



**Hewlett Packard**  
Enterprise

## **Cray ClusterStor Data Services Administrator Guide 2.0 (S-1237 Revision B)**

Part Number: S-1237  
Published: December 2021  
Edition: 1

# **Cray ClusterStor Data Services Administrator Guide 2.0 (S-1237 Revision B)**

## **Abstract**

This guide contains instructions for managing, monitoring, and troubleshooting ClusterStor data services.

Part Number: S-1237

Published: December 2021

Edition: 1

© Copyright 2021, 2021 Hewlett Packard Enterprise Development LP

## Table of contents

### 1 Notices

#### 1.1 Acknowledgments

### 2 Support and other resources

#### 2.1 Accessing Hewlett Packard Enterprise Support

#### 2.2 Accessing updates

#### 2.3 Remote support

#### 2.4 Warranty information

#### 2.5 Regulatory information

#### 2.6 Documentation feedback

### 3 About the Cray ClusterStor data services Administrator Guide

#### 3.1 Command Example Conventions

### 4 How ClusterStor data services Benefits ClusterStor Users

#### 4.1 What ClusterStor data services Provides

##### 4.1.1 Lustre Tiering Limitations

#### 4.2 How ClusterStor data services Improves Upon Lustre

### 5 ClusterStor data services Architecture and Components

#### 5.1 Scalable Data Movement Infrastructure

#### 5.2 Lustre HSM Emitter

#### 5.3 ClusterStor data services API Agent

#### 5.4 Tiering Engine

#### 5.5 Connector

#### 5.6 Policy Engine

##### 5.6.1 Trigger Agent

##### 5.6.2 Query Agent

##### 5.6.3 Scalable Search Service

#### 5.7 Scalable Search Service Index Directories

### 6 Configure ClusterStor data services After Installation

#### 6.1 Configure OST Pools

##### 6.1.1 Automatically Configure OST Pools Using CSCLI

##### 6.1.2 Manually Configure OST Pools Using CSCLI

#### 6.2 Default Data Placement Policy

##### 6.2.1 Set a Default Data Placement Policy

#### 6.3 Create a Template

#### 6.4 Install Scalable Search Tools on a Lustre Client

#### 6.5 Configure an External Administration System

##### 6.5.1 Install CSMS CLI

#### 6.6 Restrict Access to Policy Directory

#### 6.7 Add or Remove Users in Keycloak

#### 6.8 Configure Automatic Data Movement

#### 6.9 Update High Speed Network, LNet, and Lustre Client Configuration

### 7 Move File Data with ClusterStor data services

#### 7.1 Manual Data Migration

- [7.1.1 Request a File Migration from a Lustre Client](#)
    - [7.1.2 Request File Data Movement with CSMS CLI](#)
  - [7.2 ClusterStor data services Policies](#)
    - [7.2.1 Policy Syntax](#)
  - [7.3 Check Policy Files for Errors](#)
  - [7.4 Inspect Parser Interpretation of Policy Files](#)
- [8 Check File Movement Status](#)
  - [8.1 Check File Movement Status with lfs getstripe](#)
  - [8.2 Check File Movement Status Using File ctime](#)
  - [8.3 Query Data Movement Request Status with CSMS CLI](#)
- [9 Find Files with Scalable Search](#)
  - [9.1 Search for Files Using Query](#)
  - [9.2 Run Arbitrary Commands on Query Search Results](#)
  - [9.3 Delete Files in Parallel with Query](#)
- [10 Monitor ClusterStor data services](#)
  - [10.1 Monitor ClusterStor data services Activity with Lustre Jobstats](#)
  - [10.2 View ClusterStor data services Kibana Dashboards](#)
  - [10.3 Quickly Assess Overall Health Using Kibana](#)
  - [10.4 Verify Scalable Search Service Health with Kibana](#)
  - [10.5 Verify Automated Data Movement with Kibana](#)
  - [10.6 View ClusterStor data services metrics in Grafana](#)
- [11 About Scalable Search](#)
  - [11.1 Administrator Search Tools](#)
- [12 About ClusterStor data services Policies](#)
  - [12.1 How Files are Purged](#)
  - [12.2 Policy Engine Limits Data Movement Requests](#)
  - [12.3 The testparse Policy Checking Tool](#)
  - [12.4 Data Movement mtime Verification](#)
  - [12.5 Flash Maintenance](#)
- [13 Troubleshoot ClusterStor data services](#)
  - [13.1 Degraded Performance on a Management Node](#)
- [14 Reference](#)
  - [14.1 ClusterStor data services Templates](#)
  - [14.2 testparse Command Syntax](#)
  - [14.3 Policy File Errors testparse Can Detect](#)
  - [14.4 query Command Syntax](#)
  - [14.5 Which Lustre HSM Requests Are Emitted to ClusterStor data services](#)
  - [14.6 ClusterStor data services Elasticsearch Fields](#)
  - [14.7 About the CSMS CLI](#)
  - [14.8 CSMS CLI Commands](#)
    - [14.8.1 Common CSMS CLI Options and Environment Variables](#)
    - [14.8.2 csms init](#)
    - [14.8.3 csms auth login](#)
    - [14.8.4 csms config](#)

14.8.5 csms base software mappings list

14.8.6 csms base software mappings update

14.8.7 csms base software releases create

14.8.8 csms base software releases describe

14.8.9 csms base software releases list

14.8.10 csms base status list

14.8.11 csms base tasks list

14.8.12 csms base tasks describe

14.8.13 csms inventory status list

14.8.14 csms inventory system summary list

14.8.15 csms inventory system summary update

14.8.16 csms cds requests create

14.8.17 csms cds requests delete

14.8.18 csms cds requests describe

14.8.19 csms cds requests list

14.8.20 csms cds status list

14.8.21 csms cds version list

## Notices

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Links to third-party websites take you outside the Hewlett Packard Enterprise website. Hewlett Packard Enterprise has no control over and is not responsible for information outside the Hewlett Packard Enterprise website.

## Acknowledgments

Kubernetes® is a registered trademark The Linux Foundation in the United States and/or other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

The Lustre® trademark is jointly owned by OpenSFS and EOFS.

UNIX® is a registered trademark of The Open Group.





## Accessing Hewlett Packard Enterprise Support

- For live assistance, go to the Contact Hewlett Packard Enterprise Worldwide website:

<https://www.hpe.com/info/assistance>

- To access documentation and support services, go to the Hewlett Packard Enterprise Support Center website:

<https://www.hpe.com/support/hpesc>

### Information to collect

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

## Accessing updates

- Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.

- To download product updates:

Hewlett Packard Enterprise Support Center

<https://www.hpe.com/support/hpesc>

Hewlett Packard Enterprise Support Center: Software downloads

<https://www.hpe.com/support/downloads>

My HPE Software Center

<https://www.hpe.com/software/hpesoftwarecenter>

- To subscribe to eNewsletters and alerts:

<https://www.hpe.com/support/e-updates>

- To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center More Information on Access to Support Materials page:

<https://www.hpe.com/support/AccessToSupportMaterials>

---

**❗ IMPORTANT:**

Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HPE Passport set up with relevant entitlements.

---

## Remote support

Remote support is available with supported devices as part of your warranty or contractual support agreement. It provides intelligent event diagnosis, and automatic, secure submission of hardware event notifications to Hewlett Packard Enterprise, which initiates a fast and accurate resolution based on the service level of your product. Hewlett Packard Enterprise strongly recommends that you register your device for remote support.

If your product includes additional remote support details, use search to locate that information.

HPE Get Connected

<https://www.hpe.com/services/getconnected>

HPE Pointnext Tech Care

<https://www.hpe.com/services/techcare>

HPE Complete Care

<https://www.hpe.com/services/completecure>

## Warranty information

To view the warranty information for your product, see the links provided below:

HPE ProLiant and IA-32 Servers and Options

<https://www.hpe.com/support/ProLiantServers-Warranties>

HPE Enterprise and Cloudline Servers

<https://www.hpe.com/support/EnterpriseServers-Warranties>

HPE Storage Products

<https://www.hpe.com/support/Storage-Warranties>

HPE Networking Products

<https://www.hpe.com/support/Networking-Warranties>

## Regulatory information

To view the regulatory information for your product, view the Safety and Compliance Information for Server, Storage, Power, Networking, and Rack Products, available at the Hewlett Packard Enterprise Support Center:

<https://www.hpe.com/support/Safety-Compliance-EnterpriseProducts>

### Additional regulatory information

Hewlett Packard Enterprise is committed to providing our customers with information about the chemical substances in our products as needed to comply with legal requirements such as REACH (Regulation EC No 1907/2006 of the European Parliament and the Council). A chemical information report for this product can be found at:

<https://www.hpe.com/info/reach>

For Hewlett Packard Enterprise product environmental and safety information and compliance data, including RoHS and REACH, see:

<https://www.hpe.com/info/ecodata>

For Hewlett Packard Enterprise environmental information, including company programs, product recycling, and energy efficiency, see:

<https://www.hpe.com/info/environment>

## Documentation feedback

Hewlett Packard Enterprise is committed to providing documentation that meets your needs. To help us improve the documentation, use the Feedback button and icons (located at the bottom of an opened document) on the Hewlett Packard Enterprise Support Center portal (<https://www.hpe.com/support/hpesc>) to send any errors, suggestions, or comments. All document information is captured by the process.



# About the Cray ClusterStor data services Administrator Guide

The Cray ClusterStor data services Administrator Guide S-1237 provides procedures to help administrators use, configure, and maintain ClusterStor data services deployments.

Table 1: Record of Revision

Publication Title	Date	Updates
Cray ClusterStor data services Administrator Guide S-1237 (2.0) Revision B	December 2021	Added a procedure for the whole workflow for enabling policy-driven data movement. Explained limitations of re-striping Lustre directories that already contain data. Numerous corrections throughout guide. Simplified and clarified many procedures.
Cray ClusterStor data services Administrator Guide S-1237 (2.0) Revision A	October 2021	Removed SDP update procedure and duplicated example policy. Updated "Add or Remove Users in Keycloak" to use <i>DOMAIN_NAME</i> .
Cray ClusterStor data services Administration Guide S-1237 (2.0)	October 2021	Added content about CSMS CLI, new Emitter configuration process, and new dashboards. The <code>filestatus</code> and <code>pfind</code> tools are deprecated. Added instructions for using <code>kubectrl</code> remotely as well as updating the HSN, LNet, and Lustre configuration after installation.
Cray ClusterStor data services Administration Guide S-1237 (1.1)	May 2021	Added content about policy support for <code>mirror</code> and <code>punch</code> actions, new <code>--json</code> option for <code>filestatus</code> . Default template and flash maintenance policy are no enabled by default. Policies can select files using the mirror state.
Cray ClusterStor data services Administration Guide S-1237 (1.0)	January 2021	Added topics about <code>pfind</code> , <code>testparse</code> , and <code>filestatus</code> tools. Added topics about the Kibana dashboards and Policy Engine request limiting.
Cray ClusterStor data services Administration Guide S-1237 (0.3)	August 2020	Added description of new Policy Engine components and content on policies
Cray ClusterStor Data Services User and Administration Guide S-1237 (0.2)	February 2020	Added Scalable Search, updated configuration procedure, added another method for monitoring file movement status.
Cray ClusterStor Data Services User and Administration Guide S-1237 (0.1)	November 2019	Initial release


## Scope and Audience

This publication is for administrators of ClusterStor data services who are familiar with ClusterStor storage systems and Lustre commands, terminology, and architecture. This guide contains procedures and reference information to support ClusterStor data services systems running release 2.0. The reference material includes a thorough explanation of the aims, design principles, and architecture of ClusterStor data services. This guide includes procedures for configuring, customizing, using, and troubleshooting ClusterStor data services.

Refer to the Cray ClusterStor data services User Guide (S-1239) for procedures and information specifically for non-administrator ClusterStor users.

## Typographic Conventions

Monospace	Indicates program code, reserved words, library functions, command-line prompts, screen output, file names, paths, and other software constructs.
-----------	---

<b>Monospaced Bold</b>	Indicates commands that must be entered on a command line or in response to an interactive prompt.
<i>Oblique or Italics</i>	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a GUI Window, GUI element, cascading menu (Ctrl > Alt > Delete), or key strokes (press Enter).
 (backslash)	At the end of a command line, indicates a shell or command-line continuation character (lines joined by a backslash are parsed as a single line).

## Trademarks

© Copyright 2019–2021 Hewlett Packard Enterprise Development LP.



# Command Example Conventions

## Host names and accounts in command prompts

The host name in a command prompt indicates on what host or type of host to run the command. The prompt also indicates the account that must run the command.

- The `root` or super-user account always has the `#` character at the end of the prompt.
- Any non-`root` account is indicated with a `$`. A user account that is not `root` or `admin` is referred to as "user".

<code>install#</code>	Run the command as <code>root</code> on the system that hosts, serves, and executes the files that install ClusterStor data services on the three management nodes.
<code>ext-adm#</code>	Run the command as <code>root</code> on an external administration system for the ClusterStor data services cluster.
<code>csms\$</code>	Run the command on a host that has (or will soon have) the CSMS CLI installed and initialized.
<code>VM-W#</code>	Run the command on a ClusterStor data services Kubernetes worker node virtual machine as <code>root</code> .
<code>CDS#</code>	Run the command on a ClusterStor data services node as <code>root</code> .
<code>CDS\$</code>	Run the command on a ClusterStor data services node as a user.
<code>CDS1#</code>	Run the command on the primary ClusterStor data services node as <code>root</code> .
<code>CDS1\$</code>	Run the command on the primary ClusterStor data services node as a user.
<code>MGMT0#</code>	Run the command on the primary ClusterStor management node as <code>root</code> .
<code>MGMT0\$</code>	Run the command on the primary ClusterStor management node as <code>admin</code> .
<code>MGMT1#</code>	Run the command on the secondary ClusterStor management node as <code>root</code> .
<code>MGMT1\$</code>	Run the command on the secondary ClusterStor management node as <code>admin</code> .
<code>OSS#</code>	Run the command on an OSS node as <code>root</code> .
<code>OSS\$</code>	Run the command on an OSS node as a user.
<code>MGS#</code>	Run the command on an MGS node as <code>root</code> .
<code>MGS\$</code>	Run the command on an MGS node as a user.
<code>MDS#</code>	Run the command on an MDS node as <code>root</code> .
<code>MDS\$</code>	Run the command on an MDS node as a user.
<code>sw0#</code>	Run the command on the primary management switch with administrative privileges.
<code>sw1#</code>	Run the command on the secondary management switch with administrative privileges.
<code>client\$</code>	Run the command on a Lustre client node as any user.
<code>client#</code>	Run the command on a Lustre client node as <code>root</code> .
<code>user@hostname\$</code>	Run the command on the specified system as any user.

## Lustre file system names

The name of the Lustre file system seen in command examples is `cls12345`, with a mount point of `/lus` on the Lustre clients. The following example demonstrates this convention:

<code>client\$ lfs df -h</code>					
UUID	bytes	Used	Available	Use%	Mounted on
<code>cls12345-MDT0000_UUID</code>	2.0T	59.1G	1.9T	4%	<code>/lus[MDT:0]</code>
<code>cls12345-MDT0001_UUID</code>	2.0T	97.6M	1.9T	1%	<code>/lus[MDT:1]</code>
<code>cls12345-OST0000_UUID</code>	112.0T	5.1T	105.8T	5%	<code>/lus[OST:0]</code>
<code>cls12345-OST0001_UUID</code>	112.0T	5.1T	105.8T	5%	<code>/lus[OST:1]</code>
<code>cls12345-OST0002_UUID</code>	15.3T	425.8G	14.8T	3%	<code>/lus[OST:2]</code>
<code>cls12345-OST0003_UUID</code>	15.3T	423.4G	14.8T	3%	<code>/lus[OST:3]</code>
<code>filesystem_summary:</code>	254.6T	11.0T	241.1T	5%	<code>/lus</code>

## Command Prompt inside a Kubernetes pod

If executing a shell inside a container of a Kubernetes pod where the pod name is *podName*, the prompt changes to indicate that it is inside the pod. Not all shells are available within every pod, the following is an example using a commonly available shell.

```
kubect1 exec -it podName /bin/sh
pod#
```

## Directory Path in Command Prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Usually, the commands can run in any directory. When a command must run within a specific directory, examples use the `cd` command to change into the necessary directory.

For example, here are actual prompts as they appear on the system:

```
client:~ # cd /etc
client:/etc# cd /var/tmp
client:/var/tmp# ls file
```

And here are the same prompts as they appear in this publication:

```
client# cd /etc
client# cd /var/tmp
client# ls file
```



## What ClusterStor data services Provides

ClusterStor E1000 introduces an additional flash storage tier into what has traditionally been a single-tier disk-based Lustre file system. ClusterStor data services provides flexible, fast, automated, and scalable movement of data between these tiers. As a result, users can fully use each tier with minimal effort.

Users of large and tiered Lustre storage systems have four vital needs:

- **Converged treatment of tiered data movement through the Lustre Hierarchical Storage Management (HSM) infrastructure :** The HSM functionality in Lustre cannot change data location or layouts within a single Lustre file system. It can only change them when moving a file to an external archive. Users of tiered storage systems, however, must be able to move data between tiers within the same Lustre file system. ClusterStor data services introduces a new `lfs migrate` option. This new option uses the Lustre HSM infrastructure to parallelize data movement on external dedicated nodes, yet is also able to move file data between different storage tiers.
- **Performant, scalable, efficient data movement:** Lustre users need a data movement solution optimized for extracting the full performance potential from the flash tier. ClusterStor data services parallelizes file data migration across multiple dedicated nodes.
- **Scalable and efficient external data movement request management:** Lustre itself only supports manual requests for data movement using `lfs` commands. It does not include, for example, an internal policy engine, forcing users to configure and integrate a third-party solution (software and hardware). Lustre does not provide a programmatic API for submitting, prioritizing, and managing data movement requests.
- **Fast and scalable search capabilities:** Data movement policy engines and user file searches on large Lustre namespaces both require search performance beyond what `find`, `lfs find`, and other search tools can deliver.

## Lustre Tiering Limitations

Separate storage tiers can be accessed as separate OST pools (for example, separate OST pools for flash and disk drives) within Lustre without ClusterStor data services. ClusterStor users, however face two design issues when trying to use existing Lustre mechanisms for data management:

1. The `lfs hsm_archive` and `lfs hsm_restore` HSM commands initiate external copytools to move data between Lustre and a separate, external archive (such as a tape library). The `lfs mirror` and `lfs migrate` commands, however, do not work this way. Therefore, copytools are not initiated for tier management which requires data movement **within** a single Lustre file system.
2. These HSM commands rely on the MDS HSM Coordinator queue. The MDS HSM Coordinator is an inflexible first-in, first-out (FIFO) queue that does not have prioritization and other capabilities.

These limitations lead to several specific problems when users try to use these commands on tiered Lustre storage systems at scale:

- **Data movement with `lfs mirror` and `lfs migrate` is highly inefficient and does not scale:** Although these commands can move data between OSTs and OST pools, Lustre does not have internal mechanisms for queuing or distributing data transfers. Every data movement request which uses these commands sends all the data serially through the requesting node. This bottlenecks file I/O throughput and can lead to long wait times for large file movements.
- **Requests cannot be properly prioritized:** Since the Lustre HSM Coordinator Queue operates as a FIFO, requests cannot be prioritized or reprioritized. The result is a mismatch between the current data movement needs of the Lustre system and what data is moved at that moment.
- **The Lustre internal HSM coordinator queue consumes MDS resources:** The MDS actively maintains the list, tracks timeouts, reissues requests, cancels completed items, and reports status. All of these operations divert compute resources on the MDS away from its primary function: providing metadata to Lustre clients.
- **No internal policy engine:** Lustre does not have methods for automating requests based on policies. Users must configure and integrate an external policy engine.
- **Movement decisions require significant understanding of Lustre layouts** by the individual users issuing the data movement requests. This is because there are no standard rules or templates to follow.
- **Lustre does not have a data movement API:** Users have no way to request data movements in a programmatic fashion.

## How ClusterStor data services Improves Upon Lustre

ClusterStor data services moves the entire HSM and tiering workflow out of Lustre and into a set of external, horizontally scalable microservices. These microservices are accessible through a REST API and the traditional `lfs` commands. ClusterStor data services expands the current HSM functionality within Lustre to efficiently manage hard disk and flash storage tiers. ClusterStor data services helps customers realize the full performance potential of the flash storage tier within hybrid ClusterStor systems.

### What ClusterStor data services brings to existing ClusterStor and Lustre

**Efficient and scalable data movement:** large single file migrations from one tier to another are distributed over multiple dedicated transfer nodes, turning serial data transfer operations into faster parallel ones. Also, ClusterStor data services moves the data movement request queue from the MDS to a database managed outside of Lustre. As a result, requests can be prioritized and scheduled flexibly and the MDS can now reserve compute resources for fulfilling metadata requests. Also, performance is improved since ClusterStor data services can serve multiple data movement requests concurrently.

**A programmatic way to move file data within Lustre through a REST API :** ClusterStor data services allows requests for moving data within Lustre to originate outside of it. The result is better integration with automated data movement requesters, such as policy engines or job schedulers.

**Integrated, full featured tiered storage management:** ClusterStor data services incorporates data movement commands which have not traditionally been within the Lustre HSM mechanism, namely `lfs migrate`. ClusterStor data services also expands the `lfs hsm` command path to handle internal tiering management. This expansion solves a major problem with attempting to use Lustre to manage flash and hard disk tiers within a single file system. ClusterStor data services integrates the Lustre tiering mechanism with the Lustre internal data movement commands.

**The ability to efficiently change the layouts of Lustre files by externalizing the `lfs migrate` :** ClusterStor data services supports a new HSM version of `lfs migrate` and parallelizes the data movement.

**Automatic space management of a flash tier in a hybrid disk-flash system:** ClusterStor data services provides software tools and example policies for automating flash tier management. This automation lessens the administrative burden for users of hybrid (flash and disk) ClusterStor systems.

**Efficient searches of file system metadata for reporting, analysis, or fileset operations:** ClusterStor data services maintains an internal database of file system metadata. ClusterStor data services provides a database query tool (Query) for administrators to use from any Lustre file system client. The Query tool can efficiently and quickly provide lists of files matching search criteria in CSV or JSON form, as well as summary information.



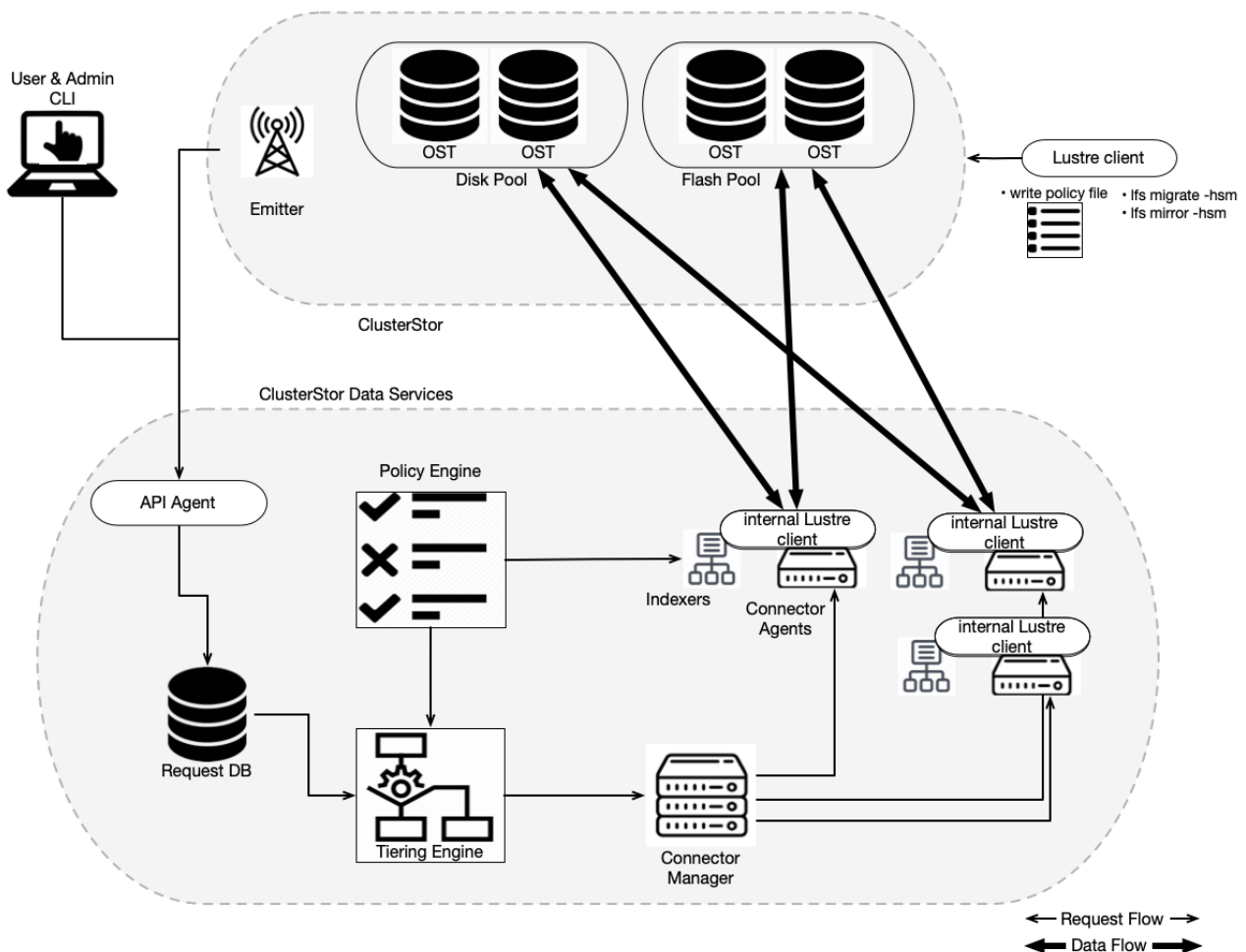
# Scalable Data Movement Infrastructure

ClusterStor data services makes management of data movement between storage tiers within hybrid ClusterStor systems easier. The core of ClusterStor data services consists of a software infrastructure designed, integrated, and configured to provide scalable data movement.

This scalable data mover infrastructure employs a microservices model to achieve the horizontal scaling necessary to meet the needs of users of large file systems. This same infrastructure provides user and administrator tools to control data movement actions.

## How the Scalable Data Mover Infrastructure Works

Figure 1: ClusterStor data services Architecture



The preceding diagram illustrates the connections between the core services of ClusterStor data services. To understand how these services work together, consider the following example data movement request:

1. A user issues a request on a Luster client to move file *myfile* from the disk tier to the flash tier:

```
client$ lfs migrate --hsm --pool flash myfile
```

2. The Luster client forwards the request to the Luster MDS.
3. The Luster MDS sends this request to ClusterStor data services through the HSM Emitter. ClusterStor data services serves as the Luster external HSM coordinator.
4. The Emitter then reformats the request to conform to the ClusterStor data services REST API before relaying it from the MDS to the API Agent. The API Agent runs on a ClusterStor data services node.
5. The API Agent stores that movement request in a database for persistence. The API Agent then immediately replies to the Emitter that the request is stored.
6. The request joins all other queued or in process requests in the database. The Tiering Engine sends a small subset of those requests to the Connector Migration Manager (CMM) as the just-in-time (JIT) list.
7. The CMM then doles out the movement requests to the pool of Connector Migration Agents (CMAs), which perform the actual data



movement. Each CMA is Lustre client internal to ClusterStor data services. In the process, the CMM determines how best to parcel out the work to use all the CMAs for the data movement. The CMM can either stripe files across CMAs or task each CMA with moving one file (that is, move multiple files simultaneously).

8. The CMAs copy the data. Meanwhile, status messages pass back from the CMAs to the CMM and from the CMM to the Tiering Engine (TE). The TE maintains the status of the request in the database.
9. Lustre file status updates through `llapi` operations on the file itself. Lustre does not rely on the status monitoring mechanisms of ClusterStor data services.

## Lustre HSM Emitter

The Lustre HSM Emitter is a ClusterStor data services component present on compatible ClusterStor systems. This component represents a change in how Lustre handles HSM requests. The Emitter bypasses the internal Lustre Coordinator Queue and sends any Lustre-generated requests out of Lustre to the ClusterStor data services API Agent. This service also reformats requests coming from Lustre into the standard form required by the API Agent.

The Emitter facilitates Lustre HSM-to-ClusterStor data services communications. The Emitter does not need any direct interaction with users during its configuration or normal operation.

The introduction of the Emitter changes Lustre into one of many possible requesters for data movements. Previously, Lustre was responsible for managing copytools directly.

## ClusterStor data services API Agent

The API Agent provides the sole external interface to ClusterStor data services, and accepts both data movement and administrative requests from the Emitter service. This REST-based API uses a message format defined in an OpenAPI specification.

The API Agent provides a unique request identifier to the requester and responds after persisting the request. Responses are sent after persisting the request rather than after performing the data movement. Due to the prioritization and just-in-time nature of the Tiering Engine, a request submitted to the API Agent is not guaranteed to be serviced immediately.

# Tiering Engine

## Data Movement Request Prioritization and Scheduling

Long work queues of data movement requests present a number of problems:

- A serial request queue cannot be easily reprioritized.
- Requests that are no longer relevant entails list searching.
- Scaling and storage of a large list may itself be problematic.

Furthermore, since data movement requests typically take substantial time, delays between issuing and servicing a request may be large. Data movement, therefore, is not dynamically responsive to the current state of the system. To avoid this situation, ClusterStor data services includes the Tiering Engine (TE) component.

The TE dynamically generates a minimal just-in-time (JIT) list of items for the data movers to work on immediately. Items are added to this JIT list only when the list length is short; enough items are added to keep the Connector Migration Agent (CMA) instances busy.

The Tiering Engine sends a small set of requests that will be serviced next - the JIT list for the Connector copytool. The TE may allow some requests to expire and return error codes for conflicting requests.

The TE also tracks the state of all requests and updates the request database.

## No Request Completion Guarantees

The TE does not generally provide completion guarantees: if a requested data movement action does not complete for any reason, the TE component will determine if it must retry the request.

Due to the dynamic priorities of a tiered storage system and resource-limited data movement capabilities, the TE may cancel some requests due to other requests. For example, a request to migrate a file from flash to disk is no longer needed after that same file is archived to tape.

Requests generally include a "time to live" indicator. When this limit is exceeded, the request will be marked as expired by the TE and dropped from future consideration. All requests also include a retry count. In case the copytool fails to service a request, the TE will reschedule it in the future a limited number of times. Beyond this count, the request is marked as permanently failed.

# Connector

## ClusterStor data services Connector

The Connector component is an evolution of Cray Connector, which was designed to handle HSM data movement requests from Lustre. In that role, Connector provided a bridge between the Lustre HSM Coordinator and an external tape archive system.

ClusterStor data services uses this HSM data movement functionality to solve a different problem: fast, scalable, and efficient management of flash and hard disk storage tiers within a single Lustre file system. To satisfy this new role, the functionality of Connector now extends to include moving data within a single Lustre file system. Specifically, Connector now moves file data from one set of OSTs to another.

The Connector is able to both coordinate movement of multiple files in parallel, and break the movement of large files down into parallel tasks. The Connector has two parts:

- Connector Migration Manager (CMM): the coordinator of requests. The CMM is a single instance service which ingests data movement requests from the TE over an internal ClusterStor data services message bus.
- A group of Connector Migration Agents (CMAs): the actual data copiers between the two storage tiers. The CMA instances are all replicates—their horizontal scaling greatly increases data movement bandwidth.

ClusterStor data services implements both the CMM and CMAs as highly available, lightweight services.

## Changes to Connector for ClusterStor data services

ClusterStor data services modifies Connector to fit this new role:

- The CMM no longer directly registers with Lustre as a copytool.
- The TE transports data movement requests to the CMM and not directly from the Lustre HSM coordinator.
- Status updates are marked directly on the file itself through metadata as layout changes or HSM status changes. The Lustre HSM Coordinator does not indicate status or completion updates.

# Policy Engine

## Purpose

ClusterStor data services supports automated data movement policies. Administrators can use these policies to:

- Reclaim storage capacity.
- Perform data backup or archiving.
- Optimize tiered storage usage for performance.
- Maintain compliance with company data management policies.

Rules regarding these use cases may be diverse and complex. The ClusterStor data services Policy Engine processes rules to find files that match selection criteria specified in a policy. To locate those files, Policy Engine searches the file system metadata indexed by the Scalable Search Service. The Query Agent submits a list of files to the Tiering Engine which match those criteria. That list also includes the actions to perform on those files. Connector then performs those actions.

Thus, the Policy Engine can quickly apply data movement policies, even on exascale-class file systems. Since ClusterStor administrators do not have to wait days for a large file system scan with ClusterStor data services, they can act immediately.

## Architecture

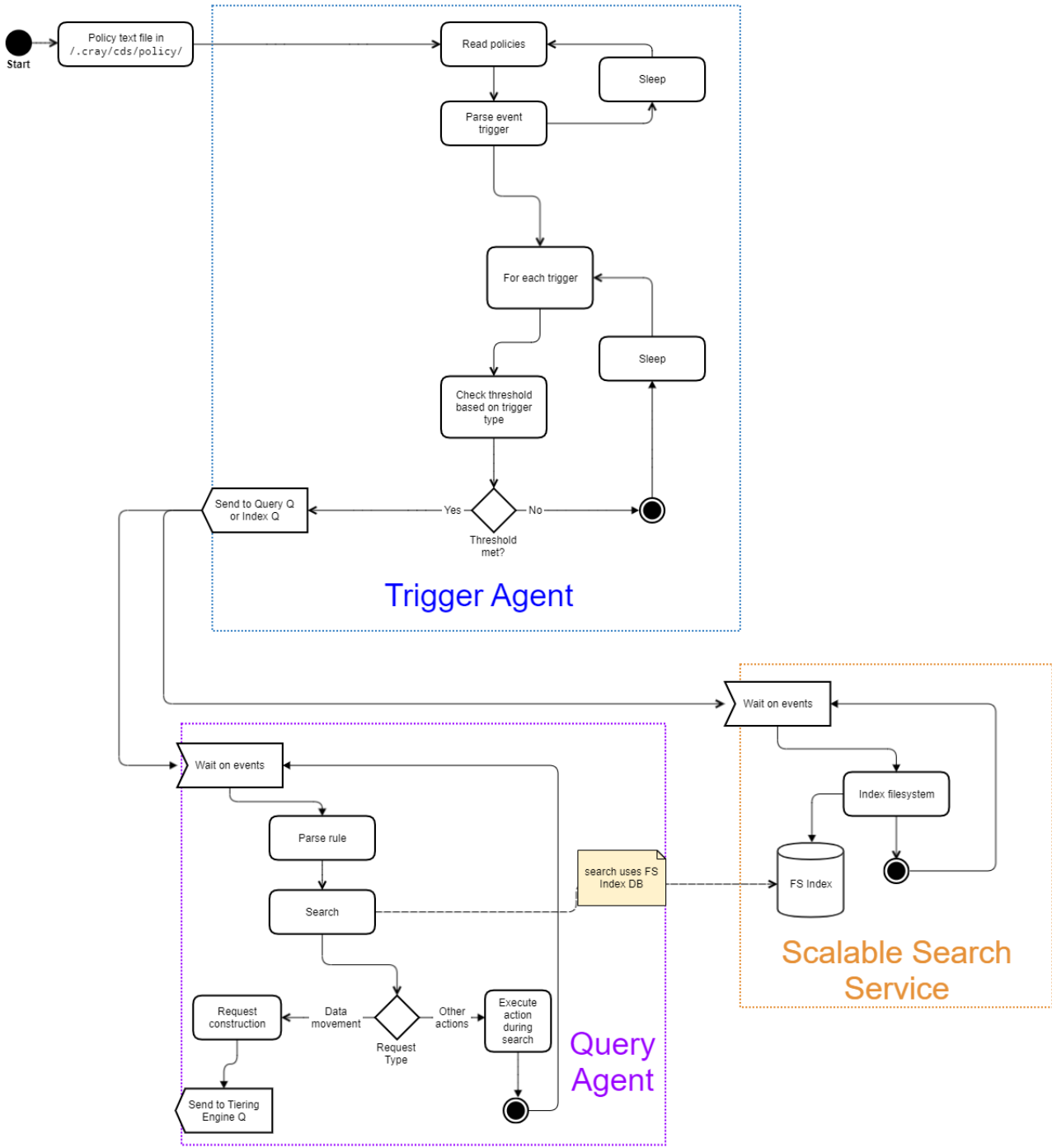
Within ClusterStor data services, the Policy Engine consists of a set of distinct roles that interact through a set of message queues:

- The triggering role, performed by Trigger Agent, includes monitoring of the Lustre file system and ClusterStor data services status, and generates trigger events. See [Trigger Agent](#) for more details.
- The file selection role is performed by Query Agent. Query Agent identifies the set of files in the file system that matches the selection criteria, and forms the results into a set of action requests. Refer to [Query Agent](#) for more information.
- The Scalable Search Service creates and maintains the distributed file system metadata database. For more information about this component, see [Scalable Search Service](#).

Each component is a separate service and is deployed in distinct pods under Kubernetes. At least one pod is always running for each service in a high availability (HA) manner. The following diagram shows how the three components of the Policy Engine work together to apply policy files.

**Figure 2: ClusterStor data services Policy Engine block diagram**

# CDS Policy Engine



## Trigger Agent

The Trigger Agent performs the following functions:

- Reads and parses the policy definitions.
- Checks the trigger conditions
- Produces trigger events on the Query Queue
- Produces index events on the Index Queue.

Approximately every five minutes, the Trigger Agent checks the policy directory ( *lustre\_mount\_point/cray/cds/policy/*) for new or modified files. If the Trigger Agent finds any, it reads and parses them, and updates its internal state accordingly. If the Trigger Agent notes a syntax error or other fatal problem with the policy file, it will produce a log message. If a named policy is no longer present, the Trigger Agent will not trigger for it.

The Trigger Agent must parse enough of the policy to determine the trigger conditions, and insure they are valid. The Trigger Agent is not responsible for verifying the correct syntax of the rule, just of the trigger condition. The Query Agent later verifies the rule syntax. The Trigger Agent may also check the rule syntax and reject the policy on rule errors during initial parsing.

The Trigger Agent maintains a list of active triggers in memory. This list is periodically updated by checking the contents of the policy directory. For each trigger, the Trigger Agent monitors the Lustre file system for the trigger condition. Trigger Agent automatically uses the appropriate monitoring method for the trigger condition.

When the file system meets the trigger condition, Trigger Agent generates a message indicating which rule was triggered and places it on the Query Queue. Policy Engine uses a separate Index Queue for any indexing action requested by a policy file.



# Query Agent

## What the Query Agent does

The Query Agent performs the following functions:

- Receives trigger messages from the Query Queue
- Parses the file selection criteria from the rule
- Maps the selection criteria to Scalable Search Service queries to generate lists of candidate files.
- Submits action requests for each candidate file to the Tiering Engine through an internal queue
- Short-duration actions (such as `delete`) directly on candidate files.

## How the Query Agent processes policies

The Query Agent waits for events arriving on the Query Queue. After an event arrives, the Query Agent identifies the policy file and rule specified in the event, rereads the policy file, and reparses the rule. The Query Agent obtains the selection criteria from the rule, generates a query suitable for the Scalable Search Service, and submits the query to that service. If the Query Agent notes a syntax error or other fatal problem with the policy file, it will produce a log message.

As results return, the Query Agent generates a data movement request for each matching file. Query Agent then submits that request to Tiering Engine to schedule the actual data movement. For short-duration actions (anything that does not require data copy), however, the Query Agent performs the actions itself using a multithreaded tool.

## Query Agent uses Scalable Search Service

The Query Agent uses the Scalable Search Service database to quickly produce lists of candidate files. Any metadata saved in the database can be used as policy selection criteria, such as file size, owner, modification time, or OST pool (for example, `flash` or `disk`).

For file deletion actions, Query Agent uses the purge functionality built into Query (the `-delete` option specifically). Query Agent deletes files using the permissions of the policy file owner. For more information on purge functionality of Query, see [How Files are Purged](#).

# Scalable Search Service

## Purpose

The Scalable Search Service component of ClusterStor data services enables three key capabilities:

- **Fast searches for files based on customizable criteria:** The Scalable Search Service creates and maintains a distributed index of file system metadata within the parallel file system itself. Administrators can create custom searches to find files of interest by using SQL syntax. The searches then execute in parallel across the distributed index shards.
- **Data movement policies:** Automated data movement requires policies that specify which files to move and when to move them. Those criteria within a policy are translated into search parameters so that Query Agent can provide a list of candidate files for movement.
- **Filesystem summaries:** Administrators must balance customer-specific user, group, or project priorities with the resources available from the file system. Administrators must also estimate capacity fill rates to plan for file system growth. Thus, administrators need the ability to provide information about the file system summarized in various ways. These include histograms of the file size distributions, total capacity consumed on flash by a particular group, and so on.

The Scalable Search Service is a parallel file system indexing service which builds and updates a distributed database. Other ClusterStor data services components and tools use that database to deliver fast search results.

## The Distributed Filesystem Metadata Database

The Scalable Search Service stores file system metadata in a small number of database shards. These shards are stored within the indexed ClusterStor file system itself. Each shard is written by a single dedicated thread, avoiding all locking contention present in monolithic database search solutions. A small number of large-size database shards are stored on the fast parallel file system. Therefore, access to the entire database is fast from any ClusterStor client.

Each index is composed of a set of shards. The shards reside in a hidden directory located at regular locations within the file system. Shards are independent SQLite databases stored as regular Lustre files. The size of each shard is dependent on the number of files in the index. For example, 500 million files in the index would produce shards ~150MB in size. Shards contain the file system metadata for a certain set of directories. Directories are mapped to shards by a simple hashing function. All files within these directories are represented in the mapped shard. The metadata stored for each file includes the information returned by `stat()`. This metadata includes the file name, owner, permissions, modification time, and size. Some Lustre-specific metadata about the location of the file data are included as well. Such metadata includes the pool name, OST index list, and FLR mirror status. Each record also includes the scan time stamp—the time the indexing process started.

## Indexing policies

The Scalable Search Service creates or updates this database when triggered to do so by the indexing policies enabled on that mounted Lustre file system. One such policy is included and enabled by default in ClusterStor data services:

`lustre_mount_point/.cray/cds/policy/indexing.plc`. This default indexing policy begins creating the initial index within three hours after ClusterStor data services installation. An incremental reindex then occurs every three hours. Administrators can modify or replace this policy if wanted.

## Indexing and File System `atime`

The Scalable Search Service only runs on ClusterStor data services nodes, and those nodes mount the Lustre file system with the `noatime` option. Therefore, the Scalable Search Service does not update the `atime` time stamp on files when they are scanned during initial or incremental indexing. This allows administrators to use both the `last_access` condition and the `atime` option in policies and Query commands, respectively.

## Metadata Database Resiliency

The underlying Lustre file system provides high availability and durability of the shards. In addition, the Scalable Search Service is resilient to the loss of its database shards. If one or more are deleted, the service will automatically recreate the lost database shards during its next run.

## Scalable Search Service Index Directories

The Scalable Search Service creates and maintains a distributed file system metadata database within the Lustre file system. This database consists of directories named `._dbindex_` placed at known locations within the file system tree. These directories contain the sqlite database files and other files used by the Policy Engine and several ClusterStor data services tools.

Do not delete the `._dbindex_` directories, nor any of the files within them. Policy Engine and several Scalable Search tools may produce errors or incorrect results if they are missing.

The Scalable Search Service will, however, recreate any missing files during the next file system re-indexing.



## Configure OST Pools

ClusterStor data services policies can automatically manage the flash and disk tiers within a Lustre file system. To manage data placement between these tiers, ClusterStor data services requires that these tiers are in separate OST pools. Therefore, the ClusterStor administrator must create and configure separate flash and disk OST pools before ClusterStor data services can manage them.

There are two methods the administrator can use to create these OST pools if they do not exist. Both methods use `cscli` commands issued from the active management node of the ClusterStor system:

1. Single-step, automatic pool configuration through the `cscli lustre pool auto_configure` command. This method is the more convenient one. See [Automatically Configure OST Pools Using CSCLI](#) for details.
2. Multiple-step, manual, custom pool configuration using individual `cscli` commands. Refer to [Manually Configure OST Pools Using CSCLI](#).

See the ClusterStor™ CSCLI Command Reference Guide S-9922 for more information on `cscli` commands.

## Automatically Configure OST Pools Using CSCLI

The `cscli lustre pool auto_configure --force` command removes any existing OST pools before creating the two pools that ClusterStor data services can use. Do not perform this procedure if the wanted pools exist.

Hewlett Packard Enterprise recommends using the `cscli lustre pool auto_configure` command to create and configure the disk and flash OST pools when customer sites require only one pool of each type. If the site will use more than one OST pool per storage tier, then perform [Manually Configure OST Pools Using CSCLI](#) instead.

### Prerequisites

- Verify that the ClusterStor system contains both flash and disk OSTs and that the ClusterStor version meets the compatibility requirements for ClusterStor data services.
- Log into the active management node of the ClusterStor system as `admin`.
- Confirm that there are no existing OST pools currently in use.

### Procedure

1. List any existing OST pools.

```
MGMT0$ cscli lustre pool show
Pools from cls12345:
    disk
    flash
```

2. Display all the OST members of a pool. Replace `POOL_NAME` with `flash` or `disk` as wanted. Skip the rest of this procedure if the wanted OST pools exist, as in the following example.

This command is useful for reporting which OSTs are members of which OST pool before modifying the pools.

```
MGMT0$ cscli lustre pool show -p POOL_NAME
```

3. Create the `flash` and `disk` OST pools on the ClusterStor system. Run either of the following commands on the active ClusterStor Management node:

- Run

```
MGMT0$ cscli lustre pool auto_configure
```

if there are currently no existing OST pools configured on the ClusterStor system.

- Run

```
MGMT0$ cscli lustre pool auto_configure --force
```

to both remove any existing pools and create new ones.

4. Confirm that the `disk` and `flash` OST pools have been created by displaying the list of pools in the system.

```
MGMT0$ cscli lustre pool show
Pools from cls12345:
    disk
    flash
```

5. Verify the OST pool configuration. Replace `POOL_NAME` with `flash` or `disk` as wanted.

```
MGMT0$ cscli lustre pool show -p POOL_NAME
```

# Manually Configure OST Pools Using CSCLI

The `cscli lustre pool auto_configure` command is the preferred method for creating and configuring the disk and flash OST pools required by ClusterStor data services. ClusterStor administrators, however, can also create these pools manually through multiple `cscli` commands if a site requires that approach. This manual method can be useful for setting up a custom ClusterStor data services pool configuration, such as multiple OST pools for flash or disk media.

Perform any or all the following steps as necessary to manually configure the ClusterStor data services OST pools. This procedure creates one pool named `flash` containing only flash OSTs, and one pool named `disk` that contains the HDD OSTs. For more information on each command, refer to the ClusterStor CSCLI Command Reference Guide S-9922.

ClusterStor administrators can create additional OST pools if wanted.

Run all the commands in this procedure as `admin` on the active ClusterStor management node.

## Prerequisites

- Verify that the ClusterStor system contains both flash and disk OSTs and that the ClusterStor version meets the compatibility requirements for ClusterStor data services.
- Log into the active management node of the ClusterStor system as `admin`.

## Procedure

1. List any existing OST pools. Skip the rest of this procedure if the wanted OST pools exist, as in the following example.

```
MGMT0$ cscli lustre pool show
Pools from cls12345:
    disk
    flash
```

2. Display all the OST members of a pool. Replace `POOL_NAME` with `flash` or `disk` as wanted.

This command is useful for reporting which OSTs are members of which OST pool before modifying the pools.

```
MGMT0$ cscli lustre pool show -p POOL_NAME
```

3. Remove OSTs from an existing pool. Replace `OST_LIST` in the following example command with a space-separated list of OSTs, and the appropriate OST pool name for `POOL_NAME`.

```
MGMT0$ cscli lustre pool remove -p POOL_NAME -t OST_LIST
```

4. Delete an empty OST pool.

Only empty pools can be deleted.

```
MGMT0$ cscli lustre pool delete -p POOL_NAME
```

5. Create a new (empty) OST pool. Substitute an appropriate name such as `flash` or `disk` for `POOL_NAME`.

```
MGMT0$ cscli lustre pool create -p POOL_NAME
```

6. Add OSTs to the newly created pool.

```
MGMT0$ cscli lustre pool add -p POOL_NAME -t OST_LIST
```

7. Confirm the OST pool configuration by displaying the list of pools in the system.

```
MGMT0$ cscli lustre pool show
Pools from cls12345:
    disk
    flash
```

8. Verify the OST pool configuration. Replace `POOL_NAME` with `flash` or `disk` as wanted.

```
MGMT0$ cscli lustre pool show -p POOL_NAME
```

## Default Data Placement Policy

By default, Lustre places new file data across all OSTs in the system, regardless of the tier each OST belongs to. Such file placement defeats the purpose of tiering, which is to place data in the appropriate location given its size and usage pattern.

For optimal results, ClusterStor administrators must set the default striping of the file system to use the flash pool for data requiring high performance flash media. HPE service personnel can help customers plan the initial data placement, as the most optimal data placement decisions depend on customer use cases. HPE provides some example layouts for ClusterStor data services in [Set a Default Data Placement Policy](#).

After data is striped on the flash OST pool, ClusterStor data services can then run one or more flash maintenance policies (see [Flash Maintenance](#)).



# Set a Default Data Placement Policy

## Lustre Striping and ClusterStor data services

Lustre segments files into objects then stripes those objects across OSTs for storage. Files and directories stored in Lustre inherit the striping parameters of their parent directory. If the parent does not have any striping parameters set, Lustre applies the global file system default. The `lfs setstripe` command can set these default stripe parameters on the file system root.

By default, Lustre will stripe files across all OSTs, both flash and disk - a suboptimal file placement pattern in a hybrid ClusterStor system. ClusterStor data services does not provide or set a global file system striping default for ClusterStor. The most optimal default placement parameters are highly dependent on the expected use cases and ClusterStor configuration. Hewlett Packard Enterprise recommends that two-tier ClusterStor systems place as much file data as possible in the flash tier by default. The flash maintenance policy will automatically move files out of flash as it gets full. Therefore, ClusterStor administrators can potentially (depending on the size of the flash tier) set an all-flash default placement setting.

Even with active flash maintenance policies, ClusterStor data services might not be able to maintain space in the flash tier in all cases. Flash maintenance depends on the flash storage consumption rate and the bandwidth available to the ClusterStor data services nodes.

## Applying a Default Layout to Non-Empty Lustre File Systems

ClusterStor administrators can set striping pattern (layout) on the root of an empty Lustre file system with `lfs setstripe`. This action causes all new directories and files that do not have a specific layout set by `lfs setstripe` to inherit the file system root layout by default.

Administrators can also apply a layout to the root of a mounted Lustre file system that already contains files and directories. In that case, Lustre determines the layout of existing directories and new files based on:

- Whether the parent directory (or its parent) has layout set by `lfs setstripe`.
- Whether a user created the file or directory with a specific layout with `lfs setstripe`.

Remember the following when using a default layout on the root of Lustre file systems that ClusterStor data services manages:

- All files created in Lustre have a fixed layout. Lustre applies the parent directory layout to a new file during the first write to that file. If a Lustre user created the file with `lfs setstripe` however, Lustre will use that explicitly set layout. Changing the file system default layout or the parent directory layout does not automatically change the file layout. The Lustre client must use its own resources to migrate the file data to the new layout.



**NOTE:** Moving this resource burden from Lustre client nodes to dedicated nodes is the primary use case for ClusterStor data services.

- Any directories created in Lustre by `mkdir` or similar methods, but not by `lfs setstripe`, inherit their layout from either:
  - The parent directory. In this case, the new directory acts as if `lfs setstripe` was executed on it in the same form as the parent.
  - The file system default if parent does not have an explicit layout set.
- Lustre users can assign new layouts to directories (including the file system root) at any time by using `lfs setstripe`. Changing the layout of a directory will affect future files (or subdirectories) created in the modified directory. This command does not affect any existing files (or subdirectories).
- Changing the default layout on a directory propagates to any subdirectory that does not have an explicit striping pattern set.
- The `lfs setstripe -d` command removes any explicitly-set directory striping. Any future files created in this directory will then inherit the file system default layout.

Consult [this section of the Lustre documentation](#) for more information about striping inheritance rules.

## Choose and Apply Default Layout Parameters

ClusterStor administrators can choose to place some, most, or even all file data in the flash tier by default. This section lists a few example default layouts to aid ClusterStor data services administrators in selecting the most optimal layout parameters for their site.

- This layout specifies an object size limit of 1MB on the flash OST pool. When that object exceeds 1MB, it will write the remaining data set onto the disk OST pool, with a stripe count of one:

```
client$ lfs setstripe -E 1M -c 1 -S 1M -p flash -E -1 -c 1 -S 4M -p disk /lus
```

This layout tends to concentrate small files in the flash tier and large files in the disk tier.

- Administrators can specify an object size of 10MB and a stripe count of two on the flash OST pool. This striping more aggressively uses the flash tier. When the object exceeds 10MB in size, Lustre writes the remaining data to the disk OST pool with a stripe count of 4:

```
client$ lfs setstripe -E 10M -c 2 -S 1M -p flash -E -1 -c 4 -S 4M -p disk /lus
```

- The following layout writes all data to the flash pool, with a stripe count of 2. ClusterStor data services policies will then migrate data off the flash tier:

```
client$ lfs setstripe -c 2 -p flash /lus
```

- The following layout will use a stripe count of 4 to place the first petabyte of a file in flash. If the flash pool runs low on space before reaching that limit, data will subsequently "spill over" onto the disk pool. Replace **1000T** in the following command with the maximum amount of data from a single file wanted in flash:

```
client$ lfs setstripe -E 1000T -c 4 -p flash -E 1 -c 8 -p disk /lus
```

## Create a Template

Both files and directories can serve as templates, but this procedure will only cover creating a template directory. Hewlett Packard Enterprise recommends using directories as templates since it is often easier to change the layout of a directory after creation.

### Prerequisites

- Read [ClusterStor data services Templates](#).
- Log into a Lustre client.

### Procedure

1. Create an empty template directory by replacing `my_template_dir` in the following command with the wanted name.

```
client# mkdir lustre_mount_point/my_template_dir
```

Templates can be located anywhere within the Lustre file system. Any policies which reference a template, however, must provide a current, correct path for that template.

Do not store any file data in template directories. These directories merely serve as layout templates for Lustre.

2. Apply a Lustre layout to the template directory through the `lfs setstripe` command.

The following example sets the OST pool of the template to `disk`. This simple template layout could be used in a policy that demotes file data from flash to disk.

```
client$ lfs setstripe --pool disk lustre_mount_point/my_template_dir
```

Consult the Lustre documentation at <https://www.lustre.org/documentation> for more guidance on Lustre layouts.

3. Use the template by creating a policy that references it.

Refer to [ClusterStor data services Policies](#) for data movement policy syntax and policy examples.

## Install Scalable Search Tools on a Lustre Client

The `LUSTRE_MOUNT_POINT/.cray/cds/tools` directory contains a CentOS RPM for the `query` and `summary` search tools. Policy Engine copies this file in on startup. Install this RPM on a Lustre client node before running `query` and `summary` from that node.

In addition, increase the limit on the number of open files to 100,000 on the client. Otherwise, `query` commands will not return search results and will instead produce error messages containing `unable to open database file: too many open files`.

Perform this procedure on every Lustre client that will use these Scalable Search tools.

### Procedure

#### Install the Scalable Search tools RPM

1. Log into a CentOS Lustre client as `root`.
2. Install the RPM containing the Scalable Search tools. Replace `/lus` in the following command with the Lustre mount point.

```
client# rpm -ivh /lus/.cray/cds/tools/cds-brindexer-tools.rpm
```

After running this command, the `query` and `summary` Scalable Search tools will appear in the `/opt/cray/brindexer/bin` directory on the client node.

#### Increase the maximum open files limit

3. Obtain the current maximum number of open files allowed on the node by the kernel.

```
client# ulimit -a | grep -i open
```

4. Skip the remaining steps in this procedure if the output of the previous command is greater than or equal to 100,000. Otherwise, continue with the next step.
5. Open `/etc/sysctl.d/99-sysctl.conf` in an editor.
6. Change the value of `fs.file-max` to `100000`.
7. Save the change and exit the editor.
8. Confirm that the new, higher limit is in effect.

```
client# ulimit -a | grep -i open
```

This command now returns `100000`.

9. Wait until the Policy Engine has completed at least one indexing of Lustre file system.

The `query` or `summary` tools use the file system metadata index.

## Configure an External Administration System

HPE does not support logging into ClusterStor data services nodes after installation. This procedure configures an external system to run `kubectl` commands as necessary for administration tasks after installation is complete.

See <https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/> for more information about kubeconfig files.

### Prerequisites

#### Procedure

1. Install `kubectl` on an external system allocated for ClusterStor data services administration.

This system must connect to the local site network and be able to route to the ClusterStor data services cluster.

2. Copy the `/root/.kube/config` file from a ClusterStor data services node to the external management system.

```
ext-adm# export CDSHOST=CDS_NODE_FQDN
scp root@$CDShost:/root/.kube/config $CDShost.config
```

`CDS_NODE_FQDN` is the external Fully Qualified Domain Name (FQDN) of a ClusterStor data services node.

3. Modify the configuration file.

```
ext-adm# sed -i -E 's/10\(\.[0-9]\{1,3\}\)\{3\}:8443/'$CDShost':6443/' $CDShost.config
ext-adm# sed -i -E 's/certificate-authority-data.*/insecure-skip-tls-verify: true/' $CDShost.config
```

These commands replace the endpoint IP address with `$CDShost` and disable TLS host verification.

4. Copy this modified configuration file to the default location expected by `kubectl`.

```
ext-adm# mv $CDShost.config ~/.kube/config
```

5. Verify that the external system can run `kubectl` commands for the ClusterStor data services system.

```
ext-adm# kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
k8s-master-cdsxmp101-m3d1          Ready    master   18h    v1.18.6
...
```

6. Install the CSMS CLI on this external administration system.

See [Install CSMS CLI](#) for instructions.

## Install CSMS CLI

This procedure downloads and installs the CSMS CLI on an external Linux system. After completing this procedure, ClusterStor administrators can use CSMS CLI commands on that system to administer ClusterStor data services and move file data.

The CSMS CLI is a command-line interface for managing and monitoring ClusterStor data services clusters. This CLI can also manage data movement requests. ClusterStor data services administrators can use the CSMS CLI to perform software upgrades and other administrative operations.

### Prerequisites

- Perform [Add or Remove Users in Keycloak](#) to create the administrator-level account in Keycloak for this procedure.
- Obtain an external Linux system that either already has, or can have, Python 3 installed.

### Procedure

1. Log into a Linux system that is not one of the ClusterStor data services nodes.

Run all the commands in this procedure from this external system.

2. Save the Keycloak access token required to download the CSMS `tar` file.

*ADMIN* is the user name of the administrator-level account and *ADMIN\_PASSWORD* is the password for this account. *DOMAIN\_NAME* is the external-facing domain name for the ClusterStor data services cluster set during installation by the `MERCURY_SYSTEM_EXTERNAL_DOMAIN` variable.

```
ext-adm$ export ACCESS_TOKEN=$(curl -k -s -d grant_type=password -d client_id=shasta \
-d username=ADMIN -d password=ADMIN_PASSWORD \
https://api.DOMAIN_NAME/keycloak/realms/shasta/protocol/openid-connect/token | awk -F'"' '{print $4}'); \
echo $ACCESS_TOKEN
```

3. Download the most recent CSMS CLI tarball.

The following command is the preferred one and downloads the `csms-latest.tar` file from one of the ClusterStor data services nodes. The `wget` command uses the API gateway, but the load balancing performed by BGP selects a specific node to provide the file.

The following example removes most of the output for brevity.

```
ext-adm$ wget --no-check-certificate -d --header="Authorization: Bearer $ACCESS_TOKEN" \
https://api.DOMAIN_NAME/apis/csms/base/v1/cli/download --content-disposition
...
Saving to: 'csms-latest.tar'100%[=====>] 140,713  --.-K/s  in 0.003s2021-11-18 16:14:46 (45.5 MB/s
- 'csms-latest.tar' saved [140713/140713]
csms-latest.tar
```

4. Extract the downloaded CSMS CLI tarball.

```
ext-adm$ tar -xvf csms-latest.tar
x csms-latest/
x csms-latest/csms-guide.pdf
x csms-latest/csms-latest-py3-none-any.whl
x csms-latest/csms.man
x csms-latest/INSTALLATIONGUIDE.md
```

5. Install the CSMS CLI from the Python `.whl` file.

The following commands install the `csms-latest.tar` file and assume that `pip` is a soft link to `pip3`.

```
ext-adm$ python3 -m venv ~/myvenv
ext-adm$ source ~/myvenv/bin/activate
ext-adm$ pip install --upgrade pip
ext-adm$ pip install --upgrade setuptools
ext-adm$ pip install csms-latest/csms-latest-py3-none-any.whl
```

6. Initialize CSMS CLI to verify that the installation completed successfully.

The `Cray Hostname` value must start with `api`.

```
ext-adm$ csms init
Cray Hostname: api.DOMAIN_NAME
Username: ADMIN_USERNAME
Password: ADMIN_PASSWORD

Success!
Initialization complete.
```

## Restrict Access to Policy Directory

The ClusterStor data services Policy Engine operates on policies stored in the `.cray/cds/policy` directory in the Lustre file system. Hewlett Packard Enterprise recommends that ClusterStor administrators use Linux file system access controls to protect this directory as well as the templates used by policies.

### Procedure

1. Set user and group permissions on the `.cray/cds/policy` directory and its contents.

Setting these permissions controls which users can create, view, modify, and delete policies.

The following `chgrp` and `chmod` commands are examples. Administrators must set the ownership and permissions to satisfy local site requirements.

```
client# chgrp policyadmins /lus/.cray/cds/policy
client# chmod 770 /lus/.cray/cds/policy
```

In release 2.0, ClusterStor data services processes might use elevated permissions to carry out the policy actions. In future releases, ClusterStor data services will perform data movements under the UID of the policy file owner. Thus ClusterStor data services will restrict the ability of policy file writers to modify only files to which they have access.

2. Repeat the previous step on all directories that contain templates used by policies in the `.cray/cds/policy` directory.

# Add or Remove Users in Keycloak

This procedure details how to use Keycloak to:

- Add one or more administrator accounts during or after installation of ClusterStor data services.
- Remove an account after ClusterStor data services installation.

Do not use this procedure to configure the account required by the Emitter service. Refer to ClusterStor data services Installation Guide S-1238 for those instructions.

Viewing ClusterStor data services dashboards requires an administrator-level account. Each system administrator must have their own administrator-level account.

## Procedure

1. **Optional:** Log into the base operating system (OpenSUSE) of a ClusterStor data services node if necessary.

This step is not necessary if the ClusterStor data services administrator already knows the Keycloak `admin` password.

2. **Optional:** Obtain the `admin` password for the Keycloak web interface.

```
ext-adm# kubectl get secret -n services keycloak-master-admin-auth \
--template={{.data.password}} | base64 --decode; echo
```

3. Open the Keycloak Administration Console in a web browser.

The web address for this console is `https://auth. SYSTEM_DOMAIN/keycloak/admin/master/console`. *SYSTEM\_DOMAIN* is the ClusterStor data services cluster domain name set by the `MERCURY_SYSTEM_EXTERNAL_DOMAIN` variable during installation.

4. Log into Keycloak using the `admin` user name and the password.
5. Click Manage > Users in the pane on the left side of the screen.
6. **Optional:** Remove a user. Refer to [https://www.keycloak.org/docs/latest/server\\_admin/index.html#delete-user](https://www.keycloak.org/docs/latest/server_admin/index.html#delete-user) for instructions.
7. Add another user account.
  - a. Click Add user in the main pane of the screen.
  - b. Click Save.
  - c. Click the Details tab.
  - d. Enter a user name for the account into the Username box.
  - e. Set User Enabled button to ON.
  - f. Click the Save button.
  - g. Click the Credentials tab.
  - h. Enter a password for the account into the Password and Password Confirmation boxes.
  - i. Set the Temporary button to OFF.
  - j. Click the Set Password button.
  - k. Click the Role Mappings tab.
  - l. Start typing `shasta` into the Client Roles box.
  - m. Select `shasta` from the Client Roles drop-down list after it appears.
  - n. Add the `admin` role to the Assigned Roles box.
8. **Optional:** Verify that Administration Console > Manage Users reflects the new or deleted user account.



## Configure Automatic Data Movement

ClusterStor data services does not provide any default data movement policies. Specific data movement needs vary between customer sites. To create and activate data movement policies, follow this procedure.

### Procedure

1. [Restrict Access to Policy Directory](#)
2. [Create the templates](#) required by the site-specific policies.
3. Create site-specific policies with guidance from [ClusterStor data services Policies](#) and [About ClusterStor data services Policies](#).

## Update High Speed Network, LNet, and Lustre Client Configuration

This procedure updates the High Speed Network (HSN), LNet, and Lustre client configuration on the ClusterStor data services nodes. Perform this procedure when either of these three aspects of the ClusterStor networking change after installation.

### Procedure

1. Download a copy of the `/opt/cray/passthrough/lustre-client-config/node_config/MANAGEMENT_NODE_NAME.extra_sls_data.json` file from the first ClusterStor data services node.

`MGMT_NODE_NAME` is the host name of the first ClusterStor data services node.

```
$scp \
root@MGMT_NODE_NAME:/opt/cray/passthrough/lustre-client-config/node_config/MGMT_NODE_NAME.extra_sls_data.json .
```

This is an example of the contents of the `extra_sls_data.json` file.

```
{
  "ExtraProperties": {
    "Networking": {
      "HSN": [
        {
          "FABRIC": "SS10",
          "DEVICE": "hsn0",
          "NAME": "hsn0",
          "IPADDR": "10.230.77.5",
          "PREFIXLEN": "16",
          "MTU": "9000"
        }
      ],
      "LNET": [
        {
          "DEVICE": "hsn0",
          "NAME": "hsn0",
          "LNET_ID": "o2ib0"
        }
      ]
    },
    "Filesystems": [
      {
        "FSTYPE": "lustre",
        "FSNAME": "lustre_filesystem",
        "MGS": "10.230.77.4@o2ib:10.230.77.6@o2ib",
        "MOUNTPPOINT": "/lus",
        "MOUNTOPTS": "_netdev,noatime,lazystatfs,flock,noexec,nodev,suid",
        "STATE": "present"
      }
    ]
  }
}
```

This is a summary of the main sections of this file:

- The `"HSN"` subsection of the `"NETWORKING"` section describes the interface connected to the High Speed Network (HSN).
  - The `"LNET"` subsection of the `"NETWORKING"` section describes the Lustre LNet configuration and maps the HSN hardware defined in the `"HSN"` subsection to the LNet network.
  - The `"Filesystems"` section specifies the Lustre file systems that ClusterStor data services manages.
2. Open the file in an editor.
  3. Update the configuration values as required.

Modify these values as necessary to match to site environment:

- `IPADDR`
- `PREFIXLEN`
- `LNET_ID`
- `FSNAME`
- `MGS`

The `IPADDR` value must be a unique IP address on the same subnet as the Lustre file system.

4. Save changes to the file.
5. Replace the existing file with the updated version.

```
$scp MANAGMENT_NODE_NAME.extra_sls_data.json \  
root@MANAGMENT_NODE_NAME:/opt/cray/passthrough/lustre-client-config/node_config
```

The node will automatically update the configuration. It may take up to 5 minutes for the node to apply the updated configuration.

6. Repeat the previous three steps for the other ClusterStor data services nodes. Verify that each node has a unique IP address value in the `IPADDR` field of the file for that node.



## Manual Data Migration

ClusterStor data services can provide automatic, policy-driven migration of file data. Sometimes, however, file data must be manually migrated between OSTs for storage management or tiering purposes. To cover this use case, ClusterStor data services offers manual data migration.

Users can initiate a manual request to migrate file data by running an `lfs migrate --hsm` command on a Lustre client. The new `--hsm` flag directs Lustre to use ClusterStor data services (or any external coordinator) to move the data.

By design, ClusterStor data services operates transparently to Lustre and is invisible to ClusterStor users. The new HSM-based migration command operates asynchronously, like all other Lustre HSM commands.

ClusterStor data services fulfills manual data migration requests by following the process listed in [Scalable Data Movement Infrastructure](#). See the following topics for commands to query the status of a manual migration request:

- [Query Data Movement Request Status with CSMS CLI](#)
- [Check File Movement Status with `lfs getstripe`](#)
- [Check File Movement Status Using File `ctime`](#)

## Request a File Migration from a Lustre Client

Lustre now provides an `--hsm` option for the `lfs migrate` command. This option instructs Lustre to forward the data movement task to external software instead of performing it on the Lustre client. ClusterStor data services uses this option to ingest data movement requests from Lustre clients.

In general, any `lfs migrate` option will work with ClusterStor data services.

### Prerequisites

- Log into a Lustre client node

### Procedure

Submit a request to ClusterStor data services from a Lustre client to migrate a file residing in one tier to another tier.

In the following command example, the path *path/to/file* specifies a file currently on the flash pool, and `disk` is the destination pool.

```
client$ lfs migrate --hsm --pool disk /lus/path/to/file
```

Lustre will forward the data movement request to ClusterStor data services, which will store and perform the data movement request.

# Request File Data Movement with CSMS CLI

Use the CSMS CLI to request ClusterStor data services file data movement operations from a host outside of the ClusterStor system. This release supports only `migrate` and `mirror` actions.

## Prerequisites

Perform "Install the CSMS CLI" from the publication [Cray ClusterStor data services Installation Guide \(S-1238\)](#) on an external system.

## Procedure

1. Activate the Python virtual environment on the external system that contains CSMS CLI.

The following command assumes that `~/myvenv` is the CSMS CLI Python virtual environment.

```
ext-adm$ source ~/myvenv/bin/activate
```

2. Obtain the Lustre file system name.

In the following example, `cls12345` is the name of the Lustre file system.

```
client$ lfs getname
cls12345-ffff9d95acadf800 /lus
```

3. Obtain the Lustre identifier of the file to mirror or migrate.

```
client$ lfs path2fid /lus/FILENAME
[0x200004abe:0xe518:0x0]
```

4. Execute a `csms cds requests create` command to migrate or mirror the file.

The following command includes the `--format json` option to show the typical output after successfully submitting a request.

- `FILE_FID` is the Lustre file identifier from the previous step.
- `TEMPLATE` is the path of the mirror or migrate template directory, relative to the Lustre mount point.
- Both the `--source-fsid` and `--target-fsid` values are both the Lustre file system name obtained in Step 2.

```
ext-adm# csms cds requests create --action migrate \
--source-fid FILE_FID \
--source-fsid cls12345 --target-fname TEMPLATE \
--target-fsid cls12345 --format json
{
  "requestId": "23",
  "requestState": {
    "message": "request received",
    "state": "PENDING",
    "time": "2021-09-02T07:36:08.715804342Z"
  },
  "traces": [
    {
      "message": "request received",
      "state": "PENDING",
      "time": "2021-09-02T07:36:08.715804342Z"
    }
  ]
}
```

## More information

[`csms cds requests create`](#)

# ClusterStor data services Policies

ClusterStor data services policies are written instructions for the optimal, automatic management of a file system. A primary use case of ClusterStor data services policy is to manage free space in a hybrid Lustre file system. Policies can trigger to run based on free space capacity in the whole file system, within a pool of OSTs, or on a single OST.

## Policies provided by ClusterStor data services

When the Policy Engine starts, it looks for specific policy-related directories in the managed Lustre file system. If it cannot find them, then the Policy Engine creates the following default structure within the `.cray` directory at the root of file system:

```
└─ cds
   ├── README
   ├── default_policy
   │   └─ indexing.plc
   ├── example_policy
   │   ├── flash_maintenance.plc
   │   └─ flash_migrate.plc
   ├── template
   ├── policy
   └─ tools
```

Policy Engine also copies the contents of the `default_policy` directory to `.cray/cds/policy` if it does not exist when Policy Engine starts.

The `cds_example_policy` directory includes useful example policies. These examples, however, are not complete since the templates they reference (`/lus/disk_layout_example` for example) do not exist. These templates do not exist because the Policy Engine automatically creates an empty `cds/template` directory. Policy Engine creates this directory for templates as a convenience; data movement templates can exist anywhere in the file system.

Therefore, policy authors must create at least one template. Then policy authors can modify these examples to reference real templates when creating site-specific policies or before using them in their current form to move data. Since policy authors must modify these policy files before using them, Hewlett Packard Enterprise recommends that using the `testparse` tool to check policy files for errors. See [Check Policy Files for Errors](#) for instructions.

ClusterStor data services includes one active default policy: `indexing.plc`. This policy routinely scans the managed Lustre file system and updates the distributed metadata database. See [Scalable Search Service](#) for more information about this database.

ClusterStor data services provides two example, inactive flash maintenance policies:

- `flash_maintenance.plc`: a policy that, every four hours, mirrors large files in flash to disk and checks if the flash tier is at or above 80% full. If the flash tier meets that criterion, the policy punches all large mirrors that are in sync.
- `flash_migrate.plc`: A flash maintenance policy that, every 10 minutes, checks if the flash tier is at or above 85% full. If the flash tier meets that criterion, the policy migrates from flash to disk all files meet these criteria:
  - Are larger than 100MB.
  - Nothing accessed the files in the last two days.

Both of these policies are in `.cray/cds/example_policy`. Sites can customize any of the three provided policies to serve their tiering needs. For example, users can change the trigger condition for `indexing.plc` to index the file system less frequently. For more information, see [Flash Maintenance](#) and the section "Policy file examples" in [Policy Syntax](#).

## Policy file terms

Fundamentally, a policy defines an action to take on set of candidate files that match a set of selection criteria. Policies also require a triggering event. A triggering event can be a particular state of the file system or a simple timer. Policies must include at least one of the following statement groups:

- `fileclass`: defines the set of candidate files
- `rule`: defines the action to take on the candidate files
- `action`: requests ClusterStor data services to perform a data movement operation on the candidate files
- `trigger`: the condition that causes ClusterStor data services to search for candidate files and perform some action on them.

Refer to [Policy Syntax](#) for more information about writing policies.

## Policy submission and template paths

Administrators define policies in text files with the extension `.plc`. Only policy files stored in `lustre_mount_point/.cray/cds/policy/`, where `lustre_mount_point` is the mount point of the Lustre file system, are run by the Policy Engine. The Policy Engine will execute all policy files found in this directory.



A policy file might describe more than one policy. Policies are referenced by their policy name in the policy file by `define_policy` statements.

ClusterStor data services uses directories that have striping settings applied through `lfs setstripe` as templates for data movement. Hewlett Packard Enterprise recommends that:

- Template directories do not contain user data.
- All template directories reside in the `.cray/cds/template` directory at the root of the mounted Lustre file system.

Template paths in policy files must be relative to the root of the mounted file system. The Connector pods mount the Lustre file system at `/lus` but external hosts might mount the same file system at a different mount point, such as `/mnt/lustre`. For example, the following policy statement directs the Policy Engine to use the `flash2_migrate_template` template in `/lus/.cray/cds/template/`:

```
action_params {  
  template = .cray/cds/template/flash2_migrate_template  
}
```

## Disable a policy

Disable a policy by removing the `.plc` extension, replacing that extension with another one, or appending another extension (such as `.disabled`) to the file name. Any of these actions prevents the policy from generating new queries. However, the Tiering Engine will still process any outstanding queries and associated data movement requests from the previous policy run.

## Policy Scope

Definitions in policy files apply only to that file. Policy writers must fully define each policy within a single policy file. A policy file might define multiple policies, as long as there are no name collisions within the file. Name collisions are a syntax error.

Policy components in different policy files might have the same names. For example, both `policy_file1.plc` and `policy_file2.plc` might have a policy named `flash_copy`. The Policy Engine considers each policy file a separate namespace.

## Policy File Access and Policy Action Permissions

The policy directory is subject to administrator-controlled access control settings. The administrator might set directory permissions to allow other users to add policy files into this directory. If so, each user may also set file permissions on their own files within this directory, making their policy readable to others if wanted.

The Policy Engine includes the UID of policy file in each request. Each request generated by the Policy Engine also includes an identifier that the Policy Engine is the source. The Connector will access the file under the UID of the requestor. Therefore, a policy will only affect files where the owner of the policy file has permissions. Similarly, file deletions attempted by a user policy will only execute if that policy owner has the necessary permissions.

## Policy Syntax

The ClusterStor data services Policy Engine uses the policy rules language of the Lustre 3rd-party open-source Robinhood policy engine. This allows ClusterStor data services to:

- Support all the use cases of large and tiered ClusterStor systems.
- Reduce the learning curve for users.
- Provide an easier transition from other solutions.

See [https://github.com/cea-hpc/robinhood/wiki/robinhood\\_v3\\_admin\\_doc#Policies](https://github.com/cea-hpc/robinhood/wiki/robinhood_v3_admin_doc#Policies) for more details on RobinHood policy syntax.

In general, any rules written for Robinhood will work with the ClusterStor data services Policy Engine as well. ClusterStor data services, however, only supports four actions: `migrate`, `mirror`, `punch`, and `delete`.

Policy files must only contain characters within the ASCII character set. If a policy file contains a character outside this set, the policy will not parse and therefore will not run. The `testparse` tool, however, may parse such a policy file, without errors.

### Statements and tokens

Robinhood syntax uses statement groups where a token (or pair of tokens) identifies the group. The statements of the group are enclosed in curly braces and separated by semicolons:

```
token1 token2 {  
    statement1;  
    statement2  
}
```

A semicolon is required between two statements, even if those statements are already separated by a newline. A parse error will result if a semicolon is omitted between two statements on separate lines.

### Fileclass

A `fileclass` defines the set of files that meet one or more metadata selection criteria. These criteria can be combined with logical `and` and `or` operators and precedence groupings. The Query Agent uses the file system metadata index to locate files that match the `fileclass` criteria. A `fileclass` defines a query constraint to choose a subset of the files from the index.

The primary content of a `fileclass` is a statement group named `definition`. The `definition` statement group defines the selection criteria for the file query. That criteria can be based on file attributes such as size, name, Lustre pool, and OST membership. A `definition` must be provided. Lack of a `definition` does not result in absence of criteria. Rather, the `fileclass` is not applied for the referring rule.

In the following example, the `fileclass` named `megabyte_files` represents a query for those files that are greater than 1 MB in size:

```
fileclass megabyte_files {  
    definition { size > 1mb }  
}
```

The following table describes the allowed fields and comparisons for policies in ClusterStor data services:

**Table 2: Allowed `definition` fields and comparisons**

Name	Comparators	Description	Notes
<code>size</code>	<code>&lt;</code> , <code>&lt;=</code> , <code>&gt;</code> , <code>&gt;=</code> , <code>=</code>	Size of file, in bytes	The size may have a multiplier appended to the value (no white space allowed between the value and the multiplier). Size multipliers are case insensitive. Both power of 10 multipliers ( <code>kb</code> , <code>mb</code> , and <code>gb</code> for example) and power of 2 multipliers (including <code>kib</code> , <code>mib</code> , and <code>gib</code> ) are supported.
<code>name</code>	<code>=</code>	Name of the file, must be quoted with <code>"</code> or <code>'</code>	Wildcard character <code>*</code> is supported for filenames.
<code>pool</code>	<code>=</code>	Name of OST pool	Must match actual pool name ( <code>flash</code> , for example) exactly.

Name	Comparators	Description	Notes
<code>ost</code>	<code>=</code>	OST identifier	<p>An integer representing an OST. OST identifiers emitted by Lustre utilities are in hexadecimal and may be prefixed with <code>OST</code>. Values provided here prefixed with <code>OST</code> or <code>0x</code> are interpreted as hexadecimal, otherwise, they are interpreted as decimal.</p> <p>Example: for OST 16, valid inputs are <code>0x10</code>, <code>16</code>, or <code>OST0010</code>.</p>
<code>mirror</code>	<code>=, !=</code>	Tests whether a mirror f the file exists, and if the mirrors (if it exists) are in sync.	<p>Examples:</p> <ul style="list-style-type: none"> <li><code>mirror = sync</code> : mirror for the file exists and is in sync</li> <li><code>mirror != sync</code> : a mirror exists and is out of sync</li> <li><code>mirror = none</code> : mirror does not exist</li> <li><code>mirror != none</code> : mirror exists</li> </ul>

ClusterStor data services internally limits the list of candidate files for any single `fileclass` to a fixed maximum number of results. This limit avoids burdening the system with large backlogs of requests. As the system works through the request backlog, subsequent triggers will rerun the rule and return remaining results that still match the search criteria.

## Policy rules

The `rules` statement group defines one or more rules for a policy. A `rule` associates an `action` with one or more `fileclass`. A `rules` statement group takes the following form:

```
policy_name_rules {
    rule rule_name {
    }
}
```

The *policy\_name* must match the policy name of the policy to which the rules will apply. One or more rule statement groups may be defined with unique rule names given by *rule\_name*.

**Table 3: Supported statements for policy rules**

Statement	Description	Notes
<code>target_fileclass</code>	Statement declaring the <code>fileclass</code> for the rule	May be specified multiple times to allow the rule to apply to more than one <code>fileclass</code> .

Statement	Description	Notes
<code>action</code>	<p>Statement declaring the action for the <code>rule</code>. It defines the wanted effect to be applied to every file in the <code>fileclass</code>.</p> <p>Supported actions:</p> <ul style="list-style-type: none"> <li>• <code>migrate</code>: Change the Lustre layout of a file to match the layout (for example, file stripe count, size, PFL) defined in a template. A <code>migrate</code> action retains the original inode.</li> <li>• <code>delete</code>: remove a file from the file system namespace and reclaim the storage used by the file data.</li> <li>• <code>mirror</code>: Create a mirrored (FLR) file from a non-FLR file or synchronize the data in an FLR file. Same result as <code>lfs mirror resync</code> or <code>lfs mirror extend</code>, depending on the file's existing mirrors. A template must be specified to describe a new mirror's layout.</li> <li>• <code>punch</code>: Remove one in-sync mirror of an FLR file from the file system. The file remains in the namespace.</li> <li>• <code>incremental_index</code>: Index a directory or set of directories. If a directory has already been indexed, an incremental reindex will occur and the index will be updated starting from that directory.</li> </ul>	<p> <b>WARNING:</b> Before running any policy using the <code>delete</code> action, verify that the <code>definition</code> statements for the <code>fileclass</code> and for the <code>condition</code> statement group are correct. Incorrect syntax may cause Query Agent to delete more files than intended by the policy.</p> <hr/> <p>ClusterStor data services only supports creating and resyncing two mirrors.</p>
<code>action_params</code>	<p>Statement group containing <code>template</code> statements, which specify the template to use for the Lustre layout settings for migrate or mirror actions. See <a href="#">Create a Template</a> for more information on templates.</p> <p>This statement group can also contain optional <code>index_root</code> and <code>index_level</code> parameters for advanced performance tuning of indexing policies.</p>	<p><code>template</code> statements are an addition to policy syntax that is unique to ClusterStor data services policies.</p> <hr/> <p> <b>WARNING:</b> Contact HPE for guidance before attempting to set either the <code>index_root</code> or <code>index_level</code> parameters. Unexpected and undesired file system actions can occur if these parameters are set improperly.</p> <hr/>

Statement	Description	Notes
<code>condition</code>	<p>Statement group containing one statement allowing the query to be constrained based on time.</p> <p>These conditions are supported:</p> <ul style="list-style-type: none"> <li><code>last_access</code></li> <li><code>last_modified</code></li> <li><code>last_changed</code></li> </ul>	<p>The only supported comparator for <code>condition</code> statements is <code>&gt;</code>.</p> <p>The operand must be an integer number of days with an immediate <code>d</code> suffix. No other comparator or time unit is allowed.</p> <p><b>⚠ WARNING: Any unexpected comparator or operand value will cause the whole condition to be discarded.</b></p>

## Policy definitions

A `define_policy` statement group is required to define a policy and must be defined in conjunction with an identically named `_trigger` and `_rules` statement group. A `default_action` statement is also required, as the `define_policy` section must not be empty. A `define_policy` statement group typically looks like the following example:

```
define_policy <policy name> {
    default_action = migrate;
}
```

## Triggers

Triggers are a statement group that defines one or more events that initiate executing a policy. In other words, a trigger defines when to run a rule. Trigger statements take the form of:

```
policy_name_trigger {}
```

The `policy_name` must exactly match the name of the policy to which the trigger will apply.

The following table lists and describes supported trigger statements.

**Table 4: Supported trigger statements**

Statement	Description	Notes
<code>trigger_on</code>	<p>Defines an event on which to trigger. The following trigger types are supported:</p> <ul style="list-style-type: none"> <li><code>pool_usage</code>: Triggers on the used capacity of a pool (measured against <code>high_threshold_pct</code>). Requires a pool name operand.</li> <li><code>ost_usage</code>: Triggers on the used capacity of an OST (measured against <code>high_threshold_pct</code>) identified by the <code>ost</code> identifier.</li> <li><code>global_usage</code>: Triggers on the used capacity of the whole file system (measured against <code>high_threshold_pct</code>).</li> <li><code>schedule</code>: Triggers on a schedule specified using <code>cron</code>'s five field syntax: minute, hour, day of month, month, and day of week. This type of trigger is useful for running incremental indexing operations.</li> </ul>	<p>All trigger statements require function notation: trigger name is the function name with a parameter within parentheses.</p> <p>The <code>global_usage()</code> trigger type does not require any operands and will ignore any that are supplied. The <code>pool_usage()</code> and <code>ost_usage()</code> trigger types, however, require a parameter to specify which pool or OST, respectively, to consider.</p> <p>For example, <code>pool_usage(flash)</code> triggers on the used capacity of the OST pool named <code>flash</code>.</p>

Statement	Description	Notes
check_interval	Number of seconds between checks of the trigger_on condition.  If unspecified, default is 5 minutes.	Supported time unit specifiers: <ul style="list-style-type: none"> <li>s - seconds</li> <li>m - minutes</li> <li>h - hours</li> <li>d - days</li> </ul>
high_threshold_pct	Percent above which the policy will run.	Integer and float values are allowed. The percent character (%) is optional.

The Scalable Search Service creates and updates the database used by the Query Agent to generate candidate files for a given selection criteria in a policy. As a result, repeated data-movement policy runs will generally produce the same set of candidates unless the databases are updated. Therefore, do not write data movement policies that are triggered to run more frequently than the active indexing policy. The default indexing policy included with ClusterStor data services, indexing.plc, is triggered every three hours. If a data movement policy must avoid stale results, that indexing schedule can be modified necessary.

## Policy file examples

ClusterStor data services provides two example flash maintenance policies (see [ClusterStor data services Policies](#)).

The flash\_migrate.plc policy moves large files in the flash pool to the disk pool if they have not been accessed in two days. The `/lus/layout_example` placeholder must be replaced with the path of a valid template before this policy can migrate data.

```
fileclass largeflash {
    definition { size > 100MB and pool = flash }
}

flash_migrate_rules {
    rule migrate_large {
        target_fileclass = largeflash;
        action = migrate;
        action_params {
            template = /lus/layout_example
        }
        condition { last_access > 2d }
    }
}

# trigger FM policy if any OST in pool 'flash' exceeds 85% of disk usage.
flash_migrate_trigger {
    check_interval = 600;
    trigger_on = pool_usage(flash);
    high_threshold_pct = 85%;
}

define_policy flash_migrate {
    default_action = migrate;
}
```

The flash\_maintenance.plc policy uses `mirror` and `punch` actions to release space in the flash tier much more quickly:

```
# Every four hours, mirror unmirrored flash files to disk.

fileclass large_unmirrored_flash {
    definition { size > 10MB and pool = flash and mirror = none }
}

flash_mirror_rules {
    rule mirror {
        target_fileclass = large_unmirrored_flash;
        action_params {
            template = /lus/disk_layout_example;
        }
    }
}
```

```

        condition { last_modified > 1d }
    }
}

flash_mirror_trigger {
    trigger_on = schedule("0 */4 * * *");
}

define_policy flash_mirror {
    default_action = mirror;
}

# Every four hours, check flash pool usage, and if above threshold,
# punch any flash mirrors.

fileclass mirrored_files {
    definition { size > 10MB and mirror = sync }
}

punch_mirror_rules {
    rule mirror {
        target_fileclass = mirrored_files;
        action_params {
            punch_mirror = flash;
        }
        condition { last_modified > 1d }
    }
}

punch_mirror_trigger {
    trigger_on = pool_usage(flash);
    check_interval = 4h;
    high_threshold_pct = 80%;
}

define_policy punch_mirror {
    default_action = punch;
}

```

The following policy is an example of incremental indexing and will reindex the subtree under `lustre_mount_point/home/user1` every 3 hours.

```

fileclass reindex_dirs {
    definition { name = "/home/user1" }
}

index_disk_rules {
    rule disk_incremental {
        target_fileclass = reindex_dirs;
        action = incremental_index;
    }
}

index_disk_trigger {
    trigger_on = schedule("0 */3 * * *");
}

define_policy index_disk {
    default_action = incremental_index;
}

```

The following policy contains multiple rules which run from a single trigger. Every 30 minutes the storage consumption of the `flash` OST pool is checked. If it is over 70% full, then the policy moves to disk any log files larger than 100MB that have not been accessed in

the last two days. The policy will also migrate to disk any files larger than 1GB that have not been accessed in the last 10 days.

```
fileclass big_log {
    definition { name = "*.log" and size > 100MB }
}

fileclass hugefiles{
    definition { size > 1GB }
}

bigflash_rules{
    rule flash_move_large {
        target_fileclass = big_log;
        action_params {
            template = .cray/cds/template/disk_pool;
        }
        condition { last_access > 2d }
    }
    rule flash_move_huge {
        target_fileclass = hugefiles;
        action_params {
            template = .cray/cds/template/disk_pool;
        }
        condition { last_access > 10d }
    }
}

bigflash_trigger {
    check_interval = 30m;
    trigger_on = pool_usage(flash);
    high_threshold_pct = 70%;
}

define_policy bigflash {
    default_action = migrate;
}
```



# Check Policy Files for Errors

Use `testparse` with the `-v` option to check one or more policy files for syntax errors. Perform this check on new and modified policy files before enabling.

## Prerequisites

Copy the `testparse` executable from the `.cray/cds/tools` directory at the root of the mounted Lustre file system to a directory in the `PATH` of a Lustre client node.

## Procedure

1. Run `testparse` against one or more policy files.

```
client$ cd /lus/.cray/cds/policy
client$ testparse -v example_policy1.plc example_policy2.plc
```

If none of files checked contain any syntax errors, `testparse` returns a YAML representation of all the files checked. Refer to [Policy File Errors `testparse` Can Detect](#) for a list of errors `testparse` can identify.

The following example demonstrates `testparse` output from checking a policy file that contains an unexpected statement group:

```
client$ testparse -v example_policy3.plc
example_policy3.plc: unexpected statement group
```

Parse errors prevent `testparse -v` from returning a YAML representation.

2. **Optional:** Fix all parse errors in all checked files if the previous step reported any.
3. Check for logical and grouping errors in the policy file by performing [Inspect Parser Interpretation of Policy Files](#) after `testparse` validates the syntax of the policy file or files.

## Inspect Parser Interpretation of Policy Files

In addition to detecting fatal syntax errors, `testparse` can optionally produce a YAML representation of all checked policy files. The YAML output reflects what a policy definition file "means" to the Policy Engine parser.

To find logical and grouping errors in policy files before enabling them on a Lustre file system, inspect this YAML reflection.

### Prerequisites

#### Check Policy Files for Errors

### Procedure

1. Obtain the YAML reflection of one or more policy files.

The following example demonstrates a successful parse and YAML reflection of two active policies.

```
client$ cd /lus/.cray/cds/policy
client$ testparse -v flash_migrate.plc indexing.plc
flash_migrate.plc:
fileclass:
  largeflash:
    definition: size > 100MB and pool = flash
policies:
- flash_migrate:
  policy:
    default_action: migrate
  rules:
    migrate_large:
      action: migrate
      action_params:
        template: .cray/cds/template/disk_pool
      condition: last_access > 2d
      target_fileclass:
        - largeflash
  trigger:
    check_interval: '600'
    high_threshold_pct: 85%
    trigger_on: pool_usage(flash)

indexing.plc:
policies:
- index_disk:
  policy:
    default_action: incremental_index
  rules:
    disk_incremental:
      action: incremental_index
      action_params: {}
      target_fileclass: []
  trigger:
    trigger_on: schedule("0 */3 * * *")
```

2. Inspect the output of the previous step for all checked files.
  - a. Verify that the `policy:`, `rules:`, and `trigger:` sections are not empty.
  - b. Verify that each statements and statement group in the checked policy files appear in the correct `policy:`, `rules:`, or `trigger:` section.
3. If necessary, modify any policies as necessary so that the expected YAML reflection is produced.

Refer to [Policy Syntax](#) as needed.



## Check File Movement Status with `lfs getstripe`

Check the progress of a file migration request by retrieving the layout information from Lustre. If the OST pool name returned by Lustre for that file is different from that the original pool, then the movement request has been completed. If the OST pool location has not changed, then the movement request is either still pending, in progress, or was canceled.

### Prerequisites

Issue a manual data movement request from one storage tier to another, following the instructions in [Request a File Migration from a Lustre Client](#).

### Procedure

1. Query Lustre for the layout information of the file for which a data movement request has been issued. The `lfs getstripe` command is issued from a Lustre client.

```
client$ lfs getstripe -p example_file
flash
```

The `-p` or `--pool` option causes `lfs getstripe` to only return the name of the OST pool in which the file resides. If the file is not assigned to a specific pool, the previous command returns an empty line. In the previous example, `example_file` is in the flash tier, but a request has been submitted to move it to the disk tier.

2. Repeat the previous command as necessary until Lustre indicates that file has moved from the source OST pool to the destination OST pool.

```
client$ lfs getstripe -p example_file
disk
```

## Check File Movement Status Using File `ctime`

Check the progress of a file movement request from one tier to another by querying the file metadata change time ( `ctime` ). This attribute updates on layout changes, and thus will update after migrating a file from one OST pool to another.

### Prerequisites

Identify a file for data movement from one storage tier to another (for example, from the `flash` OST pool to the `disk` OST pool).

### Procedure

1. Query the file `ctime` of a file for which a data movement request will be issued.

```
client$ ls -lc /lus/test_dir/example_file
-rw-rw-r-- 1 example_file 1318 Dec 12 04:00 /lus/test_dir/example_file
```

2. Issue a manual data movement request for the file. In the following example, the data movement requested is to move `example_file` from the disk tier to the flash tier.

```
client$ lfs migrate --pool flash --hsm /lus/test_dir/example_file
```

3. Repeat the `ls -lc` command as necessary until the file `ctime` returned by Lustre indicates that the file migration has completed. The new `ctime` will be later than the one returned in Step 1.

```
client$ ls -lc /lus/test_dir/example_file
-rw-rw-r-- 1 example_file 1318 Dec 12 04:03 /lus/test_dir/example_file
```

## Query Data Movement Request Status with CSMS CLI

Query the status of one or more data movement requests by using either the `csms cds request describe` or `csms cds requests list` commands.

### Prerequisites

Perform [Install CSMS CLI](#) on an external system.

### Procedure

Obtain the status of one or more data movements requests submitted to ClusterStor data services using one of the following methods:

- Query the status one specific request with `csms cds request describe` and the request ID number.

```
csms$ csms cds request describe REQUEST_ID --format text
Request id:      REQUEST_ID
Request state:
  Message:      (null)
  State:        COMPLETED
  Time:         2021-09-02T07:36:12Z
Traces:
  Message          State      Time
  -----
  request received  PENDING   2021-09-02T07:36:08.715804342Z
  Queue the request QUEUED    2021-09-02T07:36:11.602Z
  (null)           RUNNING   2021-09-02T07:36:11Z
  (null)           COMPLETED 2021-09-02T07:36:12Z
```

- Query the status of all requests currently saved in the Tiering Engine.

```
csms$ csms cds requests list
```

- Group and sort all requests by state to produce an overall status report.

```
csms$ csms cds requests list|grep -A2 requestState | egrep -v 'requestState|null|--' \
| awk -F'"' '{groups[$2];count[$2]+=1} END{for (grp in groups) {print count[grp]" "grp}}'
```

```
100 RUNNING
1 Resource temporarily unavailable
1 Operation not permitted
2 Is a directory
2 No such file or directory
1 File exists
7 ERROR
25 COMPLETED
```



## Search for Files Using Query

The Scalable Search Service records various metadata about each file in the file system database during indexing. ClusterStor administrators can use these metadata together with SQL syntax to construct file search queries which execute quickly.

Each line of Query output represents a separate matching indexed file and includes a comma-separated list of the following information (from left to right):

1. The absolute path of the matching file, including the filename
2. The name of file
3. The file size in bytes
4. The UNIX epoch time stamp

### Prerequisites

- [Install Scalable Search Tools on a Lustre Client](#)

### Procedure

1. Log into a Lustre client node as `root`.
2. Choose the file metadata to use as search criteria. Refer to [Administrator Search Tools](#) for a full list of indexed file metadata.
3. Construct and run a file search query on a Lustre client node using the metadata selected in the previous step and valid SQL syntax. Use `%s` in the place of table and column names.

The following example searches for five indexed files greater than 10,000 bytes in size.

```
client# /opt/cray/brindexer/bin/query -q "select %s from %s where size > 10000" -limit 5 /lus
/lus/4/testfile-5734-0,testfile-5734-0,68305,1598072452000000000
/lus/4/testfile-5734-1,testfile-5734-1,113722,1598072452000000000
/lus/4/testfile-5734-10,testfile-5734-10,80484,1598072452000000000
/lus/4/testfile-5734-100,testfile-5734-100,96030,1598072453000000000
/lus/4/testfile-5734-1000,testfile-5734-1000,129607,1598072465000000000
```

The `%s` characters enable searches without knowledge of the Scalable Search Service database schema. Query will replace each `%s` instance with the appropriate table or column name before running the SQL query. To use the `%` character in a SQL query, escape it with another `%`:

```
client# /opt/cray/brindexer/bin/query -q "select %s from %s where name like 'test%'"
/lus/test.c
/lus/test.log
/lus/test_input.txt
/lus/test_drv.ko
/lus/test
```



Run Arbitrary Commands on Query Search Results

Use the `-exec` option of Query to perform an arbitrary command on each file in the search results list. This option is used the same way as `find's -exec` option.

Prerequisites

- Read [Administrator Search Tools](#)
- [Install Scalable Search Tools on a Lustre Client](#)
- Refer to [Search for Files Using Query](#) for basic instructions on using `query` for file searches.

Procedure

1. Log into a Lustre client node as `root`.
2. Construct and run a complete `query` search command that includes the `-exec` option at the end before the Lustre mount point. Refer to [Administrator Search Tools](#) as needed.

The following `query` command executes the `stat` command on each found file, directory, or hard link in the Lustre file system.

```
client# /opt/cray/brindexter/bin/query -q "select %s from %s where \
size > 30000" -exec "stat {}" /lus
16777220 13009028200 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028172 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/tiny/
16777220 13009028170 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/tiny/
16777220 13009028193 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028171 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/tiny/
16777220 13009028191 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028196 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028202 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028201 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028197 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
16777220 13009028198 -rw-r--r-- 1 user1 (12790) 0 40960 "Aug 7 03:16:44 2020" "Jun 2 16:51:27 2019" "Jun 2 13:12:19 2020" "Jun 2 12:40:48 2019" 4096 80 0 /lus/next
```



## Delete Files in Parallel with Query

Adding the `--delete` option to a `query` command is a fast and efficient way to delete files that match a specific criteria expressed in SQL syntax.

### Prerequisites

- Read [Administrator Search Tools](#)
- [Install Scalable Search Tools on a Lustre Client](#)

### Procedure

1. Log into a Lustre client node as `root`.
2. Construct and run a `query` command which includes selection criteria, the `--delete` option, and the mounted Lustre file system. Refer to [Administrator Search Tools](#) as needed. Replace `/lus` in the following command with the mount point of the Lustre file system.

```
client# /opt/cray/brindexer/bin/query -q "select %s from %s where size > 30000" --delete /lus
Delete file:/lus/next copy/myfile
Delete file:/lus/next copy/3file
Delete file:/lus/next copy/another
Delete file:/lus/tiny/onefile-0
Delete file:/lus/tiny/onefile-0 copy
Delete file:/lus/next/3file
Delete file:/lus/next/another
Delete file:/lus/next/myfile
Delete file:/lus/level1/level2/tiny copy/onefile
Delete file:/lus/level1/level2/tiny copy/onefile-0
Delete file:/lus/next copy/tiny/onefile-0 copy
Delete file:/lus/next copy/tiny/onefile-0
Delete file:/lus/tiny/next/3file
Delete file:/lus/tiny/next/another
Delete file:/lus/tiny/next/myfile
Delete file:/lus/next copy/tiny/next/3file
Delete file:/lus/next copy/tiny/next/another
Delete file:/lus/next copy/tiny/next/myfile
```



## Monitor ClusterStor data services Activity with Lustre Jobstats

Administrators can monitor the Lustre activity of ClusterStor data services using jobstats tracking on Lustre servers. Use this data to verify ClusterStor data services operating health and to troubleshoot issues.

Use the Lustre `job_id` of Cray-CDS to view the ClusterStor data services performance statistics tracked by the Lustre servers. Refer to [https://doc.lustre.org/lustre\\_manual.xhtml#dbdoclet.jobstats](https://doc.lustre.org/lustre_manual.xhtml#dbdoclet.jobstats) for more information on Lustre jobstats.

### Procedure

#### Obtain ClusterStor data services Lustre metadata statistics

1. Log into an MDS as root.
2. Query the MDT for all the Lustre job metadata statistics.

The Cray-CDS `job_id` contains the statistics for ClusterStor data services.

```
MDS# lctl get_param mdt.*.job_stats
. . .
- job_id:          Cray-CDS
  snapshot_time:   1602198854
  open:            { samples:      39011, unit: reqs }
  close:           { samples:      39010, unit: reqs }
  mknod:           { samples:      24728, unit: reqs }
  link:            { samples:        0, unit: reqs }
  unlink:          { samples:      24728, unit: reqs }
  mkdir:           { samples:        0, unit: reqs }
  rmdir:           { samples:        0, unit: reqs }
  rename:          { samples:        0, unit: reqs }
  getattr:         { samples:     92231, unit: reqs }
  setattr:         { samples:     38600, unit: reqs }
  getxattr:        { samples:    1076204, unit: reqs }
  setxattr:        { samples:        0, unit: reqs }
  statfs:          { samples:        1, unit: reqs }
  sync:            { samples:     5680, unit: reqs }
  samedir_rename:  { samples:        0, unit: reqs }
  crossdir_rename: { samples:        0, unit: reqs }
  read_bytes:      { samples:        0, unit: reqs, min: 0, max: 0, sum: 0 }
  write_bytes:     { samples:        0, unit: reqs, min: 0, max: 0, sum: 0 }
  punch:           { samples:        0, unit: reqs }
. . .
```

#### Obtain ClusterStor data services Lustre object storage statistics

3. Log into an OSS as root.
4. Query the OSS for all the Lustre job data operation statistics.

The Cray-CDS `job_id` contains the statistics for ClusterStor data services.

```
OSS# lctl get_param obdfilter.testfs-OST0000.job_stats
obdfilter.testfs-OST0000.job_stats=
job_stats:
- job_id:          Cray-CDS
  snapshot_time:   1604007562
  read_bytes:      { samples:      716, unit: bytes, min: 4096, max: 24576, sum: 5984256 }
  write_bytes:     { samples:        3, unit: bytes, min: 2, max: 32768, sum: 57346 }
  getattr:         { samples:        0, unit: reqs }
  setattr:         { samples:     544, unit: reqs }
  punch:           { samples:     481, unit: reqs }
  sync:            { samples:        0, unit: reqs }
  destroy:         { samples:        0, unit: reqs }
  create:          { samples:        0, unit: reqs }
  statfs:          { samples:        1, unit: reqs }
  get_info:        { samples:        0, unit: reqs }
  set_info:        { samples:        0, unit: reqs }
  quotactl:        { samples:        0, unit: reqs }
```

## View ClusterStor data services Kibana Dashboards

ClusterStor data services includes an ELK (Elasticsearch, Logstash, and Kibana) instance for monitoring and analyzing internal log data. This ELK instance includes preconfigured Kibana dashboards. The CDS default dashboard shows the general activity of ClusterStor data services components. Use this dashboard as an indicator of overall health. The other dashboards provide more detailed information on some of these components.

The format and content of the preconfigured Kibana dashboards might change in future releases.

See <https://www.elastic.co/guide/index.html> for documentation on using Kibana.

### Procedure

1. Open the web address `https://sma-kibana.SUBDOMAIN_NAME/app/kibana` in a web browser.

*SUBDOMAIN\_NAME* is the combination of the ClusterStor data services system and domain names set during system installation.

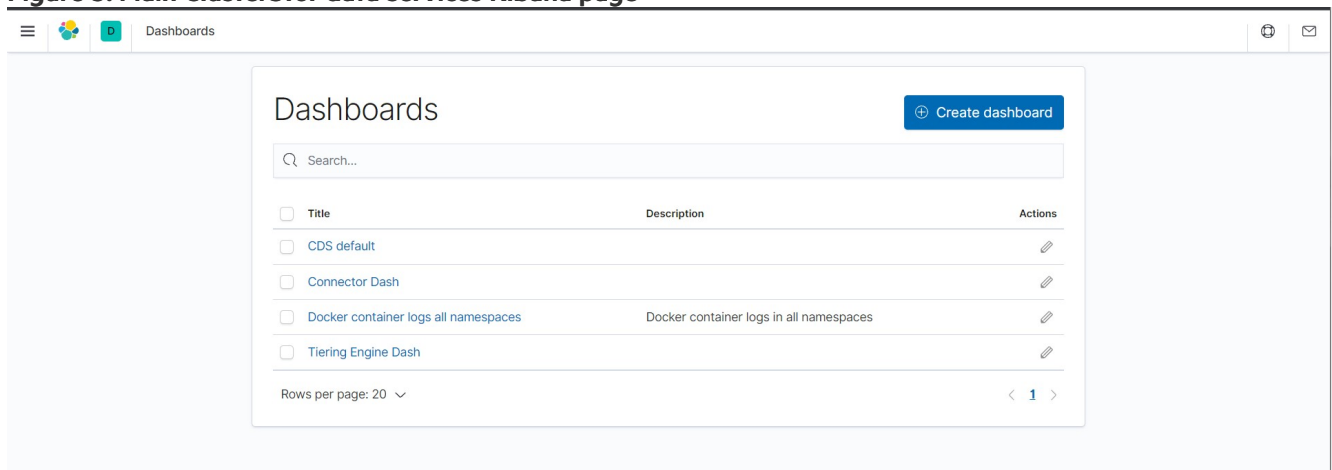
2. Authenticate with Keycloak if necessary.

Log in with a valid user account that can view the ClusterStor data services dashboards.

3. Click the icon in the upper left corner with the three horizontal lines.
4. Click Kibana > Dashboard in the menu that appears.

A page similar to the following image will appear:

**Figure 3: Main ClusterStor data services Kibana page**



5. View the dashboard or dashboards of interest.

- The CDS default dashboard contains charts that indicate the overall health of ClusterStor data services.
- The Tiering Engine Dash dashboard contains charts that visualize the request state management performed by the Tiering Engine component.
- The Connector Dash dashboard contains charts that show the data movement activity of the Connector component.
- The Podman container logs all namespaces dashboard contains tables of log-related data from the individual Podman containers that comprise a running ClusterStor data services system.

6. Verify or troubleshoot ClusterStor data services operation by performing any or all the following procedures:

- [Quickly Assess Overall Health Using Kibana](#)
- [Verify Scalable Search Service Health with Kibana](#)
- [Verify Automated Data Movement with Kibana](#)
- [ClusterStor data services Elasticsearch Fields](#)

## Quickly Assess Overall Health Using Kibana

The Stderr ratio chart indicates the overall health of ClusterStor data services over time and at the present moment. This chart displays the ratio of error messages to all system log messages. If the system is healthy, the ratio will remain at a low value.

### Prerequisites

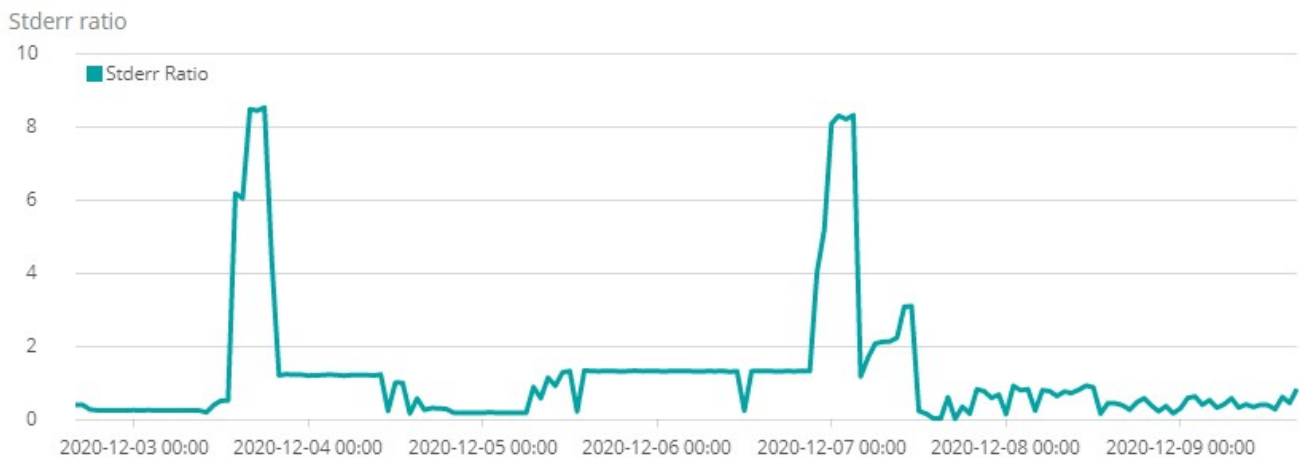
[View ClusterStor data services Kibana Dashboards](#)

### Procedure

1. Open the CDS default Kibana dashboard.
2. Adjust the Kibana Time Range to visualize the error message ratio for the wanted time span.
3. Inspect the Stderr ratio chart.

Any sustained increases over a historic baseline may indicate a problem deserving further investigation. In the following example Stderr ratio chart, there are two time periods over seven days which warrant investigation due to higher than normal levels of error messages:

**Figure 4: Stderr ratio spikes**



## Verify Scalable Search Service Health with Kibana

Modifying, creating, and deleting files on a file system managed by ClusterStor data services causes the Scalable Search Service to update its metadata database. Therefore, checking the Indexing Activity chart on the Kibana dashboard can quickly verify the following:

- The Policy Engine policy parser can read and parse policy files.
- The Scalable Search Service triggers as expected given the active indexing policy.
- The Scalable Search Service is able to update the distributed file system metadata database.

### Prerequisites

[View ClusterStor data services Kibana Dashboards](#)

### Procedure

1. Open the CDS default dashboard.
2. Adjust the Kibana Time Range to visualize the error message ratio for the time period of interest.

Select a time span that includes file system activity. It must also be long enough to capture at least one incremental indexing run, as specified by the active indexing policy.

3. Inspect the Indexing Activity chart for spikes that match the schedule of the active indexing policy.

Such spikes in the Indexing Activity chart will coincide with peaks in the number of indexing actions in the Policy Actions chart. The following screenshot shows an example of this correlation:

**Figure 5: Evidence of indexing activity in the CDS default dashboard**



## Verify Automated Data Movement with Kibana

In ClusterStor data services several components work together to move data automatically by policy:

- One or more policy files
- Policy Engine
- Tiering Engine
- Connector

Verify that each service in this process is functioning nominally by using the charts displayed in the Kibana dashboards. Alternatively, perform this procedure as a first phase in troubleshooting policy issues.

### Prerequisites

[View ClusterStor data services Kibana Dashboards](#)

### Procedure

1. Select the CDS default dashboard.

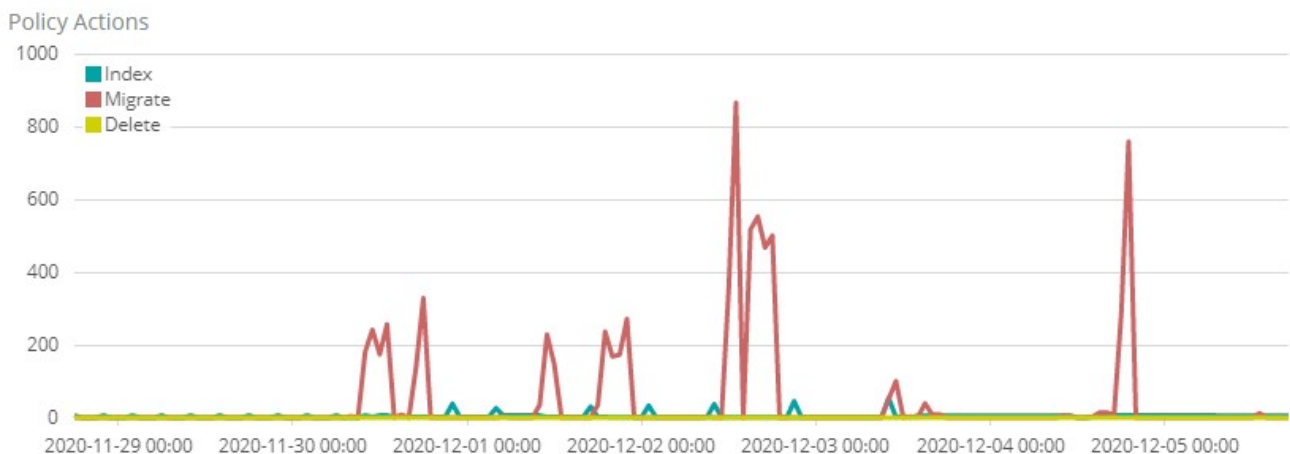
2. Select a recent time period for visualization using the Kibana Time Range pane.

Select a timespan where policy-driven data migration is expected given the active policies and the state of the Lustre file system at that time.

3. Inspect the Policy Actions chart and note the timing of any peaks in "Migrate" actions.

The following screenshot shows several peaks in policy-initiated file data migrations over a one week period:

**Figure 6: Migrate action peaks in the Policy Actions chart**



4. Open the Tiering Engine Dash dashboard and adjust the Time Range if necessary.

5. Inspect the Request State Transitions chart.

Spikes in "Queued" or "Running" transitions which coincide with the spikes observed in Step 3 indicate that:

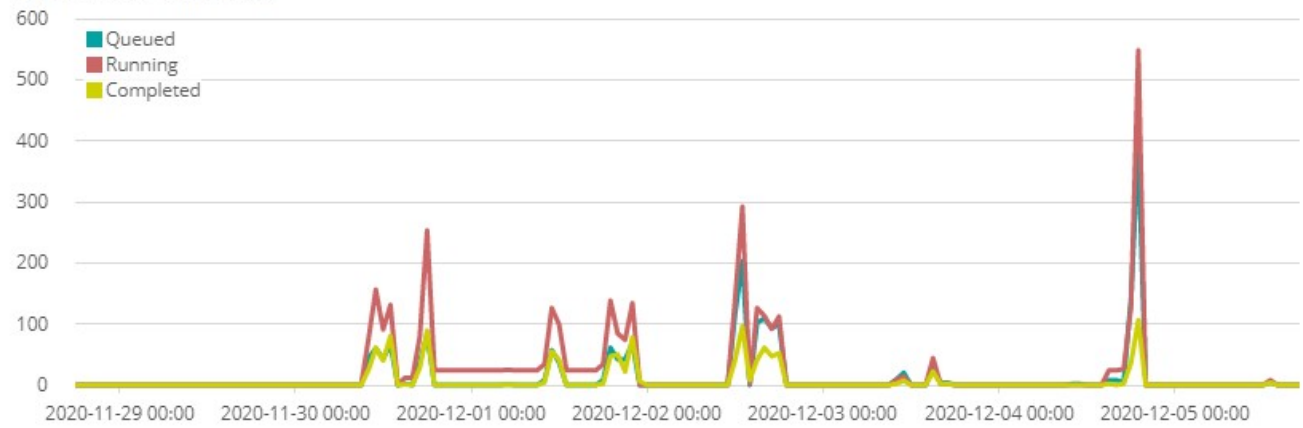
- Trigger Agent is able to forward file selection (search) requests to Query Agent.
- Query Agent is able to successfully locate files which match the `fileset` definition in at least one policy.
- Tiering Engine is able to receive and persist file data movement requests from Query Agent.

The following screenshot shows an example of peaks in request state transitions that coincide with the peaks observed in the previous example Policy Actions chart:

**Figure 7: Peaks in the Request State Transitions chart**



Request State Transitions



6. Open the Connector Dash dashboard and adjust the Time Range if necessary.

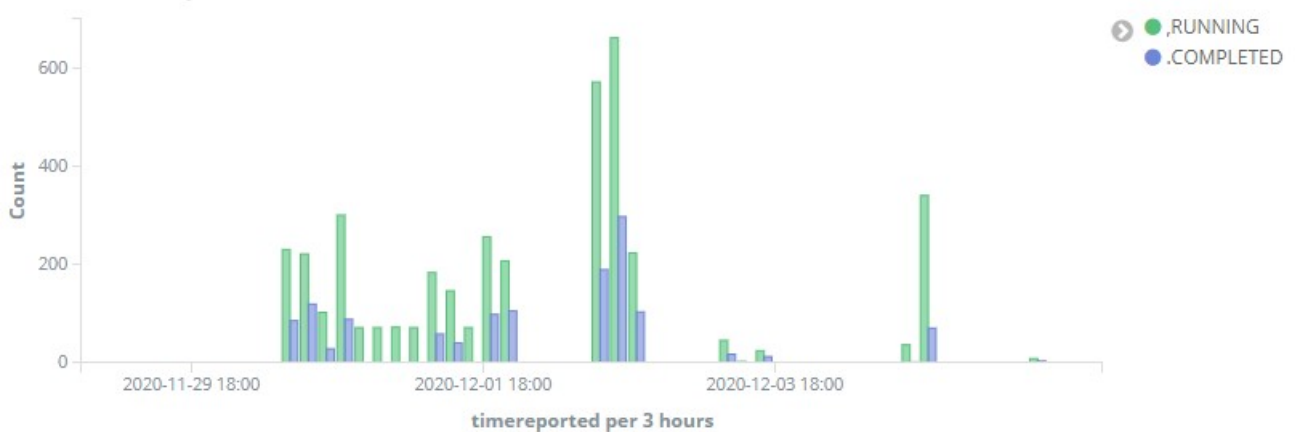
7. Inspect the Connector Activity chart.

Look for peaks in this chart which coincide with the spikes observed in Steps 3 and 5. Coinciding peaks indicate that Connector is able to retrieve and successfully complete data movement requests from the Tiering Engine.

The following screenshot shows an example of bursts of Connector activity which coincide with the peaks seen in the previous two charts:

**Figure 8: Peaks in the Connector Activity chart**

Connector Activity



## View ClusterStor data services metrics in Grafana

This procedure demonstrates how to access and display ClusterStor data services performance metrics. This release of ClusterStor data services does not provide prebuilt Grafana dashboards. For more information on how to use Grafana, refer to <https://grafana.com/docs/>.

### Prerequisites

Perform [Add or Remove Users in Keycloak](#) to add the user account or accounts that will access Grafana.

### Procedure

1. Launch a web browser and open `https://grafana.SUBDOMAIN_NAME` on an external host.

*SUBDOMAIN\_NAME* is the combination of the system and domain names of the ClusterStor data services cluster. This name is set during installation.

2. Authenticate with Keycloak if necessary.

Log in with a valid user account that can view the ClusterStor data services dashboards.

3. Click the Explore icon on the left pane.
4. Select Prometheus in the drop-down menu near the top of the page.
5. Enter a PromQL query next to Metrics.

Administrators can use the following example PromQL queries to monitor ClusterStor data services HSN bandwidth consumption:

- `rate(node_infiniband_port_data_transmitted_bytes_total[1m]) / 1024 / 1024`
- `sum (rate(node_infiniband_port_data_transmitted_bytes_total[1m])) by(endpoint) / 1024 / 1024`
- `sum (rate(node_infiniband_port_data_transmitted_bytes_total[1m])) by(instance) / 1024 / 1024`

The first query displays a separate line graph of HSN throughput data for each HSN interface on each node. The second query displays a single line graph of the combined HSN throughput for all three nodes. The third query displays the HSN throughput per node. The units are mebibytes per second for all graphs.

6. Click Run Query to generate the graph.

## About Scalable Search

ClusterStor data services provides Scalable Search tools to assist ClusterStor administrators in managing large Lustre file systems and to aid users in finding files of interest:

- **Query**: a multithreaded file search tool which uses sharded databases of file system metadata.
- **summary**: a tool for quickly producing file system usage statistics, also using sharded databases of file system metadata.

The ClusterStor data services release includes an rpm file which can install these tools on any CentOS 7 Lustre client. The package installs the command-line programs in `/opt/cray/brindexer/bin`. The command examples in this manual assume that this directory is in the PATH environment variable of the user account running the example command.

This section describes these tools in detail, as well as guidance on how to use them.

# Administrator Search Tools

Besides the search tools available to unprivileged users, ClusterStor administrators can use two additional Scalable Search tools: the Query file search tool, and `summary`, the file system summarizer.

## Query

Query uses the database shards created and maintained by the Scalable Search Service for fast parallelized file system search queries. The search results are effectively constant-time, as all shards are read in parallel. Searches are run as queries against a database and the database shards are stored within the ClusterStor file system. Therefore, file searches using the Query execute quickly from any Lustre client.

The results returned by Query reflect the file system metadata at the time of the last full or incremental indexing scan. Therefore, those results may not completely match the state of the file system when the Query search was run.

Query runs with `root` privileges as the index shards created by the Scalable Search Service are saved as files created and owned by `root`.

A file search run through `find`, `lfs find`, and a full indexing scan by the Scalable Search Service place similar loads on the metadata servers (MDSs). After the initial database shards are created, however, subsequent searches with Query will be fast, with little impact on the file system. The indices can also be updated efficiently using an incremental scan, which requires much less time than either a full scan, or a search with `find` or `lfs find`.

To run a search using Query, the user expresses the match criteria using SQL syntax and provides the indexed directory root to search through. When using this tool, administrators can use the following types of metadata as criteria:

- **Basic metadata** such as the file name, size, permission mode settings, and size
- **Time stamps** including the file `ctime`, `mtime`, and `atime`
- **Lustre-specific metadata** such as the OST pool name and OST indices containing the file

The following table contains the full list of file attributes which can be used as search criteria using Query.

Metadata Field Name in the Database	(Data Type) Description
<code>name</code>	(text) filename
<code>type</code>	(text) type of file: <code>l</code> for hard link, <code>f</code> for regular file, and <code>d</code> for directory
<code>inode</code>	(64-bit integer) Inode number
<code>mode</code>	(64-bit integer) permission mode
<code>nlink</code>	(64-bit integer) number of hard links to the file
<code>uid</code>	(64-bit integer) file owner's uid
<code>gid</code>	(64-bit integer) file owner's gid
<code>size</code>	(64-bit integer) size of the file in bytes
<code>blksize</code>	(64-bit integer) size of the file's data blocks
<code>blocks</code>	(64-bit integer) number of data blocks in the file
<code>atime</code>	(64-bit integer) time stamp of last file access
<code>mtime</code>	(64-bit integer) time-stamp of last file data modification
<code>ctime</code>	(64-bit integer) time-stamp of last file metadata modification
<code>hsmarchid</code>	(64-bit integer) Lustre archive ID of the file
<code>poolname</code>	(text) Name of the OST pool the file resided on at the time of the scan
<code>ostindices</code>	(text) Index numbers of the OSTs which store the file
<code>mirrorstate</code>	(64-bit integer) a number that reports the existence and state of FLR files: <ul style="list-style-type: none"><li>• <code>0</code> : no mirror</li><li>• <code>1</code> : mirrors are in read-only state; mirrors are synced</li><li>• <code>2</code> : write pending; preferred mirror is in a "dirty" state</li><li>• <code>3</code> : mirror resync is in progress; mirror state will return to <code>1</code> after resync completes</li></ul>

Query implements two `find` command options for performing an action on each file in the search result set:

- `-delete` : deletes the file through the `unlink()` system call from within the search process
- `-exec {}` : forks a process to execute an arbitrary command (including `rm`) on the file.

For more information on how to use Query, see [Search for Files Using Query](#), [Run Arbitrary Commands on Query Search Results](#), and [Delete Files in Parallel with Query](#).

## summary

The `summary` tool collects and displays bulk statistical information about the entire file system. This tool uses the file system metadata stored in the database shards created and maintained by the Scalable Search Service. Like Query, the `summary` tool is run from a Lustre client with `root` privileges. Administrators can use this tool to assess storage consumption, schedule file archival or deletion, and plan for file system growth. ClusterStor administrators use this tool to rapidly generate usage statistics for the whole parallel file system. Since the summarizer uses distributed (sharded) databases which reside within the parallel file system itself, summaries execute quickly.

The file system summarizer tool returns the following information about the indexed file system:

- The total number of files
- The total number of directories, including subdirectories
- The total number of hard links
- Storage consumed by all files (that is, a sum of all the individual file sizes)
- Total count of hard links, files, and directories

To generate a file system summary, run the `summary` command as the `root` user with the mount point of the Lustre file system as the argument. The following example demonstrates running this tool on a small indexed Lustre file system ( `/mnt/lus`):

```
# summary /mnt/lus
Total file count:    26
Total link count:    0
Total Dir count:     10
Total file size:     610344
Total File Objects:  36
Time elapsed: 225.730774ms
```

The `summary` tool is designed to quickly generate usage statistics for an entire Lustre file system. The `summary` tool does not support summarizing subdirectories within the mount point of the indexed file system (for example, `/mnt/lus/example_dir`).



## How Files are Purged

In ClusterStor data services, "purge" refers to operations which release storage space occupied by the data of a file. This release supports the `delete` and `punch` operations as defined in [ClusterStor data services Policies](#).

File deletion and punch are metadata-only operations which do not require any data movement and thus can execute quickly. Thus, Query Agent directly executes purge actions triggered by policies. This approach reduces overhead and accelerates the execution of purge requests triggered by policy.

Query Agent uses Query (see [Administrator Search Tools](#)) to find files which match the selection criteria stated in policy files. Since Query is a multithreaded tool, purge operations execute in parallel.

Purge actions operate under the UID of the user who owns the policy file. A policy file owned by one unprivileged user cannot delete files or punch mirrors owned by another unprivileged user.

### punch

To request a mirror punch operation in a policy, use the `punch_mirror` action and the OST pool name. The following policy excerpt requests that ClusterStor data services remove a mirror that is in the OST pool named `flash`:

```
action = punch;
action_params {
    punch_mirror = flash;
}
```

When a policy requests a punch action, ClusterStor data services:

1. Verifies that the file has a mirror in the OST pool specified.
2. Verifies that the mirror Query Agent will punch is in sync. If a policy requests to punch the preferred mirror, the nonpreferred mirror must be in sync.
3. Calls `lfs mirror split --destroy` to remove the requested mirror.

### delete

Query is able to call the `unlink()` system call directly from the search process without forking a separate process. Since this functionality is in Query, administrators can run a Query search with deletion as well. See [Delete Files in Parallel with Query](#) for more information.

# Policy Engine Limits Data Movement Requests

## Candidate file chunking

The Policy Engine internally limits the volume of active data movement requests. This throttling benefits policies that use the `trigger_on` conditions `global_usage`, `pool_usage`, and `ost_usage` to maintain a minimum percentage of free space.

When the Policy Engine runs such policies, not all the candidate files which match the `fileset` definition will be operated on. Rather, the Policy Engine aims to migrate or purge only the number of candidate files sufficient to satisfy the free space criteria.

The Policy Engine segments the list of candidate files and operates on one segment at a time during the policy run. After each segment is processed, the Trigger Agent checks the state of managed file system against the trigger condition. If the trigger condition is still met after a segment is processed, the Policy Engine proceeds with the next segment. Thus, the actual free space incrementally approaches the target amount over time. Therefore, the amount of free space may remain under the policy target until a sufficient number of segments are processed.

This throttling prevents the Policy Engine from both performing unnecessary work and greatly overshooting the free space target specified in the policy.

## Age limit for Trigger Agent messages

The Query Agent discards messages from the Trigger Agent older than one day. This age limit prevents Query Agent from acting on a backlog of work that is both large and outdated. Such a backlog will also likely contain duplicated requests from repeated runs of the same policy identifying the same candidate files.

Therefore, administrators do not need to manually purge messages if the Query Agent is temporarily unable keep up with the Trigger Agent.



## The `testparse` Policy Checking Tool

The `testparse` command is a binary executable in the `.cray/cds/tools` directory at the root of the managed file system. The `testparse` executable is compiled for SLES15, but is a static binary and has been tested on CentOS Lustre clients.

Any user can use `testparse` to check one or more policy files for errors. The `testparse` tool can uncover two main types of errors:

- Policy Engine syntax errors that can prevent a policy from running.
- Logical or grouping errors that can cause unexpected results when a syntactically correct policy is run.

### `testparse` uncovers policy file problems

For fatal syntax errors, `testparse` prints out an error message for each error in each file checked. The `testparse` command includes an option to print out a YAML representation of the parsed policy file or files. This option enables policy writers to uncover logical and grouping errors. This feature allows policy writers and administrators to verify that the "understanding" of the policy file by the policy parser matches the intent. Discrepancies between the structure of the YAML returned by `testparse` and the structure of the policy file can reveal problems that would be difficult and time-consuming to troubleshoot after the policy is active.

### `testparse` limitations

In this release of ClusterStor data services, `testparse` cannot report the line number or the statement group where a fatal syntax error occurs. If the tool cannot successfully parse a policy file, the `testparse` tool cannot print out a YAML reflection of that file. If multiple policy files are simultaneously checked with the `-v` option, no YAML reflection will be printed for any file if any one file contains an error.

## Data Movement `mtime` Verification

ClusterStor data services performs an additional check on data migration requests initiated by policies before fulfilling those requests. A policy might select a file and queue it for migration based on its `mtime`. For example, a policy might request to migrate a file to the flash tier based on a recent `mtime`.

Like many other file metadata, the distributed metadata database stores the `mtime` for indexed files. A file, however, may change after indexing but before Connector is ready to move the file. ClusterStor data services, therefore, performs a `stat()` of the file just before moving the file data. If the `mtime` returned from `stat()` does not match the value recorded in the database, ClusterStor data services aborts the migration.

This `mtime` verification is only performed on data movement requests generated by a policy.

# Flash Maintenance

## The example flash maintenance policy

ClusterStor data services administrators can both maintain available flash space for new or promoted job data on demand and achieve maximum utilization of the flash OSTs. To help administrators accomplish these goals, ClusterStor data services includes an example flash maintenance policy for sites to customize and enable on their systems.

This example policy is named `flash_maintenance.plc` and resides in `/lustre_mount_point/.cray/cds/example_policy`. ClusterStor data services administrators can modify this policy, or write an entirely new one if wanted. Refer to the subsection "Policy file examples" in [Policy Syntax](#) for an explanation and complete listing of this example policy.

The example flash maintenance policy attempts to ensure that the flash OST pool has sufficient space for future jobs. The policy achieves this goal by continuously monitoring OSTs and automatically migrating old and large files.

## How mirroring and punching benefit flash maintenance policies

The example flash maintenance policy is simple policy that uses only the migration action to release space in the flash tier. ClusterStor data services administrators can include `mirror` and `punch` actions in policies. Administrators can use these new actions in flash maintenance policies to reduce the time needed for clearing flash storage space. The example `flash_maintenance.plc` policy shows how to use `mirror` and `punch` in such a way.

For example, a ClusterStor data services policy can create Lustre mirror files for file data in the flash tier. That same policy can also periodically resync the mirrors in the disk tier. If demand for flash space increases, that same policy or a separate one can select and punch flash mirrors to increase free flash space. Since punching is faster than migrating the file data between pools:

- There is less delay for starting new jobs that require flash space.
- Data can remain in flash for as long as possible.

## Configuration required for flash maintenance

Before a flash maintenance policy can start managing the flash tier, the ClusterStor data services administrator must:

- Create flash and disk OST pools (see [Configure OST Pools](#) for details)
- Either modify one of the example flash maintenance policies or write a new one to meet site requirements.
- Create a migration template, mirror template, or both for data moved or mirrored, respectively, to the disk tier. Refer to [Create a Template](#) for the procedure.
- Replace the placeholder migration template, `/lus/layout_example`, with the site-created one if an example flash maintenance policy is used.
- Copy or move the modified example flash maintenance policy to the active policy directory.



# Degraded Performance on a Management Node

## Symptom

Data movement performance of one of the three ClusterStor data services management nodes is lower than normal. The performance of this node is also lower than the performance of the other two management nodes.

This is the only situation where HPE recommends direct logins to the ClusterStor data services management nodes and the virtual machines (VMs) running on them.

## Cause

The worker VM on the management node has abnormally high CPU usage, memory usage, or both.

## Action

1. Run the `sum (rate(node_infiniband_port_data_transmitted_bytes_total[1m])) by(instance) / 1024 / 1024` query in Grafana. Refer to [View ClusterStor data services metrics in Grafana](#) for instructions.
2. Inspect the graph for a time period where one node stops moving data on the HSN while the other two nodes continue to move data. Note IP address of the node.

Not all nodes will show peaks at the same time, nor will the peaks always be the same magnitude across all three nodes.

3. Open the Kibana dashboard and search for log messages showing high Linux OOM Killer activity on the node identified in the previous step.

A large number of these messages can confirm that high resource utilization on a worker VM is the root cause.

4. Log in as `crayadm` to the node identified in Step 2 using SSH. Use either the IP address or host name.

```
ext-adm$ ssh crayadm@CDS_NODE
CDS$
```

5. Switch to the `root` account.

```
CDS$ sudo su
CDS#
```

6. Log into the worker VM on the node using the `shell_into_vm.sh` script.

This script will log you in as the `mercury` user on the worker VM.

```
CDS# /opt/cray/terraform/bin/shell_into_vm.sh k8s-worker1
+++ dirname /opt/cray/terraform/bin/shell_into_vm.sh
++ cd /opt/cray/terraform/bin
++ pwd
+ DIR=/opt/cray/terraform/bin
+ declare -A vms
+ vms["k8s-master"]=10.17.1.71
+ vms["k8s-worker1"]=10.17.1.72
+ vms["k8s-worker2"]=10.17.1.73
+ vms["k8s-worker3"]=10.17.1.74
+ vms["k8s-worker4"]=10.17.1.75
+ vms["k8s-worker5"]=10.17.1.76
+ vms["k8s-worker6"]=10.17.1.77
. . .
VM-W$
```

7. Switch to the `root` user.

```
VM-W$ sudo su
VM-W#
```

8. Reboot the worker VM.

```
VM-W# reboot
```





## ClusterStor data services Templates

ClusterStor data services can use directories stored anywhere within the parallel file system as templates for Lustre layout settings. Such settings include stripe count, stripe size, and PFL. The `migrate` and `mirror` policy actions use templates:

- For `migrate` actions, ClusterStor data services uses templates to set the Lustre layout for file data moved between storage tiers.
- For `mirror` actions, ClusterStor data services applies the layout stored in the template to the second mirror.



## testparse Command Syntax

```
testparse [ -v ] policy_file [policy_file_2 policy_file_3 ...]
```

### About the testparse options

The `testparse` command returns 0 after a successful parse of all files specified for syntax checking. Any error detected in any checked files causes `testparse` to return a nonzero number.

The `-v` option causes `testparse` to print out a YAML representation (reflection) of all checked files. The `testparse` command will not print out a YAML reflection for a policy file if that file contains an error. If multiple policy files are checked at once with the `-v` option, no YAML reflection will print if one of the files contains an error.

## Policy File Errors `testparse` Can Detect

The `testparse` tool can detect and report the following errors in policy files:

- **Missing characters:** unmatched `{`, `}`, and `=` characters as well as key-value pairs within a statement group that are separated by a newline, but not a `;` (semicolon)
- **Unexpected statement groups:** the first statement group is not named `fileclass` or does not start with `define_policy`, statement group names that do not end in `_rules` or `_trigger`
- **Misplaced or miswritten statements:** statements in the top (root) level of a policy file, outside of a statement group, `trigger_on` values that are not in the form of `func(value)`

### Empty policy files and statement groups

Empty policy files and policy files with empty `_trigger`, `_policy`, and `_rules` statement groups are considered valid by the `testparse` tool. Such files do not cause `testparse` to return an error, and they will not modify the managed file system. Printing out the YAML reflection of the policy with the `-v` option, however, will uncover these types of files.

## query Command Syntax

query [OPTIONS] TARGET\_DIRECTORY

Option	Default	Description
<code>-json</code>	Off. Default is CSV format for results.	Print each query result in JSON format. When multiple files match, this option returns a list of JSON documents rather than a single one for all results.
<code>-header</code>	Off.	Print file result columns as a CSV header.
<code>-q</code>	Off.	Enables queries stated in SQL with <code>%s</code> in the place of table and column names. Must be included with all queries.
<code>-limit</code>	Returns all records.	Maximum number of records to return.
<code>-v</code>	Off.	Reports total results found and how long the search took.
<code>-verify</code>	Off.	Performs a <code>stat()</code> against all matching files to obtain the <code>ctime</code> and <code>mtime</code> . Then, compare those values against the values which are recorded in the file system metadata database for that file. If either time stamp does not match, that file is skipped for any <code>-delete</code> or <code>-exec</code> operations.
<code>-delete</code>	Off.	Delete the files returned by the query. Cannot be used with <code>-exec</code> .
<code>-exec</code>	Off.	Execute an arbitrary command on all the files returned by the query. Usage is identical to the <code>find -exec</code> option. Cannot be used with <code>-delete</code> .

# Which Lustre HSM Requests Are Emitted to ClusterStor data services

In ClusterStor data services, the Emitter is able to forward all types of Lustre HSM requests, except for status queries through the `lfs hsm_action file` command. In this release, issuing this command has no effect.

The following table lists the types of Lustre HSM requests which are forwarded to ClusterStor data services by the Emitter. Refer to the Lustre documentation at [www.lustre.org/documentation](http://www.lustre.org/documentation) for more information on these `lfs` commands.

Table 5: Lustre HSM requests forwarded to ClusterStor data services

Lustre Request	Command	Notes
Move a file from the current OST or OST pool to another one.	<code>lfs migrate --hsm file</code>	
Cancel a file movement request.	<code>lfs hsm_cancel file</code>	Not implemented in current release

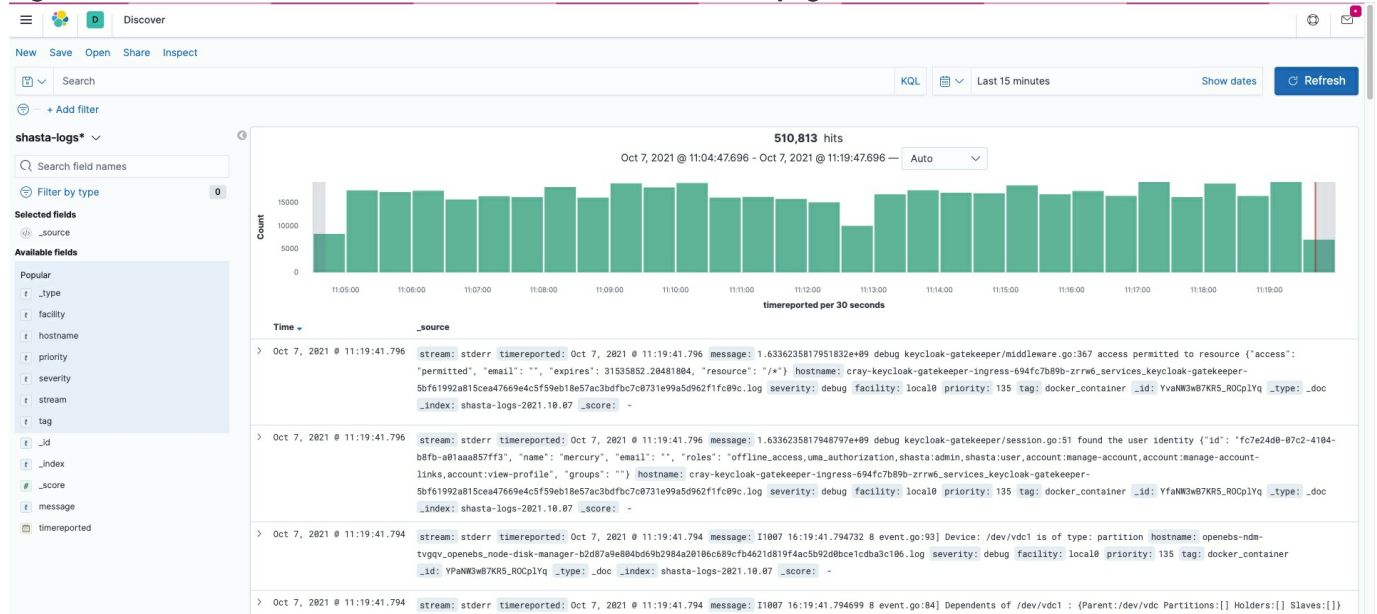
# ClusterStor data services Elasticsearch Fields

## Useful Elasticsearch fields

The ClusterStor data services Kibana dashboard visualizes data stored in an Elasticsearch database. This database is populated by data extracted from internal log files.

View all Elasticsearch database fields by performing the procedure in [View ClusterStor data services Kibana Dashboards](#) and then clicking Discover on the left. Kibana will then display a page similar to the following screenshot:

Figure 9: ClusterStor data services Elasticsearch database fields page



All of the Elasticsearch database fields which are populated from the logs are listed in a column under Available Fields. This list is on the left side of the screen. The most useful and informative fields within the ClusterStor data services Elasticsearch database are listed in the following table. The other fields can be ignored for troubleshooting and monitoring purposes in the current release.

Table 6: Informative Elasticsearch Fields

Field Name	Field Description
hostname	The log file name (which includes the Kubernetes pod name).
message	The text of the log message.
_index	A test string in the form of shasta-logs-DATE.
severity	Severity level of the log message; currently only debug and notice are used.
stream	The output stream the message was reported on, either stdout or stderr.
timereported	The time stamp from the original log entry.

# About the CSMS CLI

## Purpose

The CSMS CLI is a command-line tool for:

- Managing ClusterStor data services systems. This includes performing software upgrades, viewing system state, monitoring operations, and other administrative operations.
- ClusterStor users to use for creating and managing their data movement requests.

This CLI provides both default configurations (stored in `~/.config/csms`) and built-in authentication token stores.

The CSMS CLI is a tool written in Python that can be installed on one or more systems external to the ClusterStor data services cluster. The CSMS CLI is a client of the REST API provided by ClusterStor data services.

## CSMS CLI Security

Only users authenticated with a valid user name and password can successfully execute CSMS CLI commands. CSMS CLI users must first authenticate against Keycloak to obtain the authentication token needed to use CSMS CLI commands. The CSMS CLI uses a secure TLS channel to communicate with the ClusterStor data services API Agent.

ClusterStor data services uses the following to secure internal and external requests:

- **Istio API gateway service:** Istio provides a common access gateway for all the system management REST APIs. Istio also integrates with Keycloak to provide authorization.
- **Keycloak:** Keycloak is an open-source Identity and Access Management (IAM) solution. It provides authentication and authorization services that are used to secure access to services on the system. To learn more about Keycloak, refer to <https://www.keycloak.org/>.
- **JSON Web Tokens (JWT):** ClusterStor data services leverages the OpenID Connect standard. OpenID Connect consists of a specific application of the OAuth v2.0 standard, which leverages the use of JSON Web Tokens (JWT).

ClusterStor data services authorizes REST API calls at only place: the API gateway. The API gateway forwards every REST API call into the system to the Open Policy Agent (OPA) which then makes an authorization decision. The decision is based on the Authenticated JSON Web Token (JWT) passed into the request.

The ClusterStor data services installation process creates an X.509 Certificate Authority (CA) on the primary ClusterStor data services node. The installation process then uses the CA to create a host X.509 certificate. This host certificate is used during the installation process to configure the API gateway for TLS. TLS enables communications between API clients and the gateway to use HTTPS.

All ClusterStor data services API clients should use HTTPS to connect to services behind the API gateway. Therefore, the clients must have access to the CA certificate when making requests.

The installation process also creates a certificate for Keycloak. Keycloak signs JWTs with this certificate. The Keycloak certificate is registered with the API gateway, which uses this certificate to validate incoming JWTs. This validation verifies that the JWTs that accompany the API requests actually originate from Keycloak.



## Common CSMS CLI Options and Environment Variables

All CSMS CLI commands support a set of common command options and environment variables. This topic describes them.

### Common options

`--configuration` *CONFIG*

*CONFIG* is the name of the CSMS CLI configuration to use. Create configurations with `csms init`. All CSMS CLI commands require a configuration specified by either this option or the `CSMS_CONFIG` environment variable.

`--quiet`

`--format` *FORMAT*

*FORMAT* is either `json`, `toml`, `yaml`, or `text`.

`--token` *TOKEN\_FILE\_PATH*

*TOKEN\_FILE\_PATH* is the path of the API Agent client access token.

`--verbose`

Increase the amount and detail of command output messages.

### Common environment variables

`CSMS_CONFIG`

Sets a default for `--configuration`.

`CSMS_QUIET`

Sets a default for `--quiet`.

`CSMS_FORMAT`

Sets a default for `--format`.

`CSMS_CREDENTIALS`

Sets a default for `--token`.



## csms init

### Syntax

```
csms init OPTIONS
```

### Description

Initializes or reinitializes the CSMS CLI configuration on the host for the target ClusterStor data services system.

### Options

`csms init` supports the options common to all `csms` commands and:

**--no-auth**

Do not attempt to authenticate to the target system.

**--overwrite**

Overwrite existing configuration if it exists.

**--hostname**

The FQDN of the API Agent of the target ClusterStor data services system. The `csms init` command will prompt the user for the FQDN if this option is omitted.

### Permissions

- This command requires only unprivileged user permissions, since it only modifies the CSMS CLI configuration for the user that executes it.

### Restrictions

- This command prompts for a valid user name and password registered with Keycloak to complete successfully.
- This command requires an operational URL for the API Agent on the target ClusterStor data services cluster.

### Example input

```
csms$ csms init
```

### Example output

```
Cray Hostname: api.DOMAIN_NAME
Username: USERNAME
Password: PASSWORD

Success!
Initialization complete.
```

# csms auth login

## Syntax

```
csms auth login [OPTIONS]
```

## Description

Authenticate to the target ClusterStor data services system with the user credentials of a different account from the one used for `csms init`.

## Options

`csms auth login` supports the options common to all `csms` commands and:

**--username *USERNAME***

The user name of the wanted account.

**--password *PASSWORD***

The password of the wanted account.

**--rsa\_token *RSA\_TOKEN***

Provide an RSA authentication token to use. *RSA\_TOKEN*.

## Permissions

- This command requires only unprivileged user permissions, since it only modifies the CSMS CLI configuration for the user that executes it.

## Usage

- This command will prompt for a user name and password if run without any options.

## Restrictions

- This command prompts for a valid user name and password registered with Keycloak to complete successfully.

## Example input

```
csms auth login
```

## Example output

```
Username: USERNAME
Password: PASSWORD
Success!
```

# csms config

## Syntax

```
csms config [OPTIONS] SUBCOMMAND [ARGS]
```

## Description

View and edit `csms` configuration properties.

## Options

The `csms config` commands support options common to all `csms` commands.

## Subcommands

### describe

List all the current configuration values of the active CSMS CLI configuration.

### get *PROPERTY*

Print a specific *PROPERTY* of the active CSMS CLI configuration.

### list

List all saved CSMS CLI configurations.

### set *SECTION VALUES*

Set one or more configuration parameters. Both *SECTION* and *VALUES* are required.

### unset *PROPERTIES*

Clears one or more configuration parameters. *PROPERTIES* is required.

### use *CONFIGURATION*

Select (that is, activate) *CONFIGURATION* as the CSMS CLI configuration to use. *CONFIGURATION* is required.

## Permissions

- This command requires only unprivileged user permissions, since it only modifies the CSMS CLI configuration for the user that executes it.

## Usage

Use the `csms config` commands to manage one or more CSMS CLI configurations.

## Examples

Print the active CSMS CLI configuration.

```
csms$ csms config describe
Your active configuration is: default
[core]
hostname = "https://api.SUBDOMAIN.DOMAIN"

[auth.login]
username = "USERNAME"
```

Print the user name in the active CSMS CLI configuration.

```
csms$ csms config get auth.login.username
USERNAME
```

Set the user name to *USERNAME* in the active CSMS CLI configuration.

```
csms$ csms config set auth.login username=USERNAME
```

Clear the user name in the active CSMS CLI configuration.

```
csms$ csms config unset auth.login.username
```

# csms base software mappings list

## Syntax

```
csms base software mappings list [OPTIONS]
```

## Description

Print a list of deployment mappings.

## Options

`csms base software mappings list` supports the options common to all `csms` commands and:

**--node-group** *NODE\_GROUP*

Print mappings for a specific node group. *NODE\_GROUP* must be `mgmt` in ClusterStor data services release 2.0.

**--release-version** *RELEASE\_VERSION*

Print mappings that use a specific release version.

**--dg-version** *DG\_VERSION*

Print mappings that use a specific Deployment Group (DG) version.

**--dg-name** *DG\_NAME*

Print mappings that use a specific DG name.

**--hostname** *HOSTNAME*

Print mappings for a specific base platform virtual machine (VM) by using the host name of that VM.

**--xname** *XNAME*

Print mappings for a specific node by using the xname of that node.

## Usage

- The most useful forms of this command are:
  - `csms base software mappings list`
  - `csms base software mappings list --dg-name DG_NAME`
  - `csms base software mappings list --release-version RELEASE_VERSION`
  - `csms base software mappings list --dg-version DG_VERSION`

## Examples

Print out all software mappings for the ClusterStor data services cluster.

```
csms$ csms base software mappings list
[[results]]
xname = "x3000c0s10b0n1"
hostname = "k8s-worker1-cdslmo101-u01c"
node_group = "mgmt"
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"

[[results.running_deployment_groups]]
id = "dfe15a31-4897-f368-489c-2bd7a0730e58"
name = "ssm-base-management"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211002003006_fed69b9"

. . .
```

```

[[results]]
xname = "x3000c0s12b0n1"
hostname = "k8s-worker1-cdslmo102-suk6"
node_group = "mgmt"
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"

[[results.running_deployment_groups]]
id = "dfe15a31-4897-f368-489c-2bd7a0730e58"
name = "ssm-base-management"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211002003006_fed69b9"

. . .

[[results]]
xname = "x3000c0s14b0n1"
hostname = "k8s-worker1-cdslmo103-f386"
node_group = "mgmt"
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"

[[results.running_deployment_groups]]
id = "dfe15a31-4897-f368-489c-2bd7a0730e58"
name = "ssm-base-management"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211002003006_fed69b9"

. . .

```

Print out all the mappings for the DG with the name `csms-generic-dg-mgmt`.

```

csms# csms base software mappings list --dg-name csms-generic-dg-mgmt
[[results]]
xname = "x3000c0s10b0n1"
hostname = "k8s-worker1-cdslmo101-u0lc"
node_group = "mgmt"
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"

[[results]]
xname = "x3000c0s12b0n1"
hostname = "k8s-worker1-cdslmo102-suk6"
node_group = "mgmt"

```

```
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"

[[results]]
xname = "x3000c0s14b0n1"
hostname = "k8s-worker1-cdslmo103-f386"
node_group = "mgmt"
previous_deployment_groups = []
target_deployment_groups = []
uncommitted_deployment_groups = []
[[results.running_deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
release_version = "2.0.1-20211006123006_fed69b9"
version = "1.1.0.20211005003007_fed69b9"
```

# csms base software mappings update

## Syntax

```
csms base software mappings update [OPTIONS]
```

## Description

Deploy a different system software release to one or more Deployment Groups (DGs).

## Options

The `csms base software mappings update` command supports the option common to all `csms` commands and:

**--deployment-groups** *DEPLOYMENT\_GROUPS*

This option is required. *DEPLOYMENT\_GROUPS* is either a comma-separated list of deployment group names or `ALL` for updating all existing DGs.

**--release-id** *RELEASE\_ID*

This option is required. *RELEASE\_ID* is the release id for a deployment group.

**--dry-run** *DRY\_RUN*

Test the mapping update without actually deploying new software to DGs. *DRY\_RUN* can be either `true` or `false`.

## Restrictions

- Releases must first be uploaded from SDP files and created with `csms base software releases create` before they can be deployed with this command.

## Usage

- Use this command to update or revert the software deployed to the system.

## Example input

Update or revert the software for the `csms-cds-dg-mgmt` Deployment Group to version provided by the release id `c8365c24-0bc6-8ece-5f8e-99c9f4083ceb`.

```
csms$ csms base software mappings update --release-id c8365c24-0bc6-8ece-5f8e-99c9f4083ceb \  
--deployment-groups csms-cds-dg-mgmt
```

## Example output

A Successful `csms base software mappings update` command returns a `task_id`.

```
task_id = "653655dc-d943-11eb-962e-f29f55d47838"
```

# csms base software releases create

## Syntax

```
csms base software releases create [OPTIONS]
```

## Description

Upload an SDP file to the system and process the file so that the Deployment Groups within the file can then be deployed to the system.

## Options

The `csms base software releases create` command supports the options common to all `csms` commands and:

**`--filename` *FILENAME***

The name of the SDP file to upload, `cds-2.0.0-20210628003004_c104719.sdp` for example.

**`--path` *SDP\_PATH\_AND\_NAME***

The path and name of the SDP file to upload `./cds-2.0.0-20210628003004_c104719.sdp` for example. This option is required.

## Usage

- This command:
  - Uploads the SDP file from the host to the system.
  - Immediately returns a `task_id`.
  - Tells the system to begin extracting the DG artifacts and metadata.

## Example input

Upload the `cds-2.0.0-20210628003004_c104719.sdp` file to the ClusterStor data services cluster from an external host that is running CSMS CLI.

```
csms$ csms base software releases create --path \  
./cds-2.0.0-20210628003004_c104719.sdp
```

## Example output

A successful `csms base software releases create` command returns a `task_id` value.

```
task_id = "653655dc-d943-11eb-962e-f29f55d47838"
```



# csms base software releases describe

## Syntax

```
csms base software releases describe [OPTIONS] RELEASE_ID
```

## Description

Print out information about an uploaded and processed SDP file. This information includes:

- Original SDP file name
- The SDP version number
- Verbal description of the release.
- The following information about all the Deployment Groups (DGs) provided by the release:
  - `id`
  - `name`
  - A common `version` string for all DGs within the release

## Options

The `csms base software releases describe` command supports the options common to all `csms` commands.

## Parameters

**`RELEASE_ID`**

This argument is required and specifies the ID of the release to describe.

## Usage

- This command produces much output. Piping through a pager like `less` is recommended.

## Example input

```
csms base software releases describe 6cde4c2b-837a-393b-ba6a-5d7699417fd5
```

## Example output

```
id = "6cde4c2b-837a-393b-ba6a-5d7699417fd5"
filename = "cds-2.0.1-20211006123006_fed69b9.sdp"
version = "2.0.1-20211006123006_fed69b9"
description = "CDS Release 2.0.1.418 master"
[[deployment_groups]]
id = "f42e2f8c-5012-d4d5-995a-efdb8b63121a"
name = "cs-storage-os"
version = "1.1.0.20211003123006_fed69b9"
href = "/v1/software/deployments/groups/f42e2f8c-5012-d4d5-995a-efdb8b63121a"

[[deployment_groups]]
id = "dfe15a31-4897-f368-489c-2bd7a0730e58"
name = "ssm-base-management"
version = "1.1.0.20211002003006_fed69b9"
href = "/v1/software/deployments/groups/dfe15a31-4897-f368-489c-2bd7a0730e58"

[[deployment_groups]]
id = "d7a69511-32d7-7664-4b65-cd82297fbea3"
name = "ssm-serviceability-management"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/d7a69511-32d7-7664-4b65-cd82297fbea3"

[[deployment_groups]]
id = "26ffbcd6-57de-c2c9-3340-b70346244406"
name = "ssm-hardware-monitoring"
```

```

version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/26ffbcd6-57de-c2c9-3340-b70346244406"

[[deployment_groups]]
id = "345d663c-4747-f1c5-e660-0367b882b8af"
name = "dp-storage-node"
version = "1.0.0.20210909154054_fed69b9"
href = "/v1/software/deployments/groups/345d663c-4747-f1c5-e660-0367b882b8af"

[[deployment_groups]]
id = "88112e6c-255c-8b1d-b4d2-9998f50f13ce"
name = "csms-base-dg-mgmt"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/88112e6c-255c-8b1d-b4d2-9998f50f13ce"

[[deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
version = "1.1.0.20211005003007_fed69b9"
href = "/v1/software/deployments/groups/4aeeadcf-b09e-6c94-3189-9130b2643fc3"

[[deployment_groups]]
id = "5a5656d3-bd60-d3ec-4c96-13c801286389"
name = "csms-cds-dg-mgmt"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/5a5656d3-bd60-d3ec-4c96-13c801286389"

[[deployment_groups]]
id = "2033656a-2399-1183-c6e1-733efcbd9fb4"
name = "cs-data-services"
version = "1.0.0.20211006123006_fed69b9"
href = "/v1/software/deployments/groups/2033656a-2399-1183-c6e1-733efcbd9fb4"

[[deployment_groups]]
id = "74fb9e73-5076-93ec-2133-0892belf4aa7"
name = "cs-data-services-datamover"
version = "1.0.0.20211006123006_fed69b9"
href = "/v1/software/deployments/groups/74fb9e73-5076-93ec-2133-0892belf4aa7"

[[deployment_groups]]
id = "e4d7a781-2d8a-6cf3-6a72-a38e77a67a47"
name = "system-dump-utility-dg"
version = "1.0.1.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/e4d7a781-2d8a-6cf3-6a72-a38e77a67a47"

```

# csms base software releases list

## Syntax

```
csms base software releases list [OPTIONS]
```

## Description

Print out information about all uploaded and processed SDP files available on the system. This information includes:

- Original SDP file name
- The SDP version number
- Verbal description of the release.
- The following information about all the Deployment Groups (DGs) provided by the release:
  - `id`
  - `name`
  - A common `version` string for all DGs within the release

## Options

The `csms base software releases list` command supports the options common to all `csms`.

## Usage

- This command produces much output. Piping through a pager like `less` is recommended.

## Example input

```
csms$ csms base software releases list
```

## Example output

The following output is for a system that has only one SDP file uploaded and processed. The command output in this case is identical to the output of `csms base software releases describe RELEASE_ID`.

```
id = "6cde4c2b-837a-393b-ba6a-5d7699417fd5"
filename = "cds-2.0.1-20211006123006_fed69b9.sdp"
version = "2.0.1-20211006123006_fed69b9"
description = "CDS Release 2.0.1.418 master"
[[deployment_groups]]
id = "f42e2f8c-5012-d4d5-995a-efdb8b63121a"
name = "cs-storage-os"
version = "1.1.0.20211003123006_fed69b9"
href = "/v1/software/deployments/groups/f42e2f8c-5012-d4d5-995a-efdb8b63121a"

[[deployment_groups]]
id = "dfe15a31-4897-f368-489c-2bd7a0730e58"
name = "ssm-base-management"
version = "1.1.0.20211002003006_fed69b9"
href = "/v1/software/deployments/groups/dfe15a31-4897-f368-489c-2bd7a0730e58"

[[deployment_groups]]
id = "d7a69511-32d7-7664-4b65-cd82297fbea3"
name = "ssm-serviceability-management"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/d7a69511-32d7-7664-4b65-cd82297fbea3"

[[deployment_groups]]
id = "26ffbcd6-57de-c2c9-3340-b70346244406"
name = "ssm-hardware-monitoring"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/26ffbcd6-57de-c2c9-3340-b70346244406"
```

```

[[deployment_groups]]
id = "345d663c-4747-f1c5-e660-0367b882b8af"
name = "dp-storage-node"
version = "1.0.0.20210909154054_fed69b9"
href = "/v1/software/deployments/groups/345d663c-4747-f1c5-e660-0367b882b8af"

[[deployment_groups]]
id = "88112e6c-255c-8b1d-b4d2-9998f50f13ce"
name = "csms-base-dg-mgmt"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/88112e6c-255c-8b1d-b4d2-9998f50f13ce"

[[deployment_groups]]
id = "4aeeadcf-b09e-6c94-3189-9130b2643fc3"
name = "csms-generic-dg-mgmt"
version = "1.1.0.20211005003007_fed69b9"
href = "/v1/software/deployments/groups/4aeeadcf-b09e-6c94-3189-9130b2643fc3"

[[deployment_groups]]
id = "5a5656d3-bd60-d3ec-4c96-13c801286389"
name = "csms-cds-dg-mgmt"
version = "1.1.0.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/5a5656d3-bd60-d3ec-4c96-13c801286389"

[[deployment_groups]]
id = "2033656a-2399-1183-c6e1-733efcbd9fb4"
name = "cs-data-services"
version = "1.0.0.20211006123006_fed69b9"
href = "/v1/software/deployments/groups/2033656a-2399-1183-c6e1-733efcbd9fb4"

[[deployment_groups]]
id = "74fb9e73-5076-93ec-2133-0892belf4aa7"
name = "cs-data-services-datamover"
version = "1.0.0.20211006123006_fed69b9"
href = "/v1/software/deployments/groups/74fb9e73-5076-93ec-2133-0892belf4aa7"

[[deployment_groups]]
id = "e4d7a781-2d8a-6cf3-6a72-a38e77a67a47"
name = "system-dump-utility-dg"
version = "1.0.1.20210930003007_fed69b9"
href = "/v1/software/deployments/groups/e4d7a781-2d8a-6cf3-6a72-a38e77a67a47"

```

## csms base status list

### Syntax

```
csms base status [OPTIONS] list
```

### Description

Displays the version numbers for:

- The CSMS API on the system
- The CSMS CLI running on the system

### Options

This command supports the options common to all `csms` commands.

### Usage

- Use this command together with `csms --version` to check that CSMS versions running on the system and the host are compatible.

### Example input

```
csms base status list
```

### Example output

```
csms$ csms base status list
api-version = "1.1.0"
cli-version = "1.2.1"
system-version = ""
```

# csms base tasks list

## Syntax

```
csms base tasks list [OPTIONS]
```

## Description

Print the details of all registered tasks, or only tasks that fit certain criteria.

## Options

This command supports the options common to all `csms` commands and:

**--end-time** *END\_TIME*

Only list tasks that have the specified *END\_TIME* date and time string, such as `2017-07-21T17:32:28`.

**--start-time** *START\_TIME*

Only list tasks that have the specified *START\_TIME* date and time string, such as `2017-07-21T17:32:28`.

**--status** *PENDING | SUCCESS | FAILURE*

Only list tasks that currently match a specific status.

**--task-type** *TASK\_TYPE*

Only list tasks that currently match a specific *task\_type* message string. Examples include

`ssm_software_manager_discovery`, `sdp`, and `ssm_power_storage_management`.

## Usage

This command returns the following information about each task:

- The date and time the task started.
- The date and time the task finished.
- The date and time of the last update for the task.
- A status message from the system.
- The current progress, in percent.
- The current state of the task.
- The current status of the task: `PENDING`, `SUCCESS`, or `FAILURE`
- The task id.
- The task type.

This command will return much output if none of the filtering options are used.

## Example input

```
csms$ csms base task list
```

## Example output

```
[[results]]
end_time = "2021-10-08T10:03:08CDT"
last_update_time = "2021-10-08T10:03:08CDT"
message = "Node x3000c0s25b0n0 has been discovered and initialized. Ready for software deployment."
progress = "100 %"
start_time = "2021-10-08T08:39:39CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "2a0eec08-283d-11ec-9da6-3e48ed5dbedb"
task_type = "shm-discovery-monitor"
warnings = []

[[results]]
end_time = "2021-10-08T09:13:43CDT"
```

```

last_update_time = "2021-10-08T09:13:43CDT"
message = "Completed reboot on x3000c0s25b0n0"
progress = "100 %"
start_time = "2021-10-08T09:03:23CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "7fa30c16-2840-11ec-9799-92c2f2fa23f6"
task_type = "ssm_power_storage_management"
warnings = []

[[results]]
end_time = "2021-10-08T10:49:18CDT"
last_update_time = "2021-10-08T10:49:19CDT"
message = "Deployment mapping commit complete"
progress = "100 %"
start_time = "2021-10-08T10:36:26CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "7fa95087-284d-11ec-b5a4-3eac39f82f12"
task_type = "ssm_software_manager_commit"
warnings = []

[[results]]
end_time = "2021-10-06T21:17:31CDT"
last_update_time = "2021-10-06T21:17:31CDT"
message = "Deployment mapping commit complete"
progress = "100 %"
start_time = "2021-10-06T21:09:41CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "a1c8ee14-2713-11ec-b56f-024504761791"
task_type = "ssm_software_manager_commit"
warnings = []

[[results]]
end_time = "2021-10-06T21:09:11CDT"
last_update_time = "2021-10-06T21:09:11CDT"
message = "Processing Software Deployment Package complete"
progress = "100 %"
start_time = "2021-10-06T20:55:30CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "a6583dae-2711-11ec-97c1-024504761791"
task_type = "sdp"
warnings = []

[[results]]
end_time = "2021-10-08T10:48:46CDT"
last_update_time = "2021-10-08T10:48:46CDT"
message = "Completed reboot on x3000c0s25b0n0"
progress = "100 %"
start_time = "2021-10-08T10:37:34CDT"
state = "COMPLETED"
status = "SUCCESS"
task_id = "a844177b-284d-11ec-83cb-92c2f2fa23f6"
task_type = "ssm_power_storage_management"
warnings = []

[[results]]
end_time = "2021-10-08T09:14:10CDT"
last_update_time = "2021-10-08T09:14:10CDT"
message = "Deployment mapping commit complete"

```

```
progress = "100 %"  
start_time = "2021-10-08T08:57:47CDT"  
state = "COMPLETED"  
status = "SUCCESS"  
task_id = "b7bf7dca-283f-11ec-abd3-3eac39f82f12"  
task_type = "ssm_software_manager_discovery"  
warnings = []
```



## csms base tasks describe

### Syntax

```
csms base tasks describe [OPTIONS] TASK_ID
```

### Description

Reports the details of a single CSMS task.

### Options

This command supports the options common to all `csms` commands.

### Parameters

**TASK\_ID**

This is the id of the task to retrieve details for. This argument is required.

### Usage

This command returns the following information about the specified task:

- The date and time the task started.
- The date and time the task finished.
- The date and time of the last update for the task.
- A status message from the system.
- The current progress, in percent.
- The current state of the task.
- The current status of the task: `PENDING`, `SUCCESS`, or `FAILURE`
- The task id.
- The task type.

### Example input

```
csms base tasks describe 653655dc-d943-11eb-962e-f29f55d47838
```

### Example output

```
end_time = "2021-06-03 13:39:29"
last_update_time = "2021-06-30 13:39:29"
message = "Deployment mapping commit complete"
progress = "100 %"
start_time = "2021-06-30 13:35:48"
state = "COMPLETED"
status = "SUCCESS"
task_id = "653655dc-d943-11eb-962e-f29f55d47838"
task_type = "ssm_software_manager_commit"
warnings = []
```

## csms inventory status list

### Syntax

```
csms inventory status list [OPTIONS]
```

### Description

Prints a status message from the server.

### Options

This command supports the options common to all `csms` commands.

### Usage

In the current release, this command returns only the CSMS API version of the system.

### Example input

```
csms inventory status list
```

### Example output

```
api-version = "1.0.0"  
system-version = ""
```

# csms inventory system summary list

## Syntax

```
csms inventory system summary list [OPTIONS]
```

## Description

Provide system information.

## Options

This command supports the options common to all `csms` commands.

## Usage

- This command returns the following system-level information:
  - The current system and local time
  - The timezone
  - Current time-stamp in Unix epoch time format
  - The identifier and serial number

## Example input

```
csms inventory system summary list
```

## Example output

```
local_time = "Thu, 14 Oct 2021 21:15:58 UTC"
system_identifier = "[not-set]"
system_serial_number = "[not-set]"
system_time = "Thu, 14 Oct 2021 21:15:58"
system_timezone = "UTC"
utc_timestamp = 1634246158
```

# csms inventory system summary update

## Syntax

```
csms inventory system summary update [OPTIONS]
```

## Description

Update system info.

## Options

In addition to the options common to all `csms` commands, this command also supports these options:

**--value *VALUE***

The value you want to change. This option is required.

**--op *OP***

The operation to perform. The current release supports only the `replace` option. This option is required.

**--path *PATH***

The JSON Pointer path to the object. This option is required.

## Usage

- Use this command to set or update:
  - The system local time
  - The identifier (name) of the system
  - The system time and timezone

## Example

This example first shows a blank `system_identifier` field, followed a `csms inventory system summary update` command that sets that value to `kj1234`.

```
csms inventory system summary list
local_time = "Fri, 15 Oct 2021 16:01:49 UTC"
system_identifier = "[not-set]"
system_serial_number = "[not-set]"
system_time = "Fri, 15 Oct 2021 16:01:49"
system_timezone = "UTC"
utc_timestamp = 1634313709
csms inventory system summary update --value kj1234 --op \ replace --configuration default --path /system_identifier
local_time = "Fri, 15 Oct 2021 16:02:34 UTC"
system_identifier = "kj1234"
system_serial_number = "[not-set]"
system_time = "Fri, 15 Oct 2021 16:02:34"
system_timezone = "UTC"
utc_timestamp = 1634313754
```

# csms cds requests create

## Syntax

```
csms cds requests create [OPTIONS]
```

## Description

Create a data movement request for ClusterStor data services to execute.

## Options

This command supports the options common to all `csms` commands and:

**--action *ACTION***

The data movement action to take on the source file to produce the target file. *ACTION* can be `migrate` or `mirror`. This option is required.

**--target-fname *TEMPLATE***

The full absolute path of the migration or mirroring template.

**--target-fsid *TARGET\_FSID***

The name of the target file system or archive. For `migrate` and `mirror` actions, specify the same file system name as for `--source-fsid`.

**--source-fid *SOURCE\_FID***

A Lustre `fid` string for the source file system.

**--source-fsid *SOURCE\_FSID***

The name of the source file system or archive. For `migrate` and `mirror` actions, specify the same file system name as for `--target-fsid`.

## Usage

- ClusterStor data services will queue the request and reply after the request is stored. That reply contains the `requestId` number of the request for later tracking. The request may not be scheduled for servicing until some later time.

## Example input

```
csms cds requests create --action migrate \  
--source-fid [0x200004abe:0xe518:0x0] \  
--source-fsid cls12345 --target-fname TEMPLATE \  
--target-fsid cls12345 --format json
```

## Example output

```
{  
  "requestId": "23",  
  "requestState": {  
    "message": "request received",  
    "state": "PENDING",  
    "time": "2021-09-02T07:36:08.715804342Z"  
  },  
  "traces": [  
    {  
      "message": "request received",  
      "state": "PENDING",  
      "time": "2021-09-02T07:36:08.715804342Z"  
    }  
  ]  
}
```

## More information

[Request File Data Movement with CSMS CLI](#)

## csms cds requests delete

### Syntax

```
csms cds requests delete [OPTIONS] ID
```

### Description

Request to cancel a ClusterStor data services data movement request.

### Options

This command supports the options common to all `csms` commands.

### Arguments

**ID**

The request ID number of the request to cancel. This argument is required.

### Usage

ClusterStor data services provides no guarantee of data movement cancellation.

### Example input

```
csms cds requests delete 27
```

### Example output

# csms cds requests describe

## Syntax

```
csms cds requests describe [OPTIONS] ID
```

## Description

Print details about a ClusterStor data services data movement request.

## Options

This command supports the options common to all `csms` commands.

## Arguments

**ID**

The ID number of the request. This argument is required.

## Usage

This command returns `traces` and `requestState` objects that list:

- A message
- A time-stamp
- The state of the data movement request at that time of the time-stamp

## Example input

```
csms cds requests describe 10
```

## Example output

```
requestId = "10"
[[traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T00:37:23.080794345Z"

[[traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T00:37:23.516Z"

[[traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T00:37:24Z"

[[traces]]
message = "(null)"
state = "COMPLETED"
time = "2021-10-14T00:52:56Z"

[requestState]
message = "(null)"
state = "COMPLETED"
time = "2021-10-14T00:52:56Z"
```

# csms cds requests list

## Syntax

```
csms cds requests list [OPTIONS]
```

## Description

Return the details of all ClusterStor data services data movement requests currently stored in the request database.

## Options

In addition to the options supported by all `csms` commands, this command also supports:

`--ruid REQUESTOR_ID`

Display only the results issued by the requestor with the specified *REQUESTOR\_ID*.

`--state STATE`

Display only the results that are currently in the specified *STATE*, where *STATE* is either:

- `PENDING`
- `QUEUED`
- `RUNNING`
- `COMPLETED`
- `CANCELING`
- `CANCELED`
- `ERROR`
- `FAILED`

## Usage

On an active, long-running system the output of this command can be large. Hewlett Packard Enterprise recommends filtering the results by using the `--state` option. This command returns `traces` and `requestState` objects that list:

- A message
- A time stamp
- The state of the data movement request at that time of the time stamp

## Example input

```
csms cds requests list --state ERROR
```

## Example output

```
[[results]]
requestId = "12"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T01:23:10.308564154Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T01:23:11.893Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T01:23:11Z"
```



```

[[results.traces]]
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T01:23:12Z"

[results.requestState]
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T01:23:12Z"
[[results]]
requestId = "16"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T02:53:09.991784085Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T02:53:11.014Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T02:53:11Z"

[[results.traces]]
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T02:53:11Z"

[results.requestState]
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T02:53:11Z"
[[results]]
requestId = "18"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T03:11:20.617637160Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T03:11:24.477Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T03:11:24Z"

[[results.traces]]
message = "Re-queue the request after Connector restart"
state = "QUEUED"
time = "2021-10-14T03:13:15.053Z"

[[results.traces]]
message = "File exists"
state = "ERROR"

```

```

time = "2021-10-14T03:13:15.213Z"

[results.requestState]
message = "File exists"
state = "ERROR"
time = "2021-10-14T03:13:15.213Z"
[[results]]
requestId = "19"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T03:21:29.897423003Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T03:21:31.701Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T03:21:31Z"

[[results.traces]]
message = "Re-queue the request after Connector restart"
state = "QUEUED"
time = "2021-10-14T03:24:17.974Z"

[[results.traces]]
message = "File exists"
state = "ERROR"
time = "2021-10-14T03:24:17.997Z"

[results.requestState]
message = "File exists"
state = "ERROR"
time = "2021-10-14T03:24:17.997Z"
[[results]]
requestId = "20"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T16:28:47.331443719Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T16:28:51.222Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T16:28:50Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T16:36:27Z"

[[results.traces]]

```

```
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T17:00:14Z"

[results.requestState]
message = "Invalid argument"
state = "ERROR"
time = "2021-10-14T17:00:14Z"
[[results]]
requestId = "21"
[[results.traces]]
message = "request received"
state = "PENDING"
time = "2021-10-14T23:43:36.434808120Z"

[[results.traces]]
message = "Queue the request"
state = "QUEUED"
time = "2021-10-14T23:43:41.312Z"

[[results.traces]]
message = "(null)"
state = "RUNNING"
time = "2021-10-14T23:43:40Z"

[[results.traces]]
message = "Input/output error"
state = "ERROR"
time = "2021-10-14T23:52:16Z"

[results.requestState]
message = "Input/output error"
state = "ERROR"
time = "2021-10-14T23:52:16Z"
```

## csms cds status list

### Syntax

```
csms cds status list [OPTIONS]
```

### Description

Obtain the overall status of ClusterStor data services.

### Options

This command supports the options common to all `csms` commands.

### Usage

- This command only verifies that the ClusterStor data services API Agent is able to receive and respond to external calls. This command is not a substitute for Kibana, Grafana, `kubect1`, or other ClusterStor data services monitoring and troubleshooting tools.

### Example input

```
csms cds status list
```

### Example output

```
status = "ok"
```

## csms cds version list

### Syntax

```
csms cds version list [OPTIONS]
```

### Description

Print the CSMS version of the system.

### Options

This command supports the options common to all `csms` commands.

### Usage

This command returns the full version string of the CSMS API on the system.

### Example input

```
csms cds version list
```

### Example output

```
version = "v1.0.3-30-50d43d7"
```