conn</code>	Add a path to a network device.
<code>rem-conn</code>	Remove a path to a network device.
<code>remove-dev</code>	Remove a network device and all of its paths.
<code>set-mp-policy</code>	Update the multipath policy for a network device.

## device-manager network add-conn (ac)

Use the command below to add a network connection to a remote storage target.

```
xnr_cli device-manager network add-conn
```

You may also use `ac` in the place of `add-conn`:

```
xnr_cli dm net ac
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NetworkConnectionAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-q</code>	Specify the NQNs of the remote storage targets for which to add a network connection. If this target has not been configured in xiRAID Opus, it will be automatically added with the provided path.
<code>--nqn</code>	
"nqns"	
<code>-t</code>	Specify the transport type for this connection: <code>TCP</code> or <code>RDMA</code> .
<code>--trtype</code>	
"transport"	
<code>-a</code>	Specify the path to the remote storage target in the <code>&lt;addr-Fam&gt;/&lt;address&gt;/&lt;svcId&gt;</code> format, where:
<code>--path</code>	
"connections"	

- `addrFam` - NVMe-oF transport address family: `ipv4`, `ipv6`, `ib`, or `fc`. This value can be omitted if you use `ipv4` or `ipv6`.
- `address` - NVMe-oF transport address (IP address).
- `svcId` - Transport service ID (port number).

Examples:

```
-a ipv6/abcd::08:08:aa/4420
```

```
-a 1.2.3.4/4420
```

```
-a ipv4/4.5.6.7/4420
```

<code>-x</code>	Specify which multipath mode to use:
<code>--mp</code>	
"mp"	

"mode"

- `disable` or `no` - In this mode, multipath is disabled. Only one path can be added for a single remote storage target.
- `failover` or `fo` - Multipath operates in the failover mode. In this mode, only one path at a time can be active.
- `multipath` or `mp` - Multipath mode. In this mode, multiple paths can be active simultaneously. See [device-manager network set-mp-policy \(mpp\)](#) for details about configuring the multipath policy.

In the gRPC API, this field is an enum:

- `0` - Multipath disabled.
- `1` - Failover mode.
- `2` - Multipath mode.

xnr\_cli examples:

```
xnr_cli dm net ac -q nqn.2025-04.io.xinnor:cnodel -t TCP -a
172.16.133.242 -x no
```

grpcurl example:

```

grpcurl -plaintext -d '{
  "transport": {
    "type": "TT_TCP"
  },
  "nqns": [
    "nqn.2025-04.io.xinnor:cnodel"
  ],
  "connections": [
    {
      "addr": "172.16.133.242",
      "family": "AFAM_IPV4"
    }
  ],
  "mode": "MP_DISABLE"
}' localhost:8000 xnr.Device.NetworkConnectionAdd

```

## device-manager network rem-conn (dc)

Use the following command to delete a path to a network device:

```
xnr_cli device-manager network rem-conn
```

You may also use `dc` in the place of `rem-conn`:

```
xnr_cli dm net dc
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NetworkConnectionRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-q</code>	Specify the NQNs of the remote storage targets for which to add a network connection. If this target has not been configured in xiRAID Opus, it will be automatically added with the provided path.
<code>--nqn</code>	
<code>"nqns"</code>	
<code>-a</code>	Specify the path to the remote storage target in the <code>&lt;addr-Fam&gt;/&lt;address&gt;/&lt;svcId&gt;</code> format, where:
<code>--path</code>	

- `addrFam` - NVMe-oF transport address family: `ipv4`, `ipv6`, `ib`, or `fc`. This value can be omitted if you use `ipv4` or `ipv6`.
- `address` - NVMe-oF transport address (IP address).
- `svcId` - Transport service ID (port number).

Examples:

```
-a ipv6/abcd::08:08:aa/4420
```

```
-a 1.2.3.4/4420
```

```
-a ipv4/4.5.6.7/4420
```

If you delete the only path to a network device, the network device itself will also be removed from xiRAID Opus.

xnr\_cli examples:

```
xnr_cli dm net dc -q nqn.2025-04.io.xinnor:cnodel -a 172.16.133.242
```

grpcurl example:

```
grpcurl -plaintext -d '{
  "nqns": [
    "nqn.2025-04.io.xinnor:cnodel"
  ],
  "connections": [
    {
      "addr": "172.16.133.242",
      "family": "AFAM_IPV4"
    }
  ]
}' localhost:8000 xnr.Device.NetworkConnectionRemove
```

## device-manager network remove-dev (rm)

Use the following command to remove a network device and all of its paths:

```
xnr_cli device-manager network remove-dev
```

You may also use `rm` in the place of `remove-dev`:

```
xnr_cli dm net rm
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NetworkDetach
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify identifiers of devices that should be removed. To obtain device identifiers, use the <code>xnr_cli dm show</code> command.
<code>--ids</code>	
<code>"ids"</code>	

`xnr_cli` examples:

```
xnr_cli dm net rm -i q374.e3d6b1df.83700f0d
```

grpcurl example:

```
grpcurl -plaintext -d '{"ids":[{"name":"q374.e3d6b1df.83700f0d"}]}' \
localhost:8000 xnr.Device.NetworkDetach
```

## device-manager network set-mp-policy (mpp)

Use the command below to configure the multipath policy for remote storage targets with multiple network connections.

```
xnr_cli device-manager network set-mp-policy
```

You may also use `mpp` in the place of `set-mp-policy`:

```
xnr_cli dm net mpp
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.NetworkMultipathSet
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify identifiers of devices for which to update the multipath policy.
<code>--ids</code>	
<code>"ids"</code>	
<code>-p</code>	Specify the multipath policy:
<code>--policy</code>	
<code>"multipath.policy"</code>	
	<ul style="list-style-type: none"> <li>• <code>active_active</code> (1 in the gRPC API) - In this mode, all paths are actively used for I/O operations simultaneously.</li> <li>• <code>active_passive</code> (2 in the gRPC API) - In this mode, only one path is actively used for I/O operations at any given time.</li> </ul>
<code>-s</code>	For devices that use the <code>active_active</code> multipath policy, specify the selector:
<code>--selector</code>	
<code>"multipath.selector"</code>	<ul style="list-style-type: none"> <li>• <code>rr</code> (1 in the gRPC API) - With this selector, I/O requests are distributed in a round-robin fashion across all available paths.</li> <li>• <code>mq</code> (2 in the gRPC API) - With this selector, I/O requests are routed to the path with the lowest number of pending I/O requests.</li> </ul>
<code>-r</code>	For devices that use the <code>active_active</code> multipath policy with the <code>rr</code> (roundrobin) selector, specify the minimum number of I/O operations that should be routed to the current I/O path before switching to another path.
<code>--min_io</code>	
<code>"multipath.rr_min_io"</code>	

xnr\_cli examples:

```
xnr_cli dm net mpp -i q374.e3d6b1df.83700f0d -p active_active
```

grpcurl example:

```
grpcurl -plaintext -d '{
  "ids": [
    { "name": "q374.e3d6b1df.83700f0d" }
  ],
  "multipath": {
    "policy": "ACTIVE_ACTIVE"
  }
}' localhost:8000 xnr.Device.NetworkMultipathSet

grpcurl -plaintext -d '{
  "ids": [
    { "name": "q374.e3d6b1df.83700f0d" }
  ],
  "multipath": {
    "policy": "ACTIVE_ACTIVE",
    "selector": "ROUND_ROBIN",
    "rr_min_io": 10
  }
}' localhost:8000 xnr.Device.NetworkMultipathSet
```

## device-manager blacklist (bl)

This section describes the `blacklist` command for managing the device blacklist in xiRAID Opus. Devices added to the blacklist cannot be attached to xiRAID Opus.

```
xnr_cli device-manager blacklist
```

You may also use `bl` in the place of `blacklist` and `dm` in the place of `device-manager`.

```
xnr_cli dm bl
```

The following subcommands are available for the `blacklist` command:

add	Add a device to the blacklist.
-----	--------------------------------

---

remove	Remove a device from the blacklist.
--------	-------------------------------------

Devices in the blacklist will have their status set to `blacklisted` in the output of `xnr_cli dm show`.

## device-manager blacklist add

Use the following command to add a device to the blacklist to prevent administrators from attaching this device to xiRAID Opus:

```
xnr_cli device-manager blacklist add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.BlacklistAdd
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify identifiers of devices that should be added to the
<code>--ids</code>	blacklist. Use the <code>xnr_cli device-manager show</code> command
<code>"ids"</code>	to get device identifiers.

`xnr_cli` examples:

```
xnr_cli dm bl add -i nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
```

`grpcurl` example:

```
grpcurl -plaintext \  
-d '{  
  "ids": [  
    {"name": "nvme.8a46f40d82bc9c3e.1"},  
    {"name": "nvme.8a46f40d82bc9c3e.2"}  
  ]  
' \  
localhost:8000 xnr.Device.BlacklistAdd
```

## device-manager blacklist remove

Use the following command to remove a device from the blacklist:

```
xnr_cli device-manager blacklist remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Device.BlacklistRemove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-i</code>	Specify identifiers of devices that should be removed from the blacklist. Use the <code>xnr_cli device-manager show</code> command to get device identifiers.
<code>--ids</code>	
<code>"ids"</code>	

`xnr_cli` examples:

```
xnr_cli dm bl remove -i
nvme.8a46f40d82bc9c3e.1,nvme.8a46f40d82bc9c3e.2
```

grpcurl example:

```
grpcurl -plaintext \
-d '{
  "ids":[
    {"name":"nvme.8a46f40d82bc9c3e.1"},
    {"name":"nvme.8a46f40d82bc9c3e.2"}
  ]
}' \
localhost:8000 xnr.Device.BlacklistRemove
```

## license

This section describes the `license` command for managing the xiRAID Opus license.

```
xnr_cli license
```

The following subcommands are available for the `license` command:

<code>add</code>	Add a new license.
<code>show</code>	Show the current license.
<code>remove</code>	Remove all installed licenses.

To view the available functions for managing the xiRAID Opus license through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.License
```

## license add

Use the following command to add or update a xiRAID Opus license.

```
xnr_cli license add
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.License.Add
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-p</code>	Specify the path to the license file.
<code>--path</code>	This option is not supported by the gRPC API.
<code>-k</code>	Alternatively, specify the license key.
<code>--key</code>	
<code>"key"</code>	

`xnr_cli` examples:

```
xnr_cli license add -p license.txt
xnr_cli license add -k <license_key>
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"key": "<license_key>"}' localhost:8000
xnr.License.Add
```

## license show

Use the following command to show details about the current xiRAID Opus license. The output of this command also contains your hardware key (`hwkey`). The hardware key is used to generate xiRAID Opus licenses.

```
xnr_cli license show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.License.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-o</code>	Optionally, specify the output format. Note that the default
<code>--output</code>	<code>table</code> format does not contain some information, such as the hardware key and license keys.

- `table` (default)
- `json`

This option is relevant only for `xnr_cli`.

`xnr_cli` example:

```
xnr_cli license show
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.License.Show
```

## license remove

Use the following command to remove your xiRAID Opus license.

```
xnr_cli license remove
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.License.Remove
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>--force</code>	Specify this flag to delete the current xiRAID Opus license.
----------------------	--

This option is relevant only for `xnr_cli`.

`xnr_cli` example:

```
xnr_cli license remove --force
```

grpcurl example:

```
grpcurl -plaintext localhost:8000 xnr.License.Remove
```

## this

This section describes the `this` command for configuring the Opus CLI tool.

```
xnr_cli this
```

The following subcommands are available for the `this` command:

version	Display the current version of the Opus CLI tool.
server	Manage the server access configuration.
show-cmd-tree	Show all available Opus CLI commands in a tree view.

## this version

Run the following command to display the current version of the Opus CLI tool:

```
xnr_cli this version
```

## this server

This section describes the `this server` command for configuring access to Opus servers managed by the Opus CLI tool. This may be necessary if Opus CLI and Opus are installed on different machines or if you manage multiple Opus servers from a single machine with only Opus CLI installed.

```
xnr_cli this server
```

The following subcommands are available for the `this server` command:

add	Add an Opus server configuration.
show	Show existing Opus server configurations.
set-current	Set a configuration as default.

remove Remove a configuration.

## this server add

Use the following command to configure access to an Opus server.

```
xnr_cli this server add
```

Specify the following options:

<code>-n</code>	Specify a name for this Opus server. This name will be used only in the Opus CLI tool.
<code>--name</code>	
<code>-a</code>	Specify the IP address of the Opus server.
<code>--address</code>	
<code>-p</code>	Specify the port the Opus server listens on. By default, Opus servers listen on port 8080.
<code>--port</code>	

## this server show

Use the following command to show existing Opus server configurations.

```
xnr_cli this server show
```

The output contains the following information:

- `current-server` - The name of the Opus server that is used by default when you run commands in the Opus CLI tool.
- `all-servers` - A list of all configured Opus servers.
  - `name` - The name of the Opus server.
  - `endpoint` - The endpoint where the Opus server is available: `[address]:[port]`.

## this server set-current

Use the following command to configure of the existing Opus server configurations to be used by default when you run commands in the Opus CLI tool.

```
xnr_cli this server set-current
```

Specify the name of the server.

<code>-n</code>	Specify the name of the server to use by default. Use the
<code>--name</code>	<code>xnr_cli this server show command</code> to display the list of configured Opus servers and their names.

## this server remove

Use the following command remove an Opus server configuration.

```
xnr_cli this server remove
```

Specify the name of the server to remove.

<code>-n</code>	Specify the name of the server to remove. Use the <code>xnr_cli</code>
<code>--name</code>	<code>this server show command</code> to display the list of configured Opus servers and their names.

## this show-cmd-tree

Run the following command to show all available Opus CLI commands in a tree view:

```
xnr_cli this show-cmd-tree
```

To include experimental features in the output of show-cmd-tree, add the `-H` flag:

```
xnr_cli -H this show-cmd-tree
```

## vhost

This section describes the `vhost` command for managing vhost controller devices. This command can be used to expose Opus BDEVs (including RAIDs) to a client over VirtIO.

```
xnr_cli vhost
```

The following subcommands are available for the `vhost` command:

<code>create</code>	Create a new vhost controller device.
<code>show</code>	Show existing vhost controller devices managed by xiRAID Opus.
<code>destroy</code>	Destroy a vhost controller device.

To view the available functions for managing vhost controller devices through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Vhost
```

## vhost create

Use the following command to create a new vhost controller:

```
xnr_cli vhost create
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Vhost.BlockCreate
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the vhost controller.
<code>--name</code>	
<code>"id"</code>	
<code>-d</code>	Specify the name of the BDEV to expose.
<code>--bdev</code>	
<code>"bdev"</code>	
<code>-c</code>	Specify the socket file name for this controller.
<code>--ctrl</code>	
<code>"ctrl"</code>	
<code>-m</code>	Specify which CPU cores to use for the processing of IO load for this vhost controller device in the form of a cpumask. This cpumask must be a subset of the cpumask specified during the installation of xiRAID Opus. You can use the same cpumask for multiple vhost controller devices.
<code>--cpumask</code>	
<code>"cpu_mask"</code>	

`xnr_cli` examples:

```
xnr_cli vhost create -n vhost.1 -d nvme.c0a9eb3c13788900.0p1 -c
vhost.1.socket -m [0-3]
```

grpcurl example:

```
grpcurl -plaintext -d '{
  "id": [{ "name": "vhost.1" }],
  "bdev": [{ "name": "nvme.c0a9eb3c13788900.0p1" }],
  "ctrl": "vhost.1.socket",
  "cpu_mask": "0xF"
}' localhost:8000 xnr.Vhost.BlockCreate
```

## vhost show

Use the following command to display information about vhost controller devices managed by xiRAID Opus.

```
xnr_cli vhost show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Vhost.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name(s) of the vhost controller device for which to display information.
<code>--names</code>	
<code>"ids"</code>	
<code>-o</code>	Specify the output format: <code>table</code> or <code>json</code> . By default, data is presented in the <code>table</code> format.
<code>--output</code>	
	This option is relevant only for <code>xnr_cli</code> .

`xnr_cli` examples:

```
xnr_cli vhost show
xnr_cli vhost show -n vhost.1
xnr_cli vhost show -n vhost.1,vhost.2
```

## grpcurl example:

```
grpcurl -plaintext localhost:8000 xnr.Vhost.Show
grpcurl -plaintext -d '{"ids":[{"name":"vhost.1"}]}' \
localhost:8000 xnr.Vhost.Show
grpcurl -plaintext -d '{"ids":[{"name":"vhost.1"}], "ids":
[{"name":"vhost.2"}]}' \
localhost:8000 xnr.Vhost.Show
```

## vhost destroy

Use the following command to destroy a vhost controller device managed by xiRAID Opus.

```
xnr_cli vhost destroy
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Vhost.Destroy
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-n</code>	Specify the name of the vhost controller device to destroy.
<code>--name</code>	
<code>"id"</code>	

## xnr\_cli examples:

```
xnr_cli vhost destroy -n vhost.1
```

## grpcurl example:

```
grpcurl -plaintext -d '{"id":[{"name":"vhost.1"}]}' localhost:8000
xnr.Vhost.Destroy
```

## events

This section describes the `events` command for managing the event log.

```
xnr_cli events
```

The following subcommands are available for the `events` command:

<code>show</code>	Show events in the log.
-------------------	-------------------------

---

<code>clear</code>	Clear events in the log.
--------------------	--------------------------

To view the available functions for managing the xiRAID Opus event log through the gRPC API, run the following command:

```
grpcurl -plaintext localhost:8000 describe xnr.Event
```

## show

Use the following command to show events in the log:

```
xnr_cli events show
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Event.Show
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-c</code>	Optionally, specify the number of events to show. If this option is not specified, all events in the log will be displayed.
<code>--count</code>	
<code>"count"</code>	

`xnr_cli` examples:

```
xnr_cli events show
xnr_cli events show -c 2
```

`grpcurl` example:

```
grpcurl -plaintext localhost:8000 xnr.Event.Show
grpcurl -plaintext -d '{"count":"2"}' localhost:8000 xnr.Event.Show
```

## clear

Use the following command to clear events from the event log:

```
xnr_cli events clear
```

Alternatively, to use the gRPC API:

```
grpcurl -plaintext localhost:8000 xnr.Event.Clear
```

The following options are available for this operation. Options starting with `-` and `--` are for use with the `xnr_cli` tool, while options in quotes represent the option names in the gRPC API.

<code>-c</code>	Specify how many events to remove from the event log, beginning with the most recent entry.
<code>--count</code>	
<code>"count"</code>	
<code>--all</code>	Specify this flag to delete all events from the event log.
<code>"all"</code>	

`xnr_cli` examples:

```
xnr_cli events clear -c 1
xnr_cli events clear --all
```

`grpcurl` example:

```
grpcurl -plaintext -d '{"count":"1"}' localhost:8000 xnr.Event.Clear
grpcurl -plaintext -d '{"all":"true"}' localhost:8000
xnr.Event.Clear
```