Lightbits LightOS Administrator’s Guide
LightOS Version: v2.3.12

Lightbits Labs

January 2022
## Contents

### Introduction
- Lightbits Labs LightOS Product Family ........................................ 4
- Further Reading ............................................................................... 4

### Lightbits Storage Solution Consumption Models

### LightOS Data Protection

### LightOS Cluster Overview
- LightOS Cluster Topology ................................................................. 8
- LightOS Cluster Architecture ............................................................ 10
- Nodes .............................................................................................. 10
- Volume Assignments ........................................................................ 10
- NVMe/TCP MultiPath ........................................................................ 10
- Failure Domains .............................................................................. 12
- Dynamic Rebalancing ...................................................................... 12

### Common Administration Tasks
- Create a Volume ............................................................................. 15
- Create a Compressed Volume ............................................................ 15
- List Details About the Newly Created Volume ................................. 16

### Connecting Application Servers to LightOS
- Reviewing the LightOS Storage Cluster Connection Details .......... 18
- Obtaining the LightOS Cluster Subsystem NQN ............................. 18
- Obtaining the LightOS Nodes Data IP Addresses and TCP Ports...... 18
- Obtaining the Volume ACL String ................................................... 19
- Verifying TCP/IP Connectivity .......................................................... 19
- Connecting to the LightOS Cluster ................................................. 19
- Reviewing Block Device Details on the Application Server .......... 20

### Understanding a Volume Location in the LightOS Cluster
- Adding an NVMe SSD to a LightOS Storage Server ......................... 21
- Identifying Unused Slots .................................................................. 21
- Adding NVMe SSDs ......................................................................... 23

### Snapshots and Clones
- Volume Placement ........................................................................... 28
- Identifying a Failed SSD Drive .......................................................... 29
- Identifying a Node Failure in a LightOS Cluster .............................. 31
- Multi Tenancy CLI ...................................................................... 32

### Multi-Tenancy and LightOS
- Key Features .................................................................................. 32
  - “Projects” (tenants) support ........................................................... 32
  - Authentication (authN) ................................................................. 32
  - Authorization (authZ) ................................................................. 33
  - Role Based Access Control (RBAC) ............................................ 33
  - Common Multi Access Control (RBAC) .................................... 33
- IP ACL Configuration ................................................................. 34

### LightOS CLI Overview
- lbcli Command Syntax ................................................................. 38
- lbcli Command Summary ............................................................... 38
- Key Resources ................................................................................. 38
- Using the lbcli Commands ............................................................... 39
- lbcli Add NVMe Device ................................................................. 39
- lbcli Upgrade Server ................................................................. 39
- Upgrade Server from Remote Repo .............................................. 41
- Upgrade Server from File ............................................................ 41
- Upgrade Cluster ............................................................................. 43
- lbcli Create Server ................................................................. 45
Introduction

This guide is for storage administrators to operate, manage, and troubleshoot LightOS storage clusters.

It helps to get familiar with the Installation and Configuration Guide, which contains step-by-step information on how to install and configure a LightOS cluster.

In this guide, it is assumed that you have successfully installed and deployed a LightOS cluster or are working with an already installed and operational LightOS cluster, and are interested in the day-to-day operations of LightOS. It is important to familiarize yourself with the contents of this document, as some of the material will be a good reference for how to operate the cluster, as well as to know what to do when unexpected events occur.

Lightbits Labs LightOS Product Family

The following table is a list of Lightbits Labs supported products and their relationship to LightOS.

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightOS™</td>
<td>LightOS is a Lightbits Labs SDS that can be installed on standard x86 servers. A LightOS cluster is a group of servers with LightOS installed, and are managed as a cluster.</td>
</tr>
<tr>
<td>LightField™ Storage Acceleration Card</td>
<td>LightField is a hardware-based accelerator that can be optionally installed in a LightOS-enabled server and offloads some LightOS features such as compression.</td>
</tr>
<tr>
<td>SuperSSD</td>
<td>SuperSSD is a Lightbits Labs appliance offering that integrates LightOS on a Lightbits Labs pre-configured storage server.</td>
</tr>
</tbody>
</table>

Further Reading

This Administrator Guide is part of a documentation set that provides complete information about using Lightbits products.

This document set includes:

<table>
<thead>
<tr>
<th>Document</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightOS Installation and Configuration Guide</td>
<td>Contains instructions for installing the LightOS cluster software, installing the cluster client software, and connecting the cluster client to LightOS.</td>
</tr>
<tr>
<td>LightOS Installation Customer Addendum</td>
<td>Includes customer-specific passwords to access installation files. Also captures customer-specific configuration details if needed.</td>
</tr>
<tr>
<td>User’s Manual: LightOS REST and CLI API</td>
<td>Lists the low-level details for the REST API and CLI command usage. This document is typically used as a reference manual when building and administering the system.</td>
</tr>
</tbody>
</table>
Lightbits Storage Solution Consumption Models

Lightbits is an open storage platform solution that fits various deployment models for performance, footprint, and vendor selection. It can be consumed as software only, or in conjunction with the LightField™ hardware acceleration card and integrated onto your choice of storage server hardware. This open platform model gives you the flexibility to choose your preferred consumption model and even mix and match the deployment options. As such, Lightbits provides various solution offerings to match your individual needs.

The LightBox™ reference design delivers a fully-tested solution with management control and a full platform warranty from Lightbits.

The LightOS™ software solution with the LightField accelerator card delivers high performance with in-line compression and other data services on your choice of storage server hardware. For the simplest integration, a software-only LightOS option is also available.

<table>
<thead>
<tr>
<th>Lightbits Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="LightOS™" /></td>
<td>A LightOS license is a software-only solution that can be ported to different storage server hardware options and used with several Linux distributions.</td>
</tr>
<tr>
<td><img src="image" alt="LightOS™" /></td>
<td>LightOS software and the LightField hardware acceleration card are packaged together and integrated by a third party into your storage server hardware.</td>
</tr>
<tr>
<td><img src="image" alt="LightField™" /></td>
<td>The SuperSSD storage platform provides an integrated hardware and software solution that is fully validated and ready to deploy. This option includes LightOS software and the LightField hardware acceleration cards, plus the storage server hardware itself.</td>
</tr>
</tbody>
</table>
LightOS Data Protection

The Lightbits storage solution supports Erasure Coding (EC) for protection against SSD failure within the storage server. As shown in the figure below, this flexible self-healing feature prevents IO interruption in the case of SSD failure. Normal operation continues during reconstruction when a drive is removed.

![Diagram showing LightOS data Protection](image)

**Figure 1: LightOS data Protection**

**Lightbits Open Storage Platform Self-Healing Data Protection Rebuild Process**

The system stays in normal operation even if the drive is never returned. Replacing a missing drive does not trigger any new operations; the capacity of the storage server is simply re-expanded. The rebuild processes starts automatically following the failure or removal of an SSD. Erasure coding is enabled on a per-node basis in the hardware profile that was selected for a given customer installation. The *LightOS 2 Installation Guide* has information for enabling EC; see the `ec_enabled` variable description in Install Guide - Defining Configuration Files for Each “Ansible Host” (Server) in the Cluster.
LightOS Cluster Overview

This section provides you with information about the major components of the LightOS cluster software solution and how they work together.

It also contains recommended best practice tips for collecting information required to use the automated installation script. For the installation script to download and install the LightOS software onto your system’s storage nodes, you must have details about your specific environment—such as your specific networking details.

LightOS Cluster Topology

The following is a basic diagram that shows the components and resources required to automatically install LightOS software onto your servers in your data center.

![Diagram of LightOS Cluster Resources](image)

Based on the numbers next to each component or resource in the diagram, see the following table for a description of the components and resources in the LightOS cluster topology diagram.

**LightOS Cluster Topology Components Table**

<table>
<thead>
<tr>
<th>#</th>
<th>Component or Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lightbitslabs.com</td>
<td>Lightbits supplied configuration files and installation tools via “FTP” from our website. The LightOS software is maintained in a password-protected software repository, referred to as “The Lightbits Repo”.</td>
</tr>
<tr>
<td>2</td>
<td>dl.lightbitslabs.com</td>
<td>There are many publicly-available Linux repositories that are already configured in your environment. Some standard tools might need to be downloaded from these repos or updated Linux kernel files to allow your clients to perform NVMe/TCP functions. These standard files are not available from Lightbits but are part of your core operating system.</td>
</tr>
<tr>
<td>3</td>
<td>Preferred Linux repo.org</td>
<td></td>
</tr>
</tbody>
</table>

2022 Lightbits Labs
Proprietary And Confidential - Under NDA Only
<table>
<thead>
<tr>
<th>#</th>
<th>Component or Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Public Network Time Protocol Server</td>
<td>LightOS cluster nodes remain in sync using NTP. This is automatically configured by the Ansible installation script. However a custom NTP configuration is possible.</td>
</tr>
<tr>
<td>5</td>
<td>Management Network</td>
<td>To connect and configure a given server, the standard Secure Shell (SSH) protocol is used. Each server must be reachable and is configured over this management network. This network is separate from the network that LightOS will send and receive application data over. It is also possible to use the same network for both Management and Data Networks.</td>
</tr>
<tr>
<td>6</td>
<td>Data Network</td>
<td>Acts as the interconnection between the enterprise’s “Clients” or “Application Servers”. This is a separate network from the Management Network and carries all of the NVMe/TCP data traffic.</td>
</tr>
<tr>
<td>7</td>
<td>Cluster Installation Workstation</td>
<td>This is where you download the LightOS installation software. This is composed of an Ansible script and must be outside of the planned cluster. It will automatically download the LightOS cluster software files from the dl.lightbitslabs.com repo to each of the LightOS cluster servers. Note: For information about the system requirements for the installation workstation, see the System Requirements in the LightOS 2 Installation Guide.</td>
</tr>
<tr>
<td>8</td>
<td>LightOS Cluster Servers</td>
<td>The cluster servers store the application data. Each node is what essentially makes up the data storage portion of a LightOS cluster. All of the RPM files (from dl.lightbitslabs.com) are installed on these servers.</td>
</tr>
<tr>
<td>9</td>
<td>Clients</td>
<td>These are the enterprise’s application servers and are where your applications live. The “client” part of the cluster is connected to the LightOS cluster via the data network. It might be necessary for you to update the client “kernel” using a standard repo manager program such as “yum” in the case of CentOS or RHEL.</td>
</tr>
</tbody>
</table>
LightOS Cluster Architecture

The Lightbits™ LightOS™ cluster storage solution distributes services and replicates data across different LightOS servers to guarantee service and data availability when one or more LightOS servers experience transient or permanent failures. A cluster of LightOS servers replicates data internally and keeps it fully consistent and available in the presence of failures. From the perspective of clients accessing the data, data replication is transparent, and server failover is seamless.

LightOS also protects the storage cluster from additional failures not related to the SSDs (e.g., CPU, memory, NICs) failures, software failures, network failures, or rack power failures. It provides additional data security through in-server Erasure Coding (EC) that protects servers from SSD failures and enables non-disruptive maintenance routines that temporarily disable access to storage servers (e.g., TOR firmware upgrades).

The following sections describe the failure domain and volume components used in the LightOS cluster architecture.

Note: For more information about LightOS cluster architecture, see the Deploying Reliable High-Performance Storage with LightOS Whitepaper.

Nodes

Each server can be split into multiple logical nodes. Each logical node owns a specific set of SSDs and CPUs, and a portion of the RAM and NVRAM. The physical network can be shared or exclusive per node.

Nodes can be across NUMAs or per NUMA. There is no relation or limitation between a logical node and the NUMA of the resources used by the logical node.

Each storage server runs a single Node Manager service. The service controls all the logical nodes of the storage server.

Note: The current LightOS release only supports up to two logical nodes per server.

Volume Assignments

LightOS provides the following levels of protection for the volumes in a cluster:

- Replication Factor 1 / RF1: Volumes are stored on a single storage node.
- Replication Factor 2 / RF2: Volumes are stored on two storage nodes.
- Replication Factor 3 / RF3: Volumes are stored on three storage nodes.

For Replication Factor 2 and 3, one of the storage nodes behaves as a primary (P) node for this volume, and the other volume’s storage nodes behave as secondary (S) nodes. Replication Factor 1 only has a primary node.

Each storage node that stores data of multiple volumes can act as a primary node of one volume or as a secondary node of another volume. A primary node appears in the accessible path of the client, handles all user IO requests, and replicates data to the secondary nodes. If a primary node fails, the NVMe/TCP multipath feature changes the accessible path and reassigns the primary replica to another node.

When a user creates a volume, LightOS transparently selects the nodes that hold the volume’s data and configures the primary and secondary roles. The node selection logic balances the volumes between nodes upon volume creation.

NVMe/TCP MultiPath

NVMe multipath I/O refers to two or more independent paths between a single host and a namespace. Each path uses its controller, although multiple controllers can share a subsystem port. Multipath I/O like namespace sharing requires that the NVM subsystem contains two or more controllers.

Multipath is part of NVMe specification and is used by the LightOS cluster software as follows:

1. The primary node exposes the path to the volume.
2. Clients send read and write requests to the primary node.
3. The primary node replicates to the secondary nodes.
Figure 3: Volume Assignments
4. If the primary node fails, the secondary node exposes a path to the client so the client can continue working with the secondary node.

LightOS uses a proprietary protocol on top of TCP to replicate data between primary and secondary nodes, without requiring any changes to the client.

**Failure Domains**

Users define the Failure Domains (FD) based on data center topology and the level of protection that it strives to achieve. Each server in the cluster can be assigned to a set of FDs.

An example of an FD definition is separating racks of servers by FD labels. In this case, all servers in the same rack are assigned the same FD label, while servers in different racks are assigned distinct labels (e.g., FD label = rack ID). Two replicas of the same volume will not be located on two nodes in the same rack.

The system stores different replicas of the data on separate FDs to keep data protected from failures.

The definition of an FD is expressed by assigning FD labels to the storage nodes. Single or multiple FD labels can be assigned to every node.

Another example of an FD definition is grid topology in which every node is assigned a label of a row and a label of a column. In this case, the volume is not stored on two servers that are placed on the same row or on the same column.

**Dynamic Rebalancing**

**Fail in Place**
When fail in place mode activates, the LightOS cluster will try to move replications of volumes from failed nodes to other healthy nodes, while preserving failure domain requirements. Fail in place mode is activated according to user requests, and duration from the node failure until the cluster starts recovery of volumes is detrimented by a cluster configuration called `DurationToTurnIntoPermanentFailure`.

**Fail in Place CLI Example:**

Fail in place mode is activated by enabling feature-flag.

```
lbcli enable feature-flag fail-in-place [flags]
```

Example

```
enable a feature flag in place with given feature flag name
lbcli --jwt $JWT enable feature-flag fail-in-place
```

Fail in place mode is disabled by disabling feature-flag.

```
lbcli disable feature-flag fail-in-place [flags]
```

Example

```
disable a feature flag in place with given feature flag name
lbcli --jwt $JWT disable feature-flag fail-in-place
```

Example of set duration to cluster rebalance.

```
lbcli -J $JWT update cluster-config --parameter=DurationToTurnIntoPermanentFailure --value=20m
```

**Proactive Rebalance**

When proactive rebalance mode is enabled, the cluster will rebalance cluster capacity - automatically preventing scenarios where one storage node in a cluster can reach read-only status, while other nodes have free space to serve more capacity.

By default, proactive rebalance mode is disabled and can be enabled by an API endpoint.

During proactive rebalance of a volume, the protection state of a volume is kept according to the state of nodes in the cluster. When a volume migrates from a source node, it will create another temporary replica of the volume, and the replica of the source node will be removed when the destination node syncs all of the required volume data.

The following section describes when proactive rebalance is activated. The read-only thresholds are: 1. Storage effective capacity. 2. Available RAM for metadata.

The reasons to trigger volume migration (the rebalance of volumes) are: 1. Node nearing read-only.

- **a. A source node is eligible to migrate volumes, if the node utilization is 10% from the read-only threshold.**
- **b. A destination node is eligible for migration if it has at least 30% free capacity from the read-only threshold.**
2. Cluster capacity imbalance.
   a. There is a node in the cluster where utilization is under 20%.
   b. There is a node in a cluster where utilization is greater than 50%.

**Proactive Rebalance CLI Example:**
Proactive rebalance mode is activated by enabling feature-flag.

```bash
lbcli enable feature-flag proactive-rebalance [flags]
```

Example

```bash
enable a feature proactive rebalance with given feature flag name
lbcli --jwt $JWT enable feature-flag proactive-rebalance
```

Proactive rebalance mode is disabled by disable feature-flag.

```bash
lbcli disable feature-flag proactive-rebalance [flags]
```

Example

```bash
disable a feature flag proactive rebalance with given feature flag name
lbcli --jwt $JWT disable feature-flag proactive-rebalance
```
Common Administration Tasks

The following sections are examples of common tasks a user may perform to administer and maintain a LightOS product. These tasks are simply management commands grouped into a specific sequence with brief explanations to put the commands into context.

All commands and output are examples only and the specific values should not be used when performing similar tasks on your own LightOS storage server or application server.

Lightbits recommends that you use these examples to help you build your own commands. Be sure to refer to the latest API User Manual when creating your commands to ensure that you are using the latest syntax as defined by the product version you are managing.

Create a Volume

With the LightOS software installed and the LightOS management services running, you can create a volume and connect that volume to your application servers.

To create a volume, enter the `lbcli create volume` command.

**Sample Command**

```
$ lbcli create volume --name vol_1 --acl=test --size="10 Gib" --replica-count 2
```

Similar command in case of Multi Tenancy setup:

```
$ lbcli -J $JWT create volume --size="10 Gib" --name=vol_1 --acl="test" --replica-count=2 --project-name=default
```

**Sample Output**

```
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Size</th>
<th>Replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol_1</td>
<td>82f6a469-5cc9-449b-8f43-acc5cb887a7</td>
<td>Creating</td>
<td>10 Gib</td>
<td>2</td>
</tr>
</tbody>
</table>
```

This example command creates a 10 GB volume with two replicas, and a simple “test” string is used as the Access Control List (ACL) value. This ACL string is used for the application client when the `nvme connect` command is entered to connect to this new volume over NVMe/TCP.

This section includes:

Creating a Compressed Volume

List Details About the Newly Created Volume

Create a Compressed Volume

The `lbcli create volume` command also supports a flag to enable data to be compressed as much as possible in a volume.

Use the `lbcli create volume` command to create a volume with compression enabled (true).

**Sample Command**

```
$ lbcli create volume --name=vol2 --acl=acl1 --size="1 Tib" --replica-count=3 --compression=true
```
Create a Volume

Similar command in case of Multi Tenancy setup:

```
$ lbcli -J $JWT create volume --size="1 TiB" --name=vol2 --acl="acl1" --replica-count=2
--compression=true --project-name=default
```

Sample Output

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Protection State</th>
<th>NSID</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol2</td>
<td>105624dd-c96f-4668-a2a2-da0b754ac462</td>
<td>Creating</td>
<td>NotAvailable</td>
<td>0</td>
</tr>
<tr>
<td>1.0 TiB</td>
<td>3</td>
<td>true</td>
<td>values:&quot;acl1&quot;</td>
<td></td>
</tr>
</tbody>
</table>

The sample output is standard when using the lbcli immediately after executing the create command. The output indicates the volume is being created with the values provided.

List Details About the Newly Created Volume

You can use the `lbcli get volumes` command to see specific information related to any volume.

Sample Command

```
$ lbcli get volume --name=vol2
```

Similar command in case of Multi Tenancy setup:

```
$ lbcli -J $JWT get volume --name vol2 --project-name a
```

Sample Output
This output details all of the parameters used to create the volume such as the ACL, replication factor (replica count), and compression state. In addition, the UUID of each LightOS node used to create the volume on the LightOS cluster is provided under the output's nodeList area.

**Connecting Application Servers to LightOS**

Connecting an application server to the volumes on the LightOS storage server is accomplished through the following procedure.

**Connecting an Application Server to a Volume**

<table>
<thead>
<tr>
<th>Step</th>
<th>Command Type</th>
<th>Simplified Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get details about LightOS storage cluster</td>
<td>LightOS lbcli CLI</td>
<td>ping &lt;IP of LightOS Data NIC&gt; nvme connect &lt;your LightOS connection details&gt;</td>
</tr>
<tr>
<td>Verify network connectivity</td>
<td>Linux command</td>
<td></td>
</tr>
<tr>
<td>Connect to LightOS</td>
<td>Connect to LightOS cluster</td>
<td></td>
</tr>
<tr>
<td>Review block device details</td>
<td>Linux command</td>
<td>lsblk or nvme list</td>
</tr>
</tbody>
</table>

**Before You Begin**

Before you begin the process to connect an application server to the LightOS storage server, confirm that the following conditions are met:

* A volume exists on the LightOS storage server
* A TCP/IP connection exists to the LightOS storage server

**This section includes:**

Reviewing the LightOS Storage Cluster Connection Details
Verifying TCP/IP Connectivity
Connecting Application Servers to LightOS

COMMON ADMINISTRATION TASKS

Connecting to the LightOS Cluster

Reviewing Block Device Details on the Application Server

Reviewing the LightOS Storage Cluster Connection Details

The following table lists the required details you need to use for the `nvme connect` command on your application server. You can retrieve this information using the `nvme connect` command or an `lbcli` command.

**Required LightOS Storage Cluster Connection Details**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>NVMe Connect Command Parameter</th>
<th>lbcli Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem NQN</td>
<td>The LightOS cluster that was used to create the volume.</td>
<td>-n</td>
<td><code>lbcli get cluster</code></td>
</tr>
<tr>
<td>Data NIC IP addresses</td>
<td>The LightOS data NIC IP addresses for all the cluster nodes.</td>
<td>-a</td>
<td><code>lbcli list nodes</code></td>
</tr>
<tr>
<td>TCP ports</td>
<td>The TCP ports used by the LightOS cluster nodes.</td>
<td>-s</td>
<td><code>lbcli list nodes</code></td>
</tr>
<tr>
<td>ACL string</td>
<td>The ACL used when you created the volume on the LightOS storage cluster.</td>
<td>-q</td>
<td><code>lbcli list volumes</code></td>
</tr>
</tbody>
</table>

**Obtaining the LightOS Cluster Subsystem NQN**

On any LightOS server, enter the `lbcli get cluster` command.

**Sample Command**

```
$ lbcli get cluster
```

**Note:** In case of Multi Tenancy setup, use `-J $JWT` flag after `lbcli`.

**Sample Output**

```
UUID: 442a77f8-7f7a-4ab7-9fce-f1d1612e8b03
currentMaxReplicas: 3
subsystemNQN: nqn.2014-08.org.nvmexpress:NVMf:uuid:70492bf6-92e6-498a-872b-408ceb4d52d8
supportedMaxReplicas: 3
```

The output includes the subsystem NQN for the LightOS cluster.

**Obtaining the LightOS Nodes Data IP Addresses and TCP Ports**

On any LightOS server, enter the `lbcli list nodes` command.

**Sample Command**

```
$ lbcli list nodes
```
Note: In case of MultiTenancy setup, use -J $JWT flag after lbcli.

Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>UUID</th>
<th>State</th>
<th>NVME-Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>server00</td>
<td>192af7c0-d39f-4872-b849-7eb3dc0f7b53</td>
<td>Active</td>
<td>10.23.26.13:4420</td>
</tr>
<tr>
<td>server01</td>
<td>1f4ef0ce-0634-47c7-9e5f-d4fd910ff376</td>
<td>Active</td>
<td>10.23.26.8:4420</td>
</tr>
<tr>
<td>server02</td>
<td>6d9b8337-18cd-4b14-bea1-f56aca213d68</td>
<td>Active</td>
<td>10.23.26.4:4420</td>
</tr>
<tr>
<td>server03</td>
<td>912736af-bbc5-45c5-ba22-901ee9f9fde</td>
<td>Active</td>
<td>10.23.26.29:4420</td>
</tr>
<tr>
<td>server04</td>
<td>dc3ee1b5-0625-4a4c-b627-76fbd66db74c</td>
<td>Active</td>
<td>10.23.26.7:4420</td>
</tr>
<tr>
<td>server05</td>
<td>e157c1a2-701b-403b-bb73-e0c1f4be0096</td>
<td>Active</td>
<td>10.23.26.5:4420</td>
</tr>
</tbody>
</table>

The output’s NVME-Endpoint includes the data NIC IP addresses and TCP ports for all the LightOS cluster’s nodes.

**Obtaining the Volume ACL String**

The ACL string is the ACL you used when you created the volume on the LightOS storage cluster.

You can also review the list of existing volumes and their ACLs by executing the `lbcli list volumes` on any of the LightOS servers.

**Verifying TCP/IP Connectivity**

Before you run the `nvme connect` command on the application server, enter a Linux `ping` command to check the TCP/IP connectivity between your application server and the LightOS storage cluster.

**Sample Command**

```bash
$ ping -c 1 10.23.26.8
```

- rack02-server70: An application server
- 10.23.26.8: The Data NIC’s IP address on one of the LightOS storage cluster nodes

**Sample Output**

```
PING 10.23.26.8 (10.23.26.8) 56(84) bytes of data.  
64 bytes from 10.23.26.8: icmp_seq=1 ttl=255 time=0.032 ms  
--- 10.23.26.8 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.032/0.032/0.032/0.000 ms
```

The output indicates this application server has a good connection to the Data NIC IP address on the LightOS storage server.

It is recommended to repeat this check with all the IP addresses obtained from the `lbcli list nodes` command.

**Connecting to the LightOS Cluster**

With the IP, port, subsystem NQN and ACL values for your volume, you can execute the `nvme connect` command.

You must repeat the `nvme connect` command for each of the NVMe endpoints retrieved by the `lbcli list nodes` command.
Sample NVMe Connect Command


Note: During the connection phase to a client, the system can crash if you use NVMe/TCP drivers not supported by Lightbits, or try to connect to client machines that have more than 64 CPUs.

For more details on the nvme CLI, see the NVMe CLI Overview section of this document.

Currently, Lightbits only supports TCP for the transport type value.

Reviewing Block Device Details on the Application Server

After the nvme connect command completes, you can see the available block devices on the application server using the Linux lsblk command, or the nvme connect command.

The following examples show how to use the Linux lsblk command to list all block devices after the nvme connect command finishes.

Sample Command

$ lsblk

Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>MAJ:MIN</th>
<th>RM</th>
<th>SIZE</th>
<th>RO</th>
<th>TYPE</th>
<th>MOUNTPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvme0n1</td>
<td>259:0</td>
<td>0</td>
<td>10G</td>
<td>0</td>
<td>disk</td>
<td></td>
</tr>
</tbody>
</table>

In this example output, you can see the 10GB NVMe/TCP device with the name nvme0n1. This name indicates the device is: * From the NVMe subsystem * The first volume on the NVMe subsystem 0

Your LightOS storage cluster is now initialized and ready for use.

You can configure your applications to use this NVMe/TCP connected volume as you would with any other Linux block device.

Understanding a Volume Location in the LightOS Cluster

You can use the lbcli get volumes command for a specific volume to see which of the cluster nodes holds the data of a specific volume.

Sample Command

$ lbcli get volume --uuid=48dbbe54-1548-4444-866e-6438d4877e5f

Similar command in case of Multi Tenancy setup:

$ lbcli -J $JWT get volume --project-name=proj-1 --uuid=48dbbe54-1548-4444-866e-6438d4877e5f
Sample Output

```plaintext
name: vol_3
rebuildProgress: None
UUID: 48dbbe54-1548-4444-866e-6438d4877e5f
Acl:
  Values:
    - hostnqn1
nsid: 4
size: "107374182400"
nodeList:
  - 9a625dbf-de1b-4211-b9d3-0bdf70faa5f5
  - 9f1a3a85-5be5-4f98-a44e-4271fbdbe7cc
protectionState: FullyProtected
replicaCount: 2
state: Created
```

You can also run the `nvme list-subsys` command on the application server.

Sample Command

```
# nvme list-subsys nvme0n1
```

Sample Output

```
nvme-subsys0 -
  
  +- nvme0 tcp traddr=10.18.38.4 trsvcid=4420 live
  +- nvme1 tcp traddr=10.18.38.5 trsvcid=4420 live
  +- nvme2 tcp traddr=10.18.38.7 trsvcid=4420 live
  +- nvme3 tcp traddr=10.18.38.8 trsvcid=4420 live accessible
  +- nvme4 tcp traddr=10.18.38.29 trsvcid=4420 live inaccessible
```

This command’s output includes the cluster’s primary node (optimized) IP address for the volume, the secondary node (inaccessible) IP address, and the other nodes that do not hold data for that volume (block device nvme0n1).

Adding an NVMe SSD to a LightOS Storage Server

You can add physical SSDs to a LightOS node if:

- You have empty drive slots in your LightOS storage server chassis.
- During the initial configuration, you set the `maxDeviceCount` field to the total number of slots in the chassis.

This section includes:

Identifying Unused Slots

Adding NVMe SSDs

Identifying Unused Slots

From a Linux shell on any of the LightOS servers, enter the `lbcli list nvme-devices` command to see how many SSDs are managed by the LightOS node that you want to add SSDs to.
Adding an NVMe SSD to a LightOS Storage Server

Sample Command

```bash
$ lbcli list nvme-devices --node-uid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
```

Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Sample Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Serial</th>
<th>Server UUID</th>
<th>Node UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvme0n1</td>
<td>4T</td>
<td>700084450RGN fc9849f7-7380-48d8-904b-48a89c4da7a0</td>
<td>fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb</td>
<td></td>
</tr>
<tr>
<td>nvme1n1</td>
<td>4T</td>
<td>7000SZ450RGN fc9849f7-7380-48d8-904b-48a89c4da7a0</td>
<td>fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb</td>
<td></td>
</tr>
<tr>
<td>nvme2n1</td>
<td>4T</td>
<td>7000UM450RGN fc9849f7-7380-48d8-904b-48a89c4da7a0</td>
<td>fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb</td>
<td></td>
</tr>
<tr>
<td>nvme3n1</td>
<td>4T</td>
<td>7000V9450RGN fc9849f7-7380-48d8-904b-48a89c4da7a0</td>
<td>fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb</td>
<td></td>
</tr>
</tbody>
</table>

The output shows that this LightOS node currently uses four NVMe drives.

Next, enter the `lbcli get node` command to check the maximum number of NVMe devices this node was configured to manage, as set by `maxDeviceCount` in the initial configuration.

Sample Command

```bash
$ lbcli get node --uid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
```

Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Sample Output

```
UUID: fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
clusterManagerMode: PassiveMode
failureDomains: 
- rack03-server54
hostname: rack03-server54
maxNvmeDevices: 12
name: server07-0
nvmeEndpoint: 10.18.35.15:4420
state: Active
```

The output shows that this LightOS node supports up to 12 NVMe SSDs.

Up to eight more SSDs can be added to this node. The `lbcli list nvme-devices` command showed that the node currently uses four NVMe drives and the `lbcli get node` command showed that a maximum of 12 devices can be used on this node.
Adding NVMe SSDs

You can add SSDs after you have determined that the node currently manages less SSDs than the maximum set value (maxDeviceCount).

1. Enter the `lbcli list nvme-devices` command to list all NVMe SSDs that the Linux OS currently sees before inserting the new SSD. Unlike in the previous instance, use the `--server-uuid` flag to identify physical SSDs inserted into the server chassis that are not managed by the LightOS node.

Sample Command

```bash
$ lbcli list nvme-devices --server-uuid=fc9849f7-7380-48d8-904b-48a89c4da7a0
```

Note: In case of Multi Tenancy setup, use `-J $JWT` flag after `lbcli`.

Sample Output

```plaintext
Name    Size    Serial   State    Server UUID          Node UUID
nvme0n1 4T    700084450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          fcf-afla-70933c1dd3fb
nvme1n1 4T    7000SZ450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          fcf-afla-70933c1dd3fb
nvme2n1 4T    7000UM450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          fcf-afla-70933c1dd3fb
nvme3n1 4T    7000VF450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          fcf-afla-70933c1dd3fb
```

In this case, the output shows that all SSDs are currently managed by this node.

1. Insert the new SSD drive into any available physical slot.
2. Re-enter the `lbcli list nvme-devices` command.

Sample Command

```bash
$ lbcli list nvme-devices --server-uuid=fc9849f7-7380-48d8-904b-48a89c4da7a0
```

Note: In case of Multi Tenancy setup, use `-J $JWT` flag after `lbcli`.

Sample Output

```plaintext
Name    Size    Serial   State    Server UUID          Node UUID
nvme0n1 4T    700084450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          f2-4fcf-afla-70933c1dd3fb
nvme1n1 4T    7000SZ450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          f2-4fcf-afla-70933c1dd3fb
nvme2n1 4T    7000UM450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          f2-4fcf-afla-70933c1dd3fb
nvme3n1 4T    7000VF450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22f2-4
          f2-4fcf-afla-70933c1dd3fb
nvme4n1 4T    910005450RGN None fc9849f7-7380-48d8-904b-48a89c4da7a0 ---
```
The output shows that `nvme4n1` was added to the server but it is not managed by the LightOS node because it does not include a node UUID.

It is also possible to list only unmanaged SSDs by entering the `lbcli list nvme-devices` command with the following flags.

**Sample Command**

```
$ lbcli list nvme-devices --server-uuid=fc9849f7-7380-48d8-904b-48a89c4da7a0 --node-uuid=UNMANAGED
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Sample Output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Serial</th>
<th>State</th>
<th>Server UUID</th>
<th>Node UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvme4n1</td>
<td>4T</td>
<td>910005450RGN</td>
<td>None</td>
<td>fc9849f7-7380-48d8-904b-48a89c4da7a0</td>
<td>---</td>
</tr>
</tbody>
</table>

**Note:** If you do not filter by the server-uuid flag, you will see a list of NVMe devices across all the LightOS cluster servers.

3. Enter the `lbcli add nvme-device` command to add the new device to the LightOS pool of NVMe SSDs. This command creates a background task for adding the block device.

**Sample Command**

```
$ lbcli add nvme-device --serial-number=910005450RGN --node-uuid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

This task can take several minutes for the LightOS pool of NVMe SSDs to add the new device.

4. Enter the `lbcli list nvme-devices` command. The device is available in the LightOS pool when this command’s output shows the new device in the list with a valid node UUID.

**Sample Command**

```
$ lbcli list nvme-devices --node-uuid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Sample Output**
The output shows that the new `nvme4n1` device has been successfully added to the node.

**Snapshots and Clones**

With snapshots and clones, you can leverage fast and thin point-in-time snapshots/clones of LightOS volumes. A snapshot can be taken from a volume, and a volume can be created from a snapshot (aka clone).

Snapshots cannot be exported as a volume; only clones of that snapshot can be accessed.

Snapshots and clones will be created on the same failure domain. There is no limit on the per-volume number of snaps and clones, and no limit on the per-failure domain number of snaps and clones.

For snapshot scheduling, create a snapshot of a volume in a predetermined schedule managed by the cluster (multiple schedulers can exist for the same volume).

**Note:** Snapshots in LightOS are a separate entity from the management API perspective.

**Plugins**

OpenStack and Kubernetes are both supported and require compatible plugin versions. OpenStack and Kubernetes support must be updated to include snapshots and cloning. This applies to all plugin versions in production.

**Rollback**

The Rollback operation takes a volume and restores it back to a snapshot state (data + md). It is recommended to either remove active mounts, detach the volume, or flush caches before performing this operation.

The process should be as follows: 1. Stop process/es that use the volume. 2. Unmount FS create on top of the volume; e.g., for mount point /mnt/volumetorollback.

a. Unmount /mnt/volumetorollback.

3. Perform rollback via lbcli.


**Snapshots and Clone Commands**

List the existing volumes: `lbcli --jwt $JWT list volumes`:
Create a snapshot from the volume:

```
lbcli -J $JWT create snapshot --name=<snapshot name> --project-name=<project name> --source-volume-name=<source volume name>
```

```bash
$ lbcli create snapshot --name=vol1 --project-name=default --source-volume-name=vol1
```

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>Source volume UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol1</td>
<td>bd52d3d8-65a5-49fe-a934-b7d8b07ef040</td>
<td>a7f67ad0-1f3d-49cb-9650-b82042378014</td>
</tr>
</tbody>
</table>

Creating

Check that the snapshot was created:

```
lbcli -J $JWT list snapshots --project-name=<project name>
```

```bash
$ lbcli list snapshots --project-name=default
```

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>Source volume UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol1</td>
<td>bd52d3d8-65a5-49fe-a934-b7d8b07ef040</td>
<td>a7f67ad0-1f3d-49cb-9650-b82042378014</td>
</tr>
</tbody>
</table>

Available

Get snapshot info:

```
lbcli -J $JWT get snapshot --project-name=<project name> --uuid=<snapshot uuid> -o json
```

```bash
$ lbcli get snapshot --project-name=default --uuid=bd52d3d8-65a5-49fe-a934-b7d8b07ef040 -o json
```

```json
{
    "state": "Available",
    "UUID": "bd52d3d8-65a5-49fe-a934-b7d8b07ef040",
    "name": "vol1",
    "description": "",
    "creationTime": "2021-03-05T23:11:30.445377399Z",
    "retentionTime": null,
    "sourceVolumeUUID": "a7f67ad0-1f3d-49cb-9650-b82042378014",
    "sourceVolumeName": "vol1",
    "replicaCount": 1,
    "nodeList": [
        "40365551-f7e0-50b2-b415-ac238d150e16"
    ],
    "nsid": 1,
    "acl": null,
    "compression": true,
    "size": "4294967296",
    "IPAc1": null,
    "sectorSize": 4096,
    "statistics": {
        "physicalCapacity": "0",
        "physicalOwnedCapacity": "0",
        "physicalOwnedMemory": "0",
        "physicalMemory": "0",
        "userWritten": "0"
    },
    "ETag": "3",
    "projectName": "default"
}
```
Create a volume from snapshot (clone):

```bash
lbcli -J $JWT create volume --project-name=<project name> --name=<clone name> --source-snapshot-uuid=<snapshot uuid> --acl=<acl> --size=<size> --compression=<true/false> --replica-count=<1-3>
```

```bash
lbcli create volume --project-name=default --name=copy_vol1 --source-snapshot-uuid=bd52d3d8-65a5-49fe-a934-b7d8b07ef040 --acl=acl2 --size="4 Gib" --compression=true --replica-count=1
```

Create a snapshot-policy:

```bash
lbcli create snapshots-policy -J $JWT --project-name=<project name> --name=<policy name> --volume-uuid=<volume uuid> --description="<description>" --hours-in-cycle=<hours> --start-time=<time> --retention-time=<hours>
```

```bash
$ lbcli create snapshots-policy --project-name=default --name=policy1 --volume-uuid=a7f67ad0-1f3d-49cb-9650-b82042378014 --description="my policy" --hours-in-cycle=2 --start-hour=22 --retention-time=4h
```

```
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>Volume Name</th>
<th>State</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy1</td>
<td>3c03b2bf-e102-4c6c-8c6d-173224148f31</td>
<td>vol1</td>
<td>Creating</td>
<td>Hourly</td>
</tr>
</tbody>
</table>
```

List snapshot-policies:

```bash
lbcli list snapshots-policies -J $JWT --project-name=<project name> --volume-uuid=<volume uuid>
```

```bash
$ lbcli list snapshots-policies --project-name=default --volume-uuid=a7f67ad0-1f3d-49cb-9650-b82042378014
```

```
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>Volume Name</th>
<th>State</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy1</td>
<td>3c03b2bf-e102-4c6c-8c6d-173224148f31</td>
<td>vol1</td>
<td>Active</td>
<td>Hourly</td>
</tr>
</tbody>
</table>
```

Get snapshot-policies:
Volume Placement

With Volume Placement, you can specify which failure domain to place a volume on upon its creation. The idea behind volume placement is to statically define, separate, and manage how volumes and data can tolerate a failure and provide availability - through failure of a server, rack, row, power grid, etc. By default, LightOS will define server level failure domains. To define custom failure domains, refer to:Failure Domains.

Note that this feature is associated with the create volume command.

Current feature limitations of Volume Placement include the following: - Available only for single replica volumes. - Only failure domains labels are currently matched (e.g., fd:server00). - Up to 25 affinities can be specified. - Value (failure domain name) is limited to up to 100 characters. - Volume Placement cannot be specified for a clone (a clone is always placed on the same nodes as the parent volume/snapshot). - Dynamic Rebalancing must be disabled. - The entire LightOS cluster must be upgraded to at least release version 2.3.8.

Rollback volume:

```
$ lbcli rollback volume --project-name=<project-name> --uuid=<volume-uuid> --src-snapshot-uuid=<snapshot-uuid>
```

Delete snapshots/snapshot-policy:

```
$ lbcli delete snapshot --project-name=<project-name> --uuid=<snapshot-uuid>
```

```
$ lbcli delete snapshot --project-name=default --uuid=bd52d3d8-65a5-49fe-a934-b7d8b07ef040
```

```
$ lbcli delete snapshot-policy --project-name=<project-name> --uuid=<snapshot-policy-uuid>
```

```
$ lbcli delete snapshots-policy --project-name=default --uuid=3c03b2bf-e102-4c6c-8c6d-173224148f31
```

Volume Placement

```
lbcli get snapshots-policy -J $JWT --project-name=<project name> --volume-uuid=<volume uuid> -o json

$ lbcli get snapshots-policy --project-name=default --uuid=3c03b2bf-e102-4c6c-8c6d-173224148f31 -o json

{  
  "UUID": "3c03b2bf-e102-4c6c-8c6d-173224148f31",
  "name": "policy1",
  "resourceUUID": "a7f67ad0-1f3d-49cb-9650-b82042378014",
  "resourceName": "vol1",
  "projectName": "default",
  "schedulePolicy": {
    "snapshotSchedulePolicy": {
      "hourlySchedule": {
        "startTime": "2021-03-08T22:16:21Z",
        "hoursInCycle": 2
      }
    },
    "retentionTime": "14400s"
  },
  "description": "my policy",
  "state": "Active"
}
```

```
Rollback volume:

```
$ lbcli rollback volume --project-name=default --src-snapshot-uuid=bd52d3d8-65a5-49fe-a934-b7d8b07ef040 --uuid=05f49718-4897-4ff5-adb9-5d7ccd6fc138
```

Delete snapshots/snapshot-policy:

```
Snapshot: lbcli delete snapshot -J $JWT --project-name=<project-name> --uuid=<snapshot
```

```
lbcli delete snapshot --project-name=default --uuid=bd52d3d8-65a5-49fe-a934-b7d8b07ef040
```

```
Snapshot-policy: lbcli delete snapshot -J $JWT --project-name=<project-name> --uuid=<
```

```
lbcli delete snapshot-policy --project-name=default --uuid=3c03b2bf-e102-4c6c-8c6d-173224148f31
```

2022 Lightbits Labs Proprietary And Confidential - Under NDA Only
In LightOS 2.3.8 and above, a new flag is available - 'placement-affinity' - which can be used as follows:

**lbcli create volume**

```
lbcli -J $JWT create volume --name=vol1 --acl=acl1 --size="4 GiB" --replica-count=1 --placement-affinity="fd:Server0|fd:rack1|fd:rack0"
```

In the example above, you can ask the system to place the volume as follows: 1. On a node that includes Server0 in its failure domains. Note that the server name is used as a default failure domain configuration in node-manager.yaml. You will be responsible for determining if you want anything else other than the default yaml that Lightbits provides. 2. On a node that includes rack1 in its failure domains. 3. On a node that includes rack0 in its failure domains.

If the requirement cannot be satisfied because Lightbits did not find such a node, Lightbits will fail the request.

**Note:** If Lightbits cannot find active nodes with failure domains that match the volume placement request, create volume will fail and will not place the volume on other nodes.

### Identifying a Failed SSD Drive

With EC enabled, the LightOS software allows IOs to continue without interruption in the case of an SSD failure. There are three drive status values when troubleshooting drive failures.

<table>
<thead>
<tr>
<th>Drive Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>The SSD is functioning properly.</td>
</tr>
<tr>
<td>Rebuilding</td>
<td>The SSD has failed and data reconstruction is in progress.</td>
</tr>
<tr>
<td>Failed</td>
<td>Data reconstruction has completed. You can remove the failed SSD and insert a new SSD.</td>
</tr>
</tbody>
</table>

1. Check the nvme devices status by entering the `lbcli list nvme-devices` command to see if any SSD has failed and is in an EC rebuilding process.

**Sample Command**

```
$ lbcli list nvme-devices | egrep "Failed|Rebuilding"
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Sample Output**

```
Name  Size  Serial  State       Server UUID          Node UUID          Node
nvme0n1 4T  7000SZ450RGN  Rebuilding  fc9849f7-7380-48d8-904b-48a89c4da7a0  fb9c7b76
```

In this example, the output shows one NVMe SSD which has failed and is now during data reconstruction.

**Note:** Since this example does not use the -node-uuid or -server-uuid flags, the output shows all of the failed NVMe SSDs across the entire cluster. You can filter for specific nodes or servers using these flags. Once the data reconstruction is complete and the SSD state changes to Failed, the SSD is no longer managed by any node and is not associated with a node UUID.
2. To monitor a failed SSD’s rebuild progress, use the \texttt{lbcli get node} command with the \texttt{--node-uuid} flag for the LightOS node that is managing the failed NVMe SSD.

\textbf{Sample Command}

\begin{verbatim}
$ lbcli get node --uuid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
\end{verbatim}

\textbf{Note:} In case of Multi Tenancy setup, use \texttt{-J $JWT} flag after \texttt{lbcli}.

\textbf{Sample Output}

\begin{verbatim}
UUID: fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
clusterManagerMode: PassiveMode
c: true
failureDomains:
- rack08
hostname: rack08-server62
maxNvmeDevices: 12
name: server02-0
nvmeEndpoint: 10.17.51.7:4420
state: Active
inLocalRebuild: true
localRebuildProgress: 53
\end{verbatim}

3. Recheck the NVMe devices’ status with the \texttt{lbcli list nvme-devices} command to see if the status has changed from \texttt{Rebuilding} to \texttt{Failed} for the failed SSD. If the status is changed, the rebuild process is complete.

\textbf{Sample Command}

\begin{verbatim}
$ lbcli list nvme-devices --node-uuid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
\end{verbatim}

\textbf{Note:} In case of Multi Tenancy setup, use \texttt{-J $JWT} flag after \texttt{lbcli}.

\textbf{Sample Output}

\begin{verbatim}
Name Size Serial State Server UUID Node UUID
nvme0n1 4T 700084450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22
f2-4fcf-af1a-70933c1dd3fb
nvme1n1 4T 7000SZ450RGN Failed fc9849f7-7380-48d8-904b-48a89c4da7a0 ---
nvme2n1 4T 7000UM450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22
f2-4fcf-af1a-70933c1dd3fb
nvme3n1 4T 7000V9450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22
f2-4fcf-af1a-70933c1dd3fb
nvme4n1 4T 910005450RGN Healthy fc9849f7-7380-48d8-904b-48a89c4da7a0 fb9c7b76-22
f2-4fcf-af1a-70933c1dd3fb
\end{verbatim}

To replace the failed device, follow the steps for Adding an NVMe SSD to a Lightbits Storage Server.
Identifying a Node Failure in a LightOS Cluster

The LightOS cluster software continuously monitors the cluster nodes’ health and connectivity and responds to changes in the nodes’ status.

If a node fails, volumes that have data stored on that node can be affected. For a volume with a replication factor of 3, a single node failure may cause the volume protection state to become Degraded. If another node fails, the volume’s state may becomeReadOnly.

For a volume with a replication factor of 2, a single node failure may cause the volume to become ReadOnly.

In case all nodes that hold a volume’s replica fail, the volume becomes Inaccessible.

You can view the volume’s protection state by issuing the `lbcli list volumes` command.

Sample Command

```
$ lbcli list volumes
```

In case of Multi Tenancy setup:

```
$ lbcli -J $JWT --project-name=<Project Name> list volumes
```

Sample Output

```
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>Protection State</th>
<th>State</th>
<th>Size</th>
<th>Replicas</th>
<th>ACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol1</td>
<td>76c3eae8</td>
<td>FullyProtected</td>
<td>Created</td>
<td>200 GiB</td>
<td>3</td>
<td>values: &quot;acl1&quot;</td>
</tr>
<tr>
<td>vol2</td>
<td>3f3c3ad2</td>
<td>Degraded</td>
<td>Created</td>
<td>200 GiB</td>
<td>3</td>
<td>values: &quot;acl2&quot;</td>
</tr>
<tr>
<td>vol3</td>
<td>8700cba8</td>
<td>ReadOnly</td>
<td>Created</td>
<td>200 GiB</td>
<td>2</td>
<td>values: &quot;acl3&quot;</td>
</tr>
</tbody>
</table>
```

As you can see in the output, `vol2` and `vol3` are not in a FullyProtected volume protection state.

Now, you can use the `lbcli list nodes` command to identify which node has failed. In this command’s output you will see one of the following node states:

<table>
<thead>
<tr>
<th>Node state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activating</td>
<td>Node is being activated and is currently unable to serve IOs. This state can occur after a node is reconnected to the network, coming up from reboot, or recovering from any other failure state. After the activation is complete, the node’s state transitions to Active.</td>
</tr>
<tr>
<td>Active</td>
<td>Node is active and can serve IOs.</td>
</tr>
<tr>
<td>Deactivating</td>
<td>Node failure is detected and the LightOS cluster software is changing the roles of other nodes in the cluster to keep data accessible.</td>
</tr>
<tr>
<td>Inactive</td>
<td>Node is inactive.</td>
</tr>
</tbody>
</table>

Sample Command

```
$ lbcli list nodes
```

2022 Lightbits Labs

Proprietary And Confidential - Under NDA Only
Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>UUID</th>
<th>State</th>
<th>NVME-Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>server00-0</td>
<td>192af7c0-d39f-4872-b849-7eb3dc0f7b53</td>
<td>Active</td>
<td>10.23.26.13:4420</td>
</tr>
<tr>
<td>server01-0</td>
<td>1f4e6fece-0634-47c7-9e5f-d4fd910ff376</td>
<td>Active</td>
<td>10.23.26.8:4420</td>
</tr>
<tr>
<td>server02-0</td>
<td>6d9b8337-18cd-4b14-bea1-f56aca213d68</td>
<td>Inactive</td>
<td>10.23.26.4:4420</td>
</tr>
<tr>
<td>server03-0</td>
<td>912736af-bbc5-45c5-ba22-901e99f9fde</td>
<td>Active</td>
<td>10.23.26.29:4420</td>
</tr>
<tr>
<td>server04-0</td>
<td>dc3ee1b5-0625-4a4c-b627-76fbd66db74c</td>
<td>Active</td>
<td>10.23.26.7:4420</td>
</tr>
<tr>
<td>server05-0</td>
<td>e157c1a2-701b-403b-bb73-e0c1f4be0096</td>
<td>Active</td>
<td>10.23.26.5:4420</td>
</tr>
</tbody>
</table>

Multi Tenancy CLI

With multi-tenancy, managing a cluster or projects/tenants requires unique authentication and authorization. The basic entity is the security boundary, and cluster management and project management are separate entities.

As part of multi-tenancy authentication and authorization: * All API/CLI commands have strict permission handling. * API access and JWT token generation is integrated into lbcli. * A JWT bearer token (public-key) sign-in is required. * The API traffic requires mandatory TLS encryption. * There is an audit trail for all API/CLI actions.

Role Based Access Control (RBAC) for multi-tenancy includes the following roles: * Cluster Admin * Cluster Viewer * Tenant Admin * Tenant Viewer

Multi-Tenancy and LightOS

Multi-tenancy features tight integration with the LightOS Kubernetes CSI plugin, including: * One or more K8s clusters per tenant. * One or more K8s namespaces per tenant within a K8s cluster. * Fine-grained tenant separation based on K8s security within a K8s namespace.

The LightOS K8s plugin features CSI credentials handling as well, including: * Full support for centralized and secure storage provider credentials handling, rotation, and revocation. * Credentials (JW Ts) stored as K8s “secrets”. * CSI sidecars that provide the CSI plugin with the right JWT for every CSI call.

Notes: * OpenStack integration is only with a single tenant. * Multi-tenancy separation is only for the control path (the data path is managed via ACLs and IP ACLs).

- JWTs have an expiration (30 days is the default), and can be set as desired.
- For single tenants, the cluster admin JWT can be used for all actions.

Make note of the system:cluster-admin JWT generated by the installation (this will be different for every installation - it is only an example.)

```
$ export SYSTEMJJWT=eyJhb...s4u-w
```

Key Features

“Projects” (tenants) support

- Strict separation at the API/CLI level (resource names, visibility)
- LightOS API/CLI management by admins separate from tenants’ access

Authentication (authN)

- All accesses authenticated by pubkey signed JWT bearer tokens
- Integration with K8s secrets management, LB CSI plugin
• API access JWT tokens generation integrated into lbcli

Authorization (authZ)
• Every API/CLI action explicitly authorized by the policy module

Role Based Access Control (RBAC)
• Access rights granted based on one or more of the roles assigned to an account
• Roles assigned from a limited number of predefined roles

Common Multi Tenancy CLI Examples
Create a new project (as cluster-admin). Note the encrypted (https, port 443) endpoints and usage of the system jwt.

```
[server00]:~# lbcli --jwt $SYSTEMJWT --endpoint https://localhost:443 create project --
   name myproj --description "my project"
Name    Description
myproj  my project
```

List projects (as cluster-admin):

```
[server00]:~# lbcli --jwt $SYSTEMJWT --endpoint https://localhost:443 list projects
Name    Description
default This is the default project description
a       a description
b       b description
myproj  my project
```

Create a public/private RSA256 key pair:

```
[server00]:~# mkdir -p ${HOME}/.lightos_keys && ssh-keygen -t rsa -f ${HOME}/.
   lightos_keys/mykey -q -N "" && openssl rsa -in ${HOME}/.lightos_keys/mykey -pubout -
   out ${HOME}/.lightos_keys/mykey.pem
writing RSA key
[server00]:~# ls ${HOME}/.lightos_keys
mykey mykey.pub mykey.pem
```

Create the first credential in the project (as cluster-admin):

```
[server00]:~# lbcli --jwt $SYSTEMJWT --endpoint https://localhost:443 create credential
   --id=c1 --project-name myproj --type rsa256pubkey ${HOME}/.lightos_keys/mykey.pem
ID    Type
   c1 RS256PubKey
```

List credentials in the project (as cluster-admin):
Create a project:admin JWT for new project:

```
[server00]:~ # lbcli create jwt --key-id myproj:c1 --key ${HOME}/.lightos_keys/mykey --role myproj:admin --issuer=root@example.com --subject=tenant-1 eyJh...Nmsw
```

Create volume in the project as project:admin:

```
[server00]:~ # lbcli --jwt $MYPROJADMINJWT --endpoint https://localhost:443 create volume --name vol1 --size 1G --replica-count 3 --acl ALLOW_NONE --project-name myproj
```

List volumes in the project as project:admin or project:viewer:

```
[server00]:~ # lbcli --jwt $MYPROJADMINJWT --endpoint https://localhost:443 list credentials --project-name myproj
ID   Type
c1   RS256PubKey

[server00]:~ # export MYPROJADMINJWT=eyJh...Nmsw
```

```
[server00]:~ # lbcli --jwt $MYPROJADMINJWT --endpoint https://localhost:443 list volumes --project-name myproj
Name    UUID                State       Protection State             NSID
vol1    cdf25ac7-3829-4865-bad6-bef2d87e7b  Creating    Unknown              0
        954 MiB 3 false values:"ALLOW_NONE"

[server00]:~ # export MYPROJVIEWERJWT=`lbcli create jwt --key-id myproj:c1 --key ${HOME}/.lightos_keys/mykey --role myproj:viewer`
```

```
[server00]:~ # lbcli --jwt $MYPROJVIEWERJWT --endpoint https://localhost:443 list volumes --project-name myproj
Name    UUID                State       Protection State             NSID
vol1    cdf25ac7-3829-4865-bad6-bef2d87e7b  Available  FullyProtected              1
        954 MiB 3 false values:"ALLOW_NONE" None

IP ACL Configuration

In addition to the existing ACL parameter available for use with the `lbcli create volume` command there is also the IP ACL parameter.

IP ACL allows support for restricted/non-restricted access to a cluster. In restricted mode, the LightOS cluster will maintain an allowed list of IPs per volume that allows clients to access this volume. In unrestricted mode, any IP address can be used.
When iptables mode is enabled, the following rules are applied:

- Allow access only for specific IPs within the following distinct groups:
  - LightOS cluster servers - where internal communication ports stay open within the cluster servers IPs.
  - LightOS external user-facing control services - available to all IPs: API, Monitoring.
  - IO client servers can only connect to access volumes for which their IP is on the allowed list.
  - Any other network traffic that is not allowed by default will be blocked from any communication.

The following ports are only accessible for admin endpoints:

- TCP port default=80/443 (MT disabled / MT enabled)
- TCP default=8090 - exporter port - gRPC port default=111 - ssh port=22 - NTP connection=123

There are two APIs that can cause additional ports to be opened:

- Create an admin endpoint. This adds an IP address that will have access to admin ports (REST, gRPC, Exporter).
- Create (or update) a volume with the IP ACL flag. The allowed IPs will have access to the NVMe/TCP ports (default: 4420 and 8009) on the nodes on which the volume data is stored (but not to other cluster nodes).

Examples:

Create Endpoint

```
$ lbcli create admin-endpoint --port=22 --ips=172.16.123.18
Port  IPs
22    [172.16.123.18]
```

```
$ lbcli get admin-endpoint --port=22
Port  IPs
22    [172.16.123.18]
```

Update Endpoint

```
$ lbcli update admin-endpoint --port=22 --ips=172.16.123.18,172.16.123.45
Port  IPs
22    [172.16.123.18 172.16.123.45]
```

```
$ lbcli get admin-endpoint --port=22
Port  IPs
22    [172.16.123.18 172.16.123.45]
```

Create Volume

Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.
$ lbcli create volume --name=acl3 --acl=vol --size=10 gib --replica-count=1 --ip-acl=172.16.123.18

# in case of Multi Tenancy

$ lbcli -J $JWT create volume --size="10 Gib" --name=vol --acl="acl3" --replica-count=1
   --project-name=<Project Name> --ip-acl=172.16.123.18

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Protection State</th>
<th>NSID</th>
<th>Size</th>
<th>Replicas</th>
<th>Compression</th>
<th>ACL</th>
<th>Rebuild Progress</th>
<th>Name</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl3</td>
<td>5f69c7d3-e303-4179-8400-59fa073c1229</td>
<td>Creating</td>
<td>Unknown</td>
<td>0</td>
<td>10 GiB</td>
<td>1</td>
<td>false</td>
<td>vol</td>
<td></td>
<td>acl3</td>
<td>vol</td>
</tr>
</tbody>
</table>

$ lbcli get volume --uuid=5f69c7d3-e303-4179-8400-59fa073c1229 -o json

```
{
  "state": "Available",
  "protectionState": "FullyProtected",
  "replicaCount": 1,
  "nodeList": [
    "69399d2a-3896-5b37-978a-775f3322279b"
  ],
  "UUID": "5f69c7d3-e303-4179-8400-59fa073c1229",
  "nsid": 11,
  "acl": {
    "values": [
      "vol"
    ]
  },
  "compression": "false",
  "size": "10737418240",
  "name": "acl3",
  "rebuildProgress": "None",
  "statistics": {
    "logicalUsedStorage": "0",
    "physicalUsedStorage": "0",
    "compressionRatio": 0
  },
  "IPACL": {
    "values": [
      "172.16.123.18"  # List of IPs allowed to connect the volume
    ]
  },
  "ETag": "1",
  "connectedHosts": [
    "vol: 172.16.123.18"
  ],
  "sectorSize": 4096,
  "sourceSnapshotUUID": "",
  "sourceSnapshotName": ""
}
```

Update Volume
$ lbcli update volume --name=acl3 --ip-acl=172.16.123.18,172.16.123.45
# In case of Multi Tenancy Setup
$ lbcli -J $JWT update volume --project-name=<Project Name> --name acl3 --ip-acl=172.16.123.18,172.16.123.45

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Protection State</th>
<th>NSID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Replicas</td>
<td>Compression</td>
<td>ACL</td>
</tr>
<tr>
<td>acl3</td>
<td>10 GiB</td>
<td>1</td>
<td>false</td>
<td>5f69c7d3-e303-4179-8400-59fa073c1229</td>
</tr>
</tbody>
</table>

$ lbcli get volume --uuid=5f69c7d3-e303-4179-8400-59fa073c1229 --o json
# In case of Multi tenancy:
$ lbcli -J $JWT get volume --uuid=5f69c7d3-e303-4179-8400-59fa073c1229 --project-name <Project Name> --o json

```
{
    "state": "Available",
    "protectionState": "FullyProtected",
    "replicaCount": 1,
    "nodeList": [
        "69399d2a-3896-5b37-978a-775f3322279b"
    ],
    "UUID": "5f69c7d3-e303-4179-8400-59fa073c1229",
    "nsid": 11,
    "acl": {
        "values": [
            "vol"
        ]
    },
    "compression": "false",
    "size": "10737418240",
    "name": "acl3",
    "rebuildProgress": "None",
    "statistics": {
        "logicalUsedStorage": "0",
        "physicalUsedStorage": "0",
        "compressionRatio": 0
    },
    "IPAcl": {
        "values": [
            "172.16.123.18",
            "172.16.123.45" ← New IP was added
        ]
    },
    "ETag": "2",
    "connectedHosts": [
        "vol: 172.16.123.18",
        "vol: 172.16.123.45"
    ],
    "sectorSize": 4096,
    "sourceSnapshotUUID": "",
    "sourceSnapshotName": ""
}
```
LightOS CLI Overview

Lightbits has developed a Command Line Interface (CLI) called “lbcli” that you can use to configure and manage the LightOS platform. This section includes several examples of how to use common LightOS CLI management commands. See the LightOS API User Manual for a complete list of all commands and their specific command syntax.

lbcli Command Syntax

The lbcli command syntax includes the following parameters.

```
lbcli [command] [type] [options] [flags]
```

lbctl CLI Utility Command Syntax

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Specifies the operation performed on one or more resources. See the Key Resources section below for more.</td>
</tr>
<tr>
<td>Type</td>
<td>Sets the resource type that the command will address. See the Key Resources section below for a description of each resource that can be used. An example command syntax for creating a volume is as follows: lbcli create volume</td>
</tr>
<tr>
<td>Options</td>
<td>Options that can be given with a command and type. For example, the option is to give the new volume 4 GB of maximum capacity. lbcli create volume --name vol1 --size=4gib</td>
</tr>
<tr>
<td>Flags</td>
<td>For example, the -o flag for selecting the output format (e.g., json, yaml). lbcli create volume --name vol1 --size=4gib --output-format=yaml</td>
</tr>
</tbody>
</table>

lbcli Command Summary

The lbcli CLI utility uses commands to refer to the uppermost management level tasks that can be performed by lbcli. Commands such as create and delete are used for creating and deleting resources. Therefore, a resource type must be specified to use these commands. Other commands may require other sub-commands and parameters.

lbcli Common Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Note: Will be supported in a future product release.</td>
</tr>
<tr>
<td>create</td>
<td>Creates a resource such as a volume.</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes a resource such as a volume.</td>
</tr>
<tr>
<td>get</td>
<td>Retrieves information about a single resource such as a node or a volume.</td>
</tr>
<tr>
<td>list</td>
<td>Retrieves information about a list of resources such as nvme-devices, nodes, or volumes.</td>
</tr>
<tr>
<td>help</td>
<td>Displays help content about any command.</td>
</tr>
<tr>
<td>version</td>
<td>Retrieves lbcli version information.</td>
</tr>
</tbody>
</table>

Note: For a list of all of the lbcli commands, see the LightOS API User Manual.

Key Resources

Some of the common resources that can be modified using lbctl are defined here.
Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>A Node is the collection of physical hardware that makes up a single LightOS storage controller. This includes the CPU, LightField, memory, NIC, and storage devices themselves. A typical server running the LightOS software contains either one or two storage controller nodes.</td>
</tr>
<tr>
<td>Volume</td>
<td>A block storage address space - allocated by the LightOS system and accessible as an NVMe storage device on an application server.</td>
</tr>
<tr>
<td>NVMe Device</td>
<td>The physical NVMe SSD that is used within a server running the LightOS system. This resource is used if operations need to be performed on physical SSDs on the server. For example, installing a new NVMe SSD into an empty SSD slot in the server hardware enclosure.</td>
</tr>
<tr>
<td>Events</td>
<td>List and/or Get LightOS Cluster administrative Events. This includes but is not limited to node/cluster availability (offline/online), NVMe failures, rebalance status, capacity warnings, etc.</td>
</tr>
</tbody>
</table>

Using the lbcli Commands

To use the lbcli CLI utility, you must access the Linux operating system command line on a server that is running LightOS. You can access the Linux server via a direct connection to the physical server or remotely via Secure Shell (ssh) to the server.

Notes: - See the LightOS API User Manual for a complete list of all lbcli commands.
- Some output may be only partially displayed for illustrative purposes only. Your output details will most likely be different in many cases, depending on your specific configuration.

lbcli Add NVMe Device

Adds an unmanaged NVMe SSD to be managed by a LightOS node. The operation of adding a device may take up to a few minutes.

Sample Command

```
lbcli add nvme-device --serial-number=910005450RGN --node-uuid=fb9c7b76-22f2-4fcf-af1a-70933c1dd3fb
```

Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

lbcli Upgrade Server

You can upgrade the LightOS version by using the lbcli tool. You can upgrade the servers one by one, or the entire cluster at once.

Get the cluster status.

```
lbcli get cluster
```

Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.
Sample Output

```
lbcli get cluster
ETag: "0"
MaxAllowedVersion: 2.1.X  
--- The user can upgrade until this version.
MinAllowedVersion: 2.0.X  
--- The minimum version that this cluster can support.
MinVersionInCluster: 2.0.4  
--- The minimum version that is currently installed.
UUID: c7e9e3c6-93f6-4b9a-8ac2-746227232c37
currentMaxReplicas: 3
health:
  numDegradedVolumes: 0
  numInactiveNodes: 0
  numNotAvailableVolumes: 0
  numReadOnlyVolumes: 0
  state: OK  
--- The health must be OK in order to perform the upgrade.
statistics:
  compressionRatio: 1
  effectivePhysicalStorage: "13809663442943"
  estimatedFreeLogicalStorage: "13809663442943"
  estimatedLogicalStorage: "13809663442943"
  freePhysicalStorage: "13809663442943"
  installedPhysicalStorage: "45679219826688"
  logicalStorage: "8589934592"
  managedPhysicalStorage: "0"
  physicalUsedStorage: "0"
  physicalUsedStorageIncludingParity: "0"
subsystemNQN: nqn.2014-08.org.nvme:xpress:NVMf:uuid:f4a89ce0-9fc2-4900-bfa3-00ad27995e7b
supportedMaxReplicas: 3
```

Check the status of the nodes

```
lbcli list node
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Sample Output

```
lbcli list nodes
Name               UUID          State       NVMe endpoint
Failure domains    Local rebuild progress
server00-0         d3994d3b-d3c4-565f-8e73-54b5e0c34ab7  Active  10.10.10.100:4420  [server00]
server01-0         e964223a-fab4-51f6-b31f-6abbc0d1f9c3  Active  10.10.10.101:4420  [server01]
server02-0         edb8bd43-8090-59af-8d2d-aa8b2375793a  Active  10.10.10.102:4420  [server02]
```

Check the status of the servers
lbcli list server

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

### Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>UUID</th>
<th>State</th>
<th>RiskOfServiceLoss</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightOSVersion</td>
<td>server02 f1171a62-7f41-5330-9d5d-b826f779ad2a</td>
<td>Enabled</td>
<td>NoRiskOfServiceLoss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>server00 82dd376e-9b12-5b0b-86ea-d5b5dfeb8961</td>
<td>Enabled</td>
<td>NoRiskOfServiceLoss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>server01 2b2767c3-802b-58d8-8810-55d525bd624b</td>
<td>Enabled</td>
<td>NoRiskOfServiceLoss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Upgrade Server from Remote Repo

Run the upgrade command

```
lbcli upgrade server --install-pkg-uri="<repo uri>" --uuid=<server uuid>
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

### Example

```
lbcli upgrade server --install-pkg-uri="https://dl.lightbitslabs.com/<YOUR_TOKEN>/lightos-2-<Minor Ver>-x-ga/rpm/el/7/x86_64/lightos-2.0.5-1.x86_64" --uuid=ac57df6e-4fe9-5ba2-980c-388de6cb4740
```

**Notes:**
- The repo URL must include the lightos version (in this example: lightos-2.0.5-1.x86_64).
- For now the output of this command is {} and then has to wait until the server is up and the node is in Active state.
- Until the end of the upgrade process, the server will be in a disable state and with SourceOfRiskOfServiceLoss warning, and the node will be in an Inactive state.
- The upgrade will take ~ 10 minutes to be completed.
- All other servers will be in InRiskOfServiceLoss.

### Upgrade Server from File

Copy the RPMs to the local server (for each server in the cluster). Run createrepo.

```
createrepo <rpms folder name>
```
Example

```bash
createrepo lightos_rpms/
```

Run the upgrade command.
```
lbcli upgrade server --install-pkg-uri="<lightos file location>" --uuid=<server uuid>
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Example
```
lbcli upgrade server --install-pkg-uri=file:///root/lightos_rpms/lightos-2.0.5-1.x86_64
--uuid=82dd376e-9b12-5b0b-86ea-d5b5df6b8961
```

**Notes:**
- The file location must include the full lightos rpm name without the .rpm (in this example: lightos-2.0.5-1.x86_64).
- For now the output of this command is `{}` and then has to wait until the server is up and the node is in Active state.
- Until the end of the upgrade process, the server will be in a disable state and with SourceOfRiskOfServiceLoss warning and the node will be in an Inactive state.
- The upgrade will take ~10 minutes to be completed.
- All other servers will be in InRiskOfServiceLoss.

When the upgrade is complete, check the status of the cluster, servers, and nodes.

**Sample Output**
lbcli get cluster
ETag: "0"
MaxAllowedVersion: 2.1.X
MinAllowedVersion: 2.0.X
MinVersionInCluster: 2.0.4  <----- the minimum version, we still have 2 servers with 2.0.0-beta, when all the servers will be upgraded this will change.
UUID: c7e9e3c6-93f6-4b9a-8ac2-746227232c37
currentMaxReplicas: 3
health:
  numDegradedVolumes: 0
  numInactiveNodes: 0
  numNotAvailableVolumes: 0
  numReadOnlyVolumes: 0
state: OK  <----- Need to return to OK state
statistics:
  compressionRatio: 1
  effectivePhysicalStorage: "13809663442943"
  estimatedFreeLogicalStorage: "13809663442943"
  estimatedLogicalStorage: "13809663442943"
  freePhysicalStorage: "13809663442943"
  installedPhysicalStorage: "45679219826688"
  logicalStorage: "8589934592"
  logicalUsedStorage: "0"
  managedPhysicalStorage: "21164236111872"
  physicalUsedStorage: "0"
  physicalUsedStorageIncludingParity: "0"
subsystemNQN: nqn.2014-08.org.nvmeexpress:NVMf:uuid:f4a89ce0-9fc2-4900-bfa3-00ad27995e7b
supportedMaxReplicas: 3

lbcli list node
Name    UUID      State          NVMe endpoint
server00  d39943db-d3c4-565f-8e73-54b5e0c3ab7  Active  10.10.10.100:4420 [server00]
server01  e964223a-fab4-51f6-b31f-6abbcd1f9c3  Active  10.10.10.101:4420 [server01]
server02  edbbd43-8090-59af-8d2d-aa8b237593a  Active  10.10.10.102:4420 [server02]

lbcli list server
NAME    UUID                State          RiskOfServiceLoss State
server01  2b2767c3-802b-58d8-8810-55d525bd624b  Enabled  NoRiskOfServiceLoss
  2.0.4
server02  f1171a62-7f41-5330-9d5d-b826f779ad2a  Enabled  NoRiskOfServiceLoss
  2.0.5
server00  82dd376e-9b12-5b0b-86ea-d5b5dfeb8961  Enabled  NoRiskOfServiceLoss
  2.0.4

Upgrade Cluster
Run the upgrade command
**lbcli upgrade cluster --install-pkg-uri="<repo uri>"**

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Example**

```
lbcli upgrade cluster --install-pkg-uri="https://dl.lightbitslabs.com/<YOUR_TOKEN>/lightos-2-<Minor Ver>-x-ga/rpm/el/7/x86_64/lightos-2.0.5-1.x86_64"
```

**Notes:**
- For now the output of this command is {} and then has to wait until the all the servers are up and the nodes are in Active state.
- Until the end of the upgrade process, one server at a time will be in a failed state and the node will be in an Inactive state.
- The upgrade will take ~ 10 minutes per server to be completed.
- When the upgrade is complete, check the status of the cluster, servers, and nodes.

**Sample Output**

```
lbcli get cluster
ETag: "0"
MaxAllowedVersion: 2.1.X
MinAllowedVersion: 2.0.X
MinVersionInCluster: 2.0.5  <-------- the new minimum version
UUID: c7e9e3c6-93f6-4b9a-8ac2-746227232c37
currentMaxReplicas: 3
health:
  numDegradedVolumes: 0
  numInactiveNodes: 0
  numNotAvailableVolumes: 0
  numReadOnlyVolumes: 0
state: OK  <-------- Need to return to OK state
statistics:
  compressionRatio: 1
  effectivePhysicalStorage: "13809663442943"
  estimatedFreeLogicalStorage: "13809663442943"
  estimatedLogicalStorage: "13809663442943"
  freePhysicalStorage: "13809663442943"
  installedPhysicalStorage: "45679219826688"
  logicalStorage: "8589934592"
  logicalUsedStorage: "0"
  managedPhysicalStorage: "21164236111872"
  physicalUsedStorage: "0"
  physicalUsedStorageIncludingParity: "0"
  subsystemNQN: nqn.2014-08.org.nvmexpress:NVMf:uuid:f4a89ce0-9fc2-4900-bfa3-00ad27995e7b
  supportedMaxReplicas: 3
```
**lbcli Create Server**

Add a new server to an existing cluster (when you need to replace the existing server). Note: You will need to create a new server first. In order to create a new server, edit the Ansible inventories files (hosts, `/host_var/new_server.yaml`) and add the new server. Then run the ansible-playbook with the same parameters as before and add the following parameter: `--limit:

**Example:**

```bash
ansible-playbook -i /tmp/inventory/cluster_example/hosts playbooks/deploy-lightos.yml \
-e new_etcd_member=true \
-e extend_etcd_cluster=true \
--limit <new_server> -vv
```

In order to find the new server uuid run the following on the new server: `cat /etc/node-manager/node-manager.yaml |grep serverUUID`

**Example Command**

```bash
lbcli create server --uuid=ad69f2c6-4b9a-5a72-8ff6-daf949ef61b4 --endpoints=https://10.19.117.3:2380
```

**Note:** In case of Multi Tenancy setup, use `-J $JWT` flag after `lbcli`. 

**Example Output**
lbcli Create Volume

Creates a volume with a specified unique name, size (capacity) and replication factor (valid values: 1, 2 or 3).

An Access Control List (ACL) must also be applied to the created volume.

**Examples**

```
lbcli create volume --name=vol4 --acl=acl3 --size="2 GiB" --replica-count=2
```

Similar command with Multi tenancy enabled:

```
lbcli -J $JWT create volume --size="2 GiB" --name=vol4 --acl="acl3" --replica-count=2 --project-name=default
```

**Sample Output**

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Size</th>
<th>Replicas</th>
<th>ACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol4</td>
<td>76c3eae8-7ade-4394-82e5-056d05a92b5e</td>
<td>Creating</td>
<td>2.0 GiB</td>
<td>2</td>
<td>values:&quot;acl3&quot;</td>
</tr>
</tbody>
</table>

lbcli Replace Node

Replace a node with an unattached node.

**Notes:**
- In order to have an unattached node you need to create a server (see lbcli create server).
- In order to replace a node the replaced node needs to be in inactive state.
- In order to set a node in inactive state you need to disable the server using the lbcli command below.

```
lbcli disable server --uuid=<server uuid>
```

---

2022 Lightbits Labs

Proprietary And Confidential - Under NDA Only
Note: In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

Sample Command

$ lbcli replace node --src-node-uuid=<Inactive node uuid> --target-node-uuid=<Unattached node uuid>

Sample Output

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>NVMe endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_new-0</td>
<td>4c6a04de-ba6c-5d92-83fc-f3a86b4af081</td>
<td>Unattached</td>
<td>10.19.117.3:4420</td>
</tr>
<tr>
<td></td>
<td>server_new</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server00-0</td>
<td>55948584-acad-5318-9b8b-4b164d7623ca</td>
<td>Inactive</td>
<td>10.19.117.4:4420</td>
</tr>
<tr>
<td></td>
<td>server00</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server02-0</td>
<td>a25c1ad4-6049-5f15-9b3f-1da877b593a0</td>
<td>Active</td>
<td>10.19.117.7:4420</td>
</tr>
<tr>
<td></td>
<td>server02</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server01-0</td>
<td>a50c4a57-e6c1-5ee8-9e45-a607425ad393</td>
<td>Active</td>
<td>10.19.117.5:4420</td>
</tr>
<tr>
<td></td>
<td>server01</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

lbcli replace node --src-node-uuid=55948584-acad-5318-9b8b-4b164d7623ca --target-node-uuid=4c6a04de-ba6c-5d92-83fc-f3a86b4af081

lbcli list node

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>NVMe endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_new-0</td>
<td>4c6a04de-ba6c-5d92-83fc-f3a86b4af081</td>
<td>Active</td>
<td>10.19.117.3:4420</td>
</tr>
<tr>
<td></td>
<td>server_new</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server00-0</td>
<td>55948584-acad-5318-9b8b-4b164d7623ca</td>
<td>Unattached</td>
<td>10.19.117.4:4420</td>
</tr>
<tr>
<td></td>
<td>server00</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server02-0</td>
<td>a25c1ad4-6049-5f15-9b3f-1da877b593a0</td>
<td>Active</td>
<td>10.19.117.7:4420</td>
</tr>
<tr>
<td></td>
<td>server02</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>server01-0</td>
<td>a50c4a57-e6c1-5ee8-9e45-a607425ad393</td>
<td>Active</td>
<td>10.19.117.5:4420</td>
</tr>
<tr>
<td></td>
<td>server01</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

lbcli Delete Volume

Deletes a volume that is specified by either its name or UUID.

Sample Command

lbcli delete volume --name="vol4"

Similar command in case of Multi Tenancy setup:

lbcli -J $JWT delete volume --project-name=proj-a --name="vol4"
**lbcli Get Cluster**

Returns global information about the cluster.

**Sample Command**

```
lbcli get cluster
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Sample Output**

```
UUID: ea4fa7c1-2470-4f69-9503-1fcb005895fb
currentMaxReplicas: 3
Statistics:
  effectivePhysicalStorage: "31326416260299"
  estimatedFreeLogicalStorage: "6123549568203"
  freePhysicalStorage: "6123549568203"
  installedPhysicalStorage: "72014751793152"
  logicalStorage: "25202868092928"
  logicalUsedStorage: "25202866692096"
  managedPhysicalStorage: "36007375896576"
  physicalUsedStorage: "25202866692096"
  physicalUsedStorageIncludingParity: "25202867311353"
  subsystemQN: nqn.2014-08.org.nvmeexpress:NVMe:uuid:5aa591cd-7694-4576-bfd5-8230c1610d7d
  supportedMaxReplicas: 3
```

**lbcli Get Node**

Returns information about a specific node. You must specify a specific node using the --uuid flag.

**Sample Command**

```
lbcli get node --uuid=86b3f8ad-61a9-4423-97ec-55c301d9f09c
```

**Note:** In case of Multi Tenancy setup, use -J $JWT flag after lbcli.

**Sample Output**
lbcli Get Volume

Returns information about a volume by name or UUID.

Sample Command

```
lbcli get volume --name=vol_client_8
```

Similar command with Multi Tenancy enabled:

```
lbcli -J $JWT get volume --name vol_client_8 --project-name a
```

Sample Output
lbcli List Events

Returns a list of events in the cluster, and information on each event.

Sample Command

```
lbcli -J $JWT list events
```

Sample Output

<table>
<thead>
<tr>
<th>UUID: 80039266-6fab-4f1f-ac09-4f824616e1e0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acl:</td>
</tr>
<tr>
<td>Values:</td>
</tr>
<tr>
<td>- nqn_client_8</td>
</tr>
<tr>
<td>compression: false</td>
</tr>
<tr>
<td>name: vol_client_8</td>
</tr>
<tr>
<td>nodeList:</td>
</tr>
<tr>
<td>- 75db8358-a46e-48fc-8147-5d60735e66a0</td>
</tr>
<tr>
<td>- 73317646-41cd-4a1d-89f0-31f601c90cd4</td>
</tr>
<tr>
<td>- 2bd8ef04-d333-4492-887b-6a661df5b3ca</td>
</tr>
<tr>
<td>nsid: 9</td>
</tr>
<tr>
<td>protectionState: FullyProtected</td>
</tr>
<tr>
<td>rebuildProgress: None</td>
</tr>
<tr>
<td>replicaCount: 3</td>
</tr>
<tr>
<td>size: &quot;700079669248&quot;</td>
</tr>
<tr>
<td>state: Created</td>
</tr>
<tr>
<td>Statistics:</td>
</tr>
<tr>
<td>logicalUsedStorage: &quot;700079669248&quot;</td>
</tr>
<tr>
<td>physicalUsedStorage: &quot;700079669248&quot;</td>
</tr>
<tr>
<td>Event Name</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeInactive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
<tr>
<td>NodeActive</td>
</tr>
</tbody>
</table>

**Ibcli List Nodes**

Returns a list of nodes in the cluster, and information on each node.

**Sample Command**

```
lbcli list nodes
```

**Note:** In case of Multi Tenancy setup, use `-J $JWT` flag after Ibcli.

**Sample Output**
**lbcli List NVMe Devices**

Retrieves a list of NVMe device in the cluster. The list can be filtered to list NVMe SSDs that exist in a specific server, managed by a specific node, or unmanaged.

**Sample Command**

```
$ lbcli list nvme-devices
```

**Note:** In case of Multi Tenancy setup, use `-J $JWT` flag after `lbcli`.

**Sample Output**

```
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>NVMe Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>server15-0</td>
<td>16cf1571-756f-40e6-81a1-a68741b12789</td>
<td>Active</td>
<td>10.16.103.11:4420</td>
</tr>
<tr>
<td>server09-0</td>
<td>177a168c-ef34-415c-8a47-8d1394cdd474</td>
<td>Active</td>
<td>10.16.103.1:4420</td>
</tr>
<tr>
<td>server02-0</td>
<td>86b3f8ad-61a9-4423-97ec-55c301df09c</td>
<td>Active</td>
<td>10.16.103.7:4420</td>
</tr>
<tr>
<td>server13-0</td>
<td>9b084655-6a30-44e9-8b46-b3569c5d7d76</td>
<td>Active</td>
<td>10.16.103.6:4420</td>
</tr>
</tbody>
</table>
```

**lbcli List Volumes**

Returns the list of volumes in the LightOS cluster.

**Sample Command**

```
$ lbcli list volumes
```

Similar command with Multi Tenancy enabled:

```
lbcli --jwt $P1ADMIN list volumes --project-name project1
```

**Sample Output**
<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
<th>State</th>
<th>Protection State</th>
<th>NSID</th>
<th>Size</th>
<th>Replicas</th>
<th>Compression</th>
<th>ACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol_0</td>
<td>bea851365040</td>
<td>Created</td>
<td>FullyProtected</td>
<td>1</td>
<td>100 GiB</td>
<td>2</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>vol_1</td>
<td>92023adb23ae</td>
<td>Created</td>
<td>FullyProtected</td>
<td>2</td>
<td>100 GiB</td>
<td>2</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>vol_2</td>
<td>65c3bd571c87</td>
<td>Created</td>
<td>FullyProtected</td>
<td>4</td>
<td>100 GiB</td>
<td>2</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>vol_3</td>
<td>8cc8bb076932</td>
<td>Created</td>
<td>FullyProtected</td>
<td>3</td>
<td>100 GiB</td>
<td>2</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
LightOS Rest API Overview

You can configure and manage the LightOS cluster with a standard REST API developed by Lightbits.

This section includes several examples of how to use common management commands via REST API using a web interface and the Linux “curl” utility to send and receive data with the REST API methods:

- GET
- PUT
- DELETE

Note: For a complete reference of the LightOS REST API interface, see the LightOS API User Manual.

REST API Syntax

The following example shows how the Linux curl utility is used in this document to provide REST API usage examples:

```
curl -X <REST API VERB> http://<IP of LightOS Node>:<TCP PORT#/api/v1/<command>
```

You can use any of the cluster node’s IP addresses (not necessarily the node affected by the command).

Using REST API Commands (curl Examples)

If you use the REST API GET method, the LightOS management service returns the data in JSON format.

This section includes:

- REST get cluster
- REST get nodes
- REST get volumes
- REST create volume
- REST delete volume
- REST get nvme devices

REST Get Cluster

To get information on the cluster, use the following command:

Sample Command

```
```

Sample Output
REST Get Nodes

To get a quick summary of the status and capacity of the nodes on the server running LightOS, use the GET method with this URL `/api/v1/nodes`.

**Sample Command**


**Sample Output**

```json
{
    "UUID": "ea4fa7c1-2470-4f69-9503-1fcb005895fb",
    "subsystemQN": "nqn.2014-08.org.nvme:xpress",
    "currentMaxReplicas": 3,
    "supportedMaxReplicas": 3,
    "statistics": {
        "installedPhysicalStorage": "72014751793152",
        "managedPhysicalStorage": "36007375896576",
        "effectivePhysicalStorage": "3126416260299",
        "logicalStorage": "25202868092928",
        "logicalUsedStorage": "25202866692096",
        "physicalUsedStorage": "25202866692096",
        "physicalUsedStorageIncludingParity": "25202867311353",
        "freePhysicalStorage": "6123549568203",
        "estimatedFreeLogicalStorage": "6123549568203"
    }
}
```
Note: This example output only shows the first block of information. The command returns much more data for each node.

You can also use this command to get the information for a specific node by adding the node UUID to the command. For example:


REST Get Volumes

To review details of volumes, use the GET method with this URL /api/v1/volumes.

Sample Command


Sample Output
You can also use this command to get the information for a specific volume by adding the volume UUID to the command. For example:

curl -s -X GET http://192.168.16.153:80/api/v1/volumes/6cc0b3ae-f5fe-426e-acff-a31afbc1f1b7

**REST Create Volume**

To create a volume, use the **POST** method command for this URL /api/v1/volumes.

**Note:** This POST method command includes many parameters that are passed to the curl utility. In the following example, the text wraps across multiple lines but it is still a single “curl -x POST” command.

**Sample Command**


**Sample Output**
To delete a volume, use the DELETE method command for this URL /api/v1/volume/{volume UUID}.

Sample Command
```
curl -X DELETE http://192.168.16.187:80/api/v1/volumes/7bbeb0e4-dffc-420c-ae6d-a3b7a604d20d -H 'Accept: application/json' -H 'Content-type: application/json' -H 'Expect: '
```

To review details of volumes, use the GET method for this URL /api/v1/nvmeDevices.

Sample Command
```
```

Sample Output
```
{
  "size": "1000204886016",
  "numaNodeId": "1",
  "model": "INTEL SSDPE2XX010T7",
  "serial": "PHLF734301AX1P0GGN",
  "serverUUID": "75db8358-a46e-48fc-8147-5d60735e66a0",
  "state": "None",
  "failureTime": null,
  "rebuildCompletionTime": null,
  "name": "nvme14n1",
  "nodeUUID": "75db8358-a46e-48fc-8147-5d60735e66a0"
}
```
LightOS REST API V2 Overview

You can configure and manage the LightOS cluster with a standard REST API developed by Lightbits.

This section includes several examples of how to use common management commands via REST API using a web interface and the Linux “curl” utility to send and receive data with the REST API methods:

- GET
- CREATE
- DELETE

Note: For a complete reference of the LightOS REST API interface, see the LightOS API User Manual.

REST API Syntax

The following example shows how the Linux curl utility is used in this document to provide REST API usage examples:

```
curl -X <REST API VERB> http://<IP of LightOS Node>:<TCP PORT>/api/v2/<command>
```

You can use any of the cluster node’s IP addresses (not necessarily the node affected by the command).

Using REST API Commands (curl Examples)

If you use the REST API GET method, the LightOS management service returns the data in JSON format.

REST Get Cluster

To get information on the cluster (NQN, replicas, etc.), use the following command:

Sample Command

```
```

Sample Output
REST Get Events

Returns a list of events in the cluster, and information on each event.

To retrieve events reported in the cluster (Node Inactive, DiskFailed, Volumed Degraded), use the following command:

```
```

[JWT] would be the LightOS cluster JWT. The easiest option is to add the JWT to a variable, as shown in the example below. [LightOS Node] would be the name or IP of one of the LightOS cluster’s nodes.

Sample Command
REST Get Events

LIGHTOS REST API V2 OVERVIEW


Sample Output
```json
{
    "events": [
        {
            "ID": "0f0315f1-d6a4-4ce7-84a5-e2e422346eb",
            "Time": "2021-09-29T22:02:42.54610605Z",
            "Type": "Node",
            "Severity": "Info",
            "EventName": "NodeEnteredActiveState",
            "EventCode": 200,
            "ReportingService": "CM",
            "AssociatedEventID": "",
            "Status": "Available",
            "CauseCode": 1,
            "Description": "NoCause",
            "ComponentNodeInfo": {
                "ID": "9c623d83-cf12-5fee-b0f2-60385064d306",
                "Name": ""
            }
        },
        {
            "ID": "ab8fd2c8-1dfb-4372-8a80-4ed93f36c418",
            "Time": "2021-09-29T22:02:42.26590664Z",
            "Type": "Node",
            "Severity": "Info",
            "EventName": "NodeEnteredActiveState",
            "EventCode": 200,
            "ReportingService": "CM",
            "AssociatedEventID": "",
            "Status": "Available",
            "CauseCode": 1,
            "Description": "NoCause",
            "ComponentNodeInfo": {
                "ID": "606e6e52-4583-56fd-884b-d0f19cbb1b16",
                "Name": ""
            }
        },
        {
            "ID": "58403a7f-f381-4729-aa27-ff8e04833a2",
            "Time": "2021-09-29T22:02:42.08567598Z",
            "Type": "Node",
            "Severity": "Info",
            "EventName": "NodeEnteredActiveState",
            "EventCode": 200,
            "ReportingService": "CM",
            "AssociatedEventID": "",
            "Status": "Available",
            "CauseCode": 1,
            "Description": "NoCause",
            "ComponentNodeInfo": {
                "ID": "1cadd6d4-2499-561d-a9d5-e42cd44e20a0",
                "Name": ""
            }
        }
    ],
    "nextToken": ""
}
```
REST Get Nodes

To get a quick summary of the status and capacity of the nodes on the server running LightOS, use the GET method with this URL `/api/v2/nodes`.

Sample Command


Sample Output

```json
{
    "nodes": [
    {
        "name": "server00-0",
        "UUID": "5d8ef00-82b0-559f-a523-990ac14ddb09",
        "state": "Active",
        "status": "NoStatus",
        "nvmeEndpoint": "10.166.14.1:4420",
        "failureDomains": [
            "server00"
        ],
        "failureInfo": "",
        "hostname": "rack12-server61",
        "inLocalRebuild": false,
        "localRebuildProgress": 0,
        "numManagedDevices": 6,
        "maxNvmeDevices": 6,
        "ec": true,
        "statistics": {
            "managedPhysicalStorage": "24004722180096",
            "effectivePhysicalStorage": "17403423473664",
            "logicalStorage": "2199023255552",
            "logicalUsedStorage": "0",
            "physicalUsedStorage": "0",
            "physicalUsedStorageIncludingParity": "0",
            "freePhysicalStorage": "17403423473664",
            "estimatedFreeLogicalStorage": "17403423473664",
            "estimatedLogicalStorage": "17403423473664",
            "compressionRatio": 1
        },
        "serverUUID": "95c8298f-77d6-504f-ba15-7eacf2b9e1a",
        "ETag": "0",
        "readOnly": true,
        "powerupProgress": 100
    }
    ]
}
```

Note: This example output only shows the first block of information. The command returns much more data for each node.
REST Get Volumes

You can also use this command to get the information for a specific node by adding the node UUID to the command. For example:


REST Get Volumes

To review details of volumes - such as size, ACL, UUID, etc - use the GET method with this URL /api/v2/volumes.

Sample Command


Sample Output
You can also use this command to get the information for a specific volume by adding the volume UUID to the command. For example:

curl -s -X GET http://192.168.16.153:80/api/v2/volumes/6cc0b3ae-f5fe-426e-acff-a31afbc1f1b7

REST Create Volume

This command will create 2TB volume with replica 1 named “vol1” in the default project.
To create a volume, use the POST method command for this URL /api/v2/volumes.

Note: This POST method command includes many parameters that are passed to the curl utility. In the following example, the text wraps across multiple lines but it is still a single “curl -x POST” command.

Sample Command
Sample Output

```
{
  "state": "Creating",
  "protectionState": "Unknown",
  "replicaCount": 1,
  "nodeList": [],
  "UUID": "26be61ba-faa6-4156-bd09-4c8a32d37b39",
  "nsid": 0,
  "acl": {
    "values": ["ALLOW_NONE"]
  },
  "compression": "false",
  "size": "2199023255552",
  "name": "vol1",
  "rebuildProgress": "",
  "statistics": {
    "logicalUsedStorage": "0",
    "physicalUsedStorage": "0",
    "compressionRatio": 0
  },
  "IPACL": {
    "values": ["ALLOW_ANY"]
  },
  "ETag": "",
  "connectedHosts": [],
  "sectorSize": 4096,
  "projectName": "default"
}
```

**REST Delete Volume**

This command will delete a volume with UUID: 055ab16b-8528-46c4-940e-5493ec19f839

To delete a volume, use the DELETE method command for this URL `/api/v2/volume/{volume UUID}`.

**Sample Command**

```
```

**REST Get NVMe Devices**

To review details of volumes, use the GET method for this URL `/api/v1/nvmeDevices`.

```
Sample Command

```

2022 Lightbits Labs

Proprietary And Confidential - Under NDA Only
Sample Command


Sample Output
"NvmeDevices": [
{
    "size": "4000787030016",
    "numaNodeID": "0",
    "model": "Micron_9300_MTFHAL4TATFE",
    "serial": "1928234556E",
    "serverUUID": "8a1ee893-362c-5778-9176-fbd9986da42f",
    "state": "Healthy",
    "failureTime": null,
    "rebuildCompletionTime": null,
    "name": "nvme7n1",
    "nodeUUID": "9c4cb1c4-6e76-5741-a671-4f407eca7c94",
    "ETag": "0"
},
{
    "size": "4000787030016",
    "numaNodeID": "0",
    "model": "Micron_9300_MTFHAL4TATFE",
    "serial": "1928234558D",
    "serverUUID": "8a1ee893-362c-5778-9176-fbd9986da42f",
    "state": "Healthy",
    "failureTime": null,
    "rebuildCompletionTime": null,
    "name": "nvme6n1",
    "nodeUUID": "9c4cb1c4-6e76-5741-a671-4f407eca7c94",
    "ETag": "0"
},
{
    "size": "4000787030016",
    "numaNodeID": "0",
    "model": "Micron_9300_MTFHAL4TATFE",
    "serial": "19282345561",
    "serverUUID": "8a1ee893-362c-5778-9176-fbd9986da42f",
    "state": "Healthy",
    "failureTime": null,
    "rebuildCompletionTime": null,
    "name": "nvme6n1",
    "nodeUUID": "9c4cb1c4-6e76-5741-a671-4f407eca7c94",
    "ETag": "0"
},
{
    "size": "4000787030016",
    "numaNodeID": "0",
    "model": "Micron_9300_MTFHAL4TATFE",
    "serial": "19282345695",
    "serverUUID": "8a1ee893-362c-5778-9176-fbd9986da42f",
    "state": "Healthy",
    "failureTime": null,
    "rebuildCompletionTime": null,
    "name": "nvme0n1",
    "nodeUUID": "9c4cb1c4-6e76-5741-a671-4f407eca7c94",
    "ETag": "0"
},
{
    "size": "4000787030016",
    "numaNodeID": "0",
    "model": "Micron_9300_MTFHAL4TATFE",
    "serial": "19282345759",
    "serverUUID": "8a1ee893-362c-5778-9176-fbd9986da42f",
    "state": "Healthy",
    "failureTime": null,
    "rebuildCompletionTime": null,
    "name": "nvme6n1",
    "nodeUUID": "9c4cb1c4-6e76-5741-a671-4f407eca7c94",
    "ETag": "0"
}]}
NVME CLI Overview

Each application client that you connect to a LightOS storage server must support the NVMe Over Fabrics (NVMe-oF) network protocol.

**Note:** See the LightOS Install Guide for instructions on obtaining the correct application client nvme drivers and installing the NVMe CLI.

NVMe CLI Syntax

The NVMe CLI utility has many sub-commands for performing management tasks related to nvme device discovery, connection, and disconnection. There are many other commands outside the scope of this document as well. See the nvme documentation for more detailed and complete information.

Discovery-Client Command

The NVMe-over-Fabrics specification defines the concept of a discovery controller that an NVMe host can query on a fabric network to discover NVMe subsystems. These subsystems are contained in NVMe targets that can be connected from the network. Achieving a persistent discovery connection is possible with Kernel Supporting AEN (kernel >= 5.5 on the client).

When using a kernel < 5.5 you need to use the discovery-client command in order to achieve a persistent discovery service. The discovery-client is a deployable service running under systemd, that will poll every 5s (configurable) the discovery-services in the cluster. The purpose of this service is to overcome limitations in the old kernel - kernels that do not support AEN and kernels with bugs. The discovery-client command requires several parameters that you can query for the NVMe/TCP storage server to be discovered and connected to.

Install the discovery-client rpm

```bash
yum install -y discovery-client-2.0.5-1-9fb632c67304.x86_64.rpm
```

Start the discovery-client service

```bash
systemctl start discovery-client.service
```

Get the hostnqn

```bash
cat /etc/nvme/hostnqn
```

In case there is no hostnqn, you can generate one by using the following command:

```bash
nvme gen-hostnqn > /etc/nvme/hostnqn
```

The simplified syntax of this command is:

```bash
discovery-client connect-all -t tcp -a <ip> -s 8009 -q <hostnqn> -p
```
These are only a small sample of the possible parameters. See the discovery-client help menu for a complete list and usage of the discovery-client command. We have chosen these parameters as an example of the minimum required to connect to and use an NVMe/TCP volume.

After the application client connects to the LightOS storage cluster, the volumes that this application client can access appear as local block devices. The following example shows how to use the Linux `nvme` command to list all block devices after the `connect` command finishes.

### Sample Command

```bash
$ nvme list
```

### Sample Output

<table>
<thead>
<tr>
<th>Node</th>
<th>SN</th>
<th>Model</th>
<th>Usage</th>
<th>Format</th>
<th>FW Rev</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/nvme1n1</td>
<td>f2e1b3930c2ed3</td>
<td>LightBox</td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>/dev/nvme1n2</td>
<td>f2e1b3930c2ed3</td>
<td>LightBox</td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>/dev/nvme1n3</td>
<td>f2e1b3930c2ed3</td>
<td>LightBox</td>
<td>12.88 GB / 12.88 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>/dev/nvme1n4</td>
<td>f2e1b3930c2ed3</td>
<td>LightBox</td>
<td>12.88 GB / 12.88 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
<td>5</td>
</tr>
</tbody>
</table>

In this example, the LightOS storage server serves four volumes via NVMe/TCP. These volumes are mapped to the `/dev` Linux file system path. The NVMe/TCP block device names are listed in the `Node` column.

### NVMe Connect Command

The `nvme` `connect` command requires several parameters for the NVMe/TCP storage server to be discovered and connected to.

The simplified syntax of this command is:

```
```

These are only a small sample of the possible parameters. See the `nvme` documentation for a complete list and usage of the `nvme connect` command. We have chosen these parameters as an example of the minimum required to connect to and use an NVMe/TCP volume.

### NVMe Connect Parameter Overview

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nvme connect</code></td>
<td>The standard NVMe-oF command for a Linux application client to connect to an NVMe-oF target.</td>
</tr>
<tr>
<td><code>-t [transport type]</code></td>
<td>Indicates to the application client’s NVMe driver the transport protocol used to connect to the remote target. For Lightbits, the transport protocol is TCP.</td>
</tr>
<tr>
<td><code>-a [IP address]</code></td>
<td>The LightOS data NIC’s IP address.</td>
</tr>
<tr>
<td><code>-s [target tcp port]</code></td>
<td>The TCP port where NVMe commands are received by LightOS.</td>
</tr>
</tbody>
</table>
### NVMe Discover Command

The NVMe-over-Fabrics specification defines the concept of a discovery controller that an NVMe host can query on a fabric network to discover NVMe subsystems. These subsystems are contained in NVMe targets that can be connected from the network. Achieving a persistent discovery connection is possible with Kernel Supporting AEN (kernel >= 5.5 on the client). The `nvme discover` command requires several parameters for the NVMe/TCP storage server to be discovered and connected to.

Note that the base NVMe specification defines the NQN (NVMe Qualified Name) format, which an NVMe endpoint (device, subsystem, etc.) must follow to guarantee a unique name under the NVMe standard. In particular, the host NQN uniquely identifies the NVMe host, and can be used by the discovery controller to control what NVMe target resources are allocated to the NVMe host for a connection.

**Get the hostnqn.**

```
cat /etc/nvme/hostnqn
```

In case there is no `hostnqn` you can generate one by using the following command:

```
nvme gen-hostnqn > /etc/nvme/hostnqn
```

The simplified syntax of this command is:

### NVMe CLI Overview

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n [target nqn]</td>
<td>The NQN assigned to the LightOS cluster where the volume was created.</td>
</tr>
<tr>
<td>-q [acl]</td>
<td>The access control list (ACL) string used when the volume was created on the LightOS cluster to restrict access to the volume.</td>
</tr>
<tr>
<td>-l [timeout]</td>
<td>When connecting to a LightOS cluster, this should be set to -1.</td>
</tr>
</tbody>
</table>

After the application client connects to the LightOS storage cluster, the volumes that this application client can access appear as local block devices. The following example shows how to use the Linux `lsblk` command to list all block devices after the `connect`command finishes.

#### Sample Command

$ `lsblk`

#### Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>MAJ:MIN</th>
<th>RM</th>
<th>SIZE</th>
<th>RO</th>
<th>TYPE</th>
<th>MOUNTPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvme1n2</td>
<td>259:3</td>
<td>0</td>
<td>500G</td>
<td>0</td>
<td>disk</td>
<td></td>
</tr>
<tr>
<td>nvme0n2</td>
<td>259:1</td>
<td>0</td>
<td>500G</td>
<td>0</td>
<td>disk</td>
<td></td>
</tr>
<tr>
<td>nvme1n1</td>
<td>259:2</td>
<td>0</td>
<td>10G</td>
<td>0</td>
<td>disk</td>
<td></td>
</tr>
<tr>
<td>nvme0n1</td>
<td>259:0</td>
<td>0</td>
<td>10G</td>
<td>0</td>
<td>disk</td>
<td></td>
</tr>
</tbody>
</table>

In this example, the LightOS storage server serves four volumes via NVMe/TCP. These volumes are mapped to the `/dev` Linux file system path. The NVMe/TCP block device names are listed in the **Name** column.

### NVMe Discover Command

The NVMe-over-Fabrics specification defines the concept of a discovery controller that an NVMe host can query on a fabric network to discover NVMe subsystems. These subsystems are contained in NVMe targets that can be connected from the network. Achieving a persistent discovery connection is possible with Kernel Supporting AEN (kernel >= 5.5 on the client). The `nvme discover` command requires several parameters for the NVMe/TCP storage server to be discovered and connected to.

Note that the base NVMe specification defines the NQN (NVMe Qualified Name) format, which an NVMe endpoint (device, subsystem, etc.) must follow to guarantee a unique name under the NVMe standard. In particular, the host NQN uniquely identifies the NVMe host, and can be used by the discovery controller to control what NVMe target resources are allocated to the NVMe host for a connection.

**Get the hostnqn.**

```
cat /etc/nvme/hostnqn
```

In case there is no `hostnqn` you can generate one by using the following command:

```
nvme gen-hostnqn > /etc/nvme/hostnqn
```

The simplified syntax of this command is:
**nvme discover -t tcp -a <ip> -s 8009 -q <hostnqn> -p**

These sets of parameters are given as an example. See the nvme documentation for a complete list and usage of the nvme discover command. We have chosen these parameters as an example of the minimum required to connect to and use an NVMe/TCP volume.

The hostnqn parameter is configured in a txt file that sits in `/etc/nvme/hostnqn`. When this is modified and a the nvme discover command is invoked, a systemd unit will get called to run connect-all.

After the application client connects to the LightOS storage cluster, the volumes that this application client can access appear as local block devices. The following example shows how to use the Linux `nvme` command to list all block devices after the `connect` command finishes.

**Sample Command**

```
$ nvme list
```

**Sample Output**

<table>
<thead>
<tr>
<th>Node</th>
<th>SN</th>
<th>Model</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/nvme7n1</td>
<td>b23f8854fe984771</td>
<td>LightBox</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
</tr>
<tr>
<td>/dev/nvme7n2</td>
<td>b23f8854fe984771</td>
<td>LightBox</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
</tr>
<tr>
<td>/dev/nvme7n3</td>
<td>b23f8854fe984771</td>
<td>LightBox</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
</tr>
<tr>
<td>/dev/nvme7n4</td>
<td>b23f8854fe984771</td>
<td>LightBox</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>4.29 GB / 4.29 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
</tr>
<tr>
<td>/dev/nvme7n5</td>
<td>b23f8854fe984771</td>
<td>LightBox</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.74 GB / 10.74 GB</td>
<td>4 KiB + 0 B</td>
<td>1.0</td>
</tr>
</tbody>
</table>

In this example, the LightOS storage server serves four volumes via NVMe/TCP. Those volumes are mapped to the `/dev` Linux file system path. The NVMe/TCP block device names are listed in the **Node** column.

**Getting the Application Client’s “HostNQN”**

The application client running NVMe software drivers will have an NVMe Qualified Name, or NQN. The value is stored in the Linux file system under the following path:

**Sample Command**

```
/etc/nvme/hostnqn
```

If that directory is not automatically created upon installation of the NVMEe driver and utility software, you can generate the directory with the following procedure:

1. Create a directory and run the `nvme gen-hostnqn` command:
Sample Command

```bash
$ mkdir /etc/nvme
nvme gen-hostnqn > /etc/nvme/hostnqn
```

2. Use the following command to see the application client nvme hostnqn:

**Sample Command**

```bash
$ cat /etc/nvme/hostnqn
```

**Sample Output**

```
nqn.2014-08.org.nvmexpress:NVMf:uuid:e0411a14-fb39-4851-b0e9-5910a7d340bf
```
LightOS System Health Monitoring

LightOS supports interfacing with the Prometheus open-source systems monitoring and alerting toolkit.

Metrics and Statistics

The following groups of collectors are supported by LightOS to report node and cluster metrics and statistics on the Prometheus dashboard.

By default, the following collector groups are enabled.

<table>
<thead>
<tr>
<th>Name</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total user IOPS (read,write)</td>
<td>cluster</td>
<td>IOPs user read / user write (not including replications).</td>
</tr>
<tr>
<td>Total user throughput (read,write)</td>
<td>cluster</td>
<td>Throughput (Bps) user read / user write (not including replications).</td>
</tr>
<tr>
<td>Cluster Installed Physical Storage</td>
<td>cluster</td>
<td>All installed SSDs capacities over all servers in cluster, given in bytes. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Managed Physical Storage</td>
<td>cluster</td>
<td>All managed and healthy SSDs capacities, given in bytes. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Effective Physical Storage</td>
<td>cluster</td>
<td>Effective physical storage excluding overhead of EC and OVP, given in bytes. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Logical Storage</td>
<td>cluster</td>
<td>Sum of capacities of all allocated volumes, given in bytes. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Free Physical Storage</td>
<td>cluster</td>
<td>Free storage before entering read-only mode. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Estimated Free Logical Storage</td>
<td>cluster</td>
<td>Estimated free storage before entering read-only mode assuming the current compression ratio. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Estimated EffectiveLogical Storage</td>
<td>cluster</td>
<td>Estimated storage up to RO threshold from the point view of the client (considering compression). The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Physical Used Storage Including Parity</td>
<td>cluster</td>
<td>The sum of physical storage used on each of the nodes. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Cluster Physical Used Storage</td>
<td>cluster</td>
<td>The sum of physical storage used on each of the nodes (sum on 1.x nodes physical metric, this does not account for EC overhead). Excluding Parity. The sum includes Inactive nodes.</td>
</tr>
<tr>
<td>Name</td>
<td>Scope</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cluster Health</td>
<td>cluster</td>
<td>OK (no admin action required) == All volumes are fully protected. There is no inactive node, and no node in read-only warning (admin action required, but there is no loss of service) == There is at least one volume which is degraded. There is no volume that is read-only OR there is an Inactive or read-only node error (admin action required, there is loss of service) == There is a volume without working replicas or in a read-only state.</td>
</tr>
<tr>
<td>Number of active nodes</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Number of failed nodes</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Number of volumes</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Number of degraded volumes</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Number of volumes in read-only</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Number of volume not-available</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>Cluster Logical Used Storage</td>
<td>cluster</td>
<td>Actual User Objects saved (should be equal to the number of valid LBAs * 4K Block Size)</td>
</tr>
<tr>
<td>Compression ratio</td>
<td>cluster</td>
<td></td>
</tr>
<tr>
<td>IOPS (read, write)(user/replication)</td>
<td>node</td>
<td>IOps user read / user write / replication tx / replication rx</td>
</tr>
<tr>
<td>Throughput</td>
<td>node</td>
<td>Throughput (Bps) user read / user write / replication tx / replication rx</td>
</tr>
<tr>
<td>Latency avg/99/99.99 write/read</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Average IO Size write/read</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Managed Physical Storage</td>
<td>node</td>
<td>All managed and healthy SSDs capacities, given in bytes. Sum of capacities of all allocated volumes, given in bytes. Estimated storage up to RO threshold from the point view of the client (considering compression). Effective physical storage excluding overhead of EC and OVP, given in bytes.</td>
</tr>
<tr>
<td>Node Logical Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Estimated Effective Logical Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Effective Physical Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Free Physical Storage</td>
<td>node</td>
<td>Assuming current compression. Physical storage used by a node (incl EC). Return value before taking into account any replication factor for unused Report Capacity available after internal LightOS OP needs and up to RO threshold. Does not assume compression, may be limited by DRAM. Sum of used compressed storage of all valid LBAs in the node. Excluding Parity.</td>
</tr>
<tr>
<td>Node Estimated Free Logical Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Physical Used Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Including Parity</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Node Logical Used Storage</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>Write amplification</td>
<td>node</td>
<td>Sum of all written LBAs by user.</td>
</tr>
<tr>
<td>LF temperature</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>LF PLX switch temperature</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>gc skips</td>
<td>node</td>
<td></td>
</tr>
</tbody>
</table>
### LIGHTOS SYSTEM HEALTH MONITORING

<table>
<thead>
<tr>
<th>Name</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC-rebuild progress</td>
<td>node</td>
<td>nvme smart-log</td>
</tr>
<tr>
<td>Error stats</td>
<td>NVMe/SSD</td>
<td>nvme smart-log</td>
</tr>
<tr>
<td>Remaining life cycle</td>
<td>NVMe/SSD</td>
<td>Healthy / degraded / read-only /</td>
</tr>
<tr>
<td>State</td>
<td>volume</td>
<td>no-service</td>
</tr>
<tr>
<td>Clustering rebuild progress</td>
<td>volume</td>
<td>IOPs user read / user write /</td>
</tr>
<tr>
<td></td>
<td></td>
<td>replication tx / replication rx</td>
</tr>
<tr>
<td>IOPS (read,write)</td>
<td>volume, node</td>
<td>Throughput (Bps) user read / user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>write / replication tx / replication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rx</td>
</tr>
<tr>
<td>Throughput (read,write)</td>
<td>volume, node</td>
<td>Logical storage space used by volume,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>given in bytes.</td>
</tr>
<tr>
<td>Average IO Size write/read</td>
<td>volume, node</td>
<td>Physical storage space used by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>volume, given in bytes. Excluding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parity.</td>
</tr>
<tr>
<td>Volume Logical Used Storage</td>
<td>volume, node</td>
<td>Logical storage space used by volume,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>given in bytes.</td>
</tr>
<tr>
<td>Volume Physical Used Storage</td>
<td>volume, node</td>
<td>Physical storage space used by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>volume, given in bytes. Excluding</td>
</tr>
<tr>
<td>Compression ratio</td>
<td>volume, node</td>
<td>parity.</td>
</tr>
</tbody>
</table>

### Alerts

You can use alert rules based on the Prometheus expression language for alert notifications, which can be used to send notifications to an external service. The following list details the LightOS status alerts.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerMetricsDown</td>
<td>If lightbox exporters are not available for 1 min (period set in rules.yaml).</td>
</tr>
<tr>
<td>NodeAlmostReadOnlyMode</td>
<td>The node’s LightOS system is approaching read only mode.</td>
</tr>
<tr>
<td>RebuildInProgress/Failed/Finished Not Possible</td>
<td>The status of a node has begun a data rebuild process on a block device.</td>
</tr>
<tr>
<td>NodeReadOnlyMode, NodeSystemReadOnly</td>
<td>A node entered a read only mode. A node is in read only mode after a number of SSDs have been identified as failed.</td>
</tr>
<tr>
<td>NodeRebuildNotPossible</td>
<td>A node cannot rebuild data after a number of SSDs have been identified as failed.</td>
</tr>
<tr>
<td>NodePowerUpAfterAbruptShutdown/Failed/GracefulShutdownFailed</td>
<td>A node is powering up after an abrupt shutdown.</td>
</tr>
<tr>
<td>SSDWasReset</td>
<td>An SSD was reset in a node.</td>
</tr>
<tr>
<td>SSDWriteError</td>
<td>An SSD write error occurred in a node.</td>
</tr>
<tr>
<td>SSDReadError</td>
<td>An SSD read error occurred in a node.</td>
</tr>
<tr>
<td>SSDAddedSuccessfully</td>
<td>An SSD was successfully added in a node.</td>
</tr>
<tr>
<td>SSDAddFailure</td>
<td>An SSD failed to be added in a node.</td>
</tr>
<tr>
<td>SSDFailure</td>
<td>An SSD failed in a node.</td>
</tr>
<tr>
<td>NodeBecameInactive</td>
<td>A node became not active (switched from active to any other state).</td>
</tr>
<tr>
<td>NodeBecameActive</td>
<td>A node became active (switched to active from any state).</td>
</tr>
</tbody>
</table>
Appendixes

The following sections provide additional information for the LightOS Admin Guide.

Limitations and Boundaries

The following are the LightOS product limitations and boundaries.

<table>
<thead>
<tr>
<th>Object (Max Size)</th>
<th>Domain</th>
<th>System Hard Limit</th>
<th>System Validation Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of volumes</td>
<td>Cluster</td>
<td>64k</td>
<td>10k</td>
</tr>
<tr>
<td>Number of servers*</td>
<td>Cluster</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Size of single provisioned volume</td>
<td>Cluster</td>
<td>256 TB</td>
<td>256 TB</td>
</tr>
<tr>
<td>Number of snapshots overall</td>
<td>Cluster</td>
<td>64k</td>
<td>10k</td>
</tr>
<tr>
<td>Number of snapshots policies</td>
<td>Cluster</td>
<td>64k</td>
<td>333</td>
</tr>
<tr>
<td>Number of JWTs</td>
<td>Cluster</td>
<td>1k</td>
<td>10</td>
</tr>
<tr>
<td>Number of total projects</td>
<td>Cluster</td>
<td>1k</td>
<td>1k</td>
</tr>
<tr>
<td>Number of events</td>
<td>Cluster</td>
<td>10k</td>
<td>10k</td>
</tr>
<tr>
<td>Number of failure domains**</td>
<td>Cluster</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Number of SSDs***</td>
<td>Server</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>Size of single SSD</td>
<td>Server</td>
<td>32 TB</td>
<td>16 TB</td>
</tr>
<tr>
<td>Number of volumes</td>
<td>Project</td>
<td>64k</td>
<td>10k</td>
</tr>
<tr>
<td>Number of snapshots of a given volume/hierarchy</td>
<td>Volume</td>
<td>1k</td>
<td>1k</td>
</tr>
<tr>
<td>Number of snapshot policies</td>
<td>Volume</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note:** *The minimum number for servers per cluster is 3. One server is also allowed, but with no clustering support. The server can be a single or dual instance.

**Note:** **The minimum number of domains is 3.

**Note:** ***The minimum number of disks per server is 6 with EC enabled, and 4 with non-EC configuration.
About - Legal

Lightbits™ Labs, founded in 2016, is a software-defined storage company that brings hyperscale efficiency and flexibility to everyone. Lightbits' solution delivers composable, disaggregated storage that performs like local flash. The company pioneered NVMe/TCP so the solution is easy to deploy at scale while delivering performance that from applications’ perspective is indistinguishable from local SSDs. Lightbits Labs is backed by strategic investors including Dell Technologies Capital, Cisco Investments and Micron, as well as top investors and VCs including Avigdor Willenz, Lip-Bu Tan, Marius Nacht, SquarePeg Capital, and Walden International.

www.lightbitslabs.com
info@lightbitslabs.com

North America Office
1830 The Alameda, San Jose, CA 95126

Israel Offices * 17 Atir Yeda Street,Kfar Saba, Israel 4464313 * 3 Habankim Street, Haifa, Israel 3326115

The information in this document and any document referenced herein is provided for informational purposes only, is provided as is and with all faults and cannot be understood as substituting for customized service and information that might be developed by Lightbits Labs ltd for a particular user based upon that user's particular environment. Reliance upon this document and any document referenced herein is at the user's own risk.

The software is provided “As is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall the contributors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings with the software.

Unauthorized copying or distributing of included software files, via any medium is strictly prohibited.

COPYRIGHT (C) 2022 LIGHTBITS LABS LTD. - ALL RIGHTS RESERVED