# Cloudian HyperStore Administration Guide

## Version 7.5.2

This page left intentionally blank.

This page left intentionally blank.

# Contents

# What's New in HyperStore Version 7.5.x

This section introduces the main new features and enhancements in the Cloudian HyperStore 7.5 release and subsequent 7.5.x releases. In the lists below, all items were introduced in the 7.5 release unless marked with a parenthetical note indicating an 7.5.x release, such as "(7.5.2)"

**Note** For more granular release details including bug fixes and configuration setting changes please see the release notes.

## APIs -- New Features and Enhancements

*Admin API and IAM API calls to support Multi-Factor Authentication*

The CMC now supports multi-factor authentication (MFA). (For additional high-level information see the CMC MFA item in **"System Behavior and Management -- New Features and Enhancements"** (page 17)). To facilitate this new CMC capability, HyperStore now supports additional IAM API methods and Admin API methods. The Admin API methods are needed because the IAM methods are insufficient to allow MFA for HyperStore account root users.

Newly supported IAM API methods:

- CreateVirtualMFADevice
- DeactivateMFADevice
- DeleteVirtualMFADevice
- EnableMFADevice
- ListMFADevices
- ListVirtualMFADevices
- ResyncMFADevice

Newly supported Admin API methods:

- GET /user/mfa/list
- POST /user/mfa/deactivateDevice
- POST /user/mfa/enableDevice
- POST /user/mfa/resyncDevice
- POST /user/mfa/verify
- POST /user/mfa/createDevice (7.5.1)
- POST /user/mfa/deleteDevice (7.5.1)

Modified Admin API method:

- GET /user/list (now includes an "extended" parameter)

More information:

- The "IAM API" section of the *Cloudian HyperStore AWS APIs Support Reference*
- The "user" section of the *Cloudian HyperStore Admin API Reference*

*S3 Service now supports additional bucket policy statement elements*

The S3 Service's implementation of the S3 *PutBucketPolicy* API operation now supports the policy statement elements *NotPrincipal* and *NotAction*.

More information:

- *PutBucketPolicy* in the "S3 API" section of the *Cloudian HyperStore AWS APIs Support Reference*

*S3 Service now restricts "Version" for new bucket policies and IAM policies*

The "Version" in a bucket policy or an IAM policy is now required to be "2012-10-17" (as opposed to older AWS policy versions such as "2008-10-17"). This restriction is applied only to newly created policies. It does not apply to policies created before the upgrade to HyperStore 7.5.

*CMC's S3 client now supports bucket ownership controls*

The CMC's built-in S3 client now supports the *{Put,Get,Delete}BucketOwnershipControls* S3 API calls. In HyperStore 7.4 the S3 Service supported these API calls (if submitted by a third party S3 client) but the CMC's S3 client did not.

> **Note** For both the S3 Service and the CMC's built-in S3 client, the only bucket ownership types currently supported are "BucketOwnerPreferred" and "ObjectWriter". The "BucketOwnerEnforced" ownership type -- which is supported by AWS -- is not yet supported by HyperStore.

More information:

- While on the CMC's **Bucket -> Properties** page, click **Help**

*S3 Service and CMC's S3 client now support object ownership type 'BucketOwnerEnforced' (7.5.1)*

The HyperStore S3 Service now supports the 'BucketOwnerEnforced' object ownership type, in the S3 API calls *CreateBucket* and *PutBucketOwnerhipControls*. Previously only the 'BucketOwnerPreferred' and 'ObjectWriter' ownership types were supported.

The CMC also now supports the 'BucketOwnerEnforced' object ownership type in the bucket permissions interface.

More information:

- "CreateBucket" and "PutBucketOwnerhipControls" in the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***
- **Buckets & Objects -> Bucket Properties -> Configure Bucket Permissions** in the CMC online Help

*Admin API and CMC now support warning levels for user and group storage quotas (7.5.1)*

HyperStore's Quality of Service (QoS) feature now supports configurable warning levels ('soft limits') for number of KiBs stored and number of objects stored per user or group. Previously these QoS metrics only supported maximum levels ('hard limits') at which further upload requests would be denied.

Also, you now have the option of configuring the system so that alerts are generated if users or groups exceed warning levels for KiBs stored or number of objects stored or other QoS metrics such as request rate or bytes transfer rate. Previously if users or groups exceeded QoS warning levels this resulted only in the generation of log messages.

More information:

- **"Quality of Service (QoS) Feature Overview"** (page 213)
- **"Preparing the QoS Feature"** (page 215)
- **Users & Groups -> Set Quality of Service (QoS) Limits** in the CMC online Help
- "qos" section in the ***Cloudian HyperStore Admin API Reference***

*S3 Service and CMC's S3 client now support object size filtering for bucket lifecycles (7.5.1)*

The HyperStore S3 Service now supports object size filters in the S3 API call *PutBucketLifcycleCnfiguration*.

The CMC also now supports object size filters in the bucket lifecycle configuration interface.

This feature gives you the ability to configure bucket lifecycle rules that apply only to objects that are larger than a specified size, or only to objects that are smaller than a specified size, or only to objects that are within a specified size range.

More information:

- "PutBucketLifecycleConfiguration" and "GetBucketLifecycleConfiguration" in the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***
- **Buckets & Objects -> Bucket Properties -> Configure a Bucket Lifecycle Policy** in the CMC online Help

*IAM Service now supports 'SimulatePrincipalPolicy' call (7.5.1)*

The HyperStore IAM Service now supports the IAM API call *SimulatePrincipalPolicy*.

The CMC's IAM client does not support this call, so to utilize this API you must use a third party IAM client.

More information:

- "SimulatePrincipalPolicy" in the IAM API section of the ***Cloudian HyperStore AWS APIs Support Reference***

*S3 Service now supports condition key 's3:x-amz-content-sha256' in bucket policies (7.5.1)*

The HyperStore S3 Service's existing support for the S3 *PutBucketPolicy* call has been enhanced to include support for the condition key 's3:x-amz-content-sha256'. To deny access to a bucket via pre-signed URLs you can create a Deny policy statement in which the condition denied is "s3:x-amz-content-sha256": "UNSIGNED-PAYLOAD".

The CMC's S3 client does not support the *PutBucketPolicy* call, so to utilize this API you must use a third party S3 client.

More information:

- "PutBucketPolicy" in the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***

# System Behavior and Management -- New Features and Enhancements

*HyperStore Single-Node System*

HyperStore is now available as an appliance-based Single-Node System, designed to deliver S3 compliant object storage for AWS Outpost deployments, or for other small business or branch office environments.

More information:

- If you do not have a HyperStore Single-Node System and want to learn more, contact your Cloudian sales representative.
- If you have a HyperStore Single-Node System, your HyperStore documentation set is oriented toward Single-Node operation.

*CMC now supports Multi-Factor Authentication for user logins*

CMC users can now enable multi-factor authentication (MFA) for their own CMC account. If MFA is enabled for a user's account, when logging into the CMC the user must provide not only their user name and password but also a valid MFA code. All user types -- system admins, group admins, and regular users -- can enable MFA on their own account if they wish.

Users who enable MFA on their account can subsequently disable MFA if they no longer wish to use it.

System admins cannot enable MFA on other users' accounts. However, system admins can disable MFA on a user account for which it has been enabled by the user (which may be necessary if for example the user is unable to log into the CMC because of a problem with their MFA application).

More information:

- For enabling MFA on one's own account: While on the CMC's **Security Credentials** page, click **Help**
- For disabling MFA for a user: While on the CMC's **Manage Users** page, click **Help**

*HyperStore Shell now supports sftp access*

You can now use an *sftp* client to upload or download a file to a HyperStore node via the HyperStore Shell (HSH), if the file is one that the HSH is permitted to manage.

More information:

- **"Using the HSH"** (page 101)

*hsstool ls command now returns counts of recent disk errors*

The command *hsstool ls* -- which shows disk usage and storage token assignments for each disk on a node -- has been extended to also show counts of recent disk errors, if any.

More information:

- **"hsstool ls"** (page 515)

*hsstool whereis command now shows expected location of missing replicas or fragments*

The command *hsstool whereis* -- which shows where the replicas or erasure coded fragments of an object are located across the cluster -- has been extended to show not just the locations where replicas or erasure coded fragments are found, but also any locations where replicas or fragments are expected to be but are missing.

More information:

- **"hsstool whereis"** (page 575)

*New FIPS module to replace sunsetted default FIPS module*

The FIPS module that HyperStore uses by default has entered "sunset" status, so if you need your HyperStore system to be strictly compliant with Federal Information Processing Standard (FIPS) Publication 140-2 you must enable FIP compliance by making a HyperStore configuration change.

More information:

- FIPS Support

*Support for object metadata Search (7.5.1)*

HyperStore now supports the ability to search for objects based on the objects' metadata. This metadata search capability is provided by the HyperStore Search Service, an optional HyperStore system component. The HyperStore Search Service is designed to run on "auxiliary" nodes that are closely integrated with your HyperStore cluster, but that have lesser resource requirements than the rest of your HyperStore nodes. Once you've installed the HyperStore Search Service, you can enable metadata search on a per storage policy basis. Users who create buckets that use a search-enabled storage policy can use the CMC's new **Search** page to search for objects based on system-defined object metadata (such as object size or creation date) or user-defined object metadata.

Note that:

- In the current release the HyperStore Search Service only supports finding objects by reference to their metadata (metadata search). It does not support searching through the content of objects (content search).
- The CMC does not support the creation of user-defined object metadata. If users wish to attach user-defined metadata to their objects they must use a third party S3 client application. Users can then use the CMC to search for objects based on that metadata.

More information:

- **"Object Metadata and Search Feature Overview"** (page 196)
- **"Preparing the Metadata Search Feature"** (page 202)

**IMPORTANT !** If you have a legacy integration with Elasticsearch from an earlier version of HyperStore, after you upgrade to HyperStore 7.5.1 your integration with Elasticsearch will no longer work as is. For information about your options see **"Options if You Have a Legacy Integration with Elasticsearch"** (page 198).

*Support for File Services (7.5.1)*

HyperStore now supports SMB and NFS based File Services. This capability is provided by the HyperStore File Service, an optional HyperStore system component. The HyperStore File Service is designed to run on "auxiliary" nodes that are closely integrated with your HyperStore cluster, but that have lesser resource requirements than the rest of your HyperStore nodes. Once you've installed the HyperStore File Service you can use the CMC to configure and monitor this service component, and to choose users for whom to enable File Services functionality. Those users can then use the CMC to create and manage SMB and/or NFS file shares. The HyperStore File Service provides front end SMB and NFS access to these shares as well as a caching layer to speed performance, while the users' HyperStore buckets provide back end archival storage for the shares.

More information:

- **"File Services Feature Overview"** (page 207)
- **"Preparing the File Services Feature"** (page 208)

*Support for Hybrid Storage Policies that use EC for large objects and replication for small objects (7.5.1)*

When creating a new storage policy that uses an erasure coding data distribution scheme, you can now choose an object size threshold such that objects larger than the threshold will be erasure coded and objects at or smaller than the threshold will be protected by replication rather than erasure coding. By default the threshold is 200KiB.

This new Hybrid Storage Policy feature allows for a "best of both" storage strategy, given that key data storage objectives are best served by erasure coding for large objects and by replication for small objects.

More information:

- While on the CMC's **Storage Policies** page, click **Help** and then view the "Add a Storage Policy" topic

**Note** This feature is supported only for new storage policies created in HyperStore 7.5.1 or later.

*Support for KMIP server-side encryption (7.5.1)*

HyperStore's server-side encryption (SSE) feature has been extended to include support for the Key Management Interoperability Protocol (KMIP). You can now have HyperStore interface with a KMIP compliant third party Key Management System (KMS) when implementing server-side encryption.

**Note** Do not start using the KMIP SSE feature until the upgrade has completed for all of your HyperStore nodes.

More information:

- **"Server-Side Encryption"** (page 150)
- **"Using SSE-KMIP"** (page 155)

*Enhancements to CMC Multi-Factor Authentication feature (7.5.1)*

The CMC Multi-Factor Authentication (MFA) feature, introduced in HyperStore 7.5.0, has been enhanced in these ways in version 7.5.1:

- The CMC MFA feature no longer relies on the HyperStore IAM service. If you've disabled the IAM service in your system, the CMC MFA feature will still work because it now exclusively uses Admin API calls.
- You now have an option to **require** all CMC users to set up and use MFA. If you turn on this configuration option, this requirement will apply to system administrators as well as to group administrators and regular users.
- If you have the IAM service enabled in your system (as it is by default), the CMC's **IAM** section now includes a **Manage MFA Devices** page that lists all MFA devices under the root account of the logged-in CMC user. This includes any devices set up by and attached to IAM users by third party IAM clients calling the HyperStore IAM API, as well as devices enabled for the root account user (which the root account user can set up through the CMC, which calls the Admin API). This page also supports disabling and deleting MFA devices.
- Results of multi-factor authentication operations from user logins to the CMC are now recorded in the CMC user audit log *ui-actions.log*.

More information:

- **"Setting Up User Password and Credential Controls"** (page 165)
- common.csv: **"mfa_enforced"** (page 408)
- While on the CMC's **Security Credentials** page, click **Help**.
- While on the CMC's **Manage MFA Devices** page, click **Help**.
- **"CMC Logs"** (page 352)

**Note** In the very unlikely event that a CMC user in HyperStore 7.5.0 created two MFA devices with the same name but with different case (for example "abc" and "Abc"), during your upgrade to HyperStore 7.5.1 that user's devices will be deleted and then the user will be able to log in to the CMC without MFA (and they will need to set up MFA again if they want to use MFA). This is because starting in HyperStore 7.5.1, device names are processed as case-insensitive (for consistency with AWS). The CMC does not allow account root users to create multiple MFA device for themselves, so this "user has two MFA devices from 7.5.0 with same name but different cases" scenario could not happen unless in 7.5.0 you allowed CMC users to create MFA devices directly through the IAM API.

*Support for concurrent disk rebuilds in EC-only environments (7.5.1)*

You can now rebuild disks on multiple nodes concurrently, if there is only erasure coded data in your system. Rebuilding multiple disks concurrently is **not** supported in either of these circumstances:

- You have replication storage policies in your system (storage policies that use object replication rather than erasure coding). Note that in a multi-DC environment, "Replicated EC" is considered an erasure coding storage policy not a replication storage policy.
- You are using the new "hybrid" storage policies introduced in HyperStore 7.5.1 (which use replication for small objects and erasure coding for large objects).

Also, you cannot rebuild multiple disks on the same node concurrently. You can only rebuild multiple disks concurrently if the disks are on different nodes.

More information:

- While on the CMC's **Nodes -- Advanced** page, click **Help**. Then choose "Disk Management Comments" and then "replaceDisk".
- **"hsstool repairec"** (page 555)

*Cross-system replication now replicates canned ACLs if destination is HyperStore (7.5.1)*

The cross-system replication feature now includes replication of the following types of object ACLs, if the replication destination is an external Cloudian HyperStore system:

- Canned ACLs
- Permissions granted to S3 predefined groups

Cross-system replication does not support object ACL replication if the destination is any type of system other than Cloudian HyperStore.

More information:

- **"Cross-System Replication"** (page 182)

*Improved tracking of object data deletions (7.5.1)*

Two enhancements help to provide more information about the size of deleted object data:

- Entries in the S3 request log *cloudian-request-info.log* now include a field that shows the size of the object data deleted, if the S3 operation was a Delete.
- In the operation status metrics for *hsstool cleanup* or *hsstool cleanupec* operations, a new metric "total size of cleanup objects" has been added to indicate how much object data has been deleted by the operation in total.

More information:

- **"S3 Service Logs (including Auto-Tiering, CRR, and WORM)"** (page 365)
- **"hsstool cleanup"** (page 504)
- **"hsstool cleanupec"** (page 509)
- **"hsstool opstatus"** (page 522)

*Alert codes for additional alert types (7.5.1)*

Alphanumeric alert codes -- to aid in identifying an alert -- are now included in 'service down or unreachable' alerts and in alerts based on the crossing of thresholds defined in your alert rules (such as disk space or CPU utilization). Previously alert codes were only included in alerts based on application log events.

More information:

- While on the CMC's **Alert Rules** page, click **Help**

*Optional suppression of duplicate alerts between the CMC and HyperIQ (7.5.1)*

If you have Cloudian HyperIQ, HyperStore now supports an option to suppress CMC coverage of alerts that are covered by HyperIQ, so that you do not receive duplicate alerts from the CMC and HyperIQ about the same system events or conditions.

More information:

- While on the CMC's **Configuration Settings** page, click **Help**, then view the **Alert Settings** section of the Help.

**Note** Cloudian HyperIQ is a solution for dynamic visualization and analysis of HyperStore system monitoring data and S3 service usage data. HyperIQ is a separate product available from Cloudian that deploys as virtual appliance on VMware or VirtualBox and integrates with your existing HyperStore system. For more information about HyperIQ contact your Cloudian representative.

*Automatic restart of a crashed service (7.5.1)*

Optionally, you can now configure the system so that a service that crashes on a node is automatically restarted on that node. This feature is disabled by default. If you enable the feature it applies only to the S3 Service, Admin Service, HyperStore Service, and Cassandra Service (Metadata DB).

More information:

- common.csv: **"service_restart_on_failure"** (page 391)

*Improvements to hsstool rebalance (7.5.1)*

The *hsstool rebalance* operation -- for redistributing a portion of existing cluster data on to a newly added node -- has been improved in several ways:

- The work entailed by the operation has been re-organized in a more efficient way, to make the operation somewhat faster. (However rebalancing data to a new node remains a long-running operation).
- The command now returns one integrated set of *rebalance* operation status metrics that encompasses rebalancing activity for both replicated data and erasure coded data. Previously the *hsstool opstatus rebalance* response was partitioned into separate *rebalance* and *rebalanceec* status metrics.
- The algorithm for calculating the 'progress percentage' and 'time remaining' metrics returned by the *hsstool opstatus rebalance* command has been revised to improve the accuracy of those metrics.

- The *rebalance* command's former *-pause* option has been renamed to *-stop*, for consistency with the corresponding option for the repair and cleanup commands.
- The *rebalance* command now supports a *-resume* option, for resuming a rebalance operation that you previously stopped.

More information:

- **"hsstool rebalance"** (page 543)
- **"hsstool opstatus"** (page 522)

*Option to allow CMC users to purge the contents of a bucket they want to delete (7.5.1)*

There is now a configuration setting that, if enabled, allows users of the CMC's **Buckets** page to initiate a single-operation purge of all the contents of a bucket that they want to delete. By default this setting is disabled, so users must use the CMC's **Objects** page (or a third party S3 application) to delete all the objects in a bucket before they can delete the bucket in the **Buckets** page. This is the legacy behavior.

The new option makes it easier for users to delete a bucket that has objects in it, especially if there are very many objects in it.

More information:

- common.csv: **"cmc_purgebucket_enabled"** (page 426)
- While on the CMC's **Buckets** page, click **Help**, then view the **Delete Bucket** section of the Help.

**Note** The CMC does not allow users to purge a bucket that has Object Lock enabled.

*HyperStore Shell multi-command support (7.5.1)*

The HyperStore Shell now supports running multiple commands on one line, including the ability to pipe the output of one command to a second command.

**Note** Because the HyperStore Shell now supports command piping, the *hslog <logfile>* command no longer automatically applies *less* to the command output. Instead you can pipe its output to *less* or another command such as *grep*.

More information:

- **"Using the HSH"** (page 101)
- **"Using the HSH to View Logs"** (page 374)

*HyperStore Shell now supports the 'w' command (7.5.1)*

The HyperStore Shell now supports the Linux command 'w', which shows information about who is logged in and what they are currently doing. This command also reports the system uptime.

More information:

- **"Using the HSH"** (page 101)

*HyperStore Shell idle timeout (7.5.1)*

The HyperStore Shell now supports a configurable idle timeout. By default the timeout is 30 minutes.

More information:

- **"Configurable HSH Idle Timeout"** (page 100)

*Reduced scope of cron job that retries cross-region replication tasks (7.5.1)*

In prior versions of HyperStore, the cron job that retries cross-region replication tasks (object replication attempts that encountered temporary errors when first tried and are queued for retry) used to also do a complete sync-up of destination bucket to source bucket, so that any objects in the source bucket but not in the destination bucket would be replicated to the destination -- including older objects. This sync-up process was performed for each source bucket configured for cross-replication. Starting in HyperStore 7.5.1, the cron job only processes cross-replication task retries, without doing a complete sync-up of destination bucket to source bucket. This makes the cron job less resource intensive. A configuration setting lets you switch back to the old behavior if you wish.

More information:

- **"Setting the Scope of the Replication Retry Cron Job"** (page 185)

*All storage policy Workload Types now dynamically detect workload changes (7.5.1)*

All storage policy Workload Types now have the ability to detect a change in the type(s) of workload being written to a bucket, and to adapt the bucket's metadata storage scheme to the detected workload type(s). Previously only the "DEFAULT" Workload Type had this dynamic workload detection capability.

**Note** This enhancement applies only to **new buckets** created after you've upgraded to HyperStore 7.5.1.

More information:

- While on the CMC's **Storage Policies** page, click **Help**

*Support for more Veeam Workload Types when configuring storage policies (7.5.1)*

VEEAM12 (for Veeam version 12) and VEEAMO365 (for Veeam -- Microsoft Office 365) have been added to the list of Workload Types that you can choose when you are configuring new storage policies.

More information:

- While on the CMC's **Storage Policies** page, click **Help**

**Note** If are upgrading to HyperStore 7.5.1 from an earlier version, and in your existing HyperStore system you have buckets that use a storage policy that's configured for the "VEEAM" Workload Type (for VEEAM11), those **existing VEEAM buckets will not automatically adjust to the VEEAM12 data structure** if you upgrade your VEEAM application to VEEAM12. If you have existing VEEAM buckets that you want to convert to VEEAM12, contact Cloudian Support for assistance.

Note further that:
* If you have existing buckets that use a storage policy that's configured for the "DEFAULT" Workload Type, those bucket will automatically detect and adjust to a VEEAM12 workload, if you use those buckets as targets for a VEEAM12 application.
* From HyperStore 7.5.1 and forward, if you create **new** buckets that use storage policy that's configured for the "VEEAM" Workload Type (for VEEAM11), and then you later upgrade your VEEAM application to VEEAM12, those buckets will automatically adapt to the VEEAM12 workload.

*Support for the Veeam Smart Object Storage API (SOSAPI) (7.5.1)*

HyperStore now supports the Veeam Smart Object Storage API (SOSAPI), which provides a mechanism for reporting bucket usage information to Veeam users. By default, the settings for enabling and configuring this functionality do not appear in HyperStore configuration file templates. If your HyperStore system is used for Veeam workloads and you want to enable HyperStore's support for the Veeam SOSAPI, contact Cloudian Support for guidance on adding the applicable settings to your HyperStore configuration.

**Note** This functionality is also available in HyperStore version 7.4.2.5 and later 7.4.x versions.

*Enhancement to repair-on-read feature under Replicated EC (7.5.1)*

In a multi-DC environment if Replicated EC is being used and the read consistency requirement is LOCAL_QUORUM, in the event that a read request fails due to an insufficient number of object fragments in the local DC, the system will asynchronously repair the local copy of the object if the missing fragments can be streamed in from the other DC.

More information:

- Repair-On-Read

*Automatic log collection for Support cases (7.5.1)*

HyperStore now supports an option to automatically upload to Cloudian Support the node diagnostics logs that Cloudian Support needs to resolve support cases that you open. If you enable this feature, you will not need to collect and upload logs to Cloudian Support yourself after you open support cases. By default configuration this feature is disabled.

More information:

- **"Smart Support and Diagnostics Feature Overview"** (page 108)
- common.csv: **"logcollect_enable"** (page 402)

*Configurable blink light duration for HyperStore Appliances (7.5.1)*

The blink light duration for HyperStore Appliance drives and chassis is now configurable. The default duration is three minutes.

More information:

- common.csv: **"blink_disk_intervalsec"** (page 428)
- common.csv: **"blink_appliance_intervalsec"** (page 428)
- While on the CMC's **Node Status** page, click **Help**, then select "Locate a Disk On a HyperStore Appliance"

*Replica auto-repair on multiple nodes concurrently (7.5.1)*

Auto-repair of replicated data is now allowed to run on multiple nodes concurrently. On each node the repair is for the node's primary token ranges only.

More information:

- While on the CMC's **Configuration Settings** page, click **Help**, then select "Auto-Repair Settings"

*New behavior for deleting tiered objects if destination bucket is versioned and source bucket is not (7.5.2)*

In the case where a local source bucket configured for tiering does not have versioning enabled and the destination bucket does have versioning enabled, deleting a tiered object through the local system's S3 interface will result in a delete marker (DM) being created in the destination bucket for the tiered object.

More information:

- Accessing Auto-Tiered Objects

*Deprecation of EC storage policies created with HyperStore version 5.x or 6.x (7.5.2)*

Erasure coding storage policies created with HyperStore versions 5.x or 6.x included a functionality that has been deprecated. The functionality, which identified more than *k+m* endpoints as possible write destinations for an erasure coded object's *k+m* fragments, ultimately was found to pose potential problems that outweighed the benefits (and so erasure coding storage policies created in HyperStore 7.0.1 and later returned to using the more conventional approach of identifying only *k+m* endpoints for erasure coded writes).

Legacy storage policies with the deprecated functionality will no longer be supported as of HyperStore 8.1 (a future HyperStore release). Upon upgrade to HyperStore 7.5.2, the system will automatically detect if you have any of these deprecated storage policies in your system. If you do have any of these deprecated storage policies, then:

- A warning will display in the CMC **Dashboard**. The warning will indicate that you have one or more deprecated storage policies in your system and that you will need to contact your Cloudian Sales Representative to plan for migrating data from those deprecated storage policies to supported storage policies prior to the HyperStore 8.1 release.
- In the CMC's **Storage Policies** page -- which lists all your storage policies -- a warning symbol will display next to the names of the deprecated storage policy or policies. Hover text shows the warning description.
- You cannot edit a deprecated storage policy. However, you can delete a deprecated storage policy if no buckets are using the storage policy. (Since a bucket cannot be switched from the storage policy it's using to a different storage policy, the only way you would be able to delete a storage policy is if all buckets that had been using that policy are deleted first.)
- Users creating new buckets will not be able to use a deprecated storage policy (such policies will not display in the list of available storage policies that users see when creating a new bucket).

# Documentation -- New Features and Enhancements

*Documentation reorganization*

The HyperStore Administrator's Guide (in PDF) has been pared down by removing from it the API reference material and the CMC reference material (Help for each CMC screen). The documentation is now organized as follows:

PDF:

- Cloudian HyperStore Installation Guide
- Cloudian HyperStore Quick Start installation document (a 2-pager)
- Cloudian HyperStore Administrator's Guide
- Cloudian HyperStore Admin API Reference
- Cloudian HyperStore AWS APIs Support Reference

HTML (HyperStore Help, accessible through the CMC):

- Contains all of the information from the documents listed above, except for installation instructions
- Also contains CMC reference material (Help for each CMC screen)

**Note** The Help no longer has section numbering. There are multiple reasons for this change, including that there is no longer an exact correspondence between the Help scope and the Admin Guide scope and so it is no longer possible to have Help section numbering that matches Admin Guide section numbering.

This page left intentionally blank

# Chapter 1. Introduction to HyperStore

## 1.1. HyperStore Documentation

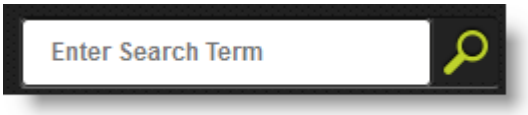
The HyperStore user documentation consists of:

- Cloudian HyperStore Help (HTML5)
- Cloudian HyperStore Administrator's Guide (PDF)
- Cloudian HyperStore Installation Guide (PDF)
- Cloudian HyperStore Quick-Start for Software-Only Users (PDF)
- Cloudian HyperStore Admin API Reference (PDF)
- Cloudian HyperStore AWS APIs Support Reference (S3, IAM, STS, & SQS APIs) (PDF)

The Help is available through the CMC (by clicking the Help button) and is also available in the directory */opt/cloudian-staging/7.5.2/doc/HyperStoreHelp* on your Configuration Master node (in that directory you can open the *HyperStoreHelp.html* file). The PDF guides are available in the directory */opt/cloudian-staging/7.5.2/doc/HyperStorePDFManuals* on the Configuration Master node.

**The Help has all of the content from the Administrator's guide and the API guides, plus information about each CMC screen.**

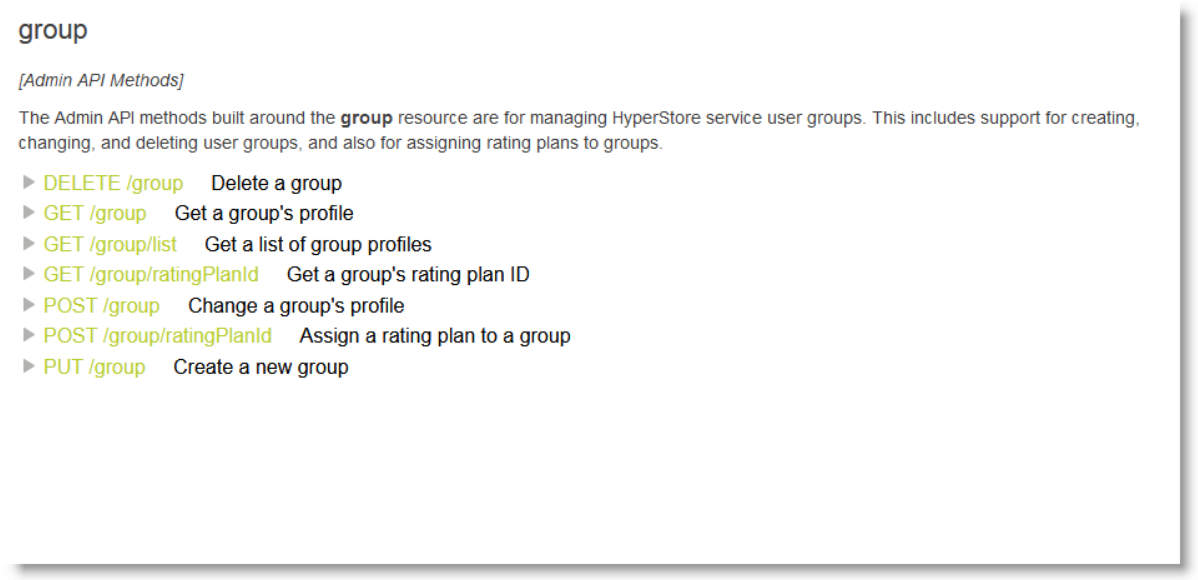
The Help features a built-in search engine. The search box is in the upper right of the interface. As with any search engine, enclose your search phrase in quotes if you want to limit the results to exact match only.




In the Help, in most cases screen shots are presented initially as small thumbnail images. This allows for a more compact initial view of the content on a page and makes it easier for you to skim through the text on the page. If you want to see the full size image simply hold your cursor over the thumbnail image. When you're done looking at the full size image just move your cursor off of it and it will disappear, leaving only the thumbnail image.

Also in the interest of presenting a compact initial view of the content on a page, the Help often makes use of expandable/collapsible text. To expand (or subsequently collapse) such text you can click on the triangle icon to the left of the text or on the text itself.

Example of collapsed text in initial view of a Help page:



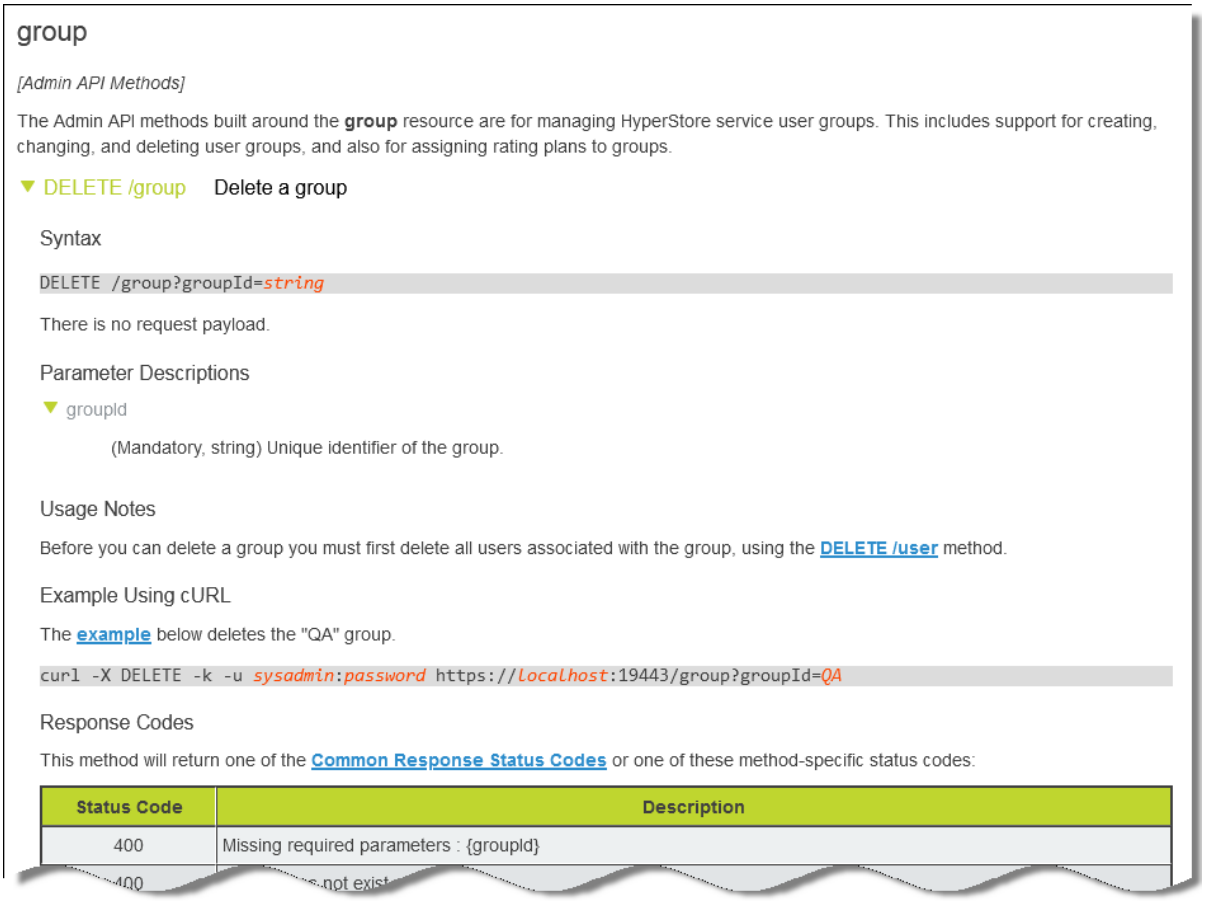
group

*[Admin API Methods]*

The Admin API methods built around the **group** resource are for managing HyperStore service user groups. This includes support for creating, changing, and deleting user groups, and also for assigning rating plans to groups.

- ▶ DELETE /group Delete a group
- ▶ GET /group Get a group's profile
- ▶ GET /group/list Get a list of group profiles
- ▶ GET /group/ratingPlanId Get a group's rating plan ID
- ▶ POST /group Change a group's profile
- ▶ POST /group/ratingPlanId Assign a rating plan to a group
- ▶ PUT /group Create a new group

Example of that same page with the first expandable text item expanded:



group

*[Admin API Methods]*

The Admin API methods built around the **group** resource are for managing HyperStore service user groups. This includes support for creating, changing, and deleting user groups, and also for assigning rating plans to groups.

▼ DELETE /group Delete a group

Syntax

```
DELETE /group?groupId=string
```

There is no request payload.

Parameter Descriptions

▼ groupId

(Mandatory, string) Unique identifier of the group.

Usage Notes

Before you can delete a group you must first delete all users associated with the group, using the DELETE /user method.

Example Using cURL

The example below deletes the "QA" group.

```
curl -X DELETE -k -u sysadmin:password https://localhost:19443/group?groupId=QA
```

Response Codes

This method will return one of the Common Response Status Codes or one of these method-specific status codes:

| Status Code | Description |
|---|---|
| 400 | Missing required parameters : {groupId} |
| 400 | ...not exist |

To expand or collapse all of the expandable/collapsible text on a page, click this button in the upper left of the Help interface:

**The Help is also responsive to the type of user logged in to the CMC**: a system administrator logged in to the CMC sees the full Help content, but a group administrator or regular user logged in to the CMC sees only the content applicable to their role. Also, in the Help that group admins and regular users see, there is no Cloudian branding and no references to HyperStore or the Cloudian Management Console.

> **Note** A system administrator logged into the CMC can download any of the HyperStore PDF documents listed above. A group administrator or regular user logged into the CMC can only download the *Cloudian HyperStore AWS APIs Support Reference* PDF document.

## 1.2. HyperStore Overview

Cloudian HyperStore is a multi-tenant object storage system that fully supports the Amazon Simple Storage System (S3) API. The HyperStore system enables any service provider or enterprise to deploy an S3-compliant multi-tenant storage cloud.

The HyperStore system is designed specifically to meet the demands of high volume, multi-tenant data storage:

- **Amazon S3 API compliance**. The HyperStore system is fully compatible with Amazon S3's HTTP REST API. Customers' existing S3 applications will work with the HyperStore service, and existing S3 development tools and libraries can be used for building HyperStore client applications.
- **Secure multi-tenancy**. The HyperStore system provides the capability to securely have multiple user groups reside on a single, shared infrastructure. Data for each user is logically separated from other users' data and cannot be accessed by any other user unless access permission is explicitly granted.
- **Quality of service controls**. HyperStore system administrators can set storage quotas and usage rate limits on a per-group and per-user basis. Group administrators can set quotas and rate controls for individual members of the group.
- **Access control rights**. Read and write access controls are supported at per-bucket and per-object granularity. Objects can also be exposed via public URLs for regular web access, subject to configurable expiration periods.
- **Reporting and billing**. The HyperStore system supports usage reporting on a system-wide, group-wide, or individual user basis. Billing of groups or users can be based on storage quotas and usage rates (such as bytes in and bytes out).
- **Tiering and replication to external systems**. The HyperStore system supports having objects be tiered (moved) to an external S3-compatible destination system on a defined schedule. The system also supports having objects be replicated to an external S3-compatible destination system.
- **Object Lock (Write Once Read Many)**. The HyperStore system supports the S3 APIs for applying Object Lock to buckets and objects, so that object versions cannot be deleted or altered for a defined retention period.
- **Horizontal scalability**. Running on commodity off-the-shelf hardware, a HyperStore system can scale up to hundreds of nodes across multiple data centers, supporting millions of users and hundreds of petabytes of data. New nodes can be added without service interruption.
- **High availability**. The HyperStore system has a fully distributed, peer-to-peer architecture, with no single point of failure. The system is resilient to network and node failures with no data loss due to the automatic replication and recovery processes inherent to the architecture. A HyperStore cluster can be deployed across multiple data centers to provide redundancy and resilience in the event of a data center scale disaster.

# 1.3. Licensing and Auditing

*Subjects covered in this section:*

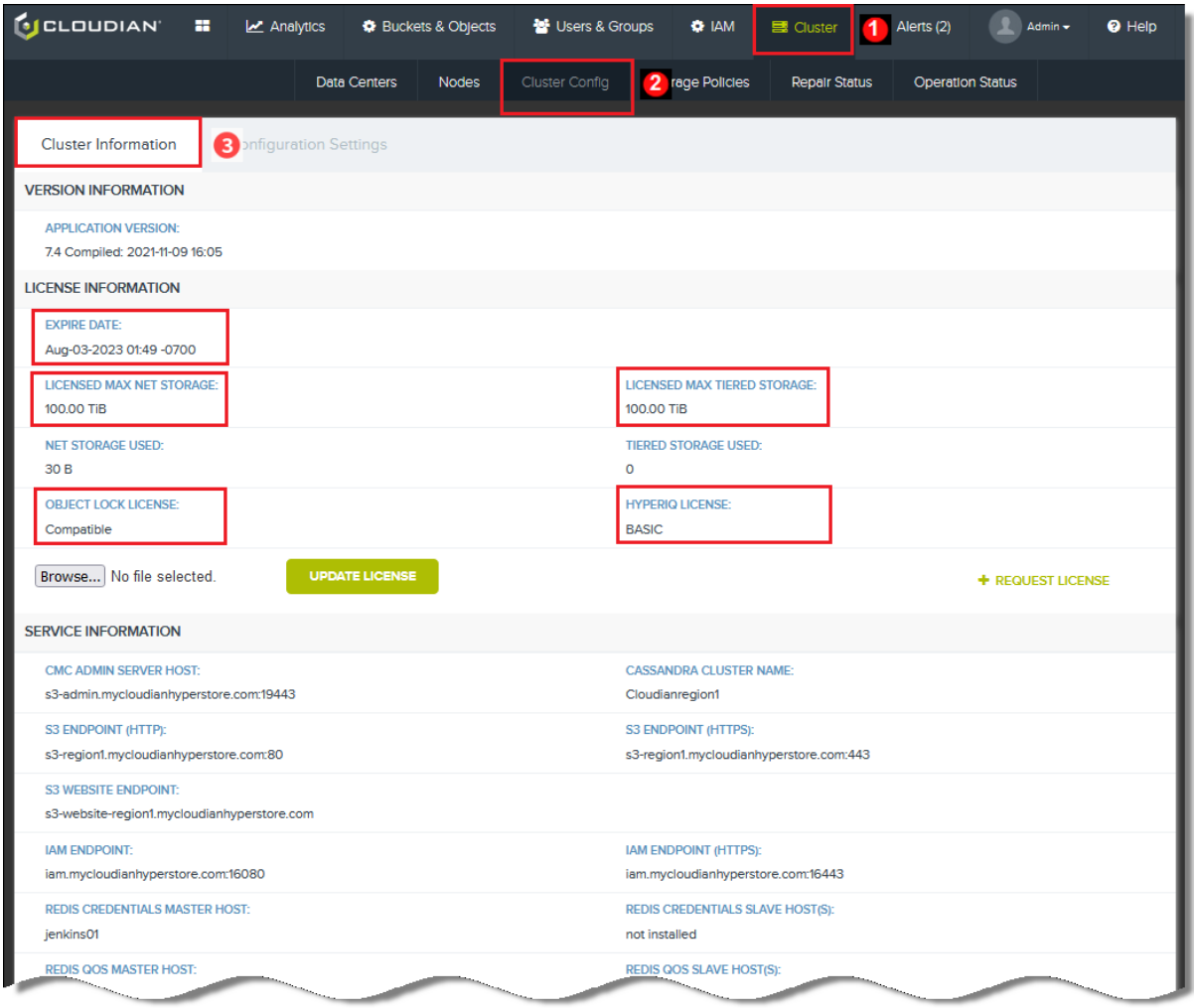


A valid Cloudian software license is required to run HyperStore software. Evaluation licenses are available as well as production licenses. Before using HyperStore software, you must obtain a license from Cloudian.

A Cloudian HyperStore license has these key attributes:

- Expiration date
- Maximum allowed on-premise storage volume
- Maximum allowed tiered storage volume
- Object lock functionality enabled or disabled
- HyperIQ license level

You can see the attributes of your particular HyperStore license by accessing the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).




The sections that follow describe these attributes and their enforcement in more detail.

## 1.3.1. License Expiration

Each Cloudian HyperStore license has an expiration date. Also as part of your license configuration there is a warning period (which commences prior to the expiration date) and a grace period (which extends beyond the expiration date).

If you reach the **warning period** preceding your license expiration, then when you use any part of the CMC, the top of the interface displays a warning that your license expiration date is approaching.

If you reach your license **expiration date** you enter a grace period, per the terms of your contract. During the grace period:

- In the CMC, the top of the screen displays a warning indicating that your license has expired and that your HyperStore system will be disabled in a certain number of days (the number of days remaining in your grace period).
- The system still accepts and processes incoming S3 requests, but every S3 response returned by the S3 Service includes an extension header indicating that the system license has expired (header name: *x-gemini-license*; value: *Expired: <expiry_time>*).

If you reach the **end of your grace period** after the license expiration date:

- No S3 service is available for end users. All incoming S3 requests will be rejected with a "503 Service Unavailable" error response. The response also includes the expiration header described above.
- You can still log into the CMC to perform system administration functions (including applying an updated license), but you will not be able to access users' stored S3 objects.
- The top of the CMC screen will display an error message indicating that your license has expired and that your HyperStore system has been disabled. Also, in the CMC's **Dashboard** page (  ) the Cluster Health panel will indicate that the system is disabled.
- If you stop the S3 Service on a node you will not be able to restart it. This applies also to the Admin Service, since that service stops and starts together with the S3 Service.

It's best to update your license well in advance of your license expiration date. See **"License Updating"** (page 38) below.

## 1.3.2. Licensed Maximum On-Premise Storage Usage

Depending on your particular license terms, your HyperStore system will have either a Net storage limit or a Raw storage limit, for on-premise data storage.

With a license based on **Net** storage volume -- also known as "Usable" volume -- the limit is on total object storage bytes minus overhead from **storage policies** (object replication or erasure coding). For example if a 1GB object is replicated three times in your system it counts as only 1GB toward a Net storage limit. A Net storage license is typically used if your cluster consists entirely of software-only nodes, with no HyperStore Appliance nodes.

With a license based on **Raw** storage volume the limit is on the total raw storage capacity used in your system. All HyperStore object data and metadata counts toward this limit, including storage overhead from replication or erasure coding. For example if a 1GB object is replicated three times in your system it counts as 3GB toward a Raw storage limit. Likewise, all object metadata and system metadata count toward a Raw storage limit.

A Raw storage license is typically used in either of two types of environments:

- Appliance-only environment. Each HyperStore Appliance has its own amount of licensed Raw storage capacity. If your system consists entirely of HyperStore Appliances, then the Raw licensed storage capacity for your whole system is the simply sum of the individual Appliance licensed capacities.
- Mixed environment of Appliances and software-only nodes. In a mixed environment, the Raw licensed storage capacity for your whole system is the sum of the individual Appliance licensed capacities plus an additional raw capacity allowance that Cloudian builds into your license to accommodate the software-only nodes.

If your license is based on Raw storage, then your total licensed Raw storage limit will be automatically increased if you add a new HyperStore Appliance node to the system. The amount of Raw storage added to your licensed system maximum depends on the particular HyperStore Appliance that you've added to your system. Conversely, if you remove an Appliance from your system this will reduce your total licensed system Raw storage maximum; and the Raw storage allowance associated with a particular Appliance machine cannot be transferred to other nodes in your system.

**Note** If your HyperStore licensed maximum storage is in terms of Net bytes, adding a HyperStore Appliance to your cluster will not change your Net storage limit. If you are interested in increasing your Net storage limit or converting to a Raw storage limit, consult with Cloudian Support.

In the CMC's **Cluster Information** page you can view the Net or Raw licensed usage maximum for your whole system and also your current system-wide Net or Raw bytes usage count. If your current usage level exceeds 70% of your licensed maximum usage the CMC displays a warning message in both the **Cluster Information** page and the **Dashboard** page. If your current usage level exceeds 90% of your licensed maximum usage the CMC displays a critical message in both of those pages.

Your total system storage usage maximum will be **automatically enforced** by the system no longer allowing S3 clients to upload data to the system. This enforcement will kick in when your system stored byte count reaches **110%** of your licensed maximum usage. At such point the system will reject S3 PUT and POST requests and return an error to the S3 clients. This will continue until one of the following occurs:

- You delete object data so that the system byte count falls below **100%** of your licensed maximum. HyperStore checks every five minutes to see if your storage usage has fallen below the licensed maximum, and if it does fall below the maximum then S3 PUTs and POSTs will again be allowed.

  **Note** In the case of Raw usage, your deletions will not impact your system's raw usage count until the next run of the automatic purging process that occurs hourly on each node. By contrast, a Net usage count is decremented immediately when you delete objects.

- You acquire and install a new license with a larger storage maximum (see **"License Updating"** (page 38)). Upon new license installation, S3 PUTs and POSTs will again be allowed.

If the system stored byte count reaches 110% of your licensed maximum usage the CMC will display a pop-up warning message to the system administrator whenever he or she logs in. This will recur on each login event until the system byte count falls below 100% of usage, or a new license with larger storage maximum has been installed.

**IMPORTANT !** Regardless of whether your system has a Net storage license or a Raw storage license, **if the data disks on a node become 90% full then that node will stop accepting new S3 writes**. This is not a license enforcement mechanism but rather a system safety feature. For more information see **"Automated Disk Usage Management Feature Overview"** (page 113).

#### 1.3.2.1. The Effect of Versioning on On-Premise Storage Volume Measurement

If some users have versioning enabled on their buckets -- so that the system retains rather than overwriting older versions of an object when the user uploads a new version of the object -- then each stored object version (the older versions as well as the current version) counts toward your system stored bytes count. For more information on versioning see Set Versioning for a Bucket.

#### 1.3.2.2. The Effect of Cross-Region Replication on On-Premise Storage Volume Measurement

If some users use the cross-region replication feature to replicate objects from one HyperStore bucket to another HyperStore bucket within the same HyperStore system, then the original source objects and the object replicas in the destination bucket both count toward your system stored bytes count. For more information on cross-region replication see **"Cross-Region Replication Versus Auto-Tiering"** (page 180).

## 1.3.3. Licensed Maximum Tiered Storage Usage

HyperStore supports an auto-tiering feature that users can enable on a per-bucket basis. With auto-tiering, objects can be automatically moved on a configurable schedule to an external destination system such as Amazon S3 or Glacier, Microsoft Azure, or Google Cloud Storage. For more information on auto-tiering see **"Auto-Tiering Feature Overview"** (page 169).

In regard to HyperStore licensing, the system treats auto-tiered data as a separate category than on-premise data. Data that's been auto-tiered out of your HyperStore system and is now stored in an external, third party system counts toward a Maximum Tiered Storage limit -- not toward your on-premise storage limit. The Maximum Tiered Storage limit is based on Net (also known as Usable) bytes, not Raw bytes.

The enforcement of this separate tiered storage limit works in largely the same way as the enforcement of the on-premise storage limit.

In the CMC's **Cluster Information** page you can view your tiered storage limit and also your current tiered storage usage level. If your tiered usage level exceeds 70% of your licensed tiered storage maximum the CMC displays a warning message in the **Cluster Information** page. This becomes a critical message if the usage exceeds 90% of licensed maximum.

The tiered usage maximum will be **automatically enforced** by the system no longer allowing auto-tiering to any third party destination system. This enforcement will kick in when your tiered storage byte count reaches **110%** of your licensed maximum. At this point auto-tiering to third party destinations will no longer work until one of the following occurs:

- Through a HyperStore interface -- i.e. the CMC or the HyperStore S3 API -- you delete auto-tiered data so that the total tiered byte count falls below 100% of your licensed maximum. HyperStore checks every five minutes to see if your tiered storage usage has fallen below the licensed maximum, and if it does fall below the maximum then auto-tiering to non-HyperStore destinations is allowed again and automatically resumes.

  **IMPORTANT !** If you're trying to reduce your tiered storage volume to below your licensed maximum, be sure to delete auto-tiered objects **through a HyperStore interface** and not directly through one of the tiering destination system's interfaces. If you do the latter, HyperStore will not detect that you've reduced your tiered storage volume. For more information see Accessing Auto-Tiered Objects.

- You acquire and install a new license with a larger tiered storage maximum (see **"License Updating"** (page 38)). Upon new license installation, auto-tiering to third party destinations will be allowed again and will automatically resume.

If the tiered byte count reaches 110% of your licensed maximum usage the CMC will display a pop-up warning message to the system administrator whenever he or she logs in. This will recur on each login event until the tiered byte count falls below the licensed maximum, or a new license with larger tiered storage maximum has been installed.

If auto-tiering to third party destination systems stops because you've exceeded 110% of your licensed maximum, and then later resumes when you come back into compliance with your license, the system will auto-tier any objects that were flagged for auto-tiering during the time period when auto-tiering was halted for license non-compliance. So as long as you come back into compliance, the period of non-compliance will not result in any permanent failures to auto-tier objects that are supposed to have been auto-tiered based on users' bucket lifecycle configurations.

#### 1.3.3.1. Tiered Storage Licensing Exceptions and Qualifiers

- Auto-tiered objects count toward your tiered storage licensed maximum, not your on-premise storage licensed maximum. However, for each auto-tiered object there is a small bit of **object metadata** (8KB per object) that is retained on-premise and counts toward your on-premise storage limit.
- The auto-tiering feature supports an option (configurable on a per-bucket basis) to **retain a local copy** of auto-tiered objects for a specified period of time. If this option is used, then the retained local copy counts toward your on-premise storage limit (while the tiered object copy counts toward your tiered storage limit), until the local copy reaches the end of its retention period and is automatically deleted from local storage.
- The auto-tiering feature supports an option to **temporarily restore** tiered objects into local storage. This works by downloading a copy of the object. During the time period that the object is locally restored, the object counts toward your on-licensed on-premise storage limit as well as toward your licensed tiered storage limit. The same is true when the "cache" (stream and restore) method is used for GETs of tiered objects -- tiered objects that have been cached count toward your on-licensed on-premise storage limit as well as toward your licensed tiered storage limit.
- The auto-tiering feature supports an option (configurable on a per-bucket basis) to tier to a HyperStore destination -- either a different service region within the same HyperStore, or an entirely separate HyperStore system. **Tiering to a HyperStore destination does not count toward the licensed tiering limit**. Instead it counts toward the target HyperStore system's licensed on-premise storage limit.

### 1.3.4. Object Lock (WORM) License

Your license may or may not include support for the HyperStore Object Lock (WORM) feature. To check whether your license supports this feature, see the CMC's **Information** page: in the "Object Lock License" field it will indicate either "Disabled" or "Compatible" or "Certified". If Disabled, you cannot use the Object Lock feature -- the system will return a 403 Forbidden response if an S3 client application attempts to create a new bucket with object lock enabled.

For more information about this feature, including descriptions of the two types of licensed Object Lock support -- "Compatible Object Lock" and "Certified Object Lock" -- see **"Object Lock Feature Overview"** (page 187).

If your current license does not support Object Lock and you want to use this feature, or if you want to switch your licensed Object Lock type from "Compatible" to "Certified" or vice versa, contact Cloudian Support.

### 1.3.5. HyperIQ License

Cloudian HyperIQ is a solution for dynamic visualization and analysis of HyperStore system monitoring data and S3 service usage data. HyperIQ is a separate product available from Cloudian that deploys as virtual appliance on VMware or VirtualBox and integrates with your existing HyperStore system. For more information about HyperIQ contact your Cloudian representative.

Your HyperStore license has a HyperIQ attribute that determines the level of HyperIQ functionality available to you if you acquire and set up the HyperIQ virtual appliance:

- Basic -- HyperIQ dashboards for OS and service status monitoring are supported indefinitely. This is the default.
- Enterprise -- HyperIQ dashboards for OS and service status monitoring are supported indefinitely, and also an S3 analytics dashboard is supported until a defined expiration date. The presence of the S3

analytics dashboard is what distinguishes Enterprise level HyperIQ support from Basic HyperIQ support.

### 1.3.6. License Updating

You may need to update your license periodically, depending on the specific terms of your Cloudian license agreement. Updating your license requires obtaining a new license file from Cloudian and applying that file on all of your HyperStore nodes. Existing customers can obtain a new license file by emailing a request to *cloudian-license@cloudian.com.*

Once you've obtained a new license file you can use the CMC's **Information** page to dynamically apply the new license file to your HyperStore system. For instructions see Install a New License File.

### 1.3.7. Auditing

If you have a production license for HyperStore software, the system will regularly transmit auditing data to Cloudian, Inc., using the system's **Smart Support** functionality.

## 1.4. Nodes, Data Centers, and Regions

*Subjects covered in this section:*



The basic units of a HyperStore system are:

- **Node** -- HyperStore software running on a host machine.
- **Data center** -- A physical data center (DC) in which multiple HyperStore nodes are running.
- **Service region** -- Also known as a **cluster**, a region is a unified set of HyperStore nodes deployed in one or multiple data centers across a particular geographic area. A region has its own unique S3 service endpoint, its own unique inventory of stored objects, and a unified Cassandra database for storing metadata (the Metadata DB).

The common deployment topologies for a HyperStore system are:

- **Single data center constituting a single region**. This is the simplest deployment topology, where the whole HyperStore system consists of a single data center in which multiple HyperStore nodes are running. The one DC constitutes its own region. The number of nodes can scale from a minimum of three -- the smallest viable size of a HyperStore system -- up to dozens of nodes, all running in one DC. Adding more nodes within a DC is the most common way to add capacity to the system. For more information see **"Capacity Monitoring and Expansion"** (page 338).
- **Multiple data centers in a single region**. With this topology, HyperStore nodes running in multiple data centers comprise one unified storage cluster. Typically the motivation for this type of deployment is replication of data across DCs, for the purposes of data protection, service resilience, and/or disaster recovery. This cross-DC replication is configurable through the use of HyperStore storage policies.

- **Multiple service regions**. With this topology the HyperStore system spans multiple service regions, each of which consists of one or more data centers. Each region has its own separate S3 service endpoint (to which S3 clients submit requests), its own independent storage cluster, and its own separate inventory of stored objects. In a multi-region HyperStore system, the regions are in most respects separate S3-compatible object storage systems -- with the significant exceptions that the same population of authorized end users has access to all the service regions, and that HyperStore affords a substantial degree of unified administration across the multiple regions. Typically the motivation for having multiple service regions is to allow users to choose one geographic region or another for storing their data, for reasons of proximity or regulatory compliance.

For a diagram showing the relation between nodes, data centers, and service regions see **"System Levels"** (page 67).

## 1.4.1. Storage Policies in Multi-DC or Multi-Region Systems

In a **multi-DC, single-region** HyperStore system, configurable storage policies determine whether and how data is replicated across DCs. You can have multiple storage policies -- for example you could have one storage policy that stores data only in one particular DC, and another storage policy that replicates data so that copies of each object are stored in each of your DCs. When users create a bucket they choose which storage policy will apply to data uploaded to that bucket.

In a **multi-region** system, each region has its own set of storage policies, and each region's storage policies operate only within that region. As noted previously, each region is essentially an independent storage cluster with its own inventory of objects -- and its own storage policies for distributing data within the region.

For more information about storage policies, including details about multi-DC storage policies, see **"Storage Policies Feature Overview"** (page 127).

For information about a supported option for asynchronously replicating data from a bucket in one region to a different bucket in a different region, see **"Cross-Region Replication Feature Overview"** (page 179).

## 1.4.2. Service Architecture in Multi-DC or Multi-Region Systems

### 1.4.2.1. Multi-DC, Single Region System

Along with common services that run on every node in a cluster (such as the S3 Service, the HyperStore Service, and the Metadata DB), the HyperStore system also includes specialized services that run on only one or a few nodes. If you deploy HyperStore across multiple DCs in a service region, the system automatically allocates the specialized services appropriately. For example, each DC will have two nodes acting as Credentials DB slave nodes and each DC will have one node acting as a QoS DB slave node.

For a summary of services and how they are allocated, see **"HyperStore Services Overview"** (page 58).

For a diagram showing typical services distribution in a multi-DC region, see **"Services Distribution -- Multi-DC, Single Region"** (page 70).

Also, in a multi-DC region each DC has its own sub-set of HyperStore nodes that are automatically configured to act as internal NTP servers. For more information see **"NTP Automatic Set-Up"** (page 496).

### 1.4.2.2. Multi-Region System

A multi-region HyperStore system has the following characteristics:

- One of the regions serves as the **default region**. The default region plays several roles in a multi-region system. For example:
  - If service users do not specify a region when they create a new S3 storage bucket, the system will create the bucket in the default region.
  - Only the Admin Service instances in the default service region support the full Admin API.
- Each region has its own S3 service endpoint (URI used by client applications for HTTP access).
- Each region has its own independent object storage cluster and by default there is no object replication across regions (although there is an option for **cross-region replication** on a bucket-to-bucket basis).
- When users create a new bucket they choose which region the bucket will be created in.
- User access credentials are valid across the system as a whole. In support of user authentication, a single, uniform Credentials DB serves the entire multi-region system. There is just one Credentials DB master node for the whole system, and that node is located in your default region. Within each region, there are two Credentials DB slave nodes per data center.
- Quality of service (QoS) controls are implemented separately in each region. The QoS limits that you establish for a service region will be applied only to user activity in that particular region. In support of QoS implementation, each region has its own independent QoS DB. Each regional QoS DB has its own master node. In each region there is also one QoS DB slave node per data center.
- The Redis Monitor application will monitor the Credentials DB and QoS DBs in all the regions (and if necessary trigger failover of the master role within each database). One primary Redis Monitor application instance serves the whole multi-region system, and if the primary Redis Monitor instance goes down the backup instance takes over. The primary Redis Monitor instance and backup Redis Monitor instance are on separate nodes in your default region.
- Group and user profile information is stored only in the default region, and is accessed there by services in the other regions. Group and user information is stored only in the Metadata DB in the default region. HyperStore services in non-default regions access the Metadata DB in the default region to retrieve this group and user information as needed.
- Just one Configuration Master node is used to propagate system configuration settings throughout the whole multi-region system, during system installation and for ongoing system configuration management.

For a diagram showing typical services distribution in a multi-region system, see **"Services Distribution -- Multi-Region"** (page 71).

## 1.4.3. Deploying HyperStore to Multiple DCs or Regions

A common way to arrive at a multi-DC or multi-region HyperStore system is to initially install and use HyperStore in a single DC constituting a single region; and then later in the life of the system, to expand the system by adding a DC or a region. The CMC supports adding a new DC to an existing region (installing HyperStore software on host machines in a new data center and updating the system configuration to reflect the addition of the new DC) or adding a new region to the system (installing HyperStore software on host machines in one or more new data centers and updating the system configuration to reflect the addition of the new service region and data center[s]). For instructions see:

- **"Adding a Data Center"** (page 276)
- **"Adding a Region"** (page 283)

HyperStore also supports the option of installing HyperStore software to multiple DCs or regions from the outset, upon the initial system installation. From a HyperStore system configuration perspective the key here is the "survey file", which the HyperStore *system_setup.sh* tool helps you to create. By responding to that tool's

interactive prompts, you create a survey file that identifies (among other things) the name of the data center and region that each node resides in. Subsequently the installer tool (*cloudianInstall.sh*) installs HyperStore software to each of those nodes and configures the system to be a single-DC, multi-DC, or multi-DC / multi-region system, in accordance with your survey file.

Below is an example of an installation survey file for a 3-node HyperStore system that will be configured as just a single service region with just a single data center. Note that you must provide a data center name and a region name even if you will have just one DC constituting just one region. Here the region name is "tokyo" and the data center name is "DC1".

```
tokyo,cloudian-vm7,66.10.1.33,DC1,RAC1
tokyo,cloudian-vm8,66.10.1.34,DC1,RAC1
tokyo,cloudian-vm9,66.10.1.35,DC1,RAC1
```

Here is a second example, this time for a system that will be installed as a single-region system with two data centers:

```
tokyo,cloudian1,66.1.1.11,DC1,RAC1
tokyo,cloudian2,66.1.1.12,DC1,RAC1
tokyo,cloudian3,66.1.1.13,DC1,RAC1
tokyo,cloudian4,67.2.2.17,DC2,RAC1
tokyo,cloudian5,67.2.2.18,DC2,RAC1
tokyo,cloudian6,67.2.2.19,DC2,RAC1
```

Below is a third example, this time for a system that will be installed as a two-region system. Note that in this example, the "tokyo" region encompasses two data centers while the "osaka" region consists of just one data center.

```
tokyo,cloudian1,66.1.1.11,DC1,RAC1
tokyo,cloudian2,66.1.1.12,DC1,RAC1
tokyo,cloudian3,66.1.1.13,DC1,RAC1
tokyo,cloudian4,67.2.2.17,DC2,RAC1
tokyo,cloudian5,67.2.2.18,DC2,RAC1
tokyo,cloudian6,67.2.2.19,DC2,RAC1
osaka,cloudian7,68.10.3.24,DC3,RAC1
osaka,cloudian8,68.10.3.25,DC3,RAC1
osaka,cloudian9,68.10.3.26,DC3,RAC1
```

For more information about installation including node preparation, DNS, and load balancer requirements, see the "Installing HyperStore" section of the HyperStore Help.

## 1.4.4. Using the CMC with Multiple DCs or Regions

The Cloudian Management Console (CMC) facilitates unified administration of a multi-DC or multi-region HyperStore system.

### 1.4.4.1. Multi-DC, Single Region System

In the CMC, in most respects a multi-DC deployment within a service region is presented as one integrated cluster. For example, performance statistics such as S3 transactions per second and S3 bytes throughput per second are reported for the cluster (the service region) as a whole. However the CMC also is data center aware and provides visibility into the individual DCs within a service region. For example:

- The **Capacity Explorer** page (**Analytics -> Capacity Explorer**) lets you see how much free storage capacity remains in each DC (as well as in each node, and also in the service region as a whole).

- The **Data Centers** page (**Cluster -> Data Centers**) shows you your node inventory in each DC and provides summary status information for each node. From this page you can also add nodes to your cluster on a per-DC basis.
- The **Object Locator** page (**Analytics -> Object Locator**) lets you see exactly where all of a specified object's replicas or erasure coded fragments are located (on which nodes, in which DC)

#### 1.4.4.2. Multi-Region System

If your HyperStore deployment is set up as a multi-region system, that will be reflected in these aspects of the CMC interface:

- User Provisioning and Administration
  - When you add a user in the **Manage Users** page (**Users & Groups -> Manage Users**), you will assign the user a rating plan (for billing purposes) for each region. You can assign the same rating plan in each region, or different rating plans in each region. The same is true when you add a group (**Users & Groups -> Manage Groups**): when you set a group-level rating plan (which serves as the default rating plan for users within the group), you choose a rating plan for each region.
  - When you set quality of service (QoS) limits for users or groups, you will assign a different set of limits for each region. QoS limits are applied on a per-region basis.
- S3 Data Storage and Access
  - When service users add a bucket in the **Buckets** page (**Buckets & Objects -> Buckets**), they choose which region to create the bucket in.
- Usage Reporting and Billing
  - When you create a usage report for a user, a user group, or the system in the **Usage By Users and Groups** page (**Analytics -> Usage By Users and Groups**), you can choose the region for which to report usage. You can also generate reports that includes usage from all regions.
  - When you use the **Account Activity** page (**Users & Groups -> Account Activity**) to generate a statement for billing a user, you generate a separate bill for each region.
- HyperStore System Monitoring
  - When you use the **Data Centers** page (**Cluster -> Data Centers**) to get a high level view of each data center's status, you choose which region's data centers you want to check on before choosing the data center.
  - When you use the **Node Status** page (**Cluster -> Nodes -> Node Status**) to check the status detail for individual nodes, you start by selecting a region, and then select a node within that region.
- HyperStore System Management
  - When you node operations in the **Nodes Advanced** page (**Cluster -> Nodes -> Advanced**), you first select a region, then select a node from within that region, then select the operation to perform on that node.
  - When you are adding a node to the system in the Data Centers page (**Cluster -> Data Centers**), you specify the region and data center in which to add it.

### 1.4.5. Using the Admin API in a Multi-Region System

If your HyperStore system has multiple regions, then:

- For some Admin API calls you can optionally use a "region" URI parameter to indicate that you want the operation applied to a particular region. For example, the syntax for retrieving a user's rating plan is:

```
GET /user/ratingPlan?userId=string&groupId=string[&region=string] HTTP/1.1
```

  For such API calls, if you do not specify a region then the default region is presumed.

- Certain Admin API calls are only supported by the Admin Service in the default region. If you submit these calls to the Admin Service in a non-default region, you will receive a 403: Forbidden response. For more information see the Introduction section in the ***Cloudian HyperStore Admin API Reference***.

# 1.5. Hosts and Network

## 1.5.1. Host Hardware and OS Requirements

*Subjects covered in this section:*



### 1.5.1.1. Hardware Requirements

The table below shows the recommended and minimum specifications for individual host machines in a HyperStore system. In general your hosts should either by HyperStore Appliances or hosts with specifications comparable to HyperStore Appliances.

| | HSA-1600 Appliance | | HSA-4200 Appliance (Per Node) | | Recommended VM Spec | | Minimum VM Spec (Test/Dev/POC) | |
|---|---|---|---|---|---|---|---|---|
| **Component** | **Description** | **Qty** | **Description** | **Qty** | **Description** | **Qty** | **Description** | **Qty** |
| Processors | Intel Xeon 5218R | 2 | Intel Xeon 5218R | 2 | 20 core CPU | 2 | 8 core CPU | 2 |
| RAM | 32GB (128GB) | 4 | 32GB (256GB) | 8 | 128GB | 1 | 128GB | 1 |
| Flash Metadata Tier | NVMe Storage 1920GB | 2 | NVMe Storage 1920GB | 2 | SSD/NVMe Storage 1920GB | 2 | SSD Storage 960GB | 2 |
| HDD Capacity Tier | 7200RPM SAS HDD -- Up to 14TB per disk | 12 | 7200RPM SAS HDD -- Up to 16TB per disk | 30 | 7200RPM SAS HDD -- Up to 14TB per disk | 4-12 | 7200RPM SAS HDD -- Up to 14TB per disk | 4-12 |
| Networking | 10/25GbE | 2 | 10/25GbE | 2 | 10GbE | 2 | 10GbE | 2 |

Note that:

- The "Minimum VM Spec" is applicable for test, development, or evaluation environments, and is not recommended for running in production.

- For production environments the recommended specifications in the table are a starting point. For guidance on scaling a HyperStore deployment to meet your workload requirements, consult with your Cloudian sales representative.
- For virtualization environments, running HyperStore on VMware ESXi and vSphere is supported so long as the VMs have specs meeting or exceeding those in the table. However, avoid KVM or Xen as there are known problems with running HyperStore in those virtualization environments. For more guidance on deploying HyperStore on VMware, ask your Cloudian representative for the "Best Practices Guide: Virtualized Cloudian HyperStore on VMware vSphere and ESXi".

### 1.5.1.2. Operating System Requirements

To install HyperStore 7.5.2, on each host machine the operating system must be **CentOS Minimal version 7.4 or newer 7.x** (or Red Hat Enterprise Linux 7.4 or newer 7.x). HyperStore 7.5.2 does **not** support installation on:

- Older versions of CentOS/RHEL
- CentOS/RHEL 8.x or newer
- Other types of Linux distribution
- Non-Linux operating systems

If you have not already done so, install CentOS Minimal 7.4 or newer 7.x (or RHEL 7.4 or newer 7.x) in accordance with your hardware manufacturer's recommendations.

> **Note** Cloudian recommends using CentOS/RHEL **7.7 or newer 7.x**.

Below, see these additional requirements related to host systems on which you intend to install HyperStore:


- **"Partitioning of Disks Used for the OS and Metadata Storage"** (page 44)
- **"Host Firewall Services Must Be Disabled"** (page 45)
- **"Python 2.7.x is Required"** (page 45)
- **"Do Not Mount /tmp Directory with 'noexec'"** (page 45)
- **"root User umask Must Be 0022"** (page 46)


#### Partitioning of Disks Used for the OS and Metadata Storage

For the disks used for the OS and metadata storage -- typically two mirrored SSDs as noted in the hardware requirements table above -- **do not accept the default partition schemes offered by CentOS/RHEL**:

- By default CentOS/RHEL allocates a large portion of disk space to a */home* partition. This will leave inadequate space for HyperStore metadata storage.
- By default CentOS/RHEL proposes using LVM. Cloudian recommends using standard partitions instead.

Cloudian recommends that you manually create a partition scheme like this:

##### For Software RAID

- 1x 1G as */boot*, Device Type RAID1, label *boot*, fs: *ext4*
- 1x 8G as SWAP, Device Type RAID1, label *swap*
- 1x remaining space as */* , Device Type RAID 1, label *root*, fs: *ext4*

### For Hardware SUDO RAID with UEFI

- 1x 1G as */boot/efi*, label *efi*
- 1x 1G as */boot*, label *boot*, fs: *ext4*
- 1x 8G as SWAP, label *swap*
- 1x remaining space as */* , label *root*, fs: *ext4*

## Host Firewall Services Must Be Disabled

To install HyperStore the following services **must be disabled on each HyperStore host machine**:

- *firewalld*
- *iptables*
- *SELinux*

To disable ***filewalld***:

```
# systemctl stop firewalld
# systemctl disable firewalld
```

RHEL/CentOS 7 uses *firewalld* by default rather than the *iptables* service (*firewalld* uses *iptables* commands but the *iptables* service itself is not installed on RHEL/CentOS by default). So you do not need to take action in regard to ***iptables*** unless you installed and enabled the *iptables* service on your hosts. If that's the case, then disable the *iptables* service.

To disable **SELinux**, edit the configuration file */etc/selinux/config* so that *SELINUX=disabled*. Save your change and then restart the host.

**HyperStore includes a built-in firewall service** (a HyperStore-custom version of the *firewalld* service) that is configured to protect HyperStore internal services while keeping HyperStore public services open. In fresh installations of HyperStore 7.2 or later, the HyperStore firewall is enabled by default upon the completion of HyperStore installation. In HyperStore systems originally installed as a version older than 7.2 and then later upgraded to 7.2 or newer, the HyperStore firewall is available but is disabled by default. After installation of or upgrade to HyperStore 7.2 or later, you can enable or disable the HyperStore firewall by using the installer's Advanced Configuration Options.

**Note** For information about HyperStore port usage see **"HyperStore Listening Ports"** (page 54).

## Python 2.7.x is Required

The HyperStore installer **requires Python version 2.7.x**. The installer will abort with an error message if any host is using Python 3.x. To check the Python version on a host:

```
# python --version
Python 2.7.5
```

## Do Not Mount */tmp* Directory with 'noexec'

The */tmp* directory on your host machines must not be mounted with the '*noexec*' option. If the */tmp* directory is mounted with '*noexec*', you will not be able to extract the HyperStore product package file and the HyperStore installer (installation script) will not function properly.

#### root User umask Must Be 0022

On hosts on which you will install HyperStore, the *root* user umask value must be '0022' (which is the default on Linux hosts). If the *root* user umask is other than '0022' the HyperStore installation will abort.

### 1.5.1.3. Note: Automatic Exclusions to OS Package Updates

As part of HyperStore installation, the HyperStore installation script will install prerequisites including Puppet, Facter, Ruby, and Salt on your HyperStore host machines. If you subsequently use *yum update* or *yum upgrade* to update your OS packages, HyperStore automatically excludes Puppet, Facter, Ruby, and Salt related packages from the update. This is to ensure that only the correct, tested versions of these packages are used together with HyperStore. After HyperStore installation, this auto-exclusion is configured in the */etc/yum/pluginconf.d/versionlock.list* file on your host machines. You can review that file if you wish to see specifically which packages are "locked" at which versions, but do not remove any entries from the lock list.

## 1.5.2. File System Requirements

*Subjects covered in this section:*



Cloudian recommends that you use the HyperStore *system_setup.sh* tool to configure the disks and mount points on your HyperStore nodes, as described in Configuring Network Interfaces, Time Zone, and Data Disks. The tool is part of the HyperStore product package (when you extract the *.bin* file).

**If you do not use the system setup tool for disk setup, use the information below to make sure that your hosts meet HyperStore file system requirements**.

### 1.5.2.1. OS/Metadata Drives and Data Drives

Although it's possible to install HyperStore on a host with just a single hard drive, for a rigorous evaluation or for production environments each host should have multiple drives (see **"Host Hardware and OS Requirements"** (page 43)). On host machines with multiple hard drives:

- HyperStore will by default use the drive that the OS is on for storing system metadata (in the Metadata DB, the Credentials DB, and the QoS DB). Cloudian recommends that you dedicate two drives to the OS and system metadata in a RAID-1 mirroring configuration. Preferably the OS/metadata drives should be SSDs.
- You must format all other available hard drives with *ext4* file systems mounted on raw disks. These drives will be used for storing S3 object data. RAID is not necessary on the S3 object data drives.

For example, on a machine with 2 SSDs and 12 HDDs:

- Mirror the OS on the two SSDs. For more detailed recommendations for partitioning these disks see **"Partitioning of Disks Used for the OS and Metadata Storage"** (page 44).

- Format each of the 12 HDDs with *ext4* file systems and configure mount points such as */cloudian1*, */cloudian2*, */cloudian3* and so on.

  **Note** On the HDDs for storing object data, HyperStore **does not support** XFS file systems; VirtIO disks; Logical Volume Manager (LVM); or Multipathing. For questions regarding these unsupported technologies, contact Cloudian Support:

### 1.5.2.2. Mount Point Naming Guidelines

If you are installing HyperStore on multiple hosts that each have multiple disks for object data storage, use the same mount point naming scheme on each of your hosts. If all your hosts have the same number of disks, then they should all have the identical set of mount points for HyperStore object storage. For example, if each host has 12 disks for object storage, then on all your hosts you could name the mount points */cloudian1*, */cloudian2*, */cloudian3*, and so on up through */cloudian12*.

If in your installation cluster some hosts have more disks than others, use as much overlap in mount point naming as possible. For example, suppose that most of your hosts have 10 disks for storing object data while one host has 12 disks. In this scenario, all of the hosts can have mount points */cloudian1*, */cloudian2*, */cloudian3*, and so on up through */cloudian10*, while the one larger host has those same mount points plus also */cloudian11* and */cloudian12*.

**Note** Although uniformity of mount point naming across nodes (to the extent possible) is desirable for simplicity's sake, the HyperStore installation does support a way to accommodate differences in the number or names mount points across nodes -- this is described in **"A Data Directory Mount Point List (fslist.txt) Is Required"** (page 48)..

### 1.5.2.3. Option for Putting the Metadata DB on Dedicated Drives Rather Than the OS Drives

Regarding the Metadata DB (built on Cassandra), another supported configuration is to put your Cassandra data on dedicated drives, rather than on the OS drives. In this case you would have:

- OS drives in RAID-1 configuration. The Credentials DB and QoS DB will also be written to these drives.
- Cassandra drives in RAID-1 configuration. On these drives will be written Cassandra data and also the Cassandra commit log.

  **Note** You must create a Cassandra data directory named as *<mountpoint>/cassandra* (for example *cassandradb/cassandra*) and a Cassandra commit log directory named as *<mountpoint>/cassandra_commit* (for example *cassandradb/cassandra_commit*).

- Multiple drives for S3 object data (with mount points for example */cloudian1*, */cloudian2*, */cloudian3* and so on), with no need for RAID protection.

### 1.5.2.4. You Must Use UUIDs in fstab

In your *fstab* file, **you must use UUIDs** to identify the devices to which you will mount HyperStore S3 object data directories. Do not use device names or LABELs.

If you are not using UUIDs in *fstab* currently, follow the instructions below to modify your *fstab* so that it uses UUIDs for the devices to which you will mount S3 object data directories (you do not need to do this for the OS/-metadata mount points).

As *root*, do the following:

1. Check whether your *fstab* is currently using UUIDs for your S3 object data drives. In the example below, there are two S3 object data drives and they are currently identified by device name, not by UUID.

```
# cat /etc/fstab
...
...
/dev/sdb1 /cloudian1  ext4  rw,noatime,barrier=0,data=ordered,errors=remount-ro 0 1
/dev/sdc1 /cloudian2  ext4  rw,noatime,barrier=0,data=ordered,errors=remount-ro 0 1
```

2. Back up your existing fstab file:

```
# cp /etc/fstab /etc/fstab.backup.<today's date>
```

3. Retrieve the UUIDs for your devices by using the *blkid* command.

```
# blkid
...
...
/dev/sdb1: UUID="a6fed29c-97a0-4636-afa9-9ba23e1319b4" TYPE="ext4"
/dev/sdc1: UUID="rP38Ux-3wzO-sP3Y-2CoD-2TDU-fjpO-ffPFZV" TYPE="ext4"
```

4. Open *fstab* in an editor.
5. For each device that you are using for S3 object storage, replace the device name with *UUID="<UUID>"*, copying the device's UUID from the *blkid* response in the previous step. For example:

```
# Original line

/dev/sdb1 /cloudian1 ext4 rw,noatime,barrier=0,data=ordered,errors=remount-ro  0 1

# Revised line

UUID="a6fed29c-97a0-4636-afa9-9ba23e1319b4" /cloudian1  ext4  rw,noatime,barrier=0,
data=ordered,errors=remount-ro   0 1
```

6. After editing *fstab* so that each device on which you will store S3 data is identified by a UUID, save your changes and close the *fstab* file.
7. Remount the host's file systems:

```
# mount -a
```

Repeat this process for **each host on which you will install HyperStore**.

### 1.5.2.5. A Data Directory Mount Point List (fslist.txt) Is Required

If you do not use the HyperStore *system_setup.sh* script to configure the data disks and mount points on your nodes, you must manually create a data directory mount point list file and place it in your installation staging directory on the Configuration Master node, as described below.

**Note** If you use the *system_setup.sh* script to configure the disks and mount points on your nodes, the script creates the needed mount point list files automatically and you can ignore the instructions below.

If all your nodes have the same data mount points -- for example if all nodes have as their data mount points */cloudian1*, */cloudian2*, and so on through */cloudian12* -- you only need to create one mount point list file. If some nodes have a different set of mount points than do other nodes -- for example if some nodes have more data disks than other nodes -- you will need to create a default mount point list file and also a node-specific mount point list file for each node that differs from the default.

In your installation staging directory create a file named *fslist.txt* and in the file enter one line for each of your S3 data directory mount points, with each line using the format below.

```
<deviceName> <mountPoint>
```

Example of a properly formatted file (truncated):

```
/dev/sdc1 /cloudian1
/dev/sdd1 /cloudian2
...
```

**Note** Use device names in your *fslist.txt* file, not UUIDs.

Optionally, you can also include an entry for the Cassandra data directory and an entry for the Cassandra commit log directory, if you do not want this data to be put on the same device as the operating system (see **"Option for Putting the Metadata DB on Dedicated Drives Rather Than the OS Drives"** (page 47)). If you do not specify these Cassandra directory paths in *fslist.txt*, then by default the system automatically puts Cassandra data and commit log directories on the same device on which the operating system resides.

Do not use symbolic links when specifying your mount points. The HyperStore system does not support symbolic links for data directories.

**If some of your hosts have data directory mount point lists that differ from the cluster default**, in the installation staging directory create a *<hostname>_fslist.txt* file for each such host. For example, along with the default *fslist.txt* file that specifies the mount points that most of your hosts use, you could also have a *cloudian-node11_fslist.txt* file and a *cloudian-node12_fslist.txt* file that specify mount points for two non-standard nodes that have hostnames *cloudian-node11* and *cloudian-node12*.

### 1.5.2.6. Reducing Reserved Space to 0% for HyperStore Data Disks

By default Linux systems reserve 5% of file system space for root user and system services. On modern large-capacity disks this can be a waste of a considerable amount of storage space. Cloudian recommends that you set the reserved space to 0% for each drive on which you will store HyperStore object data (S3 object data).

For each HyperStore data drive do the following.

```
### Check current "Reserved block count":

# tune2fs -l <device>

### Set Reserved block count to 0%:

# tune2fs -m 0 <device>

### For example:

# tune2fs -m 0 /dev/sdc1
```

## 1.5.3. DNS Set-Up

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"HyperStore Service Endpoints"** (page 50)*
- ***"Configuring Resolution of Service Endpoints"** (page 52)*
- ***"Using Customized Service Endpoints"** (page 52)*

For your HyperStore system to be accessible to external clients, you must configure your DNS name servers with entries for the HyperStore service endpoints. **Cloudian recommends that you complete your DNS configuration prior to installing the HyperStore system.** This section describes the required DNS entries.

**Note** If you are doing just a small evaluation and do not require that external clients be able to access any of the HyperStore services, you have the option of using the lightweight domain resolution utility *dnsmasq* which comes bundled with HyperStore -- rather than configuring your DNS environment to support HyperStore service endpoints. During installation of HyperStore software you can use the *configure-dnsmasq* option if you want to use *dnsmasq* for domain resolution. Details are in the software installation procedure.

### 1.5.3.1. HyperStore Service Endpoints

HyperStore includes a variety of services each of which is accessible to clients by way of a web service endpoint. On your name servers you will need to configure a DNS entry for each of these service endpoints.

By default the HyperStore system uses a standard format for each service endpoint, building on two values that are specific to your environment:

- Your organization's domain (for example *enterprise.com*)
- The name or names of your HyperStore service region or regions (for example *boston* for a single-region system, or *boston* and *chicago* for a multi-region system). Only lower case alphanumeric characters and dashes are allowed in region names.

During HyperStore installation you will supply your domain and your service region names, and the interactive installer will show you the default service endpoints derived from the domain and region names. During installation you can accept the default endpoints or specify custom endpoints instead. The table that follows below is based on the default endpoint formats.

**Note**
* Including the string "s3" in your domain or in your region name(s) is not recommended. By default HyperStore generates S3 service endpoints by prepending an "*s3-*" prefix to your *<regionname>.<domain>* combination. If you include "s3" within either your domain or your region name, this will result in two instances of "s3" in the system-generated S3 service endpoints, and this may cause S3 service requests to fail for some S3 clients.
* If you specify custom endpoints during installation, do not use IP addresses in your endpoints.
* HyperStore by default derives the S3 service endpoint(s) as *s3-<regionname>.<domain>*. However HyperStore also supports the format *s3.<regionname>.<domain>* (with a dot after the "s3" rather than a dash) if you specify custom endpoints with this format during installation.

The table below shows the default format of each service endpoint. The examples show the service endpoints that the system would automatically generate if the domain is *enterprise.com* and the region name is *boston*.

| Service Endpoint | Default Format and Example | Description |
|---|---|---|
| S3 service endpoint<br>(**one per service region**) | *s3-<regionname>.<domain>*<br>*s3-boston.enterprise.com* | This is the service endpoint to which S3 client applications will submit requests.<br>If you are installing a HyperStore system across multiple service regions, each region will have its own S3 service endpoint, and therefore you must create a DNS entry for each of those region-specific endpoints — for example *s3-boston.enterprise.com* and *s3-chicago.enterprise.com*. |
| S3 service endpoint wildcard<br>(**one per service region**) | **.s3-<regionname>.<domain>*<br>**.s3-boston.enterprise.com* | This S3 service endpoint wildcard entry is necessary to resolve virtual-hosted-style S3 requests, wherein the bucket name is specified as a sub-domain -- for example *bucket1.s3-boston.enterprise.com* and *bucket2.s3-boston.enterprise.com* and so on. |
| S3 static website endpoint<br>(**one per service region**) | *s3-website-<regionname>.<domain>*<br>*s3-website-boston.enterprise.com* | This S3 service endpoint is used for buckets configured as static websites.<br>**Note** Your S3 static website endpoint cannot be the same as your S3 service endpoint. They must be different, or else the website endpoint will not work properly. |
| S3 static website endpoint wildcard<br>(**one per service region**) | **.s3-website-<regionname>.<domain>*<br>**.s3-website-boston.enterprise.com* | This S3 static website endpoint wildcard entry is necessary to resolve virtual-hosted-style S3 requests, wherein the bucket name is specified as a sub-domain, for buckets configured as static websites. |
| Admin Service endpoint<br>(**one per entire system**) | *s3-admin.<domain>*<br>*s3-admin.enterprise.com* | This is the service endpoint for HyperStore's Admin API. The Cloudian Management Console accesses this API, and you can also access this API directly with a third party client (such as a command line tool like *cURL*). |
| IAM Service endpoint<br>(**one per entire system**) | *iam.<domain>*<br>*iam.enterprise.com* | This is the service endpoint for accessing HyperStore's implementation of the Identity and Access Management API. |
| STS Service endpoint<br>(**one per entire** | *sts.<domain>*<br>*sts.enterprise.com* | This is the service endpoint for accessing HyperStore's implementation of the Security Token Service API. |

| Service Endpoint | Default Format and Example | Description |
|---|---|---|
| **system**) | | **Note** Resolve the STS endpoint to the same address as the IAM endpoint, or use CNAME to map the STS endpoint to the IAM endpoint. |
| SQS Service endpoint<br>(**one per entire system**) | *s3-sqs.<domain>*<br>*s3-sqs.enterprise.com* | This is the service endpoint for accessing HyperStore's implementation of the Simple Queue Service (SQS) API.<br>**Note** The SQS Service is disabled by default. . |
| Cloudian Management Console (CMC) endpoint<br>(**one per entire system**) | *cmc.<domain>*<br>*cmc.enterprise.com* | The CMC is HyperStore's web-based console for performing system administrative tasks. The CMC also supports actions such as creating storage buckets or uploading objects into buckets. |

### 1.5.3.2. Configuring Resolution of Service Endpoints

**IMPORTANT !** Cloudian Best Practices suggest that a highly available load balancer be used in production environments where consistent performance behavior is desirable. For environments where a load balancer is unavailable, other options are possible. Please consult with your Cloudian Sales Engineer for alternatives.

For a production environment, in your DNS configuration each HyperStore service endpoint should resolve to the virtual IP address(es) of two or more load balancers that are configured for high availability. For more detail see **"Load Balancing"** (page 53).

### 1.5.3.3. Using Customized Service Endpoints

If you do not want to use the default service endpoint formats, the HyperStore system allows you to specify custom endpoint values during the installation process. If you intend to create custom endpoints, configure DNS entries to resolve the custom endpoint values that you intend to use, rather than the default-formatted endpoint values shown in the **"HyperStore Service Endpoints"** (page 50) table. Make a note of the custom endpoints for which you configure DNS entries, so that later you can correctly specify those custom endpoints when you perform the HyperStore installation.

If you want to use a **custom S3 endpoint** that does not include a region string, the installer allows you to do so. Note however that if your S3 endpoints lack region strings the system will not be able to support the region name validation aspect of AWS Signature Version 4 authentication for S3 requests (but requests can still succeed without the validation).

If you want to use **multiple S3 endpoints per service region** -- for example, having different S3 endpoints resolve to different data centers within one service region -- the installer allows you to do this. For this

approach, the recommended syntax is *s3-<regionname>.<dcname>.<domain>* -- for example *s3-boston.dc1.enterprise.com* and *s3-boston.dc2.enterprise.com*.

**Note** . If you change any endpoints, be sure to update your DNS configuration.

## 1.5.4. Load Balancing

**IMPORTANT !** Cloudian recommends that a highly available load balancer be used in production environments where consistent performance behavior is desirable. For environments where a load balancer is unavailable, other options are possible. Please consult with your Cloudian Sales Engineer for alternatives. The discussion below assumes that you are using a load balancer.

HyperStore uses a peer-to-peer architecture in which each node in the cluster can service requests to the S3, Admin, CMC, IAM, STS, and SQS service endpoints. In a production environment you should use load balancers to distribute S3, Admin, CMC, IAM, STS, and SQS service endpoint requests evenly across all the nodes in your cluster. In your DNS configuration the S3, Admin, CMC, IAM, STS, and SQS service endpoints should resolve to the virtual IP address(es) of your load balancers; and the load balancers should in turn distribute request traffic across all your nodes. Cloudian recommends that you **set up your load balancers prior to installing the HyperStore system**.

For high availability it is preferable to use two or more load balancers configured for failover between them (as versus having just one load balancer which would then constitute a single point of failure). The load balancers could be commercial products or you can use open source technologies such as **HAProxy** (load balancer software for TCP/HTTP applications) and **Keepalived** (for failover between two or more load balancer nodes). If you use software-defined solutions such as these open source products, for best performance you should install them on dedicated load balancing nodes -- not on any of your HyperStore nodes.

For a **single-region HyperStore system**, for each service configure the load balancers to distribute request traffic across all the nodes in the system.

For a **multi-region HyperStore system**:

- Configure each region's S3 service endpoint to resolve to load balancers in that region, which distribute traffic across all the nodes within that region.
- Configure the Admin, IAM, STS, SQS, and CMC service endpoints to resolve to load balancers in the **default service region**, which distribute traffic to all the nodes in the default service region. (You will specify a default service region during the HyperStore installation process. For example, you might have service regions *boston* and *chicago*, and during installation you can specify that *boston* is the default service region.)

For detailed guidance on load balancing set-up, request a copy of the ***HyperStore Load Balancing Best Practice Guide*** from your Cloudian Sales Engineering representative.

**Note** The HyperStore S3 Service supports **PROXY Protocol** for incoming connections from a load balancer. This is disabled by default, but after HyperStore installation is complete you can enable it by configuration if you wish.

## 1.5.5. HyperStore Listening Ports

The HyperStore system uses the listening ports specified in the table below. Only the service ports for the CMC, S3, IAM, SQS, and Admin services -- the port numbers in italics in the "Listening Port" column -- should be open to traffic originating from outside the HyperStore system. All other ports must be closed to traffic from outside the system, for system security.

Each HyperStore node includes a built-in HyperStore Firewall that implements port restrictions appropriate to a HyperStore cluster. The HyperStore Firewall is disabled by default in HyperStore systems that were originally installed as a version older than 7.2; and enabled by default in HyperStore systems that originally installed as version 7.2 or newer. You can enable/disable the firewall on all HyperStore nodes by using the installer's Advanced Configuration Options.

**Note** If you are installing HyperStore across **multiple data centers and/or multiple service regions**, the HyperStore nodes in each data center and region will need to be able to communicate with the HyperStore nodes in the other data centers and regions. This includes services that listen on the internal interface (such as Cassandra, the HyperStore Service, and Redis). Therefore you will need to configure your networking so that the internal networks in each data center and region are connected to each other (for example, by using a VPN).

| Service | Listening Port | Interface(s) Bound To | Purpose |
|---|---|---|---|
| Cloudian Management Console (CMC) | *8888* | All | Requests from administrators' or end users' browsers over HTTP |
| | *8443* | All | Requests from administrators' or end users' browsers over HTTPS |
| S3 Service | *80* | All | Requests from the CMC or other S3 client applications over HTTP |
| | *443* | All | Requests from the CMC or other S3 client applications over HTTPS |
| | *81* | All | Requests relayed by an HAProxy load balancer using the PROXY Protocol |
| | *4431* | All | Requests relayed by an HAProxy load balancer using the PROXY Protocol with SSL (if enabled by configuration) |
| | 19080 | Internal | JMX access |
| IAM Service and STS Service | *16080* | All | Requests from the CMC or other Identity and Access Management (IAM) or Security Token Service (STS) clients over HTTP |

| Service | Listening Port | Interface(s) Bound To | Purpose |
|---|---|---|---|
| | | | **Note** In the current HyperStore release, the STS Service uses the same listening ports as the IAM Service. |
| | *16443* | All | Requests from the CMC or other IAM or STS clients over HTTPS |
| | 19084 | Internal | JMX access |
| SQS Service | *18090* | All | Requests from Simple Queue Service (SQS) clients over HTTP |
| | *18443* | All | Requests from SQS clients over HTTPS (this is not supported in the current HyperStore release but will be in the future) |
| | 19085 | Internal | JMX access |
| Admin Service | *18081* | All | Requests from the CMC or other Admin API clients over HTTP |
| | *19443* | All | Requests from the CMC or other Admin API clients over HTTPS (Note: The CMC by default uses HTTPS to access the Admin Service)<br>**IMPORTANT !** The Admin Service is intended to be accessed only by the CMC and by system administrators using other types of clients (such as cURL). Do not expose the Admin Service to a public network. |
| | 19081 | Internal | JMX access |
| Redis Monitor | 9078 | Internal | Communication between primary and backup Redis Monitor instances |
| | 19083 | Internal | JMX access |

| Service | Listening Port | Interface(s) Bound To | Purpose |
|---|---|---|---|
| HyperStore Service | 19090 | Internal | Data operation requests from the S3 Service |
| | 19050 | Internal | Communication between HyperStore Service instances |
| | 19082 | Internal | JMX access |
| Credentials DB and QoS DB (Redis) | 6379 | Internal | Requests to the Credentials DB from the S3 Service, HyperStore Service, or Admin Service; and communication between Credentials DB nodes |
| | 6380 | Internal | Requests to the QoS DB from the S3 Service, HyperStore Service, or Admin Service; and communication between QoS DB nodes |
| Metadata DB (Cassandra) | 9042 | Internal | Data operations requests from the S3 Service, HyperStore Service, or Admin Service, using CQL protocol |
| | 9160 | Internal | Data operations requests from the S3 Service, HyperStore Service, or Admin Service, using Thrift protocol |
| | 7000 | Internal | Communication between Cassandra instances |
| | 7199 | Internal | JMX access |
| Cloudian Monitoring Agent | 19070 | Internal | Requests from the Cloudian Monitoring Data Collector |
| Configuration Master | 8140 | Internal | On your Configuration Master node this port will service incoming requests from Puppet agents on your other HyperStore nodes |
| | 4505 | Internal | On your Configuration Master node this is the port to which Salt agents ("minions") establish a persistent connection so that the Master can publish to the minions. |
| | 4506 | Internal | Salt minions connect to this port on the Configuration Master as needed to send results to the Master, and to request files and |

| Service | Listening Port | Interface(s) Bound To | Purpose |
|---|---|---|---|
| | | | minion-specific data values. |
| SSH | 22 | All | The HyperStore installer accesses this SSH port on each node on which you are installing HyperStore software (during initial cluster install or if you subsequently expand your cluster) |
| NTP | 123 | All | NTP port for time synchronization between nodes |
| Echo | 7 | Internal | The Cloudian Monitoring Data Collector uses Echo (port 7) to check whether each node is reachable. |
| HyperIQ | 9999 | All | If you use Cloudian HyperIQ to monitor your HyperStore system, HyperIQ accesses port 9999 on each HyperStore node |

## 1.5.6. Outbound Internet Access

The HyperStore installation process does not require outbound internet access. However, the following HyperStore features do access the internet once the system is in operation; and HyperStore does need access to NTP server(s) during the installation (see the "Pre-Configured ntpd" bullet point below). If you use forward proxying in your environment, after HyperStore installation you may want to set up forward proxying to support these HyperStore features:

- **Smart Support** — The Smart Support feature (also known as "Phone Home") securely transmits HyperStore daily diagnostic information to Cloudian Support over the internet. HyperStore supports configuring this feature to use an explicit forward proxy for its outbound internet access (after installation, the relevant settings in *common.csv* are *phonehome_proxy_host* and the other *phonehome_proxy_** settings that follow after it). To use a forward proxy with this feature you should configure your forward proxy to support access to **.s3-support.cloudian.com* (that is, to any sub-domain of *s3-support.cloudian.com*).
- **Auto-Tiering and Cross-Region Replication** — If you want to use either the auto-tiering feature or the cross-region replication feature (CRR), the S3 Service running on each of your HyperStore nodes requires outbound internet access. These features do not support configuring an explicit forward proxy, but you can use transparent forward proxying if you wish. (Setting up transparent forward proxying is outside the scope of this documentation.)
- **Pre-Configured ntpd** — Accurate, synchronized time across the cluster is vital to HyperStore service. In of your HyperStore data centers four of your HyperStore nodes are automatically configured to act as internal NTP servers. (If a HyperStore data center has only four or fewer nodes, then all the nodes in the data center are configured as internal NTP servers.) These internal NTP servers are configured to connect to external NTP servers — by default the public servers from the *pool.ntp.org* project. In order to connect to the external NTP servers, the internal NTP servers must be allowed outbound internet access. This feature does not support configuring an explicit forward proxy, but you can use transparent

forward proxying if you wish. (Setting up transparent forward proxying is outside the scope of this documentation.)

**IMPORTANT !** If you do not allow HyperStore hosts to have outbound connectivity to the internet, then during the interactive installation process -- when you are prompted to specify the NTP servers that HyperStore hosts should connect to -- you must specify NTP servers within your environment, rather than the public NTP servers that HyperStore connects to by default. If HyperStore hosts cannot connect to any NTP servers, the installation will fail.

After HyperStore installation, to see which of your HyperStore nodes are internal NTP servers, log into the CMC and go to **Cluster** → **Cluster Config** → **Cluster Information**. On that CMC page you can also see your configured list of external NTP servers.

#### 1.5.6.1. Multi-DC Considerations

If you are installing HyperStore across multiple data centers and/or multiple service regions, the HyperStore nodes in each data center and region will need to be able to communicate with the HyperStore nodes in the other data centers and regions. This includes services that listen on the internal interface (such as Cassandra, the HyperStore Service, and Redis). Therefore you will need to configure your networking so that the internal networks in each data center and region are connected to each other (for example, by using a VPN). See **"HyperStore Listening Ports"** (page 54) for HyperStore requirements regarding listening port access.

## 1.6. HyperStore Services

### 1.6.1. HyperStore Services Overview

The Cloudian HyperStoreTM system is composed of several types of services each of which plays a role in implementing the overall HyperStore object storage service. The table below shows the major HyperStore services and how the HyperStore installation script distributes these services across a multi-node cluster. There are common services that are installed to and run on every node, and specialized support services that are installed and run on only one or a sub-set of nodes.

For services distribution diagrams, see **"Services Distribution -- 3 Nodes, Single DC"** (page 69). For information on starting and stopping services, see **"Starting and Stopping Services"** (page 251).

| Service Category | Service | Where It Is Installed |
|---|---|---|
| Common Services | S3 Service | Every node. |
| | HyperStore Service | Every node. |
| | Metadata DBService | Every node. |
| | Admin Service | Every node. |
| | IAM Service | Every node in the default service region. |
| | STS Service | Every node in the default service region. |
| | SQS Service | Every node. |
| | Cloudian Management Console (CMC) | Every node. |
| Specialized Support Services | Credentials DB Master | One node per entire HyperStore system. |
| | Credentials DB Slaves | Two per data center. If you have a large cluster (25 nodes or more in a data center), consult with Cloudian Support about whether you should add more Credentials DB slaves. For instructions on adding slaves, see **"Move or Add a Credentials DB Slave or QoS DB Slave"** (page 306).<br><br>Slaves will not be on same node as the Credentials DB master |
| | QoS DB Master(s) | One node per service region. |
| | QoS DB Slave(s) | One node per data center. Slave will not be on same node as the QoS DB Master. |
| | Redis Monitor | One primary node and one backup node per entire HyperStore system. |
| | Crontab configuration and Monitoring Data Collector | One primary node and one backup node per service region. |
| | Local NTP server | One local NTP server per service region. |
| | Configuration Master | One primary node and one backup node per entire |

| Service Category | Service | Where It Is Installed |
|---|---|---|
| | | HyperStore system. |
| Auxiliary Services | HyperStore File Service | Optional service that can be installed on auxiliary nodes after the core HyperStore system is installed. |
| | HyperStore Search Service | Optional service that can be installed on auxiliary nodes after the core HyperStore system is installed. |

**Note** Within your installation cluster, the HyperStore installer **automatically chooses the hosts** for specialized support services that are not intended to run on every node. These host assignments are recorded to an installation configuration file that the installer generates when it runs (*CloudianInstallConfiguration.txt* in your installation staging directory).After your installation is completed, these host assignments can also be viewed on the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). If you want to modify these assignments after install, see **"Change Node Role Assignments"** (page 305).

## 1.6.2. Cloudian Management Console (CMC) Service

The Cloudian Management Console (CMC) is a web-based user interface for Cloudian HyperStore system administrators, group administrators, and end users. The functionality available through the CMC depends on the user type associated with a user's login ID (system admin, group admin, or regular user).

As a HyperStore system administrator, you can use the CMC to perform tasks such as:

- Monitoring the system and performing system operations
- Provisioning groups and users
- Managing quality of service (QoS) controls
- Creating and managing rating plans
- Generating usage data reports
- Generating bills
- Viewing and managing users' stored data objects (if you've configured the system to allow this)

Group administrators can perform a more limited range of admin tasks pertaining to their own group, and can also perform S3 operations such as creating and configuring buckets and uploading and downloading objects. Regular users can only perform S3 operations such as creating and configuring buckets and uploading and downloading objects.

The CMC acts as a client to several of the **API Services** including the Admin Service and the S3 Service.

## 1.6.3. API Services

### 1.6.3.1. Admin Service

The HyperStore Admin Service implements a RESTful HTTP API through which you can perform administrative operations such as:

- Provisioning groups and users
- Managing quality of service (QoS) controls
- Creating and managing rating plans
- Generating usage data reports
- Generating bills
- Retrieving system metrics and monitoring data

The **"Cloudian Management Console (CMC) Service"** (page 60) is a client to the Admin Service. You also have the option of using a command line tool such as cURL to submit Admin API commands, or building your own Admin Service client.

For more information about the Admin API see the ***Cloudian HyperStore Admin API Reference***.

### 1.6.3.2. AWS Compliant API Services

The HyperStore **S3 Service** provides comprehensive support for the AWS Simple Storage Service API.

The HyperStore **IAM, STS, and SQS Services** provide partial support for the AWS Identity and Access Management API, the AWS Security Token Service API, and the AWS Simple Queue Service API, respectively.

For the S3 Service and IAM Service you can use the **"Cloudian Management Console (CMC) Service"** (page 60) as a client application, or use a third party or custom client application. For the STS and SQS Services, the CMC does not provide client access and so you must use third party or custom client applications to access these services.

For more information about HyperStore's implementation of these services see the ***Cloudian HyperStore AWS APIs Support Reference***.

## 1.6.4. Database Services

### 1.6.4.1. Metadata DB

The HyperStore system stores object metadata, user metadata, and system metadata in the Metadata DB. The Metadata DB is built on the Apache open source storage platform **Cassandra**. The HyperStore system creates and uses several "keyspaces" within Cassandra:

- The **UserData_<policyid>** keyspaces store:
  - User bucket information
  - Object metadata. For an overview of the HyperStore system's support for object metadata, see **"Object Metadata and Search Feature Overview"** (page 196).

> **Note** There is one *UserData_<policyid>* keyspace for each storage policy in the system. For information about storage policies see **"Storage Policies Feature Overview"** (page 127).

- The **AccountInfo** keyspace stores information about HyperStore S3 user accounts and group accounts (including IAM user and group accounts)
- The **Reports** keyspace stores system-wide, per-group, and per-user S3 usage data, in support of the HyperStore usage reporting functionality. It will also store per-bucket usage data if you **enable per-bucket usage tracking**.
- The **Monitoring** keyspace stores system monitoring statistics in support of HyperStore's system monitoring functionality.
- The **ECKeyspace** keyspace does not actually store any erasure coded object data; rather, the HyperStore system creates this keyspace so that the HyperStore erasure coding feature can leverage Cassandra functions for token-based mapping of objects (erasure coded object fragments, in this case) to nodes within the storage cluster.
- The **Notification** keyspace stores bucket notification messages. For more information see the SQS section of the ***Cloudian HyperStore AWS APIs Support Reference***.

S3 client applications do not access the Metadata DB directly. Instead, all S3 client access is to the **S3 Service**, which in turn accesses the Metadata DB in support of S3 operations. The **HyperStore Service** and **Admin Service** also access the Metadata DB.

### 1.6.4.2. Credentials DB and QoS DB

The **Credentials DB** stores user credentials and additional S3 operation supporting metadata such as multipart upload session information and public URL access counters.

The **QoS DB** stores user-level and group-level **Quality of Service** settings that have been established by system administrators. The QoS DB is also used to keep count of user requests, so that Quality of Service limits can be enforced by the system.

The Credentials DB and QoS DB are both built on **Redis**, an open source, in-memory key-value data store optimized for fast performance.

The S3 Service, Admin Service, and HyperStore Service are the clients to the Credentials DB and QoS DB. Communication is through a protocol called Redis Serialization Protocol (RESP).

In a multi-region HyperStore deployment there will be:

- Just one, universal Credentials DB which serves the entire HyperStore deployment.
- A separate, independent QoS DB in each service region

#### Credentials DB and QoS DB Node Roles

Each Credentials DB and each QoS DB is implemented across two or more nodes, with the nodes playing different roles. These roles are:

- **master** — All write requests from DB clients are implemented on the master node. There is only one master node for each Credentials DB and each QoS DB. In a multi-region HyperStore deployment, the universal Credentials DB has one master node and each regional QoS DB has its own master node.
- **slave** — In each Credentials DB and each QoS DB, data from the DB master node is asynchronously replicated on to one or more slave nodes (at least one slave node per data center). The slave nodes

support doing reads for DB clients but not writes. If a master node fails, the master role is automatically failed over to a slave node. This fail-over process is managed by the **Redis Monitor Service**.

Credentials DB and QoS DB roles are assigned to your HyperStore nodes automatically during installation. For more information on the distribution of these services across the cluster see **"Services Distribution -- 3 Nodes, Single DC"** (page 69).

#### 1.6.4.3. Checksum DB

The Checksum DB stores digests (MD5 hashes) of the object data files stored in the HyperStore File System (HSFS). For more information see **"HyperStore Service and the HSFS"** (page 63), particularly the section on "File Digests".

### 1.6.5. HyperStore Service and the HSFS

The Cassandra storage platform that underlies the **Metadata DB** provides valuable built-in data management functionality including data partitioning, automatic replication, easy cluster expansion, quorum calculation, and so on. For storing small data items, Cassandra also provides good performance. But as the data size increases, storing data on the Linux file system becomes more efficient than storing it in Cassandra.

The HyperStore system uses a hybrid storage solution where Cassandra is used for storing metadata while the Linux filesystem on Cassandra nodes is used for storing object data. The area of the Linux file system where S3 object data is stored is called the **HyperStore File System (HSFS)**.

The general strategy is that Cassandra capabilities are used to determine the distributed data management information such as the nodes that a specific object's metadata should be written to and the nodes that the object's data should be written to. Then at the storage layer, the metadata is stored in Cassandra and the object data is stored in the HSFS.

Within the HSFS, objects can be stored and protected in either of two ways:

- Replicated storage
- Erasure coded storage

For more information on data storage and protection options, see **"Storage Policies Feature Overview"** (page 127).

When the system stores S3 objects, the full path to the objects will be as indicated below:

- For S3 object replicas:

```
<mountpoint>/hsfs/<base62-encoded-vNode-token>/<policyid>/<000-255>/<000-255>/<filename>
```

- For S3 object erasure coded fragments:

```
<mountpoint>/ec/<base62-encoded-vNode-token>/<policyid>/<000-255>/<000-255>/<filename>
```

- The path segments are:
  - The <mountpoint> is one of your HyperStore data mount points as configured by the **"hyperstore_data_directory"** (page 394) setting in *common.csv*.
  - The *hsfs* or *ec* segment distinguishes replicated data (designated here as "hsfs") from erasure-coded data (designated as "ec).
  - The *<base62-encoded-vNode-token>* is a base-62 encoding of the token belonging to the **vNode** to which the object replica or erasure coded fragment is assigned.

- The *<policyid>* segment indicates the storage policy used by the S3 storage bucket with which the object is associated.
- The two *<000-255>* segments of the path are based on a hash of the *<filename>*, normalized to a 255*255 number.
- The *<filename>* is a dot-separated concatenation of the object's system-assigned token and a timestamp based on the object's Last Modified Time. The token is an MD5 hash (in decimal format) of the bucket name and object name. The timestamp is formatted as *<UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>*. The last element of the timestamp is the IP address (in hexadecimal format) of the S3 Service node that processed the object upload request.

> **Note** For objects last modified prior to HyperStore version 6.1, the timestamp is simply Unix time in milliseconds. This was the timestamp format used in HyperStore versions 6.0.x and older.

- Example, for a replicated object named "HyperStoreAdminGuide.pdf":

```
/hyperstore1/hsfs/1L1tEZZCCQwdQBdGel4yNk/c4a276180b0c99346e2285946f60e59c/109/154/
5589877948126853572620057491660937218 1.1487608689783689800-0A320A15
```

In the above example:

- "hyperstore1" is one of the HyperStore data mount points configured for the system (as specified by the configuration setting *common.csv: hyperstore_data_directory*)
- "hsfs" indicates that the object is a replicated object (not an erasure-coded object)
- "1L1tEZZCCQwdQBdGel4yNk" is the Base-62 encoding of the token belonging to the vNode to which the object instance is assigned
- "c4a276180b0c99346e2285946f60e59c" is the system-generated identifier of the storage policy used by the S3 storage bucket with which the object is associated.
- "109/154" is a hash of the file name, normalized to a 255*255 number.
- "5589877948126853572620057491660937218 1.1487608689783689800-0A320A15" is the file name. The "5589877948126853572620057491660937218 1" segment is the object's system-assigned token, in decimal format. The "1487608689783689800-0A320A15" segment is the object's Last Modified Time timestamp, in format *<UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>*.

> **Note** Presuming that **versioning** is disabled (as it is by default), when an S3 client uploads an updated version of an object the system will overwrite the existing replica file with the new version. The token segment of the file name will remain constant (the object keeps the same token) and the timestamp segment of the file name will change.

### 1.6.5.1. File Digests

Each replica file and each erasure coded fragment file has a corresponding digest containing the hexadecimal MD5 hash of the file as well as a small amount of metadata including the object name (the name of the object for which the file is a replica or an erasure coded fragment) and a last modified timestamp. These digests are used by the HyperStore Service when reading and writing objects and are also used by **"hsstool "** (page 503) operations such as repair and cleanup. The digests are stored in a Checksum DB on the same mount point as

the corresponding file. On each mount point, there is one Checksum DB for storing digests for replica data files, and one Checksum DB for storing digests for erasure coded data files. The Checksum DBs are built on **RocksDB**, an open source, high-performance persistent key-value store.

For replica data files, the Checksum DB is stored under:

```
<mountpoint>/digest/hsfs/
```

For erasure coded data files, the Checksum DB is stored under:

```
<mountpoint>/digest/ec/
```

Within each Checksum DB, the *key* is a byte array consisting of the object's token and the file timestamp in binary format, and the *value* is the digest itself.

**Note**

- When an existing object is updated by an S3 client, the object's token (a decimal formatted MD5 hash of the object key) remains the same but the object's digest (including a hexadecimal formatted MD5 hash of the object data) changes.
- For an erasure coded S3 object, each fragment has the same token (based on object key) but a different digest (based on fragment content).
- For multipart S3 objects — uploaded to the system through the S3 API methods for Multipart Uploads — each part has a different token (since each part has a distinct object key incorporating a part number) and a different digest (based on part content).

## Retrieving a Digest

The HyperStore system supports a JMX command for retrieving a digest from a particular node:

- HyperStore Service listener port = 19082
- MBean = *com.gemini.cloudian.hybrid.server.digest:type=RocksDBDigestStore*
- Operation and arguments = *getDigestString=<bucketName>/<objectName>*

For example, using the command line JMX tool *cmdline-jmxclient* that comes bundled with your HyperStore system:

```
[root]# java -jar /opt/cloudian/tools/cmdline-jmxclient-*.jar -:- localhost:19082
com.gemini.cloudian.hybrid.server.digest:type=RocksDBDigestStore
getDigestString=bucket1/LocalInstallProcedure.docx

10/25/2016 06:27:30 -0700 org.archive.jmx.Client
getDigestString=bucket1/LocalInstallProcedure.docx:
688556844694319500929824031832021824 39.1477401010696032189-0A0A1608
9c741e3e7bbe03e05510071055151a6e
bucket1/LocalInstallProcedure.docx
2016-10-25T13:10:10.696Z
/var/lib/cloudian/hsfs/1mDFFH13tL1DIsYhNKDX3d/a7a28896654319cc7af4c39748a27e3d/243/187/
688556844694319500929824031832021824 39.1477401010696032189-0A0A1608
13164
```

For clarity, in the example above an empty line has been inserted between the JMX command and the response. This example is for a replicated object named *LocalInstallProcedure.docx* from the bucket named *bucket1*. In the response, *688556844694319500929824031832021824 39.1477401010696032189-0A0A1608* is the Checksum DB key for this entry (in format *<objectToken>.<timestamp>*). The subsequent lines are the digest contents. *9c741e3e7bbe03e05510071055151a6e* is the replica's MD5 hash in

hexadecimal; the */var/lib/cloudian/hsfs/...* line is the replica file path and name; and *13164* is the replica's file size in bytes.

**Note** In the command line example above, -:- is the USER:PASS value (indicating that the system is not configured to require a userId and password for JMX access). For *cmdline-jmxclient* usage information, enter the following command:

```
[root]# java -jar /opt/cloudian/tools/cmdline-jmxclient-*.jar
```

To check the current *cmdline-jmxclient* version number (replaced by the wildcard character in the command above), change to the */opt/cloudian/tools* directory and list the directory contents. Look for the *cmdline-jmxclient-<version>.jar* file.

## 1.6.6. Supporting Services

Services that play a supporting role for the HyperStore system include:

- **Configuration Master and Agents** — HyperStore's cluster configuration management system is built on the open source version of **Puppet**. For more information see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382). A Configuration Agent runs on every node. The Configuration Master runs on one node and is also configured on a backup node. Manual failover to the backup is supported if the primary Configuration Master instance goes down.
- **Cloudian Monitoring Data Collector and Agents** — The Cloudian Monitoring Data Collector runs on one node in each of your service regions (the same node on which the system maintenance cron jobs run), and regularly collects data from the Monitoring Agents that run on every node. The Monitoring Collector writes its collected node health statistics to the Metadata DB's "Monitoring" keyspace. The Monitoring Collector and the system maintenance cron jobs are also configured on a backup node, and automatic failover to the backup occurs if the primary node goes offline or if *crond* goes down on the primary.
- **Redis Monitor** — The **Credentials DB and QoS DB** are both built on **Redis**. The Redis Monitor monitors Credentials DB and QoS DB cluster health and implements automatic failover of the master node role within each of the DBs. If the Redis Monitor detects that a DB master node has gone down, it promotes an available slave node to the master node role; and informs the DB's clients (the S3 Service, IAM Service, Admin Service, and HyperStore Service) of the identity of the new master. For redundancy, the Redis Monitor runs on two HyperStore nodes, configured as primary on one node and as backup on the other node.
- **Pre-Configured *ntpd*** — Accurate, synchronized time across the cluster is vital to HyperStore service. When you install your HyperStore cluster, the installation script automatically configures a robust NTP set-up using *ntpd*. In each HyperStore data center four of your HyperStore nodes are automatically configured to act as internal NTP servers, which synchronize with external NTP servers (by default the servers from the *pool.ntp.org* project). Other HyperStore hosts in each data center are configured as clients of the internal NTP servers. For more information see **"NTP Automatic Set-Up"** (page 496). To see which of your HyperStore nodes are internal NTP servers and which external NTP servers they are synchronizing with, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

  **Note** If a HyperStore data center has only four or fewer nodes, then all the nodes in the data

center are configured as internal NTP servers.

- **Dnsmasq** — Dnsmasq is a lightweight domain resolution utility. This utility is bundled with Cloudian HyperStore software. The HyperStore interactive installation wizard gives you the option to have *dnsmasq* installed and configured to resolve HyperStore service domains (specifically the S3 service domain, the S3 website endpoint domain, and the CMC domain). The *dnsmasq* utility may be helpful if you are evaluating a small HyperStore system but it is not appropriate for production use.

### 1.6.7. Auxiliary Services

HyperStore supports two types of "auxiliary" services. These are optional HyperStore components that you can install separately on auxiliary nodes (which have lesser resource requirements than the core HyperStore nodes) after the core HyperStore system has been installed:

- **HyperStore File Service**. This component enables your HyperStore system to support Server Message Block (SMB) and Network File System (NFS) based file services. For more information see:
  - **"File Services Feature Overview"** (page 207)
  - **"Preparing the File Services Feature"** (page 208)
- **HyperStore Search Service**. With the HyperStore Search Service your HyperStore users can search for objects based on object metadata, including user-defined metadata. For more information see:
  - **"Object Metadata and Search Feature Overview"** (page 196)
  - **"Preparing the Metadata Search Feature"** (page 202)

## 1.7. System Diagrams

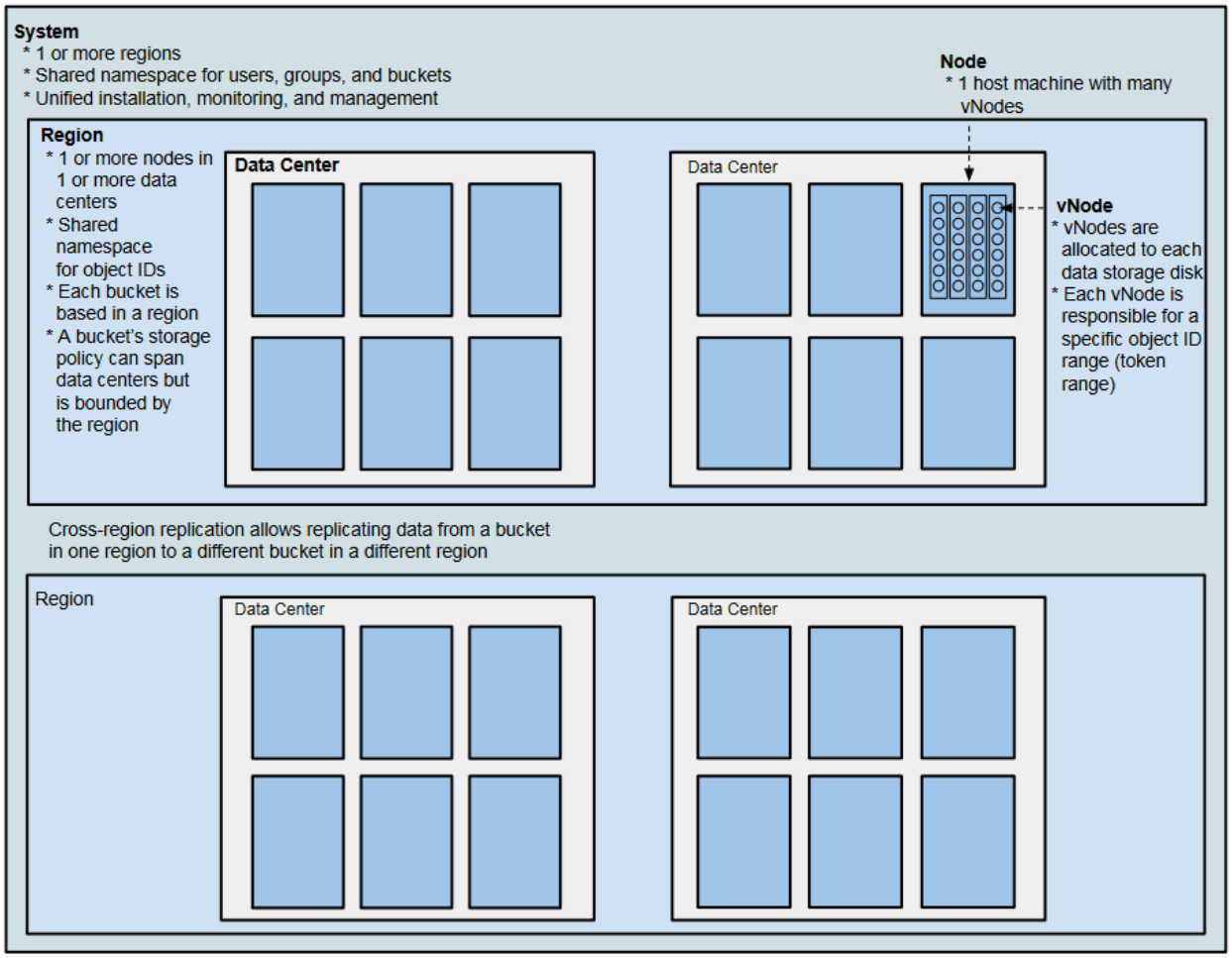


### 1.7.1. System Levels

The diagram below shows the conceptual and functional distinctions between the "levels" within a HyperStore system. From broadest to most granular the levels are:

- **System**
- **Region** (also known as a "**Cluster**")

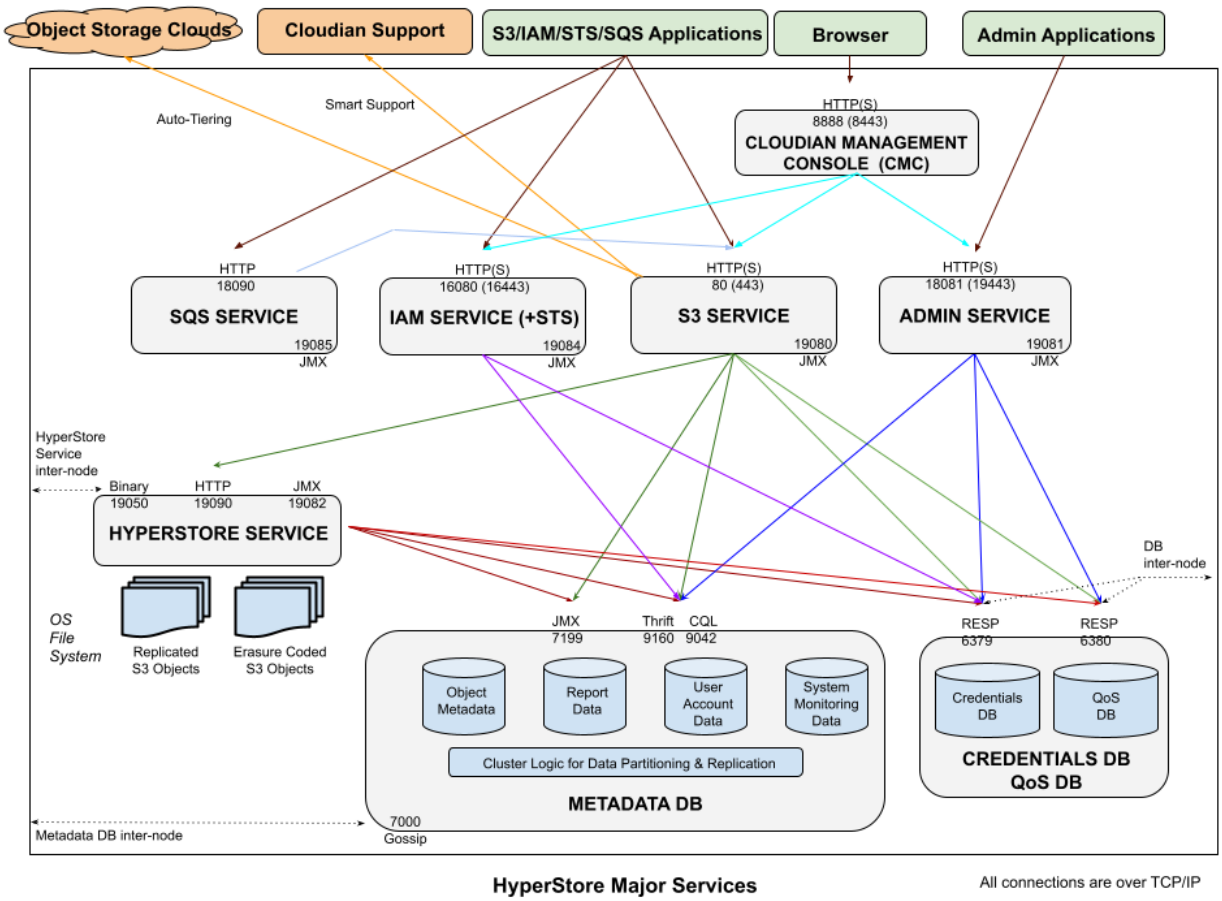
- **Data Center**
- **Node**
- **vNode**




**HyperStore System Levels**

## 1.7.2. Service Interconnections

The diagram below shows the major service components that comprise the core HyperStore system, the connections between those services, and the default listening ports to which connections are made.



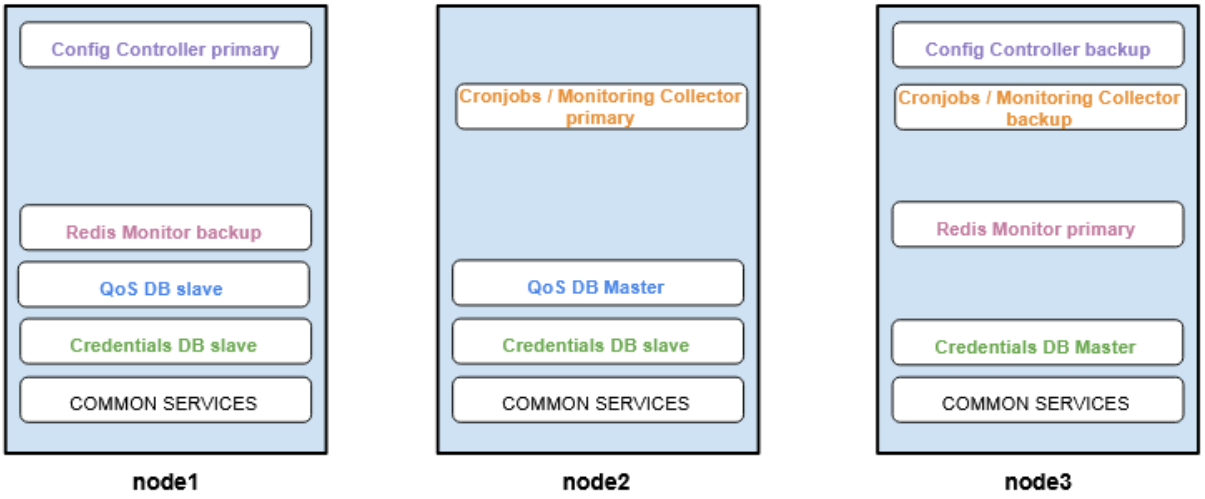

**Note** The diagram excludes certain supporting services such as the Redis Monitor, the Monitoring Data Collector, and the Configuration Master and Agents. For a complete list of HyperStore services and the listening ports they use, see **"HyperStore Listening Ports"** (page 54).

## 1.7.3. Services Distribution -- 3 Nodes, Single DC

Proper distribution of HyperStore service components across multiple physical nodes is handled automatically by the HyperStore installer. The diagram below shows a typical HyperStore services distribution in a three-node cluster within a single data center (DC). Things to note:

- On every node in your cluster are the S3 Service, IAM Service, Admin Service, HyperStore Service, Metadata DB, CMC, Configuration Agent, and Monitoring Agent. These collectively are labeled as "COMMON SERVICES" in the diagram.
- For each specialized service that has a primary instance and a backup instance (such as Configuration Master or Redis Monitor), the backup resides on a different node than the primary. Likewise the QoS DB slave will reside on a different node than the QoS DB Master, and the two Credentials DB slaves will reside on different nodes than the Credentials DB Master.
- If you have a **larger cluster** in a single DC, you will still have the same number of specialized service instances as shown in the diagram (for example, one primary Configuration Master instance and one backup instance) — the only difference is that this set of specialized service instances will be spread across your cluster rather than concentrated among three nodes as shown in the diagram. For example

if you have five or more nodes in the DC, then the Credentials DB Master, the two Credentials DB slaves, the QoS DB Master, and the QoS DB slave will all be on different nodes.




**HyperStore Services Distribution -- Three Node System**

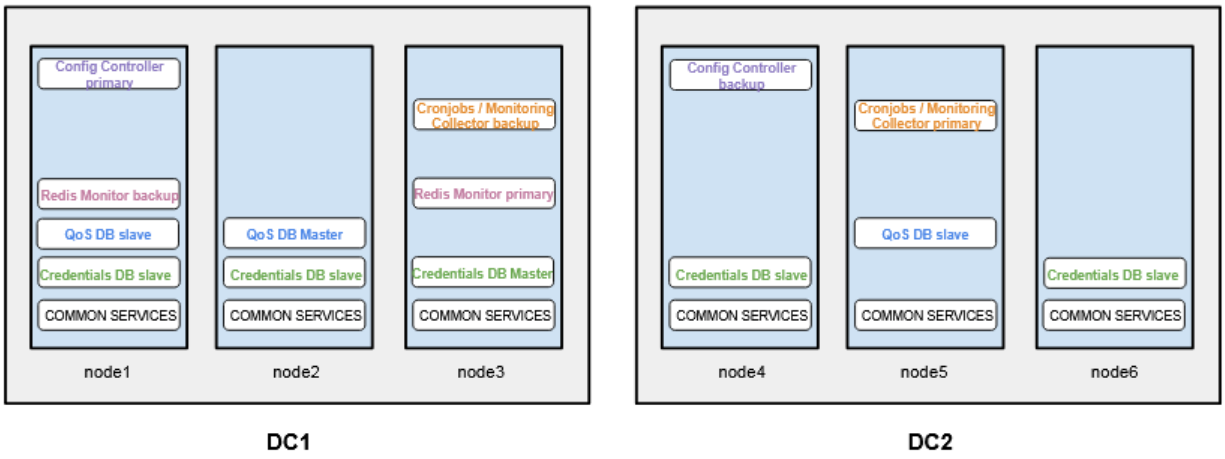
**Note** For information about the exact location of services in your HyperStore system, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). The system allows you to move services from one host to another, if you wish to do so. For instructions see **"Change Node Role Assignments"** (page 305).

**Note** If you have a very large cluster (25 nodes or more in a data center), consult with Cloudian Support about whether you should add more Credentials DB slaves. For instructions on adding Credentials DB slaves, see **"Move or Add a Credentials DB Slave or QoS DB Slave"** (page 306).

## 1.7.4. Services Distribution -- Multi-DC, Single Region

Proper distribution of HyperStore service components across multiple physical nodes is handled automatically by the HyperStore installer. The diagram below shows a typical HyperStore services distribution across a six-node system that spans two data centers. The system is configured as a single service region. Things to note:

- The "COMMON SERVICES" (S3 Service, IAM Service, Admin Service, HyperStore Service, Metadata DB, CMC, Configuration Agent, and Monitoring Agent) run on every node in your multi-DC system.
- Each data center has its own QoS DB slave and its own two Credentials DB slaves, for read performance optimization.
- The Configuration Master backup is placed in a different DC than the Configuration Master primary; and the same is true for the Cronjobs backup and primary.




**HyperStore Services Distribution -- Two Data Centers Configured as a Single Service Region**

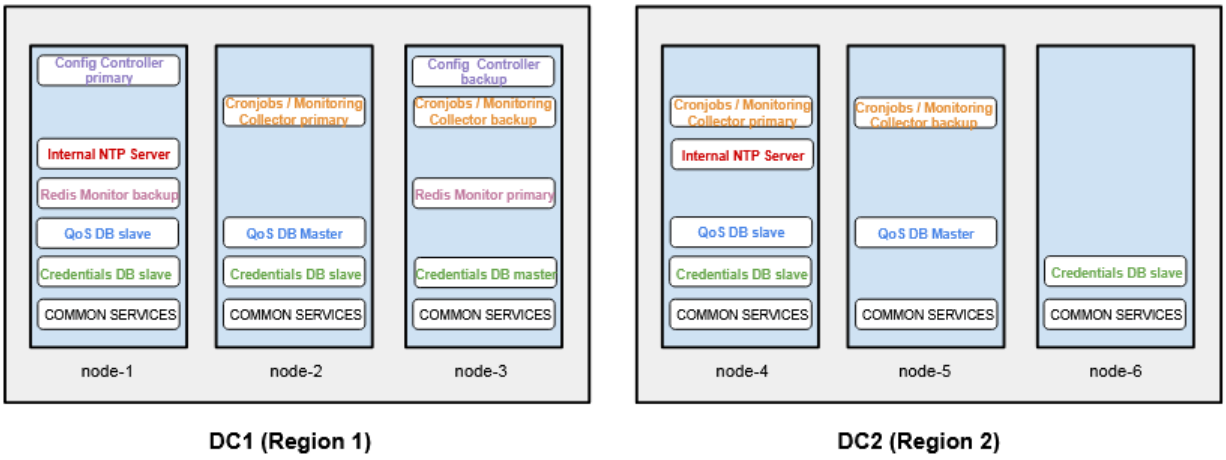
**Note** The Redis Monitor backup must remain in the same data center as the Redis Monitor primary, and this should be the same data center as where the Credentials DB master is located.

**Note** To check the current location of specialized services within your multi-DC HyperStore system, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

## 1.7.5. Services Distribution -- Multi-Region

Proper distribution of HyperStore service components across multiple physical nodes is handled automatically by the HyperStore installer. This diagram shows a six-node system that spans two data centers, and this time the system is configured as two different service regions. Things to note:

- The "COMMON SERVICES" (S3 Service, Admin Service, HyperStore Service, Metadata DB, CMC, Configuration Agent, and Monitoring Agent) run on every node in your multi-region system. The IAM Service runs on every node in the default region only.
- The whole multi-region system is served by a single active Configuration Master, a single Credentials DB Master, and a single active Redis Monitor.
- Each region has its own QoS DB Master and its own active Cronjob host.



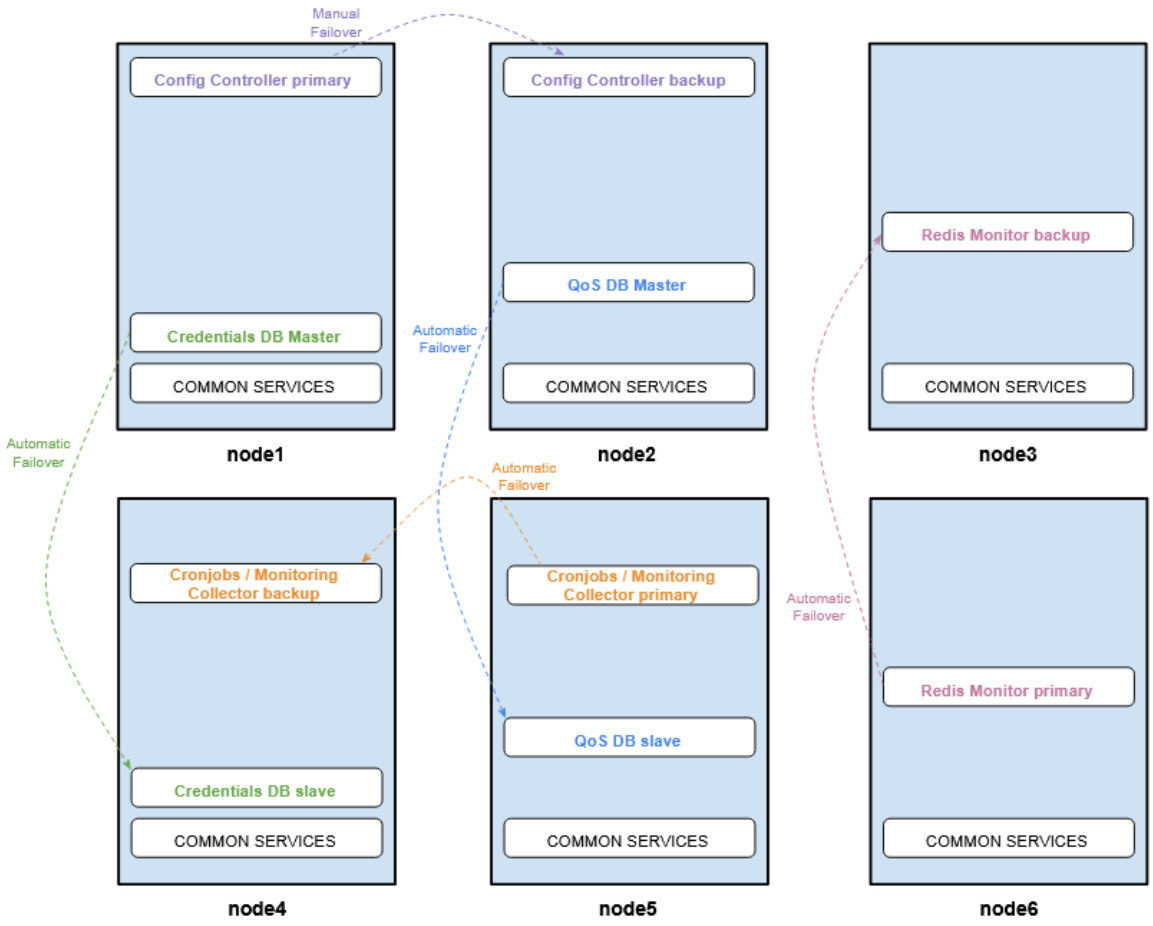

**HyperStore Services Distribution -- Two Data Centers Configured as Different Service Regions**

## 1.7.6. Specialized Services Availability

Along with the services that are common to every HyperStore node (such as the S3 Service, HyperStore Service, Metadata DB, and so on) your HyperStore system includes several specialized services that run only on certain nodes. The HyperStore installer **automatically distributes these services across your cluster**. Each specialized service has a primary instance and a backup instance, and the installer ensures that for each specialized service the primary instance and backup instance are deployed on different nodes.

The diagram below illustrates how the system ensures high availability of these specialized services by supporting failover of each service type, from the primary instance to the backup instance. For nearly all service types, the system automatically detects a failure of the primary instance and automatically fails over to the backup instance. The one exception is the Configuration Master role (for managing system configuration) — in the case of the Configuration Master you can **manually implement failover** if there's a problem with the primary instance.

The diagram shows six nodes, but the principles are the same regardless of how many nodes you have: specialized services are dispersed across the cluster, and the backup instance of any given service is deployed on a different node than the primary instance.



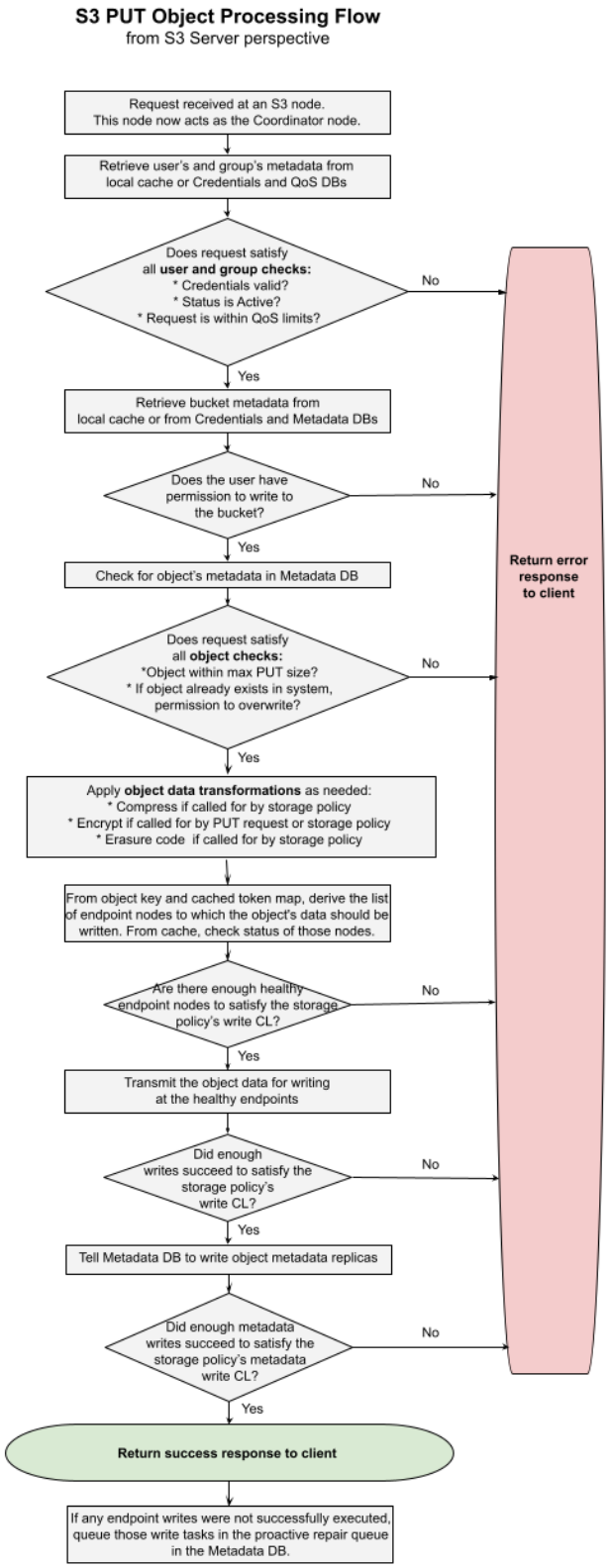

**HyperStore Specialized Services -- High Availability**

**Note** The automatic failover of the Cronjobs and Monitoring Data Collector roles from the primary to the backup instance invokes the *cloudianInstall.sh* script and will fail if *cloudianInstall.sh* is already running. When you occasionally use *cloudianInstall.sh* for system configuration tasks, remember to exit the installer when you are done — do not leave it running.

Also, the automatic failover of the Cronjobs and Monitoring Data Collector roles from the primary to the backup instance will not occur until the primary instance has been down for 10 minutes.

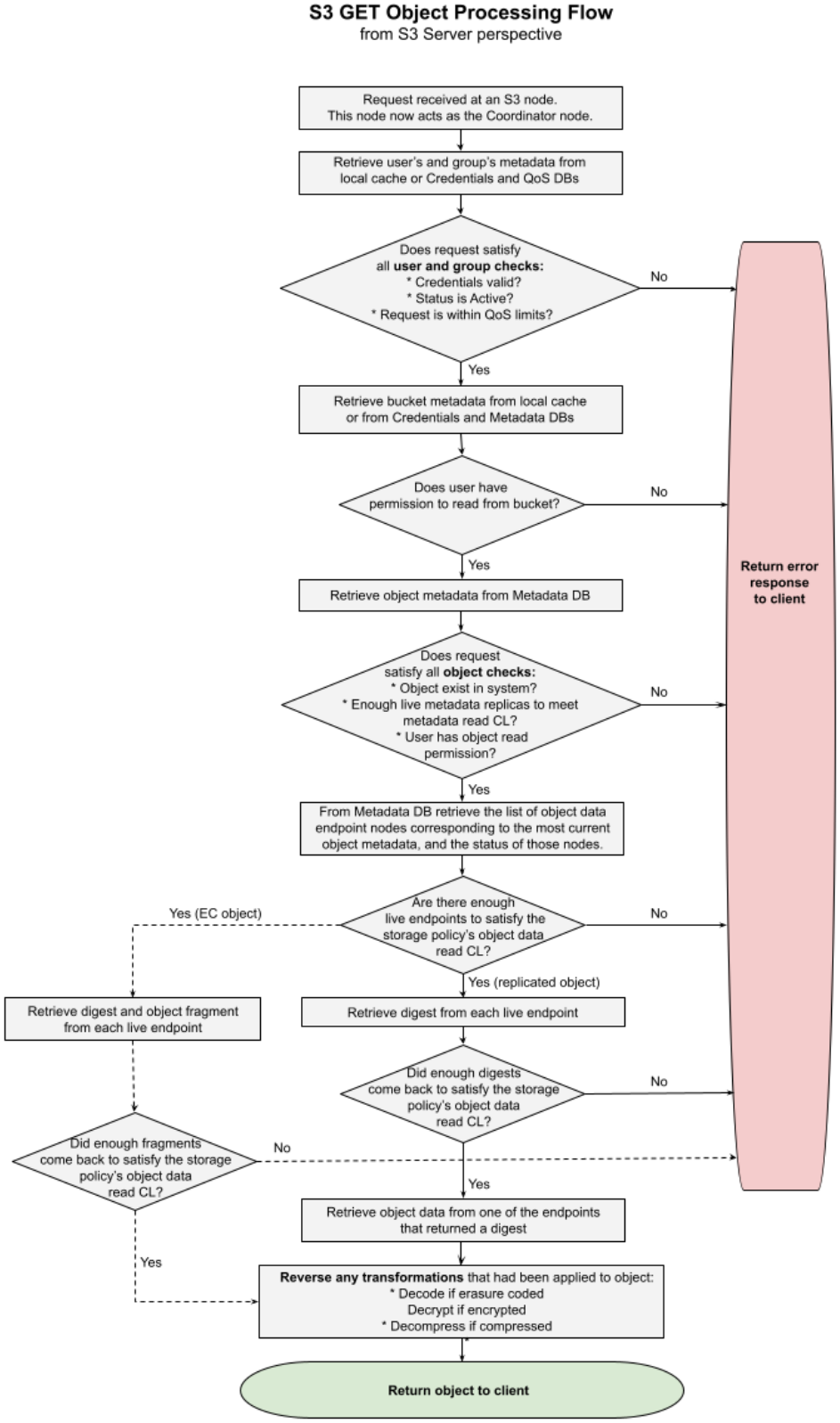
## 1.7.7. S3 PUT Processing Flow

The diagram below shows the main aspects of how the HyperStore system processes an S3 PUT Object request. The flow is presented from the perspective of the S3 Service, which handles incoming S3 requests. The S3 Service runs on all of the nodes in your cluster.


S3 PUT Object Processing Flow
from S3 Server perspective
Request received at an S3 node.
This node now acts as the Coordinator node.
Retrieve user's and group's metadata from
local cache or Credentials and QoS DBs
Does request satisfy
all user and group checks:
* Credentials valid?
* Status is Active?
* Request is within QoS limits?
No
Yes
Retrieve bucket metadata from
local cache or from Credentials and Metadata DBs
Does the user have
permission to write to
the bucket?
No
Yes
Check for object's metadata in Metadata DB
Does request satisfy
all object checks:
*Object within max PUT size?
* If object already exists in system,
permission to overwrite?
No
Yes
Apply object data transformations as needed:
* Compress if called for by storage policy
* Encrypt if called for by PUT request or storage policy
* Erasure code if called for by storage policy
From object key and cached token map, derive the list
of endpoint nodes to which the object's data should be
written. From cache, check status of those nodes.
Are there enough healthy
endpoint nodes to satisfy the storage
policy's write CL?
No
Yes
Transmit the object data for writing
at the healthy endpoints
Did enough
writes succeed to satisfy the
storage policy's
write CL?
No
Yes
Tell Metadata DB to write object metadata replicas
Did enough metadata
writes succeed to satisfy the
storage policy's metadata
write CL?
No
Yes
Return success response to client
If any endpoint writes were not successfully executed,
queue those write tasks in the proactive repair queue
in the Metadata DB.
Return error
response
to client

## 1.7.8. S3 GET Processing Flow

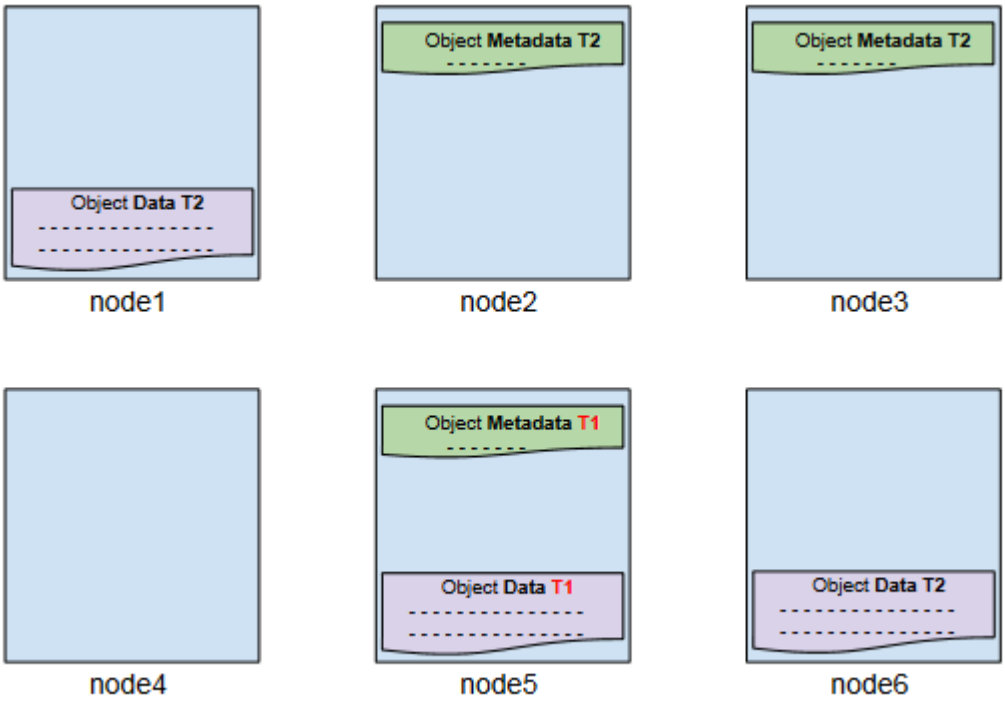
This diagram shows the main aspects of how the HyperStore system processes an S3 GET Object request, from the perspective of the S3 Service.



## 1.7.9. Strong Read-After-Write Consistency

Typically all the replicas of a given object, and all the replicas of that object's metadata, will be consistent — that is, all the replicas will be equally current. However, because HyperStore allows you to configure storage policies that utilize eventual consistency for writes, there may be times when an object's data replicas and/or metadata replicas are temporarily inconsistent. If a read request on the object comes into the system during such a time, by default HyperStore either returns the freshest data or — if no fresh replica is available — fails the request. In this way HyperStore by default ensures **strong read-after-write consistency**.

Consider a 3X replication scenario where QUORUM has been used as the write consistency level (which is the default configuration for replication storage policies). Suppose an S3 PUT of an updated version of an object has succeeded even though only two of three object data replica writes and only two of three object metadata replica writes succeeded. We then can temporarily have a condition like that shown in the following diagram, where "T2" indicates the timestamp of the new version of the data and metadata and "T1" indicates the out-dated version. (For example, perhaps *node5* was momentarily offline when the S3 write request came in; and now it's back online but **proactive repair** has not yet completed.)




If an S3 read request on the object comes into the system during this temporary period of stored data inconsistency, the system works as follows:

- As long as the read consistency level is set to at least QUORUM (the default for replication storage policies), the system will read at least two of the metadata replicas. Consequently it will read at least one of the fresh metadata replicas, with timestamp T2. If it reads one T1 metadata replica and one T2 metadata replica, it works with the metadata that has the freshest timestamp. The system then tries to retrieve an object data replica that has this same fresh timestamp.
- If object data replicas with the fresh timestamp are available, that object data is returned to the S3 client. If nodes are down in such a way that the only available object data replica is the outdated one, then the system fails the S3 request.

In this way, by default configuration HyperStore delivers strong consistency (specifically, strong read-after-write consistency).

**Note** Although not pictured here, HyperStore delivers strong read-after-write consistency for erasure coding storage policies by similar logic (in the case of erasure coded data, each of the *k* fragments required for an object data read must match against the newest timestamp found in the successful QUORUM read of object metadata).

**Note** HyperStore allows you to configure replication storage policies that use a read CL of ONE rather than QUORUM. This non-default configuration maximizes read availability and speed, but also increases the chances of returning a stale replica to the client. This is because -- if your write CL is QUORUM (the default) and your read CL is ONE (non-default) -- there is a chance of reading only a stale metadata replica and returning a stale object replica to the client.

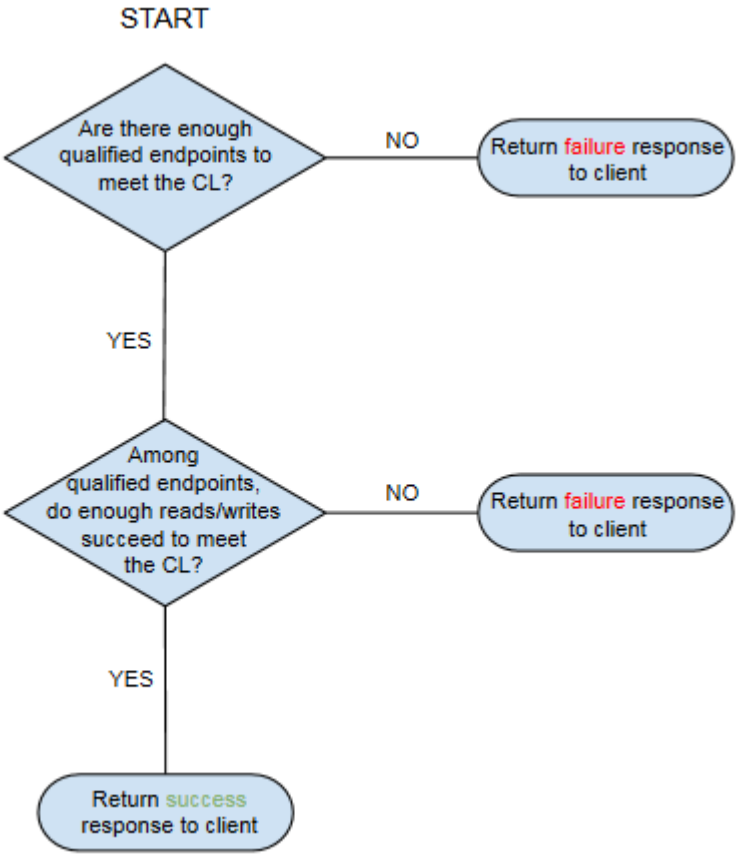
For more information on S3 write and read availability under various consistency level configurations, see **"Storage Policy Resilience to Downed Nodes"** (page 136).

## 1.7.10. Dynamic Consistency Levels

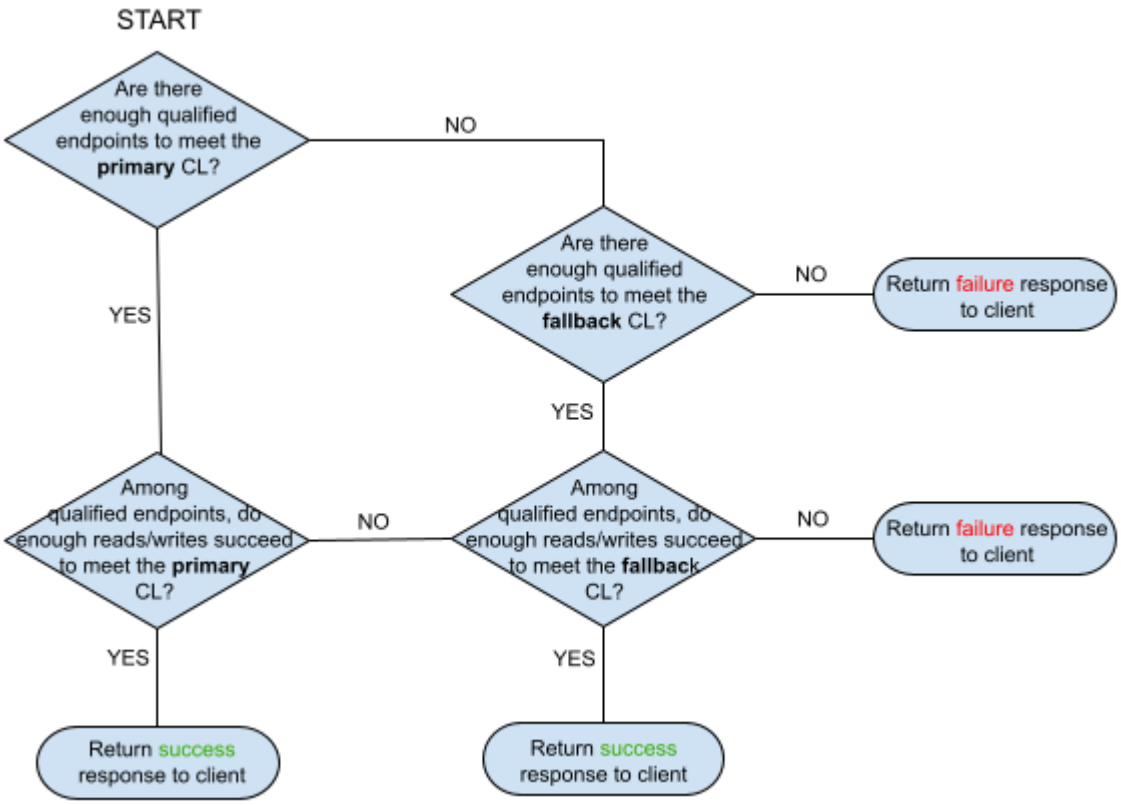
When you Add a Storage Policy to your HyperStore system one of the policy dimensions that you configure is consistency requirements for writes and reads of object data and metadata. When doing so, you have the option of configuring "dynamic" consistency level requirements. With dynamic CLs, you specify two or more consistency levels that differ in strictness. If the stricter consistency level (the "primary" consistency level) cannot be met for a given S3 request, then the system tries to meet the less strict level (the "fallback" consistency level).

The first flow chart below illustrates the standard consistency level logic when only one consistency level (CL) is used per operation type. The second flow chart illustrates the logic for a two-tier dynamic consistency level configuration. Following the flow charts is a detailed text description of how the dynamic consistency level feature works, which includes discussion of what constitutes a "qualified endpoint".

#### 1.7.10.1. Standard Consistency Level Logic



### 1.7.10.2. Dynamic Consistency Levels Logic




### 1.7.10.3. Dynamic Consistency Levels Logic Described

When the S3 Service processes an S3 PUT or GET of an object, it checks the system for a list of "endpoints" for that particular object — the nodes that the object data should be written to or read from. The system then checks whether any of the endpoint nodes is in any of these disqualifying conditions:

- HyperStore Service has been marked as down by the S3 Service (see **"hss.bring.back.wait, hss.-timeout.*, and hss.fail.*"** (page 469) in *mts.properties.erb*)
- Node has been put into Maintenance Mode by operator (see Start Maintenance Mode)
- Node is in a StopWrite condition due to all data disks being 90% full or more (relevant only for write requests)
- Disk on which requested data resides is **disabled** (relevant only for read requests)

Then, dynamic consistency levels can come into play at two different phases of S3 PUT or GET processing:

- If the number of qualifying endpoints meets the requirements of the either the primary CL **or the fall-back CL,** the system proceeds with trying to write to (or read from in the case of GET processing) those endpoints. If the number of qualifying endpoints does not meet the requirements of either the primary CL or the fallback CL, the system does not try to write to (or read from) the endpoints and the S3 request fails and an error is returned to the S3 client.
- When trying to write to (or read from) qualifying endpoint nodes, if the number of successful writes/reads meets the requirements of the either the primary CL **or the fallback CL**, a success

response is returned to the S3 client. If the number of successful writes/reads does not meet the requirements of either the primary CL or the fallback CL, the S3 request fails and an error is returned to the S3 client.

## 1.7.11. How vNodes Work

Following is an in-depth look at HyperStore vNodes, including diagrams to illustrate the role that vNodes play in supporting high-availability object storage in a HyperStore cluster.

S3 object placement and replication within a HyperStore cluster is based on a consistent hashing scheme that utilizes an integer token space ranging from 0 to $2^{127}$-1. Traditionally, in a storage cluster based on consistent hashing, each physical node is assigned an integer token from the token space. A given node is then responsible for a **token range** that extends from the next-lower token assigned to a different node (excluding the token number itself), up to and including the given node's own token. Then, an integer hash value is calculated for each S3 object as it is being uploaded to storage. The object is stored to the node responsible for the token range in which the object's hash value falls. Replication is implemented by also storing the object to the nodes responsible for the next-higher token ranges.

Advancing beyond traditional consistent hash based storage, the HyperStore system utilizes and extends the "virtual node" (vNode) functionality originally introduced in Cassandra version 1.2. This optimized design assigns multiple tokens to each physical node. In essence, the storage cluster is composed of very many "virtual nodes", with multiple virtual nodes residing on each physical node. Each virtual node is assigned its own token and has its own token range for which it is responsible.

The HyperStore system goes a significant step further by assigning a different set of tokens (virtual nodes) to **each HyperStore data disk** on each host. With this implementation, each data disk on a host is responsible for a set of different token ranges and -- consequently -- a different inventory of object data. If a disk fails it affects only the object data on that one disk. The other disks on the host can continue operating and supporting their own data storage responsibilities.

The number of tokens that the system assigns to each host is based on the total combined storage capacity of the host's HyperStore data disks. Specifically, the **system determines the number of tokens to assign to a host by taking the total number of terabytes of HyperStore data storage capacity on the host, multiplying by .7, and then rounding off** to the nearest integer. Further, the system applies a lower bound of one token per HyperStore data disk and an upper bound of 512 tokens per host.

For example:

| Number of Data Disks on Host | Size of Data Disks | Total TBs of Data Disk on Host | Number of Tokens Assigned To Host |
|---|---|---|---|
| 2 | 500GB | 1TB | 2<br>(1TB X .7 = .7, but minimum is 1 token per data disk) |
| 4 | 8TB | 32TB | 22<br>(32TB X .7 = 22.4, rounded = 22) |
| 8 | 4 X 8TB<br>4 X 10TB | 72TB | 50<br>(72TB X .7 = 50.4, rounded = 50) |
| 12 | 10TB | 120TB | 84<br>(120TB X .7 = 84) |

| Number of Data Disks on Host | Size of Data Disks | Total TBs of Data Disk on Host | Number of Tokens Assigned To Host |
|---|---|---|---|
| 24 | 16TB | 384TB | 268<br>(384TB X 0.7 = 268.8, rounded = 269) |

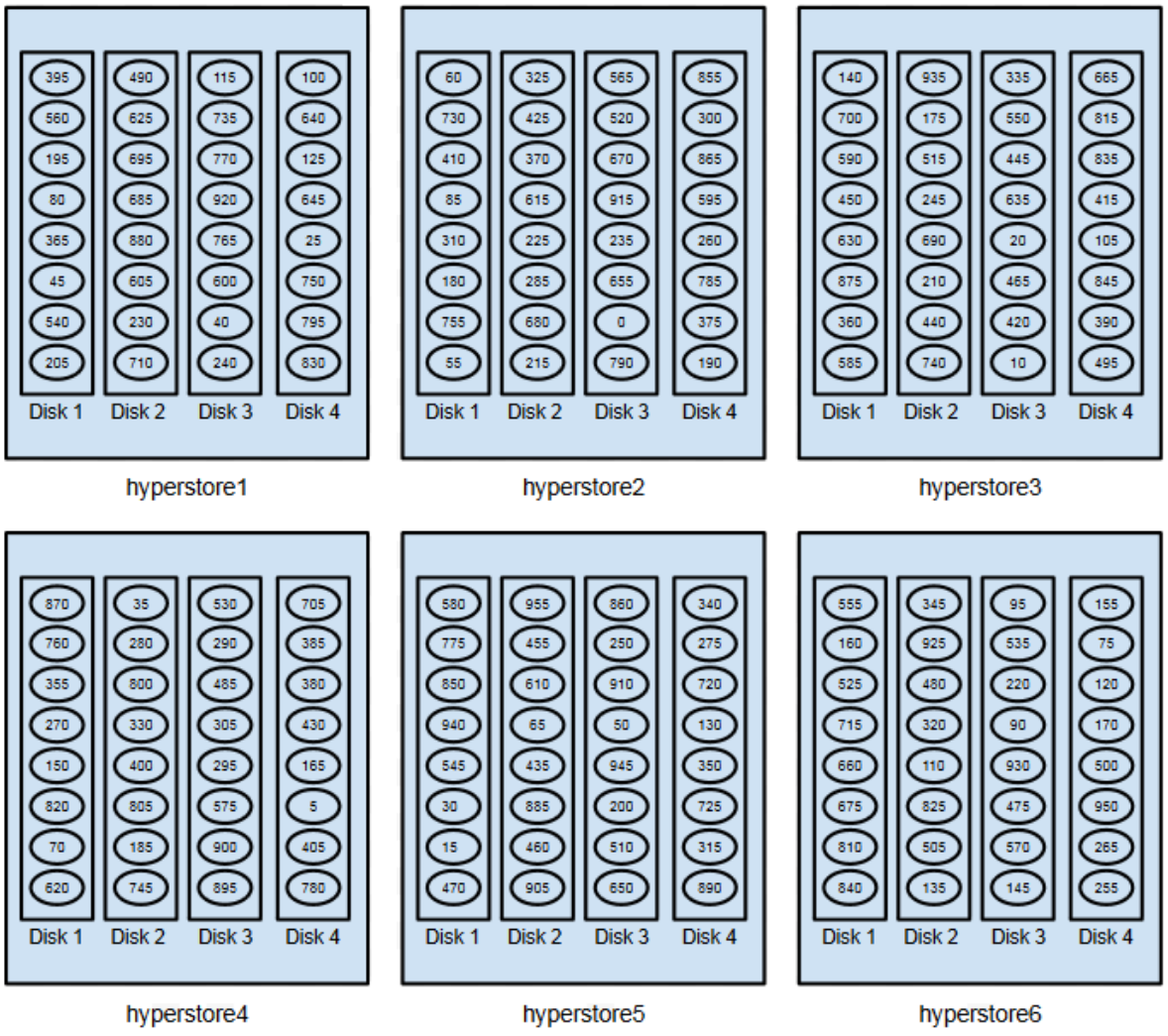
**Note** The data disk sizes in the table above are only for simple illustration of how the algorithm works. In calculations for actual hosts, the system uses not the disk raw size (the decimal-based size stated by the disk manufacturer) but rather the disk usable size after the disks are formatted and the file system is mounted (the binary-based size the operating system reports if you run the *lsblk* command on the host).

On each host, the host's assigned tokens -- and associated token ranges -- are automatically allocated to each HyperStore data disk in a manner such that storage capacity utilization should be **approximately balanced among the disks** on a given host.

In the **HyperStore File System** mounted to each HyperStore data disk there are sub-directories that demarcate each vNode's data.

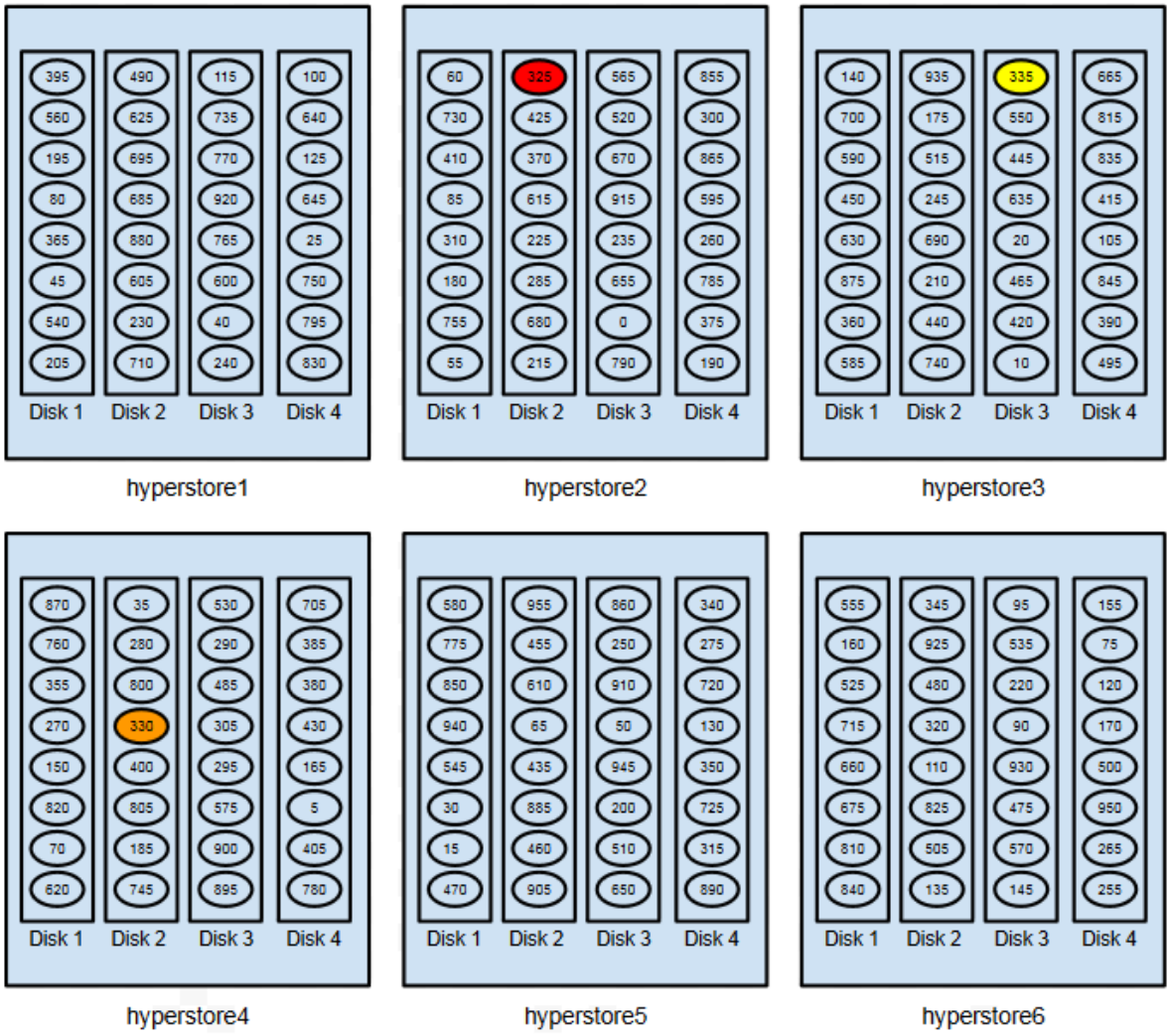
For illustration of how vNodes work to guide the distribution of data across a cluster, consider a cluster of six HyperStore hosts each of which has four disks designated for S3 object storage. Suppose that each physical host is assigned 32 tokens. And suppose for illustration that there is a simplified token space ranging from 0 to 960, and the values of the 192 tokens in this system (six hosts times 32 tokens each) are 0, 5, 10, 15, 20, and so on up through 955.

The diagram below shows one possible allocation of tokens across the cluster. Each host's 32 tokens are divided evenly across the four disks (eight tokens per disk), and that token assignment is randomized across the cluster.



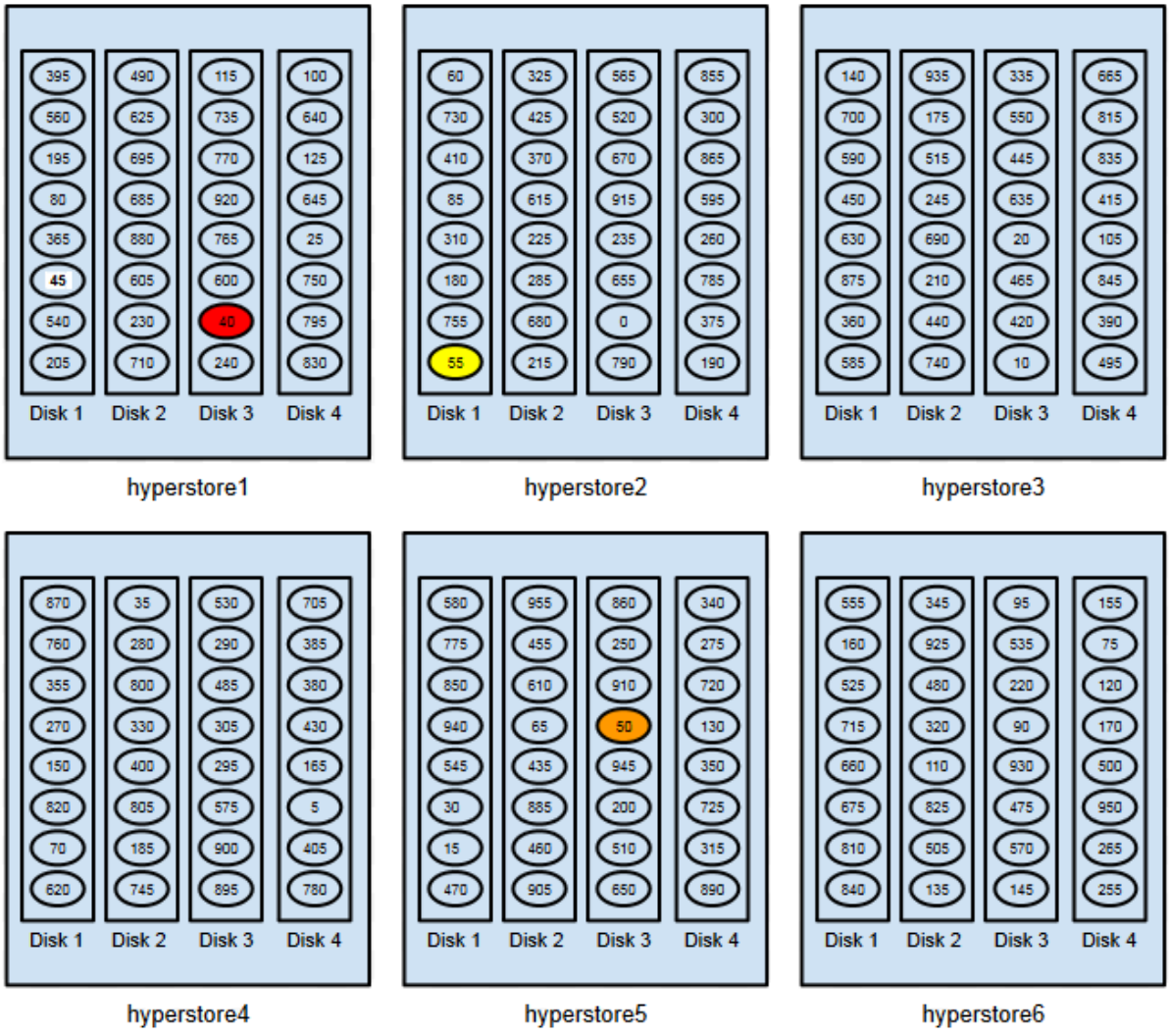

Now further suppose that you've configured your HyperStore system for 3X replication of S3 objects. And say that an S3 object is uploaded to the system and the hashing algorithm applied to the unique *<bucketname>/<objectname>* combination gives us a hash value of 322 (for this simplified example; in reality the system uses MD5 hashing). The diagram below shows how three instances or "replicas" of the object will be stored across the cluster:

- With its object name hash value of 322, the "primary replica" of the object is stored on the vNode responsible for the token range that includes the value 322. This is the vNode assigned token 325 (highlighted in red in the diagram below) -- this vNode has responsibility for a token range spanning from 320 (exclusive) up to 325 (inclusive). A simple way of identifying where the primary replica will go is that it's the vNode with the **lowest token that's higher than the object's hash value**. Note that the "primary replica" has no functional primacy compared to other replicas; it's called that only because its placement is based simply on identifying the disk that's responsible for the token range into which the object hash falls.
- The secondary replica is stored to the vNode that's assigned the next-higher token (330, highlighted in orange), which is located at hyperstore4:Disk2.
- The tertiary replica is stored to the vNode that's assigned the next-higher token after that (335, in yellow), which is at hyperstore3:Disk3.




Working with the same cluster and simplified token space, we can next consider a second object replication example that illustrates an important HyperStore vNode principle: no more than one of an object's replicas will be stored on the same physical host. Suppose that an S3 object is uploaded to the system and the object name hash is 38. The next diagram shows how the object's three replicas are placed:

- The primary replica is stored to the vNode that's assigned token 40 — at hyperstore1:Disk3 (red highlight in the diagram below).
- The vNode with the next-higher token — 45 (with white label) — is on a different disk (Disk1) on the same physical host as token 40, where the HyperStore system is placing the primary replica. Because it's on the same physical host, the system skips over the vNode with token 45 and places the object's secondary replica where the vNode with token 50 is — at hyperstore5:Disk3 (orange highlight).
- The tertiary replica is stored to the vNode with token 55, at hyperstore2:Disk1 (yellow highlight).




## The Disk Perspective

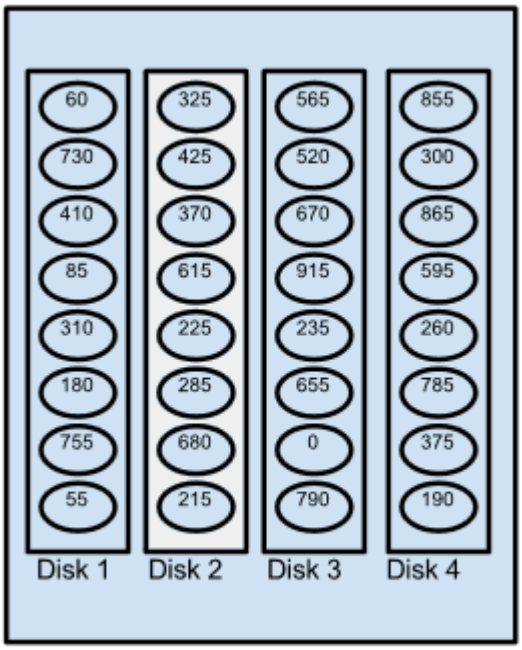
Now let's change perspective and see how things look for a particular disk from within the cluster. Recall that we've supposed a simplified token space with a total of 192 tokens (0, 5, 10, 15, and so on up to 955), randomly distributed across the cluster so that each host has 32 tokens and each host's tokens are evenly divided among its disks. We can zero in on hyperstore2:Disk2 – highlighted in the diagram below — and see that this disk has been assigned tokens 325, 425, 370, and so on.

Assuming the cluster is configured for 3X replication, on hyperstore2:Disk2 will be stored the following:

- In association with the disk's assigned token 325:
  - Primary replicas of objects for which the hash values are from 320 (exclusive) to 325 (inclusive)
  - Secondary replicas of objects for which the hash values are from 315 (exclusive) to 320 (inclusive)
  - Tertiary replicas of objects for which the hash values are from 310 (exclusive) to 315 (inclusive)

- In association with the disk's assigned token 425:
  - Primary replicas of objects for which the hash values are from 420 (exclusive) to 425 (inclusive)
  - Secondary replicas of objects for which the hash values are from 415 (exclusive) to 420 (inclusive)
  - Tertiary replicas of objects for which the hash values are from 410 (exclusive) to 415 (inclusive)
- And so on.

As noted previously, the HyperStore system when placing secondary and tertiary replicas may in some cases skip over tokens so as not to store more than one replica of an object on the same physical host. So this dynamic could result in additional responsibilities for hyperstore2:disk2 as a possible endpoint for secondary or tertiary replicas.




hyperstore2

In the event that Disk 2 fails, Disks 1, 3, and 4 will continue fulfilling their storage responsibilities. Meanwhile, objects that are on Disk 2 persist within the cluster because they've been replicated on other hosts. (Whether those objects will still be readable by S3 clients will depend on how you have configured **consistency level** requirements.)

## Multi-Data Center Deployments

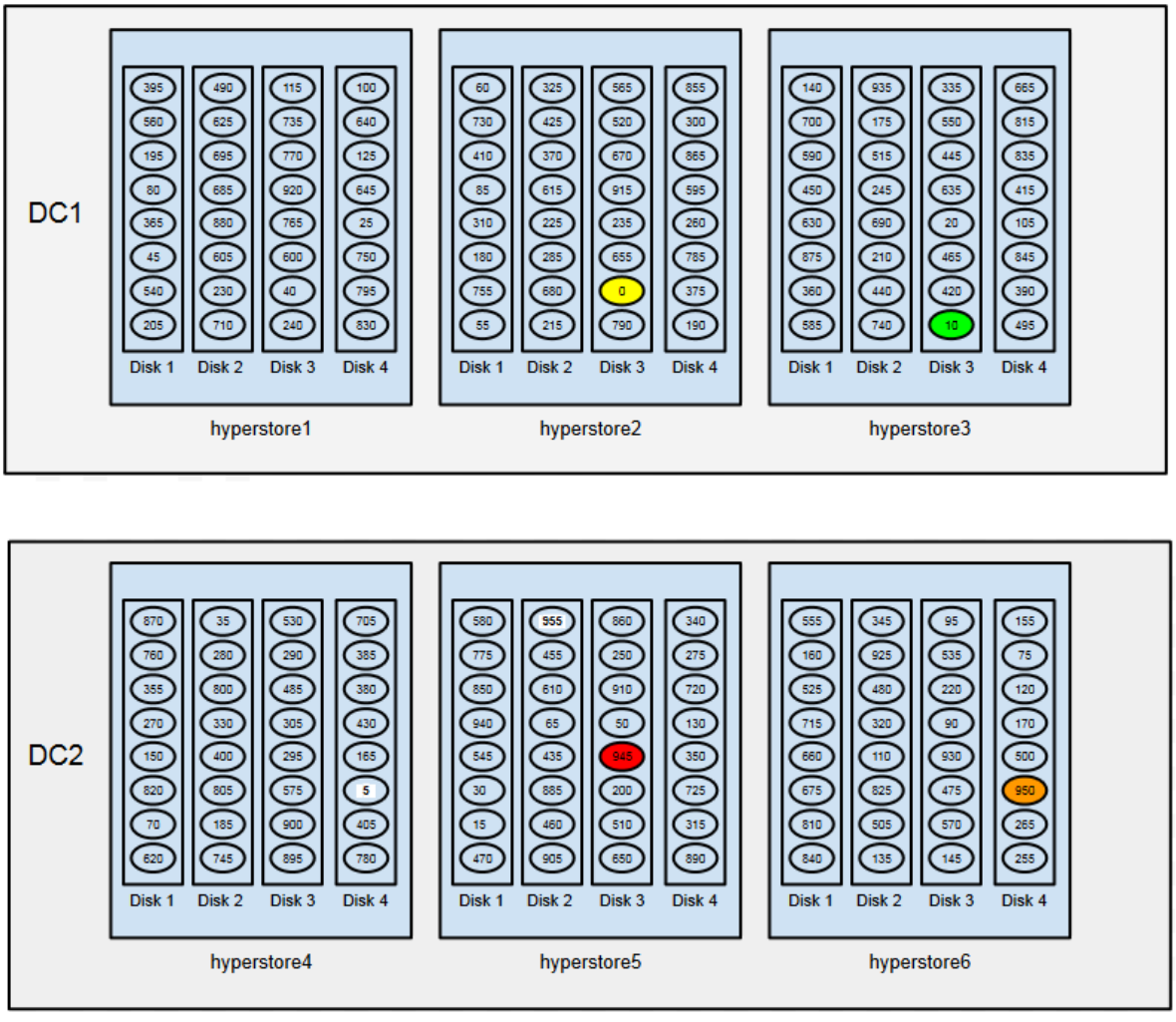
If you deploy your HyperStore cluster across multiple data centers within the same service region, your multiple data centers will be integrated in one unified token space.

Consider an example of a HyperStore deployment that spans two data centers — DC1 and DC2 — each of which has three physical nodes. As in our previous examples, each physical node has four disks; each host is assigned 32 tokens (vNodes); and we're supposing a simplified token space that ranges from 0 to 960. In this multi-DC scenario, the token space is divided into 192 tokens — 32 for each of the six physical hosts — which are randomly distributed across the six hosts.

Suppose also that S3 object replication in this deployment is configured at two replicas in each data center.

We can then see how a hypothetical S3 object with a hash value of 942 would be replicated across the two data centers:

- The first replica is stored to the vNode that's assigned token 945 (in red in the diagram below) — which is located in DC2, on hyperstore5:Disk3.
- The second replica is stored to vNode 950 (orange) — DC2, hyperstore6:Disk4.
- The next-higher vNode (955, with high-contrast label) is in DC2, where we've already met the configured replication level of two replicas — so we skip that vNode.
- The third replica is stored to vNode 0 (yellow) — DC1, hyperstore2:Disk3. Note that after the highest-numbered token (955) the token "ring" circles around to the lowest token (0). (In a more realistic token space there would be a token range spanning from the highest vNode token [exclusive] through the top of the token space and around to the lowest vNode token (inclusive]).
- The next-higher vNode (5, high-contrast label) is in DC2, where we've already met the configured replication level — so we skip that vNode.
- The fourth and final replica is stored to vNode 10 (green) — DC1, hyperstore3:Disk3.




## Multi-Region Deployments

If you deploy a HyperStore system across multiple service regions, each region has its own independent

storage cluster -- with each cluster having its own 0 to $2^{127}$-1 token space, its own set of vNodes, and its own independent inventory of stored S3 objects. There is no per-object replication across regions.

## Erasure Coded Data

When the HyperStore erasure coding feature is used, vNodes are the basis for distribution of encoded object fragments. Each of an object's *k+m* erasure coded fragments is assigned a hash value (token) by the system, and then each fragment is stored to the vNode responsible for the token range that contains the fragment's token.

## Cassandra Data

In a HyperStore system, Cassandra is used for storing object metadata and system metadata. In a typical deployment, on each HyperStore node Cassandra data is stored on the same RAID-mirrored disks as the OS. Cassandra data is not stored on the HyperStore data disks (the disks whose mount points are specified by the configuration setting *common.csv: hyperstore_data_directory*).

When vNodes are assigned to a host machine, they are allocated across only the host's HyperStore data mount points. vNodes are not allocated to the mirrored disks on which Cassandra data is stored.

Within a cluster, metadata in Cassandra is replicated in accordance with your storage policies. Cassandra data replication leverages vNodes in this manner:

- When a new Cassandra object -- a row key and its associated column values -- is created, the row key is hashed and the hash (token) is used to associate the object with a particular vNode (the vNode responsible for the token range that contains the Cassandra object's token). The system checks to see which host machine that vNode is located on, and the "primary" replica of the Cassandra object is then stored on the Cassandra disk(s) on that host.
- For example, suppose a host machine is assigned 96 vNodes, allocated across its multiple HyperStore data disks. Cassandra objects whose hash values fall into the token ranges of **any of those 96 vNodes** will get written to the Cassandra disk(s) on that host.
- Additional replicas of the Cassandra object (the number of replicas depends on your configuration settings) are then associated with next-higher-up vNodes and stored to whichever hosts those vNodes are located on — with the condition that if necessary vNodes will be "skipped" in order to ensure that each replica of the Cassandra object is stored on a different host machine.

## vNode Benefits

vNodes provide several advantages over conventional one-token-per-host schemes, including:

- Token assignment is performed automatically by the system — there is no need to manually assign tokens when you first set up a storage cluster, or when you resize a cluster.
- For cluster operations that involve transferring data across nodes — such as data repair operations or replacing a failed disk or host machine — the operations complete faster because data is transferred in small ranges from a large number of other hosts.
- The allocation of different vNodes to each disk means that failure of a disk affects only a known portion of the data on the host machine. Other vNodes assigned to the host's other disks are not impacted.
- The allocation of different vNodes to each disk, coupled with **storage policies for replication or erasure coding**, enable you to efficiently and safely store S3 object data without the overhead of RAID.

This page left intentionally blank

# Chapter 2. Accessing HyperStore Interfaces and Tools

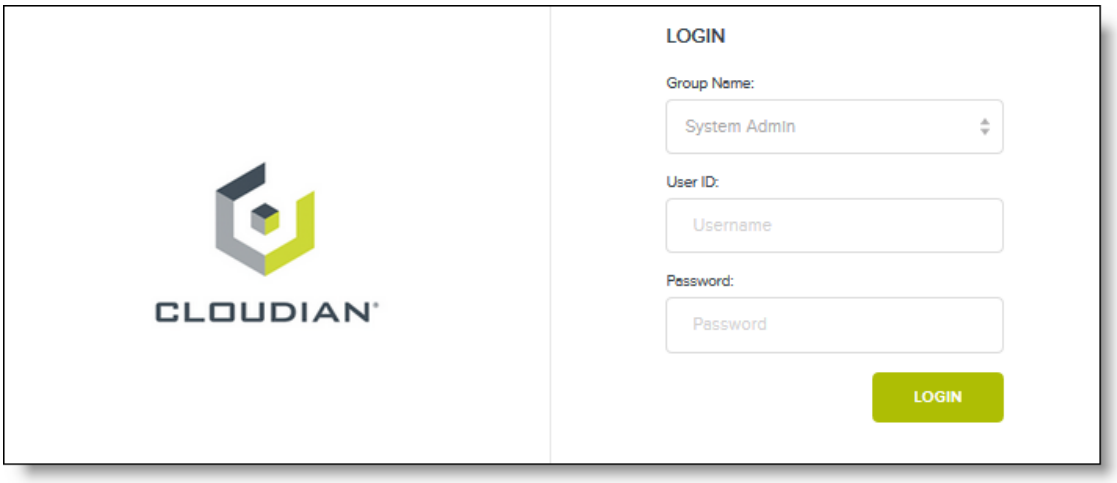
## 2.1. Accessing the Cloudian Management Console

To access the **Cloudian Management Console (CMC)**, do the following:

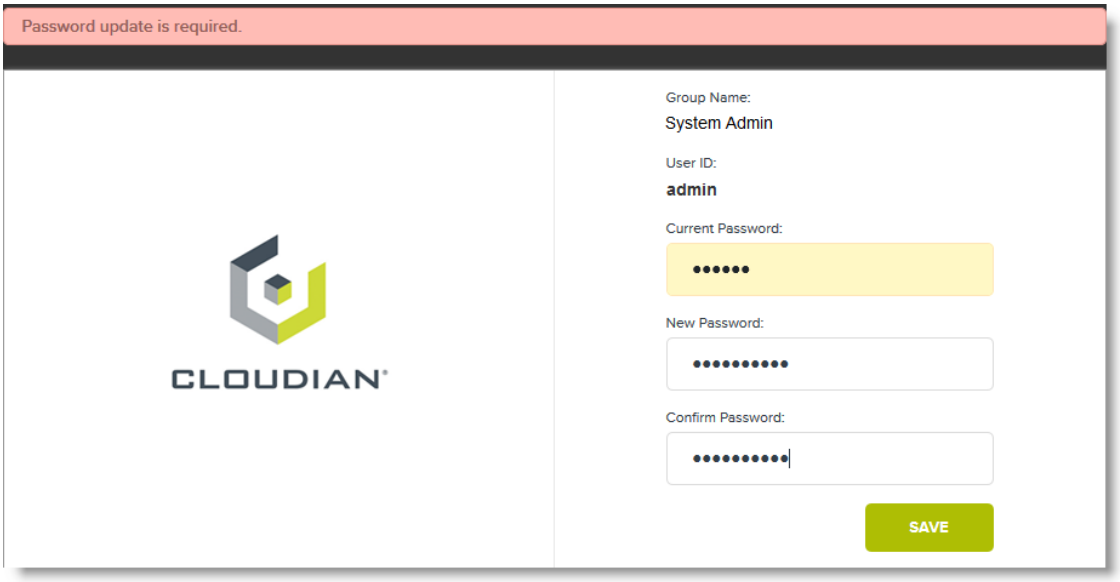
1. Point a browser to *https://<CMC_host>:8443/Cloudian*

   Since the CMC runs on all of your HyperStore nodes, for *<CMC_host>* you can use the fully qualified domain name (FQDN) or IP address of any node.

2. The first time you log in to the CMC you will get an SSL certificate warning (since the CMC's default certificate is a self-signed certificate). Follow the prompts to add an exception for the certificate. You should then see the CMC's login screen.




3. Enter the system administrator user ID *admin* and -- if this is your first time logging in -- the default password *public*. When you do so, the login screen will display additional fields in which you must create a new password for the *admin* user. After you create the new password and click **Save** you will be logged into the CMC.




**Note** The first time you log into the CMC the system requires you to create a new password for the *admin* user. On subsequent logins to the CMC as the *admin* user, use the password that you created. For security purposes you should periodically change the *admin* user password. You can do so through the CMC's **Security Credentials** page (accessible from a drop-down menu under the login user name in the upper right corner of the CMC interface).

**Note** For information about managing SSL certificates in HyperStore -- including the option to replace the CMC's default self-signed certificate with a CA-signed certificate -- see **"Security and Privacy Features"** (page 141).

## 2.2. Accessing Tools and Scripts

The tool that you will likely use most often is the HyperStore installer ***cloudianInstall.sh***. The installer is used not only for HyperStore installation but also to **restart services**, to **implement certain semi-automated configuration tasks** such as SSL certificate management, and to **apply configuration file changes to the system**. The installer is located in the installation staging directory */opt/cloudian-staging/7.5.2* on your Configuration Master node. To use this tool, change into the installation staging directory and then launch the tool as follows:

```
./cloudianInstall.sh
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a Trusted user, from any directory on the Configuration Master node you can launch the installer with this command:

```
hspkg install
```

Also located in the installation staging directory is the HyperStore host set-up tool ***system_setup.sh***. You (or a Cloudian representative working with you) would have used this tool to prepare individual host machines for the installation of your HyperStore cluster, and -- if you are expanding your cluster -- you also use this tool **to prepare new hosts to be added to the cluster**. To use this tool, change into the installation staging directory and then launch the tool as follows:

```
./system_setup.sh
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the system setup tool with this command:

```
$ hspkg setup
```

Another useful tool is the HyperStore node and cluster management tool ***hsstool***. You can launch this tool from any directory on any HyperStore node as follows:

```
hsstool
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** you can run *hsstool* from the HSH command line:

```
hsstool
```

For more information about using this tool -- including the option to run *hsstool* commands from the CMC interface rather than from the command line -- see **"hsstool "** (page 503).

HyperStore also includes several special-purpose tools that Cloudian Support might have you use in certain circumstances. These tools are located in the directory */opt/cloudian/tools* on each node. These tools are intended for use with the guidance of Cloudian Support.

## 2.3. Accessing Configuration Files

Many HyperStore configuration tasks can be managed through the CMC's **Configuration Settings** page or the installer's **Advanced Configuration Options** menu. But on occasion you may want to make edits to HyperStore configuration files. All configuration files that you might work with are located under this directory on your Configuration Master node:

```
/etc/cloudian-7.5.2-puppet/
```

For more information see HyperStore Configuration Files.

## 2.4. Accessing Log Files

Most HyperStore log files are located in this directory on each HyperStore node:

```
/var/log/cloudian
```

For more information see **"HyperStore Logs"** (page 349).

## 2.5. Accessing the Admin API

The HyperStore Admin API is a RESTful HTTP API that you can access at **port 19943** on any HyperStore node, using a command line tool such as cURL or a client application of your own creation. HTTP calls to the Admin API require HTTP Basic Authentication. You can check the user name and password for the HTTP Basic Authentication requirement by issuing these commands on any HyperStore node:

```
hsctl config get admin.auth.username
```

```
hsctl config get admin.auth.password
```

For more information see the Introduction section of the ***Cloudian HyperStore Admin API Reference***.

# 2.6. Accessing HyperStore Implementations of AWS APIs

HyperStore provides standards-compliant implementations of several popular Amazon Web Services (AWS) APIs:

- Simple Storage Service (S3) API
- Identity and Access Management (IAM) API
- Security Token Service(STS) API
- Simple Queue Service (SQS) API

Each of these APIs has a service endpoint (URI) at which they can be accessed by whichever third party client applications you use. The service endpoints are specific to your environment and are derived from information that you provided during HyperStore installation.

**To access any of these APIs with a client application, your users need S3 security credentials** (access key and secret key). These credentials are created automatically as part of the HyperStore user provisioning process. For an overview of user provisioning see **"Group and User Provisioning Feature Overview"** (page 243). Once you've provisioned users in the system they can obtain their S3 security credentials from the CMC's **Security Credentials** page (via the drop-down menu under their user name in the upper right of the CMC). Programmatically a user's security credentials can be obtained through the Admin API call *GET /user-/credentials/list/active* (for detail see the "user" section of the ***Cloudian HyperStore Admin API Reference***).

## 2.6.1. S3 API

The HyperStore S3 Service provides comprehensive support for the AWS Simple Storage Service API. Third party S3 client applications can access this service at your HyperStore system's S3 service endpoint. To find the S3 service endpoint for your system go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

For more information on HyperStore support for the S3 API see the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

> **Note** The CMC has a built-in S3 client through which your HyperStore users can perform many S3 operations such as creating and configuring buckets and uploading and downloading objects.

## 2.6.2. IAM API

The HyperStore IAM Service provides majority support for the AWS Identity and Access Management API. Third party IAM client applications can access this service at your HyperStore system's IAM service endpoint. To find the IAM service endpoint for your system go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

For more information on HyperStore support for the IAM API see the IAM API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

**Note** The CMC has a built-in IAM client through which your HyperStore users can perform many IAM operations such as creating IAM groups and users under their HyperStore root account.

### 2.6.3. STS API

The HyperStore STS Service provides partial support for the AWS Security Token Service API. The default STS Service endpoint URLs for HTTP and HTTPS are:

- *http://sts.<your-organization-domain>:16080*
- *https://sts.<your-organization-domain>:16443*

For more information on HyperStore support for the STS API see the STS API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

### 2.6.4. SQS API

The HyperStore SQS Service provides partial support for the AWS Simple Queue Service API. The SQS Service is disabled by default, and if you enable the service you can configure the service endpoint.

For more information on HyperStore support for the SQS API, and for information about enabling and configuring the SQS Service, see the SQS API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

This page left intentionally blank

# Chapter 3. Setting Up Administration Features

## 3.1. HyperStore Shell (HSH)

### 3.1.1. HyperStore Shell (HSH) Feature Overview

The HyperStore Shell (HSH) is a restrictive command-line interface that allows administrators to log into and administer HyperStore nodes without having or needing *root* access to the nodes. A user logged into the HyperStore Shell can run common HyperStore commands and tools such as *hsstool* or *cloudianInstall.sh* as well as being able to run a limited range of Linux OS commands. But the HSH is more restrictive, and therefore more secure, than having administrators log into HyperStore nodes as *root* and use a standard Linux shell. With the HSH each administrator has their own unique login ID and password; the commands that administrators can execute are limited by the shell; and each administrator's actions are recorded to an audit log.

The HSH is present on each HyperStore node but **by default the HSH is disabled**. For information on enabling the HSH, and provisioning HSH users, see **"Enabling the HSH and Managing HSH Users"** (page 96). That section also describes an option to disable *root* password access to HyperStore nodes, so that administration of the nodes is performed exclusively by way of the HSH.

For information on the HyperStore commands and Linux OS commands supported by the HSH, see **"Using the HSH"** (page 101).

> **Note** The HSH is not supported on "auxiliary nodes" (nodes on which you install the optional **HyperSearch Service** or **HyperFile Service**.)

#### 3.1.1.1. HSH Logging

Every user login to the HSH, and every command that an HSH user runs while logged in, is recorded to a dedicated HSH log. Also recorded are commands that an authenticated HSH user runs remotely, without initiating a login session.

Each HyperStore node has its own HSH log, recording HSH logins and commands run on that node. The log is configured as append only, and HSH users cannot modify this configuration.

For more information on HSH logging, see **"HyperStore Shell (HSH) Log"** (page 359).

#### 3.1.1.2. The HSH and "Certified" Object Lock

If you want to use the "Certified" type of Object Lock, you must first enable the HSH and disable the root password. With the "Certified" type of Object Lock, the system will not allow the creation of Object Lock enabled buckets until after the HSH is enabled and the root password is disabled.

For information about Object Lock, see **"Object Lock Feature Overview"** (page 187).

For information about enabling HSH and disabling the root password, see **"Enabling the HSH and Managing HSH Users"** (page 96).

## 3.1.2. Enabling the HSH and Managing HSH Users

*Subjects covered in this section:*



### 3.1.2.1. Enabling the HSH

By default the HyperStore Shell (HSH) is installed but disabled.

> **Note** The exception is that HyperStore appliances delivered for new deployments in certain industry sectors with stringent security requirements may arrive on site with HSH already enabled (and the root password already disabled). If you're unsure whether this applies to you, consult with your Cloudian representative.

To enable the HSH on all of your HyperStore nodes do the following:

1. Log into the Configuration Master node as the *root* user.
2. Check to confirm that the HSH is currently disabled. (Here and in the steps that follow you will use *hsctl*, a new node management tool that remains mostly behind the scenes in HyperStore 7.5.2 but will be more prominent in future HyperStore releases. You can run the *hsctl* commands in this procedure from any directory.)

   ```
   # hsctl config get hsh.enabled
   False
   ```

3. Set *hsh.enabled* to *true*:

   ```
   # hsctl config set hsh.enabled=true
   ```

4. Push the configuration change out to the cluster:

   ```
   # hsctl config apply hsh
   ```

5. Confirm that the HSH is now enabled:

   ```
   # hsctl config get hsh.enabled
   True
   ```

**The HSH is now enabled in your system, but by default no users are able to log into the HSH and use it**. To provision users who can use the HSH, see **"Adding HSH Users"** (page 96) below.

### 3.1.2.2. Adding HSH Users

HSH users map to your System Admin users in the CMC. For each System Admin user in the CMC, HyperStore will automatically create a corresponding HSH user account if triggered to do so as described below.

| CMC System Admin User Type | How to Trigger Creation of Corresponding HSH User |
|---|---|
| The CMC default System Admin user, with user name "admin". | After installing or upgrading to HyperStore 7.2 or later, log into the CMC as the "admin" user and change the "admin" user's password (in the drop-down list under the user login name select **Security Credentials**). (Alternatively, the password can be changed through the Admin API -- see the "user" section in the ***Cloudian HyperStore Admin API Reference***.) This password change causes the system to create a corresponding HSH user. |
| Additional CMC System Admin users, created in HyperStore 7.1.x or earlier. | After the system has been upgraded to HyperStore 7.2 or later, a legacy CMC System Admin User can log into the CMC and change their password. (Alternatively, the password can be changed through the Admin API.) This password change causes the system to create a corresponding HSH user. |
| Additional CMC System Admin users, created in HyperStore 7.2 or later. | When an additional CMC System Admin user is created in HyperStore 7.2 or later (either in the CMC or through the Admin API), the system automatically creates a corresponding HSH user. |

When HyperStore creates an HSH user corresponding to a CMC System Admin user:

- The **HSH user name is the CMC user name prefixed by "sa_"**. For example, for CMC user "admin" the HSH login name is "sa_admin"; and for CMC user "tom" the HSH login name is "sa_tom".
- The **HSH user login password is the same as the CMC user login password**. The HSH user login password cannot be managed separately from -- or differ from -- the corresponding CMC user login password. If an HSH user wants to change their password they should change their CMC password, and their HSH password will automatically change to match their new CMC password.

> **Note** If you have **LDAP authentication** enabled for the System Admin group, then when HyperStore creates an HSH user corresponding to a CMC System Admin user the HSH user name will **not** have an "sa_" prefix (instead it will be the same as the CMC user name); and when the user logs into the HSH they will use their LDAP credentials and the system will verify those credentials against your LDAP service. Note that if a local Unix user account with that same user name already exists, a new HSH user account will not be created (the existing local user account will not be overwritten). For more information on LDAP authentication see **"LDAP Integration"** (page 247) and especially the sub-section **"LDAP Authentication of System Administrators and HSH Users"** (page 249).

Once an HSH user has been created, that user can use SSH to log into any HyperStore node. Upon login, the user's shell will be the HyperStore Shell. The prompt will appear as follows:

```
<username>@<hostname>$
```

For example:

```
sa_admin@hyperstore1$
```

You can confirm that you are in the HyperStore Shell by typing *help*:

```
sa_admin@hyperstore1$ help
HyperStore Shell
Version: 1.0.1-2, d6b3c8d46ecaaaa69b62af25c4e7d4270f8b4d7e(otp protocol: 1)

Commands:
...
```

For information on using the HyperStore Shell see **"Using the HSH"** (page 101).

> **Note** HyperStore does not create corresponding HSH users for CMC Group Admin level users (or for regular users). Only System Admin level users are allowed HSH access.

### 3.1.2.3. Elevating a Regular HSH User to the "Trusted" Role

The HSH supports two types of HSH users:

- Regular HSH users
- Trusted HSH users (HSH users who have been granted the "Trusted" role)

While regular HSH users can run **most** of the commands that the HSH supports, Trusted HSH users can run **all** of the commands that the HSH supports. Put differently, there are some commands that only Trusted HSH users can run. For information about supported commands and which ones are available only to Trusted users, see **"Using the HSH"** (page 101).

The **HSH user *sa_admin*** -- the HSH user corresponding to the CMC default System Admin user -- **is automatically a Trusted HSH user**. By contrast, HSH users corresponding to additional System Admin users that have been created in the CMC are by default regular HSH users who do not have the Trusted role.

As a Trusted user the *sa_admin* user can elevate one or more regular HSH users so that they too are Trusted:

1. Log into the Configuration Master node as the *sa_admin* user. Upon login you will be in the HyperStore Shell.
2. Run this command to add a regular HSH user to the Trusted role:

   ```
   $ hspkg role -a <username> trusted
   ```

   Be sure to specify the HSH user name (including the *sa_* prefix), not the CMC user name.

   For example:

   ```
   $ hspkg role -a sa_admin2 trusted
   hsh.roles.trusted.users=sa_admin2
   ```

   > **Note** The response shows the current list of users who have been explicitly granted the Trusted role (which is just one user in the example above). The *sa_admin* user, who is automatically Trusted, will not appear in this list.

3. If you want to elevate any other regular HSH users to the Trusted role at this time, run the command from Step 2 again, with another HSH user name.
4. Apply the configuration change to the cluster. (Here you are again using the *hsctl* tool that was mentioned previously).

   ```
   $ hsctl config apply hsh
   ```

Once additional HSH users have been elevated to the Trusted role, then those Trusted users are also able to elevate regular HSH users to the Trusted role. That is, any Trusted HSH user has the ability to elevate regular HSH users to the Trusted role.

**To see the current list of HSH users who have been explicitly granted the Trusted role**, any Trusted user can run this command:

```
$ hspkg role trusted
hsh.roles.trusted.users=sa_admin2
```

Note again that the response will exclude the *sa_admin* user, who is automatically Trusted.

**To remove an HSH user from the list of Trusted users**, a Trusted user can run these commands:

```
$ hspkg role -d <username> trusted
$ hsctl config apply hsh
```

This does not delete the HSH user; it only demotes them to being a regular HSH user rather than a Trusted user.

> **Note** If an HSH user that you elevate to the Trusted role (or remove from the Trusted role) is logged in to the HSH when you make the change, the change in their role status will not take effect until after the user logs out and then logs back in again.

### 3.1.2.4. Deleting an HSH User

If a System Admin user is deleted or suspended (made inactive) in the CMC or through the Admin API, the system automatically deletes that user's HSH user account.

In the case of a System Admin user who is made inactive, and then subsequently made active again, after being made active again that user must change their password in the CMC or through the Admin API in order to trigger the system to recreate a corresponding HSH account for the user.

The system does not support deleting or disabling the HSH account of a CMC System Admin user while their CMC account remains active.

### 3.1.2.5. Disabling the *root* Password

Optionally, you can disable the *root* account password on all HyperStore nodes so that no user has *root* password access to those nodes. You can do this if for security reasons it's desirable to have all administrators use only the restrictive HyperStore Shell when logging into and administering HyperStore nodes .

> **Note** Disabling the *root* password is **required** if you want to use the **"Certified" type of Object Lock**.

Before you can disable the *root* account password:

- HSH must be enabled (see **"Enabling the HSH"** (page 96))
- An HSH user account must be created for the CMC's default System Admin user (see **"Adding HSH Users"** (page 96))

Also before you disable the *root* account password, it's important to be aware that:

- If you disable the *root* password, and then you subsequently want *root* password access to your HyperStore nodes again, you will need to contact Cloudian Support for assistance. Once you disable *root* password access to HyperStore nodes **you cannot regain *root* password access without assistance from Cloudian Support.**
- Disabling the *root* password will prevent users from logging in to a HyperStore node as *root* using a password but it **will not prevent a *root* user from accessing a HyperStore node using an SSH key**, if in your environment SSH key access has been set up for the *root* user.
- Disabling the *root* password **will have no effect on any users who were granted *sudo* privileges** before you disabled the *root* password. Such users (if any) will continue to have the privileges granted to them by a node's *etc/sudoers* configuration. To help secure your HyperStore nodes, Cloudian, Inc. recommends that you manually remove the *sudo* privileges of all users, on all HyperStore nodes (including yourself if you have *sudo* privileges). Also if you have granted elevated permissions to any user or application through any other mechanism, remove those permissions.

To disable *root* password access to all HyperStore nodes:

1. As *root*, log into the Configuration Master and then change into the installation staging directory (*/opt/cloudian-staging/7.5.2*).

   **Note** You must be logged in as *root* to disable the *root* password -- you cannot do it while logged in as an HSH user.

2. Launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

3. In the installer main menu enter **4** for "Advanced Configuration Options", then at the next menu enter **m** for "Disable the root password".
4. Follow the prompts to disable the *root* password.

After exiting the installer, log out from the node. Then try to log back in as *root*, using the *root* password -- the login attempt should fail. Then log in as *sa_admin* or another HSH user, with that user's password. The login should succeed, and you should be in the HyperStore Shell.

### Regaining *root* Password Access After Having Disabled It

If you disable *root* password access to your HyperStore nodes as described above, the only way to regain *root* password access is to contact Cloudian Support for assistance. This *root* password access will be temporary.

## 3.1.2.6. Configurable HSH Idle Timeout

The HyperStore Shell has a default idle timeout of 30 minutes. This means that by default an HSH session will terminate automatically if the HSH user has been completely inactive for 30 minutes. The idle timer:

- Is reset to zero whenever the HSH user presses any key. For example if the HSH user has been completely idle for 20 minutes and then they press any key, the timer is reset and the user could then be idle again for up to 30 minutes before their session is automatically terminated.
- Does not apply to long-running commands or utilities that the HSH user has initiated.

To change the idle timeout, do:

```
hsctl config set hsh.idleTimeout=<value>
hsctl config apply hsh
```

- Specify the *<value>* as *<n>s* for number of seconds or *<n>m* for number of minutes or *<n>h* for number of hours -- for example *90s* or *5m* or *2h*.
- The minimum allowed value is one minute and the maximum allowed value is twelve hours.
- This configuration can be performed by the root user, or while in the HSH if the HSH user has the Trusted role. If performed while in the HSH, the idle timeout configuration change will not apply until the HSH user logs out of the HSH and then logs back in again.

To check the current idle timeout configuration value, do:

```
hsctl config get hsh.idleTimeout
```

**Note** In the rare case of a root mode HSH session -- when assisted by Cloudian Support as noted in **"Regaining root Password Access After Having Disabled It"** (page 100) -- there is a dedicated root mode idle timeout which by default is 15 minutes. This timeout can be configured as described above, except that the setting name is *hsh.rootMode.idleTimeout*. This cannot be set to a value larger than the current *hsh.idleTimeout* setting value.

## 3.1.3. Using the HSH

*Subjects covered in this section:*



### 3.1.3.1. Starting and Ending an HSH Session

Once the HyperStore Shell (HSH) has been enabled and your HSH user account has been created (as described in **"Enabling the HSH and Managing HSH Users"** (page 96),) you can SSH into any HyperStore node using your HSH user name and password. Upon login to the node, your shell will be the HyperStore Shell. The prompt will appear as follows:

```
<username>@<hostname>$
```

For example:

```
sa_admin@hyperstore1$
```

To confirm that you are in the HyperStore Shell and **to view a list of HSH commands**, type *help*:

```
$ help
HyperStore Shell
Version: 1.0.1-2, d6b3c8d46ecaaaa69b62af25c4e7d4270f8b4d7e(otp protocol: 1)

Commands:
...
```

**To view HSH command inline help** type *<command> --help* (or *<command> -h*):

```
$ hslog --help
View protected log files under the /var/log directory.
```

```
Usage:
hslog [flags] FILENAME
Flags:
-h, --help   help for hslog
```

**To end your HSH session** and disconnect from the HyperStore node, type *exit*.

```
$ exit
```

**Note**
* On each node an HSH user's home directory is */home/<username>* -- for example */home/sa_admin*.
* By default the HSH has an idle timeout of 30 minutes. For more information see **"Configurable HSH Idle Timeout"** (page 100).

### Reattaching to an HSH Session

If you are logged into the HSH on a node and your SSH connection gets cut:

- Any long-running commands that were in-progress in that HSH session will continue to run
- You can log back into the HSH on that same node and you will be given the choice to reattach to the existing session or to start a new session

You can also manually detach from a session by using the keystroke sequence *ctrl-g d*. Then when you log in again you can choose whether to reattach to the existing session or to start a new session.

**Note** If an HSH user gets disconnected -- or intentionally detaches -- from a session, and then logs in and starts a new session, the user can have multiple concurrent sessions on a node. The system sets a limit of 9 concurrent sessions for a single HSH user on a node.

**To end an existing session from which you are detached**, reattach to it and then type *exit* on the command line.

## 3.1.3.2. HyperStore Commands and Utilities Supported by the HSH

The HyperStore Shell supports the use of these commands that are specific to HyperStore. Commands marked as "(requires Trusted role)" are available only to HSH users who have been granted the Trusted role (for more information on this role see **"Elevating a Regular HSH User to the "Trusted" Role"** (page 98)).

**Note** Do not precede these commands with a path -- for example, run *hsstool* as simply *hsstool* not as *./hsstool*.

| Command | Purpose | More Information |
|---|---|---|
| cloudian_sysinfo.sh | This command serves the same purpose as the "Collect Diagnostics" function in the CMC. | CMC Cluster -> Nodes -> Advanced |
| GDPR_decryption.py | Decrypt encrypted log field values in log file copies that have been uploaded to Cloudian Support. | **"Decrypting an Encrypted Log Field Value"** (page |

| Command | Purpose | More Information |
|---|---|---|
| | | 112) |
| hsctl (requires Trusted role) | *hsctl* is a new HyperStore node management tool that is of limited use in the current HyperStore release but will be more prominent in future releases | **"Security and Privacy Features"** (page 141)<br>**"FIPS Support"** (page 168)<br>"HTTP(S) Basic Authentication for Admin API Access" in the ***Cloudian HyperStore Admin API Reference*** |
| hslog | View a HyperStore log file | In shell, *hslog --help*<br>Or for more detailed information see:<br>**"Using the HSH to View Logs"** (page 374) |
| hspkg<br>Below is a partial list of sub-commands; for the complete list see the inline help. | Manage HyperStore installation and configuration | In shell, *hspkg --help* |
| • hspkg setup (requires Trusted role) | Launch the HyperStore host setup tool (*system_setup.sh*) | In shell, *hspkg setup --help*<br>See also:<br>"Preparing Your Nodes For HyperStore Installation" in the ***Cloudian HyperStore Installation Guide*** |
| • hspkg install (requires Trusted role) | Launch the HyperStore installer (*cloudianInstall.sh*) | In shell, *hspkg install --help*<br>See also: |

| Command | Purpose | More Information |
|---|---|---|
| | | **"Upgrading Your HyperStore Software Version"** (page 255)<br><br>**"Installer Advanced Configuration Options"** (page 377)<br><br>**"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382)<br><br>**Note** When launching the installer from the HyperStore Shell, the installer's "Uninstall Cloudian HyperStore" menu option is disabled. |
| • hspkg config (requires Trusted role) | View or edit a HyperStore configuration file | In shell, *hspkg config --help*<br><br>Or for more detailed inform- |

| Command | Purpose | More Information |
|---|---|---|
| | | ation see:<br>**"Using the HSH to Manage Configuration Files"** (page 386) |
| • hspkg role (requires Trusted role) | Manage HSH "Trusted" role membership | In shell, *hspkg role --help*<br>Or for more detailed information see:<br>**"Elevating a Regular HSH User to the "Trusted" Role"** (page 98) |
| • hspkg version | Check the HyperStore software version | -- |
| hsrun | Run a binary file or script that is cyrptographically signed by Cloudian (such as a HyperStore release package file). | In shell, *hsrun --help* |
| hsstool | The HSH supports running any *hsstool* command, such as *hsstool repair*, *hsstool cleanup*, and so on | **"hsstool "** (page 503) |
| install_appliance_license.sh (requires Trusted role) | Install a new license for a HyperStore Appliance | In shell, *install_appliance_license.sh --help* |
| jetty_password.sh | Create a Jetty-obfuscated password, in connection with modifying the HTTP/S Basic Authentication password for the Admin Service. | "HTTP(S) Basic Authentication for Admin API Access" in the ***Cloudian HyperStore Admin API Reference*** |
| nodetool | Cassandra's native node and cluster management tool. Typically you should use *hsstool* instead. | -- |
| rebrand_cmc.sh (requires Trusted role) | Copy image and resource files to or from the Puppet configuration directory, in con- | **"Rebranding the CMC UI"** |

| Command | Purpose | More Information |
|---|---|---|
| | nection with rebranding the CMC interface. | (page 236) |
| reset_expired_password (requires Trusted role) | This can be used if working with your Cloudian representative to set up a newly delivered HyperStore appliance. | -- |
| search-mgr.sh | Relevant only if you have deployed the optional HyperStore Search Service. | **"Using the search-mgr.sh Tool For On-Demand Indexing"** (page 205) |

### Commands for HyperStore Appliances

The following commands pertain only to HyperStore appliances and would typically be used only on instructions from Cloudian Support. **All** of these commands require the Trusted role.

- slot_disk_map_1500.sh
- slot_disk_map_4000.sh
- slot_disk_map_hsx.sh
- slot_disk_map_x3650.sh

## 3.1.3.3. Linux Commands and Utilities Supported by the HSH

The HyperStore Shell allows the use of **only** the following Linux commands and utilities. No other Linux commands or utilities can be run from within the shell. Commands marked as "(requires Trusted role)" are available only to HSH users who have been granted the Trusted role (for more information on this role see **"Elevating a Regular HSH User to the "Trusted" Role"** (page 98)).

> **Note** For security reasons, for some commands the HyperStore Shell forbids certain arguments / subcommands. If you try to run a command with one of its forbidden arguments the shell returns a message indicating that the argument is not permitted.

| | | |
|---|---|---|
| blkid | hostname | ps |
| cat | iostat | pwd |
| cd | ip | rm |
| chmod | ipmitool (requires Trusted role) | rmdir |
| cp | keytool | scp |
| curl | kill (requires Trusted role) | sftp |
| date | less | smartctl (requires Trusted role) |
| df | ls | ssh |

| | | |
|---|---|---|
| diff | lsblk | systemctl |
| dmesg | lspci | tail |
| dmidecode | lsscsi | tar |
| domainname | md5sum | top |
| du (requires Trusted role) | mdadm (requires Trusted role) | uname |
| echo | mkdir | unzip |
| exit | mv | vi |
| file | netstat | vmstat |
| grep | ntpq | w |
| gzip | openssl (requires Trusted role) | wget |
| head | pgrep | zgrep |
| host | ping | |

### 3.1.3.4. Using *sftp* with the HSH

You can use an *sftp* client to upload or download a file to a HyperStore node via an HSH account, but only if the file is one that the HSH is permitted to manage. For example:

```
# sftp sa_admin@127.0.0.1
sa_admin@127.0.0.1's password:
Connected to 127.0.0.1.
sftp> put survey.csv
Uploading survey.csv to /home/sa_admin/survey.csv
survey.csv                                    100%   28     6.3KB/s   00:00
sftp> get survey.csv
Fetching /home/sa_admin/survey.csv to survey.csv
/home/sa_admin/survey.csv                     100%   28    13.1KB/s   00:00
sftp> put abc /etc/passwd
Uploading abc to /etc/passwd
remote open("/etc/passwd"): Permission denied
```

**Note** For the list of files that the HSH has permission to access, run this command:

*$ hspkg config --help*

For more information see **"Using the HSH to Manage Configuration Files"** (page 386).

### 3.1.3.5. Notes On Command Usage

Tab completion for command names and file names is supported.

Running multiple commands on one line is supported, using the operators "**;**", "**&&**", "**||**", and/or "**|**".

For security reasons, the HSH enforces the following **restrictions** on command usage:

- PATH is fixed for each command; you cannot specify a path when running a command.
- *sudo* is not allowed. Instead, commands that need root privilege (such as *systemctl*) are automatically given that privilege inside the HSH. (The HSH accomplishes this by careful management of the effective UID, which is only elevated to root when running a command as root is required.)
- Job control commands such as *ctrl-z* are not allowed.
- When listing files, the use of the wildcard character "*" is not allowed.

# 3.2. Smart Support

## 3.2.1. Smart Support and Diagnostics Feature Overview

*Subjects covered in this section:*



HyperStore includes two automated features that help you collaborate with Cloudian Support to keep your system running smoothly: Smart Support and on-demand Node Diagnostics.

### 3.2.1.1. Smart Support

Smart Support is a feature that enables Cloudian Support to help you maximize the uptime and performance of your HyperStore system. With Smart Support, your HyperStore system automatically transmits detailed system and node performance data to Cloudian Support once a day over a secure internet connection. Cloudian Support applications and personnel continually analyze this data to detect critical issues. Cloudian Support notifies you to make you aware of such issues and, if appropriate, to advise you on the actions that should be taken to keep your system running well.

**IMPORTANT !** Cloudian's Smart Support feature is not a comprehensive remote monitoring program. Cloudian Support will notify you only in the case of critical issues, and not in real time. Primary responsibility for monitoring and managing your HyperStore system lies with you, the customer.

Specifically, the Smart Support mechanism provides Cloudian Support the following system and node information:

- Cluster topology information
- HyperStore license information
- Packages and dependencies version information
- Node hardware, OS, and software information, including (if applicable) information specific to HyperStore appliances
- Network interface configuration and network statistics for each node
- Storage capacity utilization information for each node
- RAM and CPU utilization information for each node
- Disk diagnostics data for each node

- Service status (service UP/DOWN) history for each node
- Service JMX statistics
- S3 transaction performance data for each node
- System performance data such as disk I/O and per-disk capacity usage for each node
- Current map of tokens to disks for each node
- Redis information to confirm sync between master and slave nodes
- Data repair operation status and history for each node
- Proactive repair queue information for the cluster
- HyperStore configuration files for each node
- System crontab configuration for each node
- Alerts for each node(the same as you would see in the CMC's **Alerts** page [**Alerts -> Alerts**])
- System log files and HyperStore log files

This information is collected daily on to one of your nodes (the node identified as the "System Monitoring/Cronjob Primary Host" in the CMC's **Cluster Information** page [**Cluster -> Cluster Config -> Cluster Information**]). Under */var/log/cloudian* on the node there are these files:

- *diagnostics.csv* -- This is the live file into which the current day's performance statistics are continuously written.
- *diagnostics_<date/time>_<version>_<region>.tgz*-- Once a day the current *diagnostics.csv* file is packaged -- together with application and transaction log files -- into a *diagnostics_<date/time>_<version>_<region>.tgz* file. This file is transmitted to Cloudian Support once each day. By default a local copy of each *diagnostics_<date/time>_<version>_<region>.tgz* file remains on the node for 15 days before being automatically deleted.

The Smart Support feature is **enabled by default**. You have the option of disabling the feature, although this is not recommended.

### Automatic Support Case Creation for Failed Disks

As part of the Smart Support feature, if a data disk on a HyperStore node fails (becomes disabled), information about the failed disk is automatically sent to Cloudian Support within minutes. This triggers the automatic opening of a Support case for the failed disk. This occurs in addition to the alerting functions that bring the failed disk to your attention.

For HyperStore Appliances, automatic case creation is also performed for failed OS disks.

For more information on automated disk failure handling see **"Automated Disk Usage Management Feature Overview"** (page 113).

## 3.2.1.2. On-Demand Node Diagnostics

HyperStore also supports a mechanism that allows you to easily collect deep diagnostic data for a node that you are having a problem with. You can use the CMC's Collect Diagnostics function (**Cluster -> Nodes -> Advanced**) to trigger the collection and packaging of this Node Diagnostics data and send the package to Cloudian Support so that they can help you troubleshoot the problem. So while the fully automated Smart Support feature allows for proactive monitoring and support of your system as a whole, the on-demand Node Diagnostics feature allows for deep-dive reactive troubleshooting for problems that have occurred with specific nodes.

The Node Diagnostics package includes:

- System log files and HyperStore log files for the target node(s)
- Outputs from various system commands and HyperStore application commands for the target node(s)
- MBean data for the Java-based HyperStore services for the target node(s)
- Configuration files from the target node(s)
- Puppet and Salt configuration files and logs from the system

On the target node(s),the Node Diagnostics mechanism packages all this data into */var/log/cloudian/cloudian_sysinfo/<hostname>_<YYYYMMDDhhmm>.tar.gz*. The system automatically sends this package to Cloudian Support, if you chose that option when you used the CMC to generate the package.

### Automatic Upload of Node Diagnostics for Support Cases

If you want, you can have the system automatically upload to Cloudian Support the node diagnostics logs that Cloudian Support needs whenever you open a support case. If you enable this feature, you will not need to collect and upload logs to Cloudian Support yourself when you open support cases. By default configuration this feature is disabled. For more information see *common.csv:* **"logcollect_enable"** (page 402).

## 3.2.2. Configuring Smart Support and Node Diagnostics

The HyperStore Smart Support and Node Diagnostics features work out of the box and do not require any configuration changes. Optionally you can do the following with configuration settings:

- Have the daily Smart Support upload go to an S3 destination other than Cloudian Support, by setting the *phonehome_uri*, *phonehome_bucket*, *phonehome_access_key*, and *phonehome_secret_key* settings in **common.csv**. For more information see **"phonehome_uri"** (page 401) and the subsequent setting descriptions.
- Have the daily Smart Support upload use a local forward proxy by setting the *phonehome_proxy_host*, *phonehome_proxy_port*, *phonehome_proxy_username*, and *phonehome_proxy_password* settings in *common.csv*. For more information see **"phonehome_proxy_host"** (page 400) and the subsequent setting descriptions.
- Disable the daily Smart Support upload by setting *phonehome.enabled* in **mts.properties.erb** to false. **This is not recommended.**
- Have on-demand Node Diagnostics uploads (triggered by your using the CMC's Collect Diagnostics function [**Cluster -> Nodes -> Advanced**]) go to an S3 destination other than Cloudian Support, by setting the *sysinfo.uri*, *sysinfo.bucket*, *sysinfo.accessKey*, and *sysinfo.secretKey* properties in *mts.properties.erb*. For more information see **"sysinfo.uri"** (page 465) and the subsequent property descriptions.
- Have the system automatically upload to Cloudian Support the node diagnostics logs that Cloudian Support needs to process a case that you've opened. If you enable this feature, you will not need to collect and upload logs to Cloudian Support yourself when you open support cases. By default configuration this feature is disabled. For more information see *common.csv:* **"logcollect_enable"** (page 402).

**Note** Be sure to do a Puppet push and restart the S3 Service if you edit any of these configuration settings. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

### 3.2.2.1. Other Configuration Considerations

#### Changing the Timing of the Daily Diagnostics Upload

When enabled, the daily diagnostics upload to an S3 URI is triggered by a HyperStore cron job. The timing of the cron job is configured in */etc/cloudian-<version>-puppet/modules/cloudians3/templates/cloudian-crontab.erb* on the Configuration Master node. If you want to edit the cron job configuration, look for the job that includes the string "phoneHome". If you edit the crontab, do a Puppet push. No service restart is necessary.

#### Deletion of Old Daily Diagnostics and Node Diagnostics Packages

The deletion of old diagnostics packages is managed by Puppet, as configured in *common.csv* by the **"cleanup_directories_byage_withmatch_timelimit"** (page 391) setting (for daily system diagnostics packages associated with the Smart Support feature) and the **"cleanup_sysinfo_logs_timelimit"** (page 392) setting (for on-demand Node Diagnostics packages that you've generated). By default these settings have Puppet delete the diagnostics packages after they are 15 days old. This presumes that you have left the Puppet daemons running in your HyperStore cluster, which is the default behavior. If you do not leave the Puppet daemons running the diagnostics logs will not be automatically deleted. In that case you should delete the old packages manually, since otherwise they will eventually consume a good deal of storage space.

If you edit either of these settings in *common.csv*, do a **Puppet push**. No service restart is necessary.

#### Encrypting Sensitive Fields in Uploaded Log File Copies to Protect Data Privacy

If you need to comply with the European Union's General Data Protection Regulation (GDPR), or if it's in keeping with your own organization's data privacy policies, you can have your HyperStore system encrypt personally identifiable user information and other sensitive values -- such as user IDs, email addresses, IP addresses, hostnames, and role session information -- in the log copies that get uploaded to Cloudian Support as part of the Smart Support and Node Diagnostics features. The encryption process uses an encryption key that is unique to your HyperStore system and is generated automatically when you do a fresh installation of -- or upgrade to -- HyperStore version 7.4 or newer. Cloudian does not have access to the encryption key, so Cloudian cannot decrypt the encrypted log field values. Only you can decrypt an encrypted log field value, using a tool that comes with your HyperStore system. Your ability to decrypt an encrypted log field value will come into play if Cloudian Support is analyzing logs while working with you to troubleshoot an issue, and Cloudian Support asks you to provide the decrypted value of a particular field from a relevant log entry.

> **Note** This encryption feature encrypts sensitive log field values only in the log file **copies** that are uploaded to Cloudian Support. It does not alter in any way the original log files.

This encryption feature is **disabled by default**. To enable this feature, so that user IDs, email addresses, IP addresses, hostnames, and role session information are encrypted in the log file copies that are uploaded to Cloudian Support as part of the Smart Support and Node Diagnostics features:

1. In **common.csv** set *phonehome_gdpr* to *true*.
2. Optionally, if you want to also encrypt bucket names and object names in the copies of the S3 request log (*cloudian-request-info.log*) that get uploaded to Cloudian Support as part of the Smart Support and Node Diagnostics features, in *common.csv* also set *phonehome_gdpr_bucket* to *true*.
3. After saving your change(s) to *common.csv*, do a Puppet push and then restart the S3 Service, the IAM Service, and the CMC. If you need instructions for this step see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

### Decrypting an Encrypted Log Field Value

If you enable the feature that encrypts sensitive fields in log copies uploaded to Cloudian Support (as described above), and the Cloudian Support team is analyzing log copies while working with you to troubleshoot a system issue, the Cloudian Support team may ask you to decrypt an encrypted value from a relevant log entry.

Here is an example of a log file entry with an encrypted value. An encrypted field appears as "Enc:<value>". In the example below the encrypted value is shown in bold:

```
2021-09-03 00:00:05,503|Enc:<3Wi5Nacd9lMJgNQJ:kk2krSUIcNWiHRlY>|healthCheck|115|0|0|0|115
|95|200|977d68ed-1f8c-1fa3-a7d9-a81e84492d6f|0|0|||Apache-HttpClient/4.5.9 (Java/11.0.
```

In this example, if Cloudian Support needed you to decrypt the value to aid in resolving a support case, Cloudian Support would provide you the encrypted string *3Wi5Nacd9lMJgNQJ:kk2krSUIcNWiHRlY* through the Support case.

You would then log into your Configuration Master node and run this command to decrypt the encrypted string:

```
# /opt/cloudian/tools/GDPR_decryption.py encrypted-string
```

In this example it would be:

```
# /opt/cloudian/tools/GDPR_decryption.py 3Wi5Nacd9lMJgNQJ:kk2krSUIcNWiHRlY
10.50.40.125
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)**, from any directory on the Configuration Master node you can run the decryption tool without specifying a path to the tool:

```
$ GDPR_decryption.py 3Wi5Nacd9lMJgNQJ:kk2krSUIcNWiHRlY
10.50.40.125
```

You would then provide Cloudian Support with the decrypted string value -- 10.50.40.125 in this example.

**Note** The *GDPR_decryption.py* tool uses your system's unique encryption key to decrypt the string.

## 3.2.3. Executing Node Diagnostics Collection

To generate deep-dive Node Diagnostics in support of troubleshooting one or more problem nodes, use the CMC's Collect Diagnostics function (**Cluster -> Nodes -> Advanced**). When using that function you will be able to select one or more nodes for which to collect diagnostics. You will also be able to choose whether to have the diagnostics package(s) that you generate get uploaded to Cloudian Support (or an alternative S3 destination, if you have **configured the system to support this option**).

A node's diagnostics package will be created on the node at path */var/log/cloudian/cloudian_sysinfo/<hostname>_<YYYYMMDDhhmm>.tar.gz*.

### 3.2.3.1. Automatic Upload of Node Diagnostics for Support Cases

If you want, you can have the system automatically upload to Cloudian Support the node diagnostics logs that Cloudian Support needs to process a case that you've opened. If you enable this feature, you will not need to collect and upload logs to Cloudian Support yourself when you open support cases. By default configuration this feature is disabled. For more information see *common.csv:* **"logcollect_enable"** (page 402).

### 3.2.4. Cloudian Support Tunnel

In some circumstances Cloudian Support may want to work with you through a "support tunnel" to help you resolve an issue with your HyperStore system. The support tunnel is implemented by a service that is bundled with the HyperStore product, but it is **disabled by default**.

At Cloudian Support's direction, you may be asked to use some of these support tunnel related commands on a HyperStore node:

| Action | Command |
|---|---|
| Enable the support tunnel service | *systemctl enable support-tunnel* |
| Start the support tunnel service | *systemctl start support-tunnel* |
| Check the support tunnel service status | *systemctl status support-tunnel*<br>(this status check is one method of obtaining a support "token" that Cloudian Support may transmit to you in some circumstances) |
| Stop the support tunnel service | *systemctl stop support-tunnel* |
| Disable the support tunnel service | *systemctl disable support-tunnel* |

Also, in some circumstances Cloudian Support may direct you to edit this support tunnel configuration file on a HyperStore node:

```
/etc/cloudian/support-tunnel/support-tunnel.yaml
```

**Note** Do not edit this file unless directed to do so by Cloudian Support.

If you are using the **HyperStore Shell** as a trusted user you can perform any of the above tasks, at the direction of Cloudian Support (you can run the *systemctl* commands and you can use **hspkg config commands** to view or edit the *support-tunnel.yaml* file).

## 3.3. Automated Disk Usage Management

### 3.3.1. Automated Disk Usage Management Feature Overview

*Subjects covered in this section:*

HyperStore has mechanisms for automatically correcting imbalances of data disk utilization on each node, and automatically discontinuing new writes to disks or nodes that are near capacity. Some aspects of these automated mechanisms are configurable.

#### 3.3.1.1. Automatic Correction of Substantial Disk Usage Imbalance

HyperStore is implemented in such a way that among multiple data storage disks on the same host the disk usage will typically be well balanced. However, the system also supports an automated mechanism for detecting and rectifying disk usage imbalance if it does occur. On a configurable interval (by default 72 hours) the system checks for a configurable degree of disk usage imbalance on each node (by default a 10% delta between a given disk's utilization and the average disk utilization on the node). If imbalance is detected whereby a particular disk's usage exceeds the node's average disk utilization by more than the configured delta, the system migrates one or more storage tokens from the over-used disk to less-used disks on the same node. From that point forward the less-used disks will store an increased share of newly uploaded object data while the over-used disk stores a reduced share of newly uploaded object data. Note that this mechanism **does not move existing data from one disk to another**; rather, it impacts the share of new data load allocated to each disk going forward.

The same imbalance-correcting logic applies if the system detects that a particular disk's usage is lower than the node's average disk utilization by more than the configured delta. The system automatically migrates one or more storage tokens from the node's more heavily utilized disks to the under-utilized disk.

You do not need to run a repair operation in connection with this disk usage balancing feature. The disk usage rebalancing mechanism is not intended to move existing data between disks. It is designed to impact only objects uploaded after the time that the automated token migration was executed. If you were to perform a repair, existing objects on the over-used disk will not follow a migrated token to the token's new home on a less-used disk. For more about how the token migration works see **"Dynamic Object Routing"** (page 116).

Note that the disk usage balancing feature applies only to **HyperStore data disks** (where S3 object data is stored). It does not apply to disks that store only Cassandra, Redis, or the OS.

For more information on configuring this feature see **"Configuring Disk Usage Balancing"** (page 117).

#### 3.3.1.2. Automatic Stop of Writes to a Disk at 90% Usage

Every 30 minutes the capacity usage level of each HyperStore data disk in the system is checked. If 90% or more of a disk's capacity has been filled the system automatically moves all of the disk's tokens to other disks that are at less than 90% usage, on the same node. The system moves a 90% full disk's tokens in a manner that takes into account the current usage levels among the remaining disks on the node: the lower a disk's current usage level, the more likely it is to receive some of the transferred tokens.

The object data on the 90% full disk remains readable and the disk will still support S3 get requests and delete requests, but any S3 writes associated with the token ranges that used to be on the disk are now directed to the new locations of those token ranges, on other disks on the same node. This disk-level **"stop-write"** condition triggers a warning message in the HyperStore application log, which in turn results in an alert being generated in the CMC's **Alerts** page (**Alerts -> Alerts**).

For a disk that reaches stop-write condition, if its capacity usage level later falls back below 90% it exits stop-write condition and the tokens that had been moved away from it are moved back to it.

**Note** When a disk at 90% usage level enters stop-write condition it stops receiving new S3 writes, but object data writes associated with **hsstool repair**, **hsstool repairec**, **hsstool rebalance**, and *hsstool*

*decommission* operations continue to be supported until the disk reaches 95% usage. If the disk reaches 95% usage, then subsequent writes that *hsstool* operations target to that disk will fail (and in the operation's status metrics these failures will increment the "Failed count").

**Note** If you want to customize the disk usage check interval (default 30 minutes) or the "stop-write" threshold (default 90%) or the *hssstool* operation write threshold (default 95%) or the "start-write" threshold (default 85%), consult with Cloudian Support.

### 3.3.1.3. Automatic Stop of Writes to a Node at 90% Usage

If **all** of a node's data disks have reached a stop-write condition -- because each disk has hit 90% disk usage -- then the **whole node goes into a "stop-write" condition**. When a node is in "stop-write" condition:

- **The S3 Service is not able to write object data to the token ranges assigned to that node. This may result in the failure of some S3 PUT requests from client applications**. Whether failures occur or not depends on the consistency requirements that you have configured for your **storage policies**, and on the availability of the other nodes in your system. For example if you are using 3X replication with a write consistency level of ALL, then an S3 PUT of an object will fail if one of the three endpoint token ranges for the object (as determined automatically by a hash of the bucket name and object name) is a token range on the node that's in stop-write condition. With 3X replication and a write consistency level of QUORUM, then an S3 PUT of an object will fail if one of the object's three endpoint token ranges is on the stop-write node and either of the nodes hosting the object's other two endpoint token ranges is also unavailable (such as if one of those other nodes is down, or is in stop-write condition). Note that HyperStore does **not** reallocate tokens from a node that's in stop-write condition to other nodes in the system. **Dynamic token reallocation** is supported only between the disks on a node -- not between nodes.

  **Note** After detecting that a node is in stop-write condition, the S3 Service will mark that node as unavailable for object data writes and will stop sending object data write requests to that node (rather than continuing to send object data write requests to that node and having all those requests fail).

- The S3 Service is still able -- while implementing requests from S3 client applications -- to read object data from that node and to delete object data from that node.
- Object metadata can still be written to the Cassandra database on that node, since this is on the OS and metadata disk(s) not the data disks. The stop-write feature applies only to the data disks.
- The **hsstool repair**, **hsstool repairec**, and *hsstool decommission* operations cannot write to a node that's in stop-write condition. If a node is in stop-write condition, then writes that *hsstool* operations direct to that node will fail (and in the operation's status metrics these failures will increment the "Failed count").
- When a node goes into stop-write condition a critical message appears in the CMC's **Dashboard** page (  ); an alert is generated in the **Alerts** page (**Alerts -> Alerts**); and the node is marked by a red disk-stack icon in the **Data Centers** page (**Cluster -> Data Centers**).

**IMPORTANT !** To avoid having disks and nodes go into a stop-write condition, closely monitor your system's current and projected disk space usage and **expand your cluster well in advance of disks and nodes becoming nearly full**. See **"Capacity Monitoring and Expansion"** (page 338).

### Getting a Node Out of Stop-Write Condition

A node will exit the "stop-write" condition -- and the S3 layer will resume sending object data write requests to the node -- if enough data is deleted from the node's data disks to reduce the node's **average disk utilization to 85% usage or lower**. At this point all of the node's tokens -- and consequently all of the S3 object data writes on the node -- will be allocated to the data disks that are currently at 85% usage or lower. Disks that are still above 85% utilization will not be assigned any tokens and will not support writes.

When you have a node in stop-write condition there are two ways in which disk space utilization on the node can be reduced such that the node starts accepting writes again:

- You can **delete objects from your HyperStore system.** In HyperStore each object's replicas or erasure coded fragments are stored on multiple nodes and there is not a way to target only the replicas and fragments on a specific node for deletion. Rather, you can delete objects from the system as a whole so that the overall disk space utilization in the system is reduced. This will have the effect of also reducing disk space utilization on the node that's in stop-write condition. You can delete objects either through the S3 interface or through the special Admin API call that lets you efficiently delete all the objects in a specified bucket (see bucketops). You can use the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**) to periodically check the node's disk space usage.

  **Note** When you delete objects using the S3 interface or Admin API, the objects are immediately flagged for deletion but the data is not actually removed from disk until the running of the hourly batch delete cron job.

- You can **add nodes to your cluster**, and execute the associated rebalancing and cleanup operations (for instructions see **"Adding Nodes"** (page 264)). This will have the effect of reducing data utilization on your existing nodes, including the node that's in the stop-write condition. Note that rebalance and cleanup are long-running operations and so this approach to getting a node out of stop-write condition may take several days or more depending on how much data is in your system.

## 3.3.1.4. Dynamic Object Routing

In implementing its disk usage balancing and disk stop-write features the system uses a dynamic token-to-disk mapping scheme that allows tokens to be migrated from one disk to another disk on the same host without existing data following the migrated tokens. With Dynamic Object Routing, an object replica (or EC fragment) associated with a given token range **stays with the disk where that token range was located when the object was originally uploaded**. The system keeps track of the location of each token -- the host and mount-point to which each token is assigned -- across time. Then, based on an object's creation timestamp, the system can tell where the object's replicas (or EC fragments) are located based on where the relevant token ranges were located at the time that the object was first uploaded. In this way object data can be kept in place -- and read or updated at its original location -- even if tokens are moved.

Dynamic Object Routing allows for low-impact automated token migrations in circumstances such as disk usage imbalance remediation, disk stop-write implementation, and **automated disk failure handling**.

**Note** This automated token migration occurs **only between disks on a node -- not between nodes**.

## 3.3.2. Configuring Disk Usage Balancing

With the HyperStore data disk usage balancing feature there are the questions of how often to check the disk usage balance on each host and what degree of imbalance should trigger a token migration. Both of these factors are configurable.

By default, each node is checked for disk imbalance every 72 hours, and token migration is triggered if a disk's utilization percentage differs from the average disk utilization percentage on the node by more than 10%. For example, if the average disk space utilization on a node is 35%, and the disk space utilization for Disk4 is 55%, then one or more tokens will be migrated away from Disk4 to other disks on the node (since the actual delta of 20% exceeds the maximum allowed delta of 10%). For another example, if the average disk utilization on a node is 40%, and the disk utilization for Disk7 is 25%, then one or more tokens will be migrated to Disk7 from the other disks on the node.

The settings for adjusting the frequency of the disk balance check or the delta that triggers disk migration are:

- **"hyperstore_disk_check_interval"** (page 400) in **common.csv** (default = 72 hours)
- **"disk.balance.delta"** (page 439) in **hyperstore-server.properties.erb** (default = 10 percent)

After editing either of these settings, be sure to push your changes to the cluster and restart the HyperStore Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**Note** In connection with the HyperStore "**stop-write**" feature, if you want to customize the disk usage check interval (default 30 minutes) or the "stop-write" threshold (default 90%) or the "start-write" threshold (default 85%), consult with Cloudian Support. These settings are not in HyperStore configuration files by default.

## 3.3.3. Triggering a Disk Usage Balance Check

If you don't want to wait until the next automatic check of disk usage balance (which by default occurs every 72 hours for each node), you have the option of using a JMX command to immediately trigger a disk balance check on a specific node.

To trigger an immediate disk usage balance check:

1. Use *JConsole* to access the HyperStore Service's JMX port (19082 by default) on the host for which you want the balance check to be performed.
2. Access the *com.gemini.cloudian.hybrid.server.disk → VirtualNodePartitioner* MBean, and under "Operations" execute the "shuffletoken" operation.

The operation will run in a background thread and may take some time to complete. If the space utilization for any disk is found to differ from the node's average disk utilization by more than the **configured maximum delta** (10% by default), then tokens will automatically be migrated between disks.

### 3.3.4. Checking Disk Usage and Health Status

You can use the CMC to check disk status for each disk on each node in your HyperStore system. Go to the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**), select a node, and then review the "Disk Detail Info" section of the page. Here you will see

- The current space utilization information for each disk on the selected node.
- The current health status of each disk on the selected node. Each data disk's status is communicated by a color-coded icon, with the status being one of OK, Error, or Disabled.

## 3.4. Automated Disk Failure Management

### 3.4.1. Automated Disk Failure Management Feature Overview

In the event of failure of a HyperStore data disk — a disk where S3 object data is stored — the HyperStore system by default automatically detects the failure and takes the disk offline. To detect disk failures, for each node the HyperStore system does the following:

- Continuously monitors the HyperStore Service application log for error messages indicating a failure to read from or write to a disk (messages containing the string "HSDISKERROR").
- At a configurable interval (default is once each hour), tries to write one byte of data to each HyperStore data disk. If any of these writes fail, */var/log/messages* is scanned for messages indicating that the file system associated with the disk drive in question is in a read-only condition (message containing the string "Remounting filesystem read-only"). This recurring audit of disk drive health is designed to proactively detect disk problems even during periods when there is no HyperStore Service read/write activity on a disk.

If HyperStore Service application errors regarding a drive occur in excess of a configurable error rate threshold, or if the proactive audit detects that a drive is in read-only condition, then HyperStore by default **automatically disables the drive**.

When a drive is automatically disabled, the system will no longer direct writes or reads to that drive. The storage tokens from the disabled disk are automatically moved to the other disks on the host, so that new writes associated with those token ranges can be directed to the other disks. The disabled disk's data is not recreated on the other disks, and so that data is unreadable on the host.

For more information on configuring this feature see **" Configuring Disk Failure Handling"** (page 119).

**Note** You can tell that a disk is disabled by viewing its status in the "Disk Detail Info" section of the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**).

**Note** As part of the **Smart Support** feature, if a data disk on one of your HyperStore nodes fails (becomes disabled), information about the failed disk is automatically sent to Cloudian Support within minutes. This triggers the automatic opening of a Support case for the failed disk. For HyperStore Appliances, automatic case creation is also performed for failed OS disks.

#### 3.4.1.1. Limitations on Automatic Disk Failure Handling

The automatic disk disabling feature works only if you have multiple HyperStore data disks on the host. If there is only one HyperStore data disk on the host, the system will not automatically disable the disk even if errors are detected.

Also, the automatic disk disabling feature works only if you are using *ext4* file systems mounted on raw disks (which is the only officially supported configuration). If you've installed HyperStore on nodes with unsupported technologies such as Logical Volume Manager (LVM) or XFS file systems, the automatic disk disabling feature will be deactivated by default and the HyperStore system will not take any automatic action in regard to disk failure. Also, the automatic disk disabling feature does not work in Xen or Amazon EC2 environments. Contact Cloudian Support if you are using any of these technologies.

### 3.4.2. Configuring Disk Failure Handling

**IMPORTANT !** The automatic disk failure handling feature does not work correctly in Xen, Logical Volume Manager (LVM), or Amazon EC2 environments. Contact Cloudian Support if you are using any of these technologies.

Several aspects of the HyperStore automated disk failure handling feature are configurable.

In the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**), in the "System Settings" section, you can configure a "HyperStore Disk Failure Action". This is the automated action for the system to take in the event of a detected disk failure, and the options are "Disable Disk + Move Its Tokens" (the default) or "None". (For more detail, while on the CMC's **Configuration Settings** page click **Help**.) If you change this setting in the **Configuration Settings** page, your change is dynamically applied to the cluster — no service restart is necessary.

Additional settings are available in **hyperstore-server.properties.erb** on your Configuration Master node. These include settings for establishing an error rate threshold for disk-related errors in the HyperStore Service application log (which if exceeded for a given disk will result in the automatic triggering of the Disk Failure Action):

- **"disk.fail.error.count.threshold"** (page 439)
- **"disk.fail.error.time.threshold"** (page 439)

By default the threshold is 100 errors in the space of 30 minutes, for a particular disk.

Also in *hyperstore-server.properties.erb* is this setting for the interval at which to conduct the proactive disk drive audit:

- **"disk.audit.interval"** (page 439)

By default the proactive audit occurs hourly.

After editing any of these *hyperstore-server.properties.erb* settings, push the changes to the cluster and restart the HyperStore Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**Note** As part of the **Smart Support** feature, if a data disk on one of your HyperStore nodes fails (becomes disabled), information about the failed disk is automatically sent to Cloudian Support within minutes. This triggers the automatic opening of a Support case for the failed disk. For HyperStore Appliances, automatic case creation is also performed for failed OS disks.

## 3.4.3. Disk Error Alerts

HyperStore sends a "Disk Error" alert email to the system administrator(s) if a disk read/write error occurs in the HyperStore Service application log. An alert also appears in the CMC, in both the **Alerts** page (**Alerts -> Alerts**) and the **Node Status** page (**Cluster -> Nodes -> Node Status**).

Note that such an alert does not necessarily indicate that the disk has been automatically disabled. This is because the alert is triggered by the appearance of a single "HSDISKERROR" message in the HyperStore Service application log, whereas the automatic disabling action is triggered only if such messages appear at a rate exceeding the **configurable threshold**.

## 3.4.4. Responding to a Disabled Disk

If a disk has been automatically disabled by HyperStore in response to disk failure, you have two options for restoring service on that drive:

- **Re-enable the same disk**. You might choose this option if, for example, you know that some cause other than a faulty disk resulted in the drive errors and the automatic disabling of the disk.

  HyperStore provides a highly automated method for bringing the same disk back online. For instructions see **"Enabling a HyperStore Data Disk"** (page 330).

- **Replace the disk**. You would choose this option if you have reason to believe that the disk is bad.

  HyperStore provides a highly automated method for replacing a disk and restoring data to the new disk. For instructions see **"Replacing a HyperStore Data Disk"** (page 332).

With either of these methods, **any tokens that were migrated away from the disabled disk will be automatically migrated back to it (or its replacement)**. Object data that was written in association with the affected tokens while the disk was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks (utilizing HyperStore **"Dynamic Object Routing"** (page 116)).

**Note** As part of the **Smart Support** feature, if a data disk on one of your HyperStore nodes fails (becomes disabled), information about the failed disk is automatically sent to Cloudian Support within minutes. This triggers the automatic opening of a Support case for the failed disk. For HyperStore Appliances, automatic case creation is also performed for failed OS disks.

## 3.4.5. Checking Disk Usage and Health Status

You can use the CMC to check disk status for each disk on each node in your HyperStore system. Go to the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**), select a node, and then review the "Disk Detail Info" section of the page. Here you will see

- The current space utilization information for each disk on the selected node.
- The current health status of each disk on the selected node. Each data disk's status is communicated by a color-coded icon, with the status being one of OK, Error, or Disabled.

## 3.5. Setting Up Alerts and Notifications

HyperStore has a variety of pre-configured alert rules, mostly geared toward hardware utilization levels (such as disk capacity usage or CPU utilization) and services status (such as a service being down or having errors in its application log). An alert rule defines the condition that triggers a particular alert, and how the system will communicate the alert. You can review the pre-configured alert rules, edit the rules, and create additional rules in the CMC's **Alert Rules** page.. For more information, while logged into the CMC's **Alert Rules** page click **Help**.

By default, alerts are communicated to you only by appearing in the CMC's **Alerts** page. However, **it is strongly recommended that you supply a system administrator email address** to which HyperStore will send notification emails when alerts are triggered. You can supply the email information (system administrator address and other required information such as your SMTP server's FQDN) in the CMC's **Configuration Settings** page. . For more information, while logged into the CMC's **Configuration Settings** page click **Help**. Once you've supplied the system with the information required to send emails to a system administrator, the pre-configured alert rules will include sending email to the administrator's address when an alert is triggered.

If you wish you can also have HyperStore send SNMP traps when alerts are triggered. To do so you need to take two steps:

1. Supply the SNMP destination information in the CMC's **Configuration Settings** page. . For more information, while logged into the CMC's **Configuration Settings** page click **Help**.
2. In the CMC's **Alert Rules** page, modify each of the pre-configured alert rules so that they include sending an SNMP trap. (Or modify a subset of the alert rules if you want SNMP traps sent for some alerts and not for others.) The pre-configured alert rules do not include sending an SNMP trap by default.

## 3.6. Creating Additional System Admin Users

HyperStore comes with one pre-provisioned system administrative user, with user name *admin*. You can log into the CMC as the *admin* user -- as described in **"Accessing the Cloudian Management Console"** (page 89) -- and perform a variety of system monitoring and administration tasks, including provisioning service users and user groups.

While logged into the CMC as the *admin* user you can use the CMC's **Manage Users** page (**Users & Groups -> Manage Users**) to create additional system admin users if you wish (with different user names). When creating a new user in that page you can select the user type from a drop-down list, and "System Admin" is one of the options. Once created, a new system admin user will have the same capabilities as the *admin* user: they will be able to log into the CMC and perform system monitoring and administration tasks, including user provisioning (and including the ability to create additional system admin users).

System admin users can also delete other system admin users, with the exception that the pre-provisioned system admin user named *admin* cannot be deleted.

If you have the **HyperStore Shell** enabled, each system admin user will be able to log into and use the HyperStore Shell. For more information see **"Enabling the HSH and Managing HSH Users"** (page 96).

**Note** You can also create and manage system admin users through the Admin API, similar to creating and managing other types of users. For more information see the "user" section of the ***Cloudian HyperStore Admin API Reference***.

## 3.7. Aggregating Logs to a Central Server

If you wish you can have application logs and request logs from the S3 Service, Admin Service, and HyperStore Service — as well as system logs from the host machines in your cluster — aggregated to a central logging server using *rsyslog*. This is not enabled by default, but you can enable it by editing configuration files. This procedure sets up your HyperStore logging so that logs are written to a central logging server **in addition to** being written on each HyperStore node locally.

The procedure presumes that:

- You have a central logging server running *rsyslog*.
- Each of your HyperStore nodes has *rsyslog* installed.

**IMPORTANT !** The central logging server must **not** be one of your HyperStore nodes.

**Note** *rsyslog* is included in the HyperStore Appliance and also in standard RHEL/CentOS distributions. This procedure has been tested using *rsyslog* v5.8.10.

To aggregate HyperStore application logs, request logs, and system logs to a central logging server follow the instructions below.

**1. On the central logging server**, do the following:

**a)** In the configuration file */etc/rsyslog.conf*, enable UDP by uncommenting these lines:

```
# BEFORE EDITING

#$ModLoad imudp
#$UDPServerRun 514

# AFTER EDITING

$ModLoad imudp
$UDPServerRun 514
```

**b)** Still on the central logging server, create a file */etc/rsyslog.d/cloudian.conf* and enter these configuration lines in the file:

```
$template CloudianTmpl,"%HOSTNAME% %TIMESTAMP:::date-rfc3339% %syslogseverity-text:::uppercase%
%msg%\n"
$template CloudianReqTmpl,"%HOSTNAME% %msg%\n"
:programname, isequal, "ADMINAPP" /var/log/cloudian-admin.log;CloudianTmpl
:programname, isequal, "ADMINREQ" /var/log/cloudian-admin-request-info.log;CloudianReqTmpl
:programname, isequal, "HSAPP"   /var/log/cloudian-hyperstore.log;CloudianTmpl
:programname, isequal, "HSREQ" /var/log/cloudian-hyperstore-request-info.log;CloudianReqTmpl
:programname, isequal, "S3APP"  /var/log/cloudian-s3.log;CloudianTmpl
```

```
:programname, isequal, "S3REQ"  /var/log/cloudian-request-info.log;CloudianReqTmpl
:programname, isequal, "s3-worm"  /var/log/s3-worm.log;CloudianReqTmpl
```

**c)** Still on the central logging server, restart *rsyslog* by "service rsyslog restart".

**d)** Still on the central logging server, enable rotation on the centralized HyperStore logs. For example, to use *logrotate* for rotating the HyperStore logs, create a file */etc/logrotate.d/cloudian* and enter the following configuration lines in the file. (Optionally adjust the rotated file retention scheme — 14 rotations before deletion in the example below — to match your retention policy.)

```
/var/log/cloudian-*.log
{
   daily
   rotate 14
   create
   missingok
   compress
   delaycompress
   sharedscripts
   postrotate
      /bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true
   endscript
}
```

**2. On your HyperStore Configuration Master node**, do the following:

**a)** Go to the */etc/cloudian-7.5.2-puppet/modules/cloudians3/templates/* directory. Here you will edit three configuration files:

- *log4j-admin.xml.erb*
- *log4j-hyperstore.xml.erb*
- *log4j-s3.xml.erb*

In each of these three files **search for and uncomment all sections marked with a "*#syslog*" tag**. When you are done uncommenting, there should be no remaining "*#syslog*" tags.

**Also**, in the first *#syslog* section at the top of each file — nested within the "Properties" block — in addition to uncommenting the *#syslog* section set the *sysloghost* property to the hostname or IP address or your central syslog server and set the *syslogport* property to the central syslog server's UDP port (which by default is port number is 514).

Here is a before and after example for uncommenting and editing the *#syslog* section within a "Properties" block. In the BEFORE EDITING text, the commenting boundaries (which need to be removed) are highlighted in **red**. In the AFTER EDITING text, the commenting boundaries have been removed and the central logging server hostname has been set to "regulus".

```
# BEFORE EDITING

<Properties>
   <!-- #syslog
   <Property name="sysloghost">localhost</Property>
   <Property name="syslogport">514</Property>
   -->
</Properties>

# AFTER EDITING
```

```
<Properties>
   <Property name="sysloghost">regulus</Property>
   <Property name="syslogport">514</Property>
</Properties>
```

Be sure to uncomment **all of the "*#syslog*" tagged sections in each of the three files**. Remember that only the *#syslog* section within the "Properties" block at the top of each file requires editing an attribute value. The rest of the *#syslog* sections only require uncommenting.

In this example of a *#syslog* section in *log4j-s3.xml.erb* you would remove the commenting boundaries that here are highlighted in **red**.

```
<!--  #syslog
<Syslog name="SYSLOG-S3APP" format="RFC5424" host="${sysloghost}" port="${syslogport}"
   protocol="UDP" appName="S3APP" mdcId="mdc" includeMDC="true" facility="USER" newLine="true">
</Syslog>
<Syslog name="SYSLOG-S3REQ" format="RFC5424" host="${sysloghost}" port="${syslogport}"
   protocol="UDP" appName="S3REQ" mdcId="mdc" includeMDC="true" facility="USER" newLine="true">
</Syslog>
<Syslog name="SYSLOG-S3WORM" format="RFC5424" host="${sysloghost}" port="${syslogport}"
   protocol="UDP" appName="s3-worm" mdcId="mdc" includeMDC="true" facility="USER" newLine="true">
</Syslog>
-->
```

Here is a second example of a section that needs uncommenting, from that same file:

```
<!-- Request Logger -->
<Logger name="com.gemini.cloudian.RequestLogger" additivity="false" level="INFO">
  <AppenderRef ref="S3REQ" />
  <!-- #syslog
  <AppenderRef ref="SYSLOG-S3REQ" />
  -->
</Logger>
```

**b)** Still on your HyperStore Configuration Master node, go to the */etc/cloudian-<version>-puppet/modules/rsyslog/templates/* directory. Then edit *loghost.conf.erb* to uncomment the following line and replace "<loghost>" with the hostname (or IP address) of your central syslog server host:

```
# BEFORE EDITING
#*.* @<loghost>:514

# AFTER EDITING

*.* @regulus:514
```

**IMPORTANT !** The central log host must **not** be one of your HyperStore hosts. (The reason is that if you have one of your HyperStore hosts acting as the central log host, then that host is sending logs to itself which results in a loop and rapid proliferation of log messages.)

**c)** Still on your HyperStore Configuration Master node, use the installer to push your changes to the cluster and to restart the HyperStore Service and the S3 Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**3. Back on the central logging server**:

**a)** Confirm that logs are being written in */var/log/cloudian-** files.

**b)** Confirm that system messages from HyperStore nodes appear on the log host (for example in */var/-log/messages*). If you want you can proactively test this by running the command "logger test 1" on any HyperStore node.

This page left intentionally blank

# Chapter 4. Setting Up Service Features

## 4.1. Storage Policies

### 4.1.1. Storage Policies Feature Overview

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Supported Erasure Coding Configurations for a Single DC"*** *(page 129)*
- ***"Multi- Data Center Storage Policies"*** *(page 129)*
- ***"Hybrid Storage Policies"*** *(page 130)*

Storage policies are ways of protecting data so that it's durable and highly available to users. The HyperStore system lets you pre-configure one or more storage policies. When users create a new storage bucket they choose which pre-configured storage policy to use to protect data in that bucket. **Users cannot create buckets until you have created at least one storage policy.**

For each storage policy that you create you can choose from either of two data protection methods:

- **Replication** — With replication, a configurable number of copies of each data object are maintained in the system, and each copy is stored on a different node. For example, with 3X replication 3 copies of each object are stored, with each copy on a different node.
- **Erasure coding** — With erasure coding, each object is encoded into a configurable number (known as the "k" value) of data fragments plus a configurable number (the "m" value) of redundant parity fragments. Each of an object's "k" plus "m" fragments is unique, and each fragment is stored on a different node. The object can be decoded from any "k" number of fragments. To put it another way: the object remains readable even if "m" number of nodes are unavailable. For example, in a 4+2 erasure coding configuration (4 data fragments plus 2 parity fragments), each object is encoded into a total of 6 unique fragments which are stored on 6 different nodes, and the object can be decoded and read so long as any 4 of those 6 fragments are available.

In general, erasure coding requires less storage overhead -- the amount of storage consumption above and beyond the original size of the stored objects, in order to ensure data persistence and availability -- than replication. Put differently, for a given degree of data protection erasure coding is more efficient in utilizing raw storage capacity than is replication.

For example, while 3X replication incurs a 200% storage overhead, 4+2 erasure coding incurs only a 50% storage overhead. Or stated in terms of storage capacity utilization efficiency, 3X replication is 33% efficient (for instance with 12TB of available storage capacity you can store 4TB of net object data) whereas 4+2 erasure coding is 67% efficient (with 12TB of available storage capacity you can store 8TB of net object data). On the other hand, erasure coding results in somewhat longer request processing latency than replication, due to the need for encoding/decoding.

In light of its benefits and drawbacks, erasure coding is best suited to **long-term storage of large objects that are infrequently accessed**.

**Note** Storage policies that you create with HyperStore 7.5.1 or later can use a hybrid approach as noted in **"Hybrid Storage Policies"** (page 130) below.

Regardless of whether you use replication or erasure coding, if your HyperStore system spans multiple data centers, for each storage policy you can also choose how data is allocated across your data centers — for example, you could have a storage policy that for each S3 object stores 3 replicas of the object in each of your data centers; and a second storage policy that erasure codes objects and stores them in just one particular data center (for more information see **"Multi- Data Center Storage Policies"** (page 129)).

Also as part of the configuration options for each storage policy, you can choose whether to compress and/or encrypt stored objects. You can also optionally choose a workload type, such as one of the major backup applications. The workload type feature leverages rules-based partitioning of object metadata storage to optimize *ListObjects* performance. Also, if you have set up **object metadata search** in your HyperStore system, for each storage policy you can choose whether to enable object metadata search for the buckets that use that storage policy.

Individual storage policies are **not confined to dedicated nodes or disks**. Instead, all policies utilize all the resources of your cluster, and data stored in association with a particular policy will tend to be spread fairly evenly across the cluster (with the exception that you can limit a policy to a particular data center as noted above). This helps to ensure that regardless of how many or what types of storage policies you configure, and regardless of how much data is stored in association with particular policies, the physical resources of your entire cluster — disks, CPU, RAM — will be used in an approximately even manner.

**Note** By default the system allows a maximum of 25 storage policies. This maximum is controlled by the setting **"cloudian.protection.policy.max"** (page 466) in the configuration file **mts.properties.erb**. Edit this setting if you want to have more than 25 storage policies in your HyperStore system.

**IMPORTANT ! Erasure coding storage policies created with HyperStore versions 5.x or 6.x included a functionality that has been deprecated.** The functionality, which identified more than *k+m* endpoints as possible write destinations for an erasure coded object's *k+m* fragments, ultimately was found to pose potential problems that outweighed the benefits (and so erasure coding storage policies created in HyperStore 7.0.1 and later returned to using the more conventional approach of identifying only *k+m* endpoints for erasure coded writes). Legacy storage policies with the deprecated functionality will no longer be supported as of HyperStore 8.1 (a future HyperStore release). Starting with HyperStore 7.5.2, the system automatically detects if you have any of these deprecated storage policies in your system. If you do have any of these deprecated storage policies, then:

* A warning will display in the CMC **Dashboard**. The warning will indicate that you have one or more deprecated storage policies in your system and that you will need to contact your Cloudian Sales Representative to plan for migrating data from those deprecated storage policies to supported storage policies prior to the HyperStore 8.1 release.
* In the CMC's **Storage Policies** page -- which lists all your storage policies -- a warning symbol will display next to the names of the deprecated storage policy or policies. Hover text shows the warning description.
* You cannot edit a deprecated storage policy. However, you can delete a deprecated storage policy if no buckets are using the storage policy. (Since a bucket cannot be switched from the storage policy it's using to a different storage policy, the only way you would be able to delete a storage policy is if all buckets that had been using that policy are deleted first.)

* Users creating new buckets will not be able to use a deprecated storage policy (such policies will not display in the list of available storage policies that users see when creating a new bucket).

#### 4.1.1.1. Supported Erasure Coding Configurations for a Single DC

HyperStore supports several erasure coding configurations, in terms of "k" value (number of data fragments) and "m" value (number of parity fragments):

- 4+2
- 6+2
- 8+2
- 9+3
- 12+4

The choice among these supported EC configurations is largely a matter of how many HyperStore nodes you have in the data center. For example, compared to a 4+2 configuration, 6+2 EC provides the same degree of data availability assurance (objects can be read even if 2 of the involved nodes are unavailable), while delivering a higher level of storage efficiency (4+2 is 67% efficient whereas 6+2 is 75% efficient). So 6+2 may be preferable to 4+2 if you have at least 8 HyperStore nodes in the data center.

Likewise, 9+3 EC provides a higher degree of protection and availability than 6+2 EC (since with 9+3 EC, objects can be read even if 3 of the involved nodes are unavailable) while delivering the same level of storage efficiency (both 6+2 and 9+3 are 75% efficient). So 9+3 may be preferable to 6+2 if you have at least 12 HyperStore nodes in the data center.

**Note** If you want to use a *k+m* configuration other than those mentioned above, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

**Note** For detailed information on S3 write and read availability under various combinations of cluster size and storage policy configuration, see **"Storage Policy Resilience to Downed Nodes"** (page 136).

#### 4.1.1.2. Multi- Data Center Storage Policies

If your HyperStore system is deployed across multiple data centers, for each storage policy that you create you can configure a data center assignment scheme for the policy. This determines which of your data centers to use for storing data, for each storage policy.

**For storage policies that use replication only**, in a multiple data center environment you can choose how many replicas to store in each data center -- for example, for each object store 3 replicas in DC1 and 2 replicas in DC2.

For erasure coding storage policies, you have the option of replicating the k+m fragments in each of the participating DCs (so that each participating DC stores k+m fragments), or distributing the k+m fragments across the participating DCs (so that there are a combined total of k+m fragments across the participating DCs).

**For replicated erasure coding**, by default your k+m options are:

- 4+2
- 6+2
- 8+2
- 9+3
- 12+4

In each of the above configurations the k+m fragments can be replicated across multiple DCs.

**For distributed erasure coding**, the supported options depend on how many data centers you are using in the storage policy. You must use at least 3 DCs for this type of policy, and by default your k+m options are as indicated in the table below:

| # of Participating DCs | Supported "k"+"m" | How Fragments Will Be Distributed |
|---|---|---|
| 3 | 5+4 | 3 fragments per DC |
| | 7+5 | 4 fragments per DC |
| 4 | 8+4 | 3 fragments per DC |
| 5 | 6+4 | 2 fragments per DC |
| 6 | 8+4 | 2 fragments per DC |
| | 7+5 | 2 fragments per DC |
| 7 | 10+4 | 2 fragments per DC |
| 8 | 10+6 | 2 fragments per DC |
| 9 | 10+8 | 2 fragments per DC |

**Note** If you want to use a *k+m* configuration other than those mentioned above, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

**Note** For any type of storage policy in a multiple data center environment, you have the option of configuring the policy such that data is stored in some of your data centers and not others — for example, you can create a policy that stores data in DC1 and DC2 but not in DC3. Note, however, that DC3 may be involved in processing S3 requests associated with buckets that use this policy. By default there is only one S3 service endpoint per region, and incoming S3 requests may resolve to any DC within the region. If the S3 Service in DC3 receives an S3 PUT request in association with a policy that stores data only in DC1 and DC2, it will transmit the uploaded object on to DC1 and D2 (it will not be stored in DC3). Likewise, if DC3 receives an S3 GET request in association with a policy that stores data only in DC1 and DC2, then DC3's S3 Service will get the object from DC1 or DC2 and pass it on to the client. If you want more absolute barriers so that for example DC3 never touches DC2's data and vice-versa, you need to set up your system so those DCs are in different **service regions**.

### 4.1.1.3. Hybrid Storage Policies

Starting with HyperStore version 7.5.1: When creating a new storage policy that uses an erasure coding data distribution scheme, you can set an object size threshold such that objects larger than the threshold will be

erasure coded and objects at or smaller than the threshold will be protected by replication rather than erasure coding. By default the threshold is 200KiB.

This Hybrid Storage Policy feature allows for a "best of both" storage strategy, given that key data storage objectives are best served by erasure coding for large objects and by replication for small objects.

While on the CMC's **Storage Policies** page, click **Help** and then view the "Add a Storage Policy" topic

> **Note** This feature is supported only for new storage policies created in HyperStore 7.5.1 or later.

## 4.1.2. Consistency Levels

To boost data durability and availability, HyperStore implements replication or erasure coding for object data and replication for object metadata. This entails distributing each object's data and metadata to multiple endpoint nodes across the cluster. When you create storage policies, along with configuring a replication or erasure coding scheme you will also configure consistency levels for writes and reads. Consistency levels impose requirements as to **what portion of the data and metadata writes or reads associated with each S3 request must be successfully completed before the system can return a success response** to the S3 client. If the consistency requirements cannot be met for a given S3 request at a given time -- due to one or more endpoint nodes being unavailable -- an HTTP 503 error response is returned to the client. An endpoint node could be unavailable for example because the node is down, or is unreachable on the network, or (in the case of writes of object data) is in **"stop-write" condition**.

Below is the list of consistency levels supported by the HyperStore system. Your consistency level options when configuring a storage policy will be limited by the data distribution scheme (replication or erasure coding, single DC or multi-DC) that you have selected for that policy.

- ALL
- QUORUM
- EACH QUORUM
- LOCAL QUORUM
- ANY QUORUM
- ONE

For descriptions of these consistency settings, while on the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) click **Help**.

For detailed information on S3 write and read availability under various combinations of cluster size, data distribution scheme, and consistency level settings, see **"Storage Policy Resilience to Downed Nodes"** (page 136).

For a description of how HyperStore by default ensures **strong read-after-write consistency**, see **"Strong Read-After-Write Consistency"** (page 76).

> **Note** In the case of writes, if the consistency requirement is met by something less than completing writes of all replicas (or all erasure coded fragments), then after returning a success response to the client the system continues to try to complete the remaining writes. If any of these writes fail they will later be recreated by **automatic data repair**.

**Note** As an advanced option you can also configure "dynamic" consistency levels, whereby the system will try to achieve a "fallback" consistency level if the primary consistency level cannot be achieved. For more information see **"Dynamic Consistency Levels"** (page 77).

#### 4.1.2.1. Note About Object Data Replica Reads

For replication based storage policies, the descriptions and examples in this documentation state that part of the read consistency requirement is being able to read X number of object data replicas. This is a simplification. Technically, what needs to be readable in order to satisfy a read consistency requirement is the **file digests** of X number of object data replicas. A **file digest** is an object data file "header" -- a small bit of file-identifying information including file name, size, timestamp, and MD5 hash -- which is stored in RocksDB on the same disk as the corresponding object data replica file. To determine whether or not an object data replica file is present on a given endpoint, the system tries to read that object data replica's file digest. This is much faster than reading the object data file itself.

If the read consistency requirements are met for an S3 GET operation -- for reading the required number of object metadata replicas (in the Metadata DB) and the digests for the required number of object data replicas -- the system then retrieves just one object data replica file in order to return the object data to the S3 client. For example to meet a read consistency requirement of ALL, the system must be able to read all the object's metadata replicas in the Metadata DB, and all the object's data replicas' file digests in the Checksum DB -- and then it retrieves one object data replica and returns it to the client.

#### 4.1.2.2. Note About Bucket Content List Reads

In the documentation of the supported consistency levels such as "ALL", "QUORUM", and so on, when read consistency requirements are discussed the focus is on reads of individual objects -- that is, the consistency requirements for successfully implementing S3 *GetObject* requests. It's worth noting however that your configured read consistency requirements also apply to bucket content list reads -- that is, when implementing S3 *ListObjects* requests.

Metadata for objects is stored in two different types of record in the Metadata DB: object-level records (with one such record for each object) and bucket-level records that identify the objects in a bucket (along with some metadata for each of those objects). Both types of object metadata are replicated to the same degree. So for example, in a 3X replication storage policy, for each object the object-level metadata record is replicated three times in the cluster and for each bucket the bucket-level object metadata records are replicated three times in the cluster.

A *GetObject* request requires reading the object's object-level metadata record and a *List Objects* request requires reading the bucket's bucket-level object metadata records. Whatever read consistency requirements you set for a storage policy apply not only to reads of individual objects but also to reads of bucket content lists. So for example if you use a QUORUM read consistency requirement, then in order to successfully execute a *List Objects* request the system must be able to read a QUORUM of the bucket-level object metadata records for the bucket.

### 4.1.3. Metadata Replication

*Subjects covered in this section:*

- ***"Object Metadata Replication"** (page 133)*
- ***"System Metadata Replication"** (page 134)*

#### 4.1.3.1. Object Metadata Replication

HyperStore **object metadata** is stored in the Metadata DB, and is protected by replication. The degree to which object metadata is replicated depends on the type of storage policy being used.

### Object Metadata in Replication Storage Policies

In the case of storage policies that protect object data by replication, the corresponding object metadata is replicated to the same degree as the object data. For example with a 3X replication storage policy, the system stores three replicas of each object (with each replica stored on a different node, in the HyperStore file system) and three replicas of each object's metadata (with each replica stored on a different node, in the Metadata DB). In a multi-DC replication storage policy that retains two replicas of each object in DC1 and one replica of each object in DC2, the object metadata will also be replicated in this same manner -- two replicas in DC1 and one replica in DC2.

### Object Metadata in Erasure Coding Storage Policies

In the case of storage policies that protect object data by erasure coding, the object metadata is protected by replication -- not erasure coding. This is because the object metadata associated with a given object is small in size, and therefore not appropriate for erasure coding.

Specifically, for erasure coding storage policies the system will store **2*m*-1 replicas of object metadata**. For example, with a 4+2 erasure coding storage policy, the object metadata is protected by 3X replication.

Additional examples of 2*m*-1 object metadata replication:

- Single-DC, *k*+*m* = 6+2 --> 3 replicas of object metadata
- Single-DC, *k*+*m* = 9+3 --> 5 replicas of object metadata
- Replicated multi-DC, *k*+*m* = 4+2 --> 3 replicas of object metadata in each DC
- Distributed multi-DC, *k*+*m* = 5+4 --> 7 replicas of object metadata distributed across the DCs

### Two Types of Object Metadata Record

Metadata for an object is stored in two different types of record in the Metadata DB: one record specific to the object (called "skinny row" object metadata) and one or more bucket-level records that get updated with metadata for the object (called "wide row" object metadata). The latter are used for listing the contents of a bucket, among other purposes. For more detail see **"Object Metadata Storage in the Metadata DB"** (page 199).

Both types of object metadata are replicated to the same degree. So for example, in a 3X replication storage policy, for each object the "skinny row" object metadata is replicated three times in the cluster and the "wide row" object metadata is replicated three times in the cluster.

The skinny row metadata has the same key format as the object data (*bucketname/objectname*) and so has the same hash token and will be written to the same nodes as the object data -- or a subset of those nodes in the case of an erasure coding storage policy. The wide row metadata has a different key format and hash token and so may be written to different nodes than the object data if the cluster exceeds the minimize size required for the storage policy.

Within the Metadata DB, both types of object metadata record are part of the *UserData_<policyid>* keyspaces.

### 4.1.3.2. System Metadata Replication

HyperStore stores system metadata in several Metadata DB keyspaces: the *AccountInfo* keyspace (for user account information), the *Reports* keyspace (for usage reporting data), and the *Monitoring* keyspace (for system monitoring data). The initial replication level for this system metadata is set by the operator during HyperStore system installation (with a default of 3 replicas per service region). Subsequently, the system automatically adjusts the system metadata replication level in response to your creation of storage policies. The system uses whichever of these criteria yields the **highest** replication level:

- The system metadata replication level set during installation
- Matching the replication factor of any replication storage policies created in the system
- Having *2m-1* replicas in each DC for any regular erasure coding or replicated erasure coding policies created in the system
- Having *2m+1* replicas distributed across DCs for any distributed erasure coding policies created in the system

The table below shows examples of what the automatic system metadata replication level would be in different system configuration scenarios.

| System Configuration | System Metadata Replication Level | Comment |
|---|---|---|
| • Single data center<br>• System metadata replication level configured during install = 2<br>• Just one storage policy created in system: a 3X replication policy | 3 | The highest replication level is yielded by matching the level of the 3X replication storage policy |
| • Single data center<br>• System metadata replication level configured during install = 5<br>• Just one storage policy created in system: a 3X replication policy | 5 | The highest replication level is yielded by using the level configured by the operator during system installation |
| • Single data center<br>• System metadata replication level configured during install = 3 (the default)<br>• Two storage policies created in system: a 3X replication policy and a 9+3 erasure coding policy | 5 | The highest replication level is yielded by using *2m-1* from the 9+3 erasure coding policy |
| • Two data centers in a single service region<br>• System metadata replication level configured during install = 3 (the default; implemented as 1 replica in one DC and 2 in the other)<br>• One storage policy created in system: a replicated 4+2 erasure coding policy | 3 in each DC | The highest replication level is yielded by using *2m-1* per DC from the replicated 4+2 erasure coding policy |
| • Three data centers in a single service region<br>• System metadata replication level configured during install = 3 (the default; implemented as 1 in each DC) | 3 in each DC | The highest replication level is yielded by using *2m+1* (distributed across DCs) from the distributed 5+4 erasure coding |

| System Configuration | System Metadata Replication Level | Comment |
|---|---|---|
| • Two storage policies created in system: a replication policy at 2X per DC; and a 5+4 distributed erasure coding policy | | policy |

The general logic behind this automated adjustment of system metadata replication level is that the **greater the resilience of your configured storage policies** -- in terms of ability to read and write object data and object metadata when a node or nodes are unavailable -- **the greater will be the resilience built into the system metadata storage configuration**.

### Consistency Requirements for System Metadata Reads and Writes

Consistency requirements for object data and object metadata reads and writes are set per storage policy, when you create storage policies. By contrast consistency requirements for reads and writes of system metadata are set at the system level, by these configuration properties in mts.properties.erb:

- **"cloudian.cassandra.default.ConsistencyLevel.Read"** (page 446) (default = LOCAL_QUORUM,ONE)
- **"cloudian.cassandra.default.ConsistencyLevel.Write"** (page 446) (default = QUORUM,LOCAL_QUORUM)

## 4.1.4. Creating and Managing Storage Policies

In the CMC's **Storage Policies** page you can do everything you need to do in regard to creating and managing storage policies:

- Add a storage policy
- Edit a storage policy
- Designate a default storage policy
- Disable a storage policy
- Delete a storage policy

For details, while on the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) click **Help**.

At all times you must have one and only one **default storage policy** defined in each of your HyperStore service regions. The default policy is the one that will be applied when users create new buckets without specifying a policy.

**Note** By default the system allows a maximum of 25 storage policies. This maximum is controlled by the setting **"cloudian.protection.policy.max"** (page 466) in the configuration file mts.properties.erb. Edit this setting if you want to have more than 25 storage policies in your HyperStore system.

### 4.1.4.1. Assigning a Storage Policy to a Bucket

When users create a new bucket they can select a storage policy to apply to the data that they will store in that bucket. This can be done either through the CMC or through other S3 applications that invoke HyperStore extensions to the standard S3 API.

**IMPORTANT !** After a bucket is created, it cannot be assigned a different storage policy. The storage policy assigned to the bucket at bucket creation time will continue to be bucket's storage policy for the life of the bucket.

### Assigning a Storage Policy to a Bucket (CMC)

CMC users can select a storage policy when they create a new bucket in the CMC's **Buckets** page (**Buckets & Objects -> Buckets**). For details, while on the CMC's **Buckets** page click **Help**.

If a user does not explicitly select a policy when creating a new bucket, the system's current default storage policy is automatically applied to the bucket.

### Assigning a Storage Policy to a Bucket (S3 API)

To select a storage policy for a new bucket, S3 client applications use an "x-gmt-policyid" request header when submitting a *PUT Bucket* request. For details see the S3 section in the ***Cloudian HyperStore AWS APIs Support Reference***

#### 4.1.4.2. Retrieving Storage Policy Usage Through the Admin API

Creating or modifying storage policies through the Admin API is not supported. However, you can use the API to retrieve a list of buckets that use each storage policy, with the **GET /bppolicy/bucketsperpolicy** method. For more information see the "bppolicy" section of the ***Cloudian HyperStore Admin API Reference***.

## 4.1.5. Finding an Object's Replicas or EC Fragments

HyperStore lets you quickly determine which nodes are storing the replicas or erasure coded fragments of a specified S3 object. You can do this through either the CMC or the command line.

#### 4.1.5.1. Finding an Object's Replicas or EC Fragments (CMC)

In the CMC you can find an object's replicas or fragments by using the **Object Locator** page (**Analytics -> Object Locator**).

#### 4.1.5.2. Finding an Object's Replicas or EC Fragments (Command Line)

To find an object's replicas or erasure coded fragments locations using *hsstool* on the command line:

- **hsstool whereis**

## 4.1.6. Storage Policy Resilience to Downed Nodes

When nodes are down in your cluster, HyperStore S3 service availability for object writes and reads is a function of several factors including:

- The storage policy applied to the objects -- particularly the data distribution scheme (such as 3X replication or 4+2 erasure coding) and the configured consistency level requirements.
- The number of nodes in the cluster.
- The number of nodes that are down.

The tables that follow below indicate HyperStore S3 write and read availability for common single-DC storage policy configurations, **in scenarios where either one or two nodes are down**. For simplicity the tables refer to nodes as being "down", but the same logic applies if nodes are unavailable for other reasons such as being inaccessible on the network, or in a **stop-write condition**, or in **maintenance mode**.

**Note** The erasure coding tables assume that the system default behavior applies so that for each object with its *k+m* number of data fragments, the number of object metadata replicas created is *2m-1* (for example 3 object metadata replicas per object in the context of a 4+2 erasure coding policy). You may wish to discuss with your Cloudian representative whether the default of *2m-1* is right for your use case, as versus the alternative of creating *2m+1* number of object metadata replicas (which can be achieved by customizing a 'hidden' configuration setting during system set-up). The latter behavior places more storage load on the Metadata DB but also improves read availability in some scenarios.

## Single DC, 3X Replication

With a **3X replication storage policy**, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have 3, 4, or 5 or more nodes in the cluster.

| S3 Operation Type | Configured Consistency Level (CL) | Number of Nodes Down | 3 Nodes in Cluster | 4 Nodes in Cluster | 5 or More Nodes In Cluster |
|---|---|---|---|---|---|
| Writes | ALL | 1 | All writes fail | Writes succeed for some objects and fail for others. | Writes succeed for some objects and fail for others. |
| | | 2 | All writes fail | All writes fail | Writes succeed for some objects and fail for others |
| | QUORUM (default) | 1 | All writes succeed | All writes succeed | All writes succeed |
| | | 2 | All writes fail | Writes succeed for some objects and fail for others | Writes succeed for some objects and fail for others |
| Reads | ALL | 1 | All reads fail | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |
| | | 2 | All reads fail | All reads fail | Reads succeed for some objects and fail for others |
| | QUORUM (default) | 1 | All reads succeed | All reads succeed | All reads succeed |
| | | 2 | All reads fail | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |
| | ONE | 1 or 2 | All reads succeed | All reads succeed | All reads succeed |

## Single DC, *k*+2 Erasure Coding

By default HyperStore supports several *k*+2 erasure coding schemes: 4+2, 6+2, and 8+2. With a storage policy based on *k*+2 erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have *k*+2, *k*+3, or *k*+4 or more nodes in the cluster. For example with a 4+2 policy (*k* = 4), write and read resilience depends in part on whether you have 6, 7, or 8 or more nodes in the cluster; and with a 6+2 policy (*k* = 6), resilience depends in part on whether you have 8, 9, or 10 or more nodes in the cluster.

| S3 Operation Type | Configured Consistency Level (CL) | Number of Nodes Down | *k*+2 Nodes in Cluster | *k*+3 Nodes in Cluster | *k*+4 or More Nodes in Cluster |
|---|---|---|---|---|---|
| Writes | ALL | 1 | All writes fail | Writes succeed for some objects and fail for others | Writes succeed for some objects and fail for others |
| | | 2 | All writes fail | All writes fail | Writes succeed for some objects and fail for others |
| | QUORUM (default) | 1 | All writes succeed | All writes succeed | All writes succeed |
| | | 2 | All writes fail | Writes succeed for some objects and fail for others | Writes succeed for some objects and fail for others |
| Reads | ALL | 1 or 2 | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |
| | QUORUM (default) | 1 | All reads succeed | All reads succeed | All reads succeed |
| | | 2 | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |

## Single DC, *k*+3 Erasure Coding

By default HyperStore supports one *k*+3 erasure coding scheme: 9+3. With a storage policy based on *k*+3 erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have *k*+3, *k*+4, or *k*+5 or more nodes in the cluster. For example with a 9+3 policy (*k* = 9), write and read resilience depends in part on whether you have 12, 13, or 14 or more nodes in the cluster.

| S3 Operation Type | Configured Consistency Level (CL) | Number of Nodes Down | *k*+3 Nodes in Cluster | *k*+4 Nodes in Cluster | *k*+5 or More Nodes in Cluster |
|---|---|---|---|---|---|
| Writes | ALL | 1 | All writes fail | Writes succeed for some objects and | Writes succeed for some objects and |

| S3 Operation Type | Configured Consistency Level (CL) | Number of Nodes Down | *k*+3 Nodes in Cluster | *k*+4 Nodes in Cluster | *k*+5 or More Nodes in Cluster |
|---|---|---|---|---|---|
| | | | | fail for others | fail for others |
| | | 2 | All writes fail | All writes fail | Writes succeed for some objects and fail for others |
| | QUORUM (default) | 1 or 2 | All writes succeed | All writes succeed | All writes succeed |
| Reads | ALL | 1 or 2 | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |
| | QUORUM (default) | 1 or 2 | All reads succeed | All reads succeed | All reads succeed |

## Single DC, *k*+4 Erasure Coding

By default HyperStore supports one *k*+4 erasure coding scheme: 12+4. With a storage policy based on *k*+4 erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have *k*+4, *k*+5, or *k*+6 or more nodes in the cluster. For example with a 12+4 policy (*k* = 12), write and read resilience depends in part on whether you have 16, 17, or 18 or more nodes in the cluster.

| S3 Operation Type | Configured Consistency Level (CL) | Number of Nodes Down | *k*+4 Nodes in Cluster | *k*+5 Nodes in Cluster | *k*+6 or More Nodes in Cluster |
|---|---|---|---|---|---|
| Writes | ALL | 1 | All writes fail | Writes succeed for some objects and fail for others | Writes succeed for some objects and fail for others |
| | | 2 | All writes fail | All writes fail | Writes succeed for some objects and fail for others |
| | QUORUM (default) | 1 or 2 | All writes succeed | All writes succeed | All writes succeed |
| Reads | ALL | 1 or 2 | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others | Reads succeed for some objects and fail for others |
| | QUORUM (default) | 1 or 2 | All reads succeed | All reads succeed | All reads succeed |

## Additional Considerations for S3 Availability

*S3 Availability for Large Objects*

For uploading larger objects, S3 client applications typically use Multipart Upload -- which breaks the object data into contiguous parts and uploads each part separately. Amazon recommends that client applications use Multipart Upload for objects 100MB and larger. Within HyperStore, each part is assigned its own hash value and is separately replicated or erasure coded within the cluster.

Also, for each object larger than 10MB -- or for Multipart Uploads, for each object part larger than 10MB -- the HyperStore system breaks the object or part into multiple "chunks" of 10MB or smaller (for more detail on this configurable feature see System Settings). Each chunk is assigned its own hash value and is separately replicated or erasure coded within the cluster.

For S3 write and read availability for large objects, within HyperStore the write or read of **each part and/or chunk** must satisfy the object data consistency level requirement in order for the S3 object upload or object read operation as a whole to succeed. (The metadata requirements are not impacted by object size since the metadata records are for the object as a whole and not for each part or chunk.)

In terms of the Result categories in the tables above, the consequences of an object being large are as follows:

- "**All writes/reads succeed**" -- No effect. Just as writes or reads of all individual small objects succeed, so too do writes or reads of all large object parts and chunks.
- "**All writes/reads fail**" -- No effect. Large objects fail just as small ones do.
- "**Writes/Reads succeed for some objects and fail for others**." -- Here, the object data consistency criteria that determine which objects succeed (as displayed when you click the Result text in the tables) must be met by **each part and/or chunk** in order for the S3 object upload or object read operation as a whole to succeed. Consequently, in these scenarios, the write or read of a large object with multiple parts and/or chunks has a greater chance of failing than the write or read of a small object.

*Writes of Object Metadata Per Bucket*

For an S3 write operation to succeed, your configured write consistency requirement must be met for the object data and also for the object metadata. The system actually writes two types of object metadata record -- a record specific to the object and a bucket-level record that gets updated with metadata for each object in the bucket (the bucket-level record is used for listing the contents of a bucket, among other things).

For a given object, the per-object metadata record has the same key format as the object data (*bucketname/objectname*) and therefore is assigned the same hash token as the object data and will be written to the same endpoint nodes as the object data -- or a subset of those nodes, in the case of erasure coding storage policies. By contrast, the per-bucket object metadata record has a different key format and therefore a different hash token, and so may be written to different endpoint nodes than the object data. The per-bucket object metadata record is replicated to the same degree as the per-object metadata record -- for example, three times in a 3X replication storage policy -- and must meet the same configured write consistency requirements.

To limit the complexity of the tables above, in the Result descriptions the references to object metadata are referring only to the per-object metadata record. In terms of the Result categories in the tables, the consequences of the system's need to also write per-bucket object metadata -- potentially to different endpoint nodes than the object data and per-object metadata -- are as follows:

- "**All writes succeed**" -- No effect. In these scenarios writes succeed for the per-bucket object metadata also.

- "**All writes fail**" -- No effect. In these scenarios S3 writes fail regardless of the per-bucket object metadata considerations.
- "**Writes succeed for some objects and fail for others**." -- In these scenarios, writes succeed for objects for which the consistency requirement can be met for object data, per-object metadata, **and** per-bucket object metadata. Writes fail for objects for which the consistency requirement cannot be met for either the object data, or the per-object metadata, or the per-bucket metadata. In such down node scenarios where writes succeed for some objects and fail for others, whether the write of a given object succeeds or fails is determined not only by the hash token assigned to the object's data and per-object metadata record but also by the hash token assigned to the per-bucket object metadata record.

**Note** Per-bucket object metadata is not used for S3 object reads and has **no impact on any of the read availability scenarios** in the tables above. Only the per-object metadata record is relevant to object reads.

*Redis DB Access*

The HyperStore system's S3 Service needs information from the Redis Credentials database and the Redis QoS database in order to process S3 write and read requests. In most cases -- particularly in larger clusters -- 1 or 2 nodes being down in your system will not impact the availability of these databases (which are implemented across multiple nodes in master-slave relationships). If problems within your system were to lead to either of these databases being completely offline -- such as if all the nodes running Redis Credentials are down, or all the nodes running Redis QoS are down -- the S3 layer can use cached Redis data for a while. But if the cached data expires and a Redis database is still completely offline, then S3 requests will start to fail.

# 4.2. Security and Privacy Features

## 4.2.1. HTTPS (SSL/TLS)

### 4.2.1.1. HTTPS Feature Overview

By default the following HyperStore services support client access via HTTPS, with each service using its own self-signed TLS/SSL certificate that is automatically generated during HyperStore installation:

- Cloudian Management Console (CMC) Service
- Admin Service
- IAM Service
- S3 Service

**Note** HTTPS for the S3 Service is enabled by default only in fresh installations of HyperStore version 7.4 or newer. For systems that were originally installed as a version older than 7.4, and then subsequently upgraded, HTTPS for the S3 Service is disabled by default (the upgrade does not automatically enable it). If your original HyperStore installation was older than version 7.4 and you have not yet enabled HTTPS for the S3 Service, you can enable it by **generating a self-signed certificate** for the S3 Service or **importing a CA-signed certified** for the S3 Service.

Either of those operations automatically enables HTTPS for the S3 Service if it is not already enabled.

HyperStore provides you tools that simplify several SSL certificate management tasks that you may wish to perform for the CMC, Admin, IAM, and/or S3 services:

- Generate a new self-signed certificate (see **"Generating a New Self-Signed Certificate for a Service"** (page 142))
- Generate a Certificate Signing Request (CSR) to submit to a Certificate Authority (CA) (see **"Generating a Certificate Signing Request (CSR)"** (page 144))
- Import a CA-signed certificate and associated intermediate and root certificates (see **"Importing a CA-Signed Certificate"** (page 147))

You may also wish to block regular HTTP access to these HyperStore services, so that only HTTPS access is allowed. For information on this task see **"Disabling Regular HTTP Access to HyperStore Services"** (page 150).

Because HyperStore services sometimes make outbound HTTPS connections -- when implementing cross-region replication or auto-tiering, for example -- you may also wish to import one or more root Certificate Authorities' certificates to the truststore used by HyperStore services, so that HyperStore services making outbound HTTPS connections can trust certificates signed by those CAs. This is necessary only for private root CAs, since the major public CAs are already in the truststore by default.

Adding a private root CA's certificate to HyperStore's truststore is also necessary if you import a CA-signed certificate to the S3 Service, the Admin Service, or the IAM service, and the root CA in the certificate chain is a private CA. This is because the CMC acts as a client to those services.

For information on adding a root CA's certificate to the truststore, see **"Adding a Private Root CA's Certificate to HyperStore's Truststore"** (page 149).

**Note**
* HyperStore HTTPS listeners support **TLS v1.2 and TLS v1.3** and will not accept client connections that use TLS versions older than 1.2.
* The Admin Service, along with supporting HTTPS, also supports HTTP(S) Basic Authentication. For more information -- including how to find or change the default password for authentication -- see the Introduction section in the ***Cloudian HyperStore Admin API Reference***.
* In the current HyperStore release, the Simple Queue Service (SQS) does not support HTTPS. Only regular HTTP access is supported for SQS.

### 4.2.1.2. Generating a New Self-Signed Certificate for a Service

**Note** This task is supported by the HyperStore installer, and the procedure below describes how to use the installer to perform this task. If you prefer, you can use the HyperStore command line tool *hsctl* to perform this task rather than the installer. Using *hsctl* for this task provides you additional options not available through the installer: the option to generate self-signed certificates for all HTTPS-enabled services in one operation; the option to assign a non-default lifespan to the certificate(s) (the default

lifespan is one year); and for the S3 Service in a multi-region deployment, the option to generate different certificates for each region. However, if you use *hstcl* to generate a self-signed certificate, afterwards you still must use the installer to push the change out to the cluster and restart the affected service (Steps 5 and 6 in the procedure below). For more information about using *hstcl* to generate self-signed certificates, log into the Configuration Master node and then on the terminal command line enter *hsctl cert self-signed --help*

Follow the steps below to use the HyperStore installer to generate a new self-signed certificate for an HTTPS-enabled HyperStore service. This operation will **overwrite** the certificate that the service is currently using.

1. Log into the Configuration Master node and change into */opt/cloudian-staging/7.5.2* (the installation staging directory). Then launch the installer:

   ```
   # ./cloudianInstall.sh
   ```

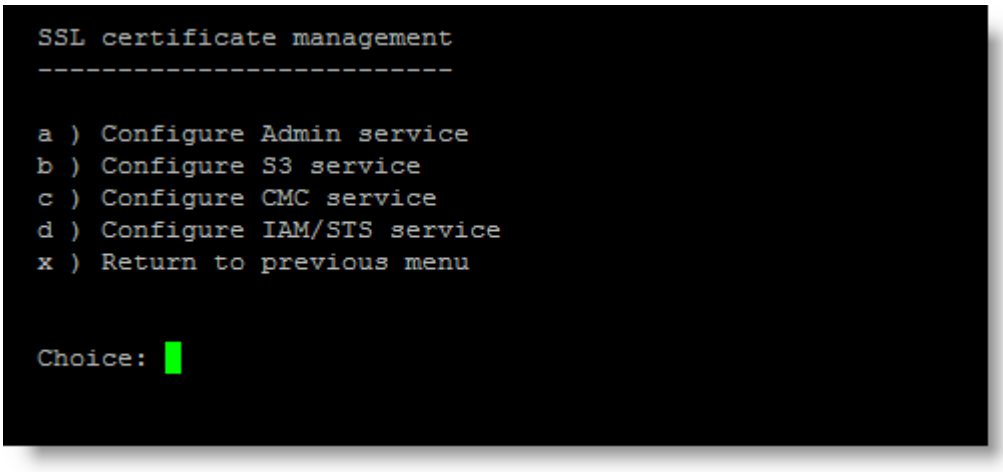
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

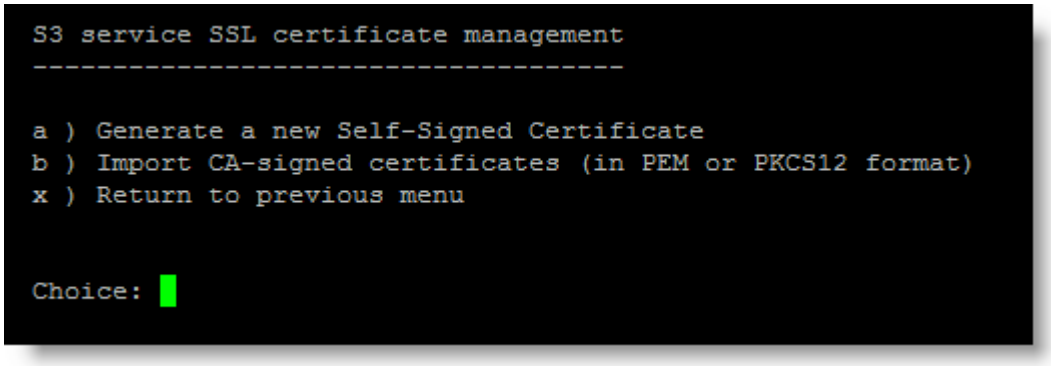
   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. At the installer's main menu enter **4** for "Advanced Configuration Options". Then in the Advanced Configuration Options menu enter **e** for "Configure SSL for Admin, CMC, S3, and IAM/STS services". This takes you to the "SSL Certificate Management" menu:

3. From the SSL Certificate Management menu select the service for which you want to generate a new self-signed certificate. This takes you to the "<Service Name> SSL Certificate Management" menu. The example below is for the S3 Service, but regardless of which service you are working with you will see the same menu options.




4. Enter **a** for "Generate a new Self-Signed Certificate". Then at the prompt, confirm that you want to generate a new self-signed certificate for this service. The installer will then generate the certificate.
5. Navigate back to the installer's main menu, and then enter **2** for "Cluster Management". Then from the Cluster Management menu enter **b** for "Push Configuration Settings to Cluster", and follow the prompts to push to the cluster.
6. Navigate back to the Cluster Management menu, then enter **c** for "Manage Services". Then from the "Service Management" menu, restart the service for which you generated a new self-signed certificate.

### 4.2.1.3. Generating a Certificate Signing Request (CSR)

**Note** This task is not supported by the HyperStore installer. You must use the HyperStore command line tool *hsctl* to perform this task. The *hsctl* tool can be run by the *root* user or by a **HyperStore Shell** user who has the Trusted role.

To generate a certificate signing request (CSR) for one or more HyperStore services: Log into the Configuration Master node, change into a working directory, and run the *hsctl* command described below. This will generate one or more CSR / private key pairs in the working directory in which you run the command.

```
hsctl cert req [--combined] [--service=enum] [--domains=string] [--country="string"]
[--state="string"] [--locality="string"] [--organization="string"] [--organization-unit="string"]
[--email="string"]
```

By default the *hsctl cert req* command:

- Generates a separate CSR and corresponding private key for each of the HTTPS-enabled HyperStore services -- the S3 service, the CMC service, the Admin service, and the IAM service. If you have multiple service regions in your system, then for the S3 service a separate CSR (and private key) will be generated for each service region.
- Uses the service endpoints configured in the HyperStore system, including a wildcarded endpoint for the S3 service in each region.
- Uses Cloudian and generic organization identifiers rather than identifiers for your organization.

You can modify this behavior by using any of the command options described below:

*--combined*

By default, separate CSRs will be generated for each of the HTTPS-enabled HyperStore services. If you prefer to generate a **single combined CSR** for all of the HTTPS-enabled services, use the *--combined* option (and omit the *--service* option described below).

**Note** This is a valid option only if the CA that you are using supports the use of the subject alternative name (SAN) field. A combined CSR for HyperStore services will list multiple service endpoints in the SAN field.

*--service*

Use the *--service* option if you want to generate a CSR for just one HyperStore service. Supported values are:

- s3
- cmc
- admin
- iam

**Note** The *--service* value must be lower case.

If you do not use the *--service* option, a CSR will be generated for each of the services listed above. (Or, if you omit the *--service* option and use the *--combined* option, a single combined CSR will be generated for all of the services listed above.)

*--domains*

By default, CSRs for HyperStore services will be created using the service domains that are configured in the system. Use the *--domains* option if you want to explicitly specify the domain(s) to use in the CSR for a service.

You can check to see the default domains that will be used in CSRs by running the command *hsctl cert list-domains*. For example:

```
[root]# hsctl cert list-domains
cmc:
cmc.mycloudianhyperstore.com

iam:
iam.mycloudianhyperstore.com

admin:
s3-admin.mycloudianhyperstore.com

s3-region1:
s3-region1.mycloudianhyperstore.com
*.s3-region1.mycloudianhyperstore.com
```

Note that by default, the CSR for the S3 service will place all endpoints in the Subject Alternative Name (SAN) field and will include wildcarded versions of the explicit endpoints. In the example above, the wildcarded endpoint is **.s3-region1.mycloudianhyperstore.com*. The wildcarded S3 endpoint is needed if your S3 service is going to support Virtual Host Style Access by S3 clients -- an access method for which the bucket name is used as a sub-domain of the S3 endpoint. This is the S3 access method that AWS recommends.

The most likely use case for the *--domains* option is if you intend to submit the S3 service CSR to a CA that does not support SAN and/or does not support wildcards in endpoints. In that case you could generate a CSR for the S3 service like this, for example:

```
hsctl cert req --service=s3 --domains=s3-region1.mycloudianhyperstore.com
```

This command would generate an S3 service CSR in which there is only one endpoint -- *s3-region1.-mycloudianhyperstore.com* -- and no wildcarding.

Note that the domain that you specify **must** match one of the domains configured for the service in the system (one of the domains returned by *hsctl cert list-domains*).

**IMPORTANT !** If the certificate for your S3 service does not include a wildcarded endpoint, then S3 client applications using your service will have to use Path-Based Access (in which the bucket name is part of the URI path) rather than Virtual Host Style Access.

**Note** If you wish you can use the *--domains* option to specify multiple domains with comma separation (*--domains=<domain1>,<domain2>,...*). You might use this approach for example if your CA supports SAN but not wildcarding. If you list multiple domains, the first-listed domain will be placed in the Common Name (CN) field of the CSR and any additional domains will be placed in the SAN field.

*--country, --state, etc*

Use these options to identify your organization.

- Quote-enclose each value
- Because the values are quote-enclosed, you can use spaces within the values. For example, *--locality="San Francisco"*.
- For *--country* use a 2-character ISO format country code.
- If you omit any of these identifiers, default values will be used as follows: *--country="US" --state="California" --locality="San Mateo" --organization="Cloudian, Inc" --organization-unit="Self-Signed Certificates Unit" --email="example@example.com"*. For example if you specify most of the identifiers but omit the *--organization-unit*, then within the CSR that's generated the organization unit will be indicated as "Self-Signed Certificates Unit".

Example (CSRs for each service):

```
[root]# hsctl cert req --country="US" --state="MA" --locality="Foxborough"
--organization="Patriots" --email="bill@gillettestadium.com"
Generating certificate signing request (CSR) and private key for 'cmc'
Writing CSR to file: './cmc.csr.pem'
Writing key to file: './cmc.key.pem'
Generating certificate signing request (CSR) and private key for 'iam'
Writing CSR to file: './iam.csr.pem'
Writing key to file: './iam.key.pem'
Generating certificate signing request (CSR) and private key for 'admin'
Writing CSR to file: './admin.csr.pem'
Writing key to file: './admin.key.pem'
Generating certificate signing request (CSR) and private key for 's3-region1'
Writing CSR to file: './s3-region1.csr.pem'
Writing key to file: './s3-region1.key.pem'
```

Example (combined CSR for all services):

```
[root]# hsctl cert req --combined --country="US" --state="MA" --locality="Foxborough"
--organization="Patriots" --email="bill@gillettestadium.com"
Generating certificate signing request (CSR) and private key for 'combined'
```

```
Writing CSR to file: './combined.csr.pem'
Writing key to file: './combined.key.pem'
```

Example (CSR for one specified service):

```
[root]# hsctl cert req --service=cmc --country="US" --state="MA" --locality="Foxborough"
--organization="Patriots" --email="bill@gillettestadium.com"
Generating certificate signing request (CSR) and private key for 'cmc'
Writing CSR to file: './cmc.csr.pem'
Writing key to file: './cmc.key.pem'
```

### 4.2.1.4. Importing a CA-Signed Certificate

**Note** This task is supported by the HyperStore installer, and the procedure below describes how to use the installer to perform this task. If you prefer, you can use the HyperStore command line tool *hsctl* to perform this task rather than the installer. Using *hsctl* for this task provides you additional options not available through the installer: the option to import CA-signed certificates for all HTTPS-enabled services in one operation (or a wildcard certificate to be applied to all services); and for the S3 Service in a multi-region deployment, the option to import different CA-signed certificates for each region. However, if you use *hstcl* to import a CA-signed certificate, afterwards you still must use the installer to push the change out to the cluster and restart the affected service (Steps 6 and 7 in the procedure below). For more information about using *hstcl* to import CA-signed certificates, log into the Configuration Master node and then on the terminal command line enter *hsctl cert import --help*

Follow the steps below to use the HyperStore installer to import a certificate signed by a public or private CA, for use by an HTTPS-enabled HyperStore service.

1. Log into the Configuration Master node, and in either */opt/cloudian-staging/7.5.2* (the installation staging directory) or a working directory, place the file(s) from which you are going to import a CA-signed certificate:
   - The file(s) must be in PEM format or PKCS12 format (with filename extension *.p12* or *.pfx*). No other formats are supported.
   - The entire trust chain must be present in the file(s): the final CA-signed certificate, any intermediate certificates, and the CA root certificate. These may all be present in a single file, or may be divided into multiple files.
   - The private key from which the CSR was generated must be present, either as a dedicated key file (such as a *.key.pem* file) or within one of the other files.
   - If you are using the **HyperStore Shell (HSH)**, you will not be able to save files into */opt/cloudian-staging/7.5.2*. Instead, save the files into a working directory under your home directory.
2. Still logged into the Configuration Master node, change into */opt/cloudian-staging/7.5.2*. Then launch the installer:

   ```
   # ./cloudianInstall.sh
   ```

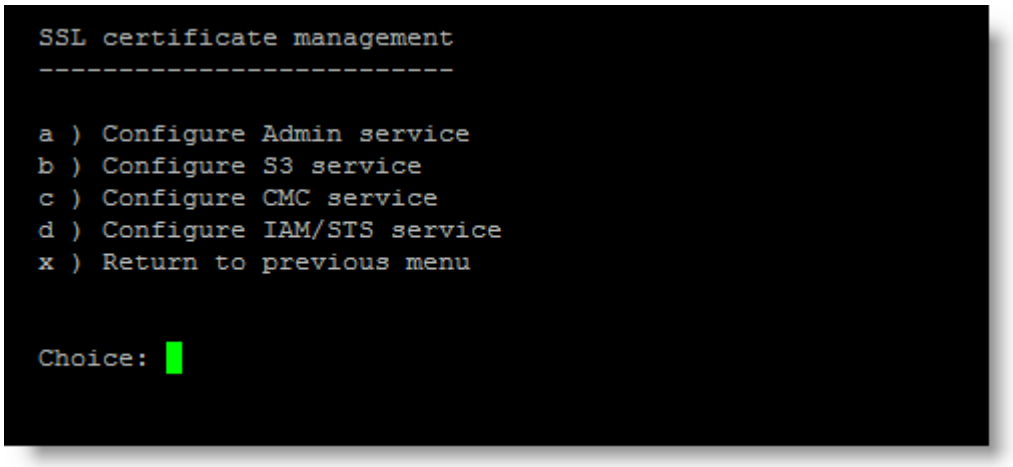
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

3. At the installer's main menu enter **4** for "Advanced Configuration Options". Then in the Advanced Configuration Options menu enter **e** for "Configure SSL for Admin, CMC, S3, and IAM/STS services". This takes you to the "SSL Certificate Management" menu:




4. From the SSL Certificate Management menu select the service for which you want to import a CA-signed certificate. This takes you to the "<Service Name> SSL Certificate Management" menu. The example below is for the S3 Service, but regardless of which service you are working with you will see the same menu options.


5. Enter **b** for "Import CA-Signed Certificates (in PEM or PKSC12 format)". You will then be prompted to enter filenames:
   - If the file(s) are in any directory other than */opt/cloudian-staging/7.5.2*, include the full path to the file(s).
   - At each successive "Enter filename" prompt, type one filename (including path if applicable) then press Enter.
   - When there are no more files to enter, at the "Enter filename" prompt simply press Enter without typing a filename. This will trigger the import of the CA-signed certificate file(s).
   - If any of the files that you specified are password-protected you will be prompted to supply the password.
6. After the import operation completes, navigate back to the installer's main menu, and then enter **2** for "Cluster Management". Then from the Cluster Management menu enter **b** for "Push Configuration

Settings to Cluster", and follow the prompts to push to the cluster.

7. Navigate back to the Cluster Management menu, then enter **c** for "Manage Services". Then from the "Service Management" menu, restart the service for which you imported a CA-signed certificate (for example, restart the S3 Service if you imported a CA-signed certificate for the S3 Service to use.)

### 4.2.1.5. Adding a Private Root CA's Certificate to HyperStore's Truststore

**Note** This task is not supported by the HyperStore installer. You must use the HyperStore command line tool *hsctl* to perform this task. The *hsctl* tool can be run by the *root* user or by a **HyperStore Shell** user who has the Trusted role.

In some contexts HyperStore services may act as clients to other services that present SSL certificates. Examples include, potentially:

- When the S3 Service implements cross-region replication, or auto-tiering.
- When the S3 Service connects to a KMIP-compliant Key Management System (KMS)
- When the CMC connects to an LDAP server (to authenticate users logging into the CMC, if you have enabled LDAP authentication for one or more user groups)
- When the CMC connects to the Admin Service
- When the CMC connects to the S3 Service (applicable only if you've set *common.csv:* **"cmc_storageuri_ssl_enabled"** (page 422) to *true* so that the CMC connects to the S3 Service's HTTPS port; it defaults to *false* so that the HTTP port is used)
- When the CMC connects to the IAM Service (applicable only if you've set *common.csv:* **"iam_secure"** (page 407) to *true* so that the CMC connects to the IAM Service's HTTPS port; it defaults to *false* so that the HTTP port is used)

By default HyperStore's truststore includes all the major public root CAs. If HyperStore services are connecting to services that are using a private CA as their root CA, you can add that CA's certificate to HyperStore's truststore so that HyperStore services trust certificates signed by that CA.

To add a private root CA's certificate to HyperStore's truststore:

1. Log into the Configuration Master node, and copy the private root CA's certificate file into a working directory.
2. Change into the working directory and run this *hsctl* command:

   ```
   # hsctl cert ca trust add <CA certificate filename>
   ```

   This operation will import any PEM formatted private CA root certificates found inside the input file.

3. Run this command to apply the change to the cluster:

   ```
   # hsctl config apply pki.trust
   ```

4. Change into */opt/cloudian-staging/7.5.2* (the installation staging directory) and launch the installer:

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

Once launched, the installer's menu options (such as referenced in the step below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

5. In the installer main menu, enter **2** for "Cluster Management". Then from the Cluster Management menu enter **c** for "Manage Services", and restart the services that you want to trust the private root CA that you added. Typically this would be the S3 Service (such as in the case of cross-region replication) and/or the CMC Service (such as if the CMC is accessing an LDAP server when authenticating users).

#### 4.2.1.6. Disabling Regular HTTP Access to HyperStore Services

For security purposes you may wish to disable regular (non-secure) HTTP access to HyperStore public services so that access to the services is exclusively through HTTPS. For best protection you can block regular HTTP access in **both** of these ways:

- Configure a **firewall** to block S3 HTTP access (port 80), CMC HTTP access (port 8888), IAM HTTP access (port 16080), and Admin Service HTTP access (port 18081). If you are using the HyperStore firewall, see **"Customizing the HyperStore Firewall"** (page 494) for instructions. (If you have not yet enabled the HyperStore firewall, and want to do so, see **"Enabling or Disabling the HyperStore Firewall"** (page 492)).
- Set the **HyperStore system configuration** so that the regular HTTP listeners are disabled for the Admin Service, for the IAM Service, and for the CMC. On the Configuration Master node, in the configuration file **common.csv**, there are settings that enable or disable the regular HTTP listeners for the Admin Service, for the IAM Service, and for the CMC:
  - For the Admin Service, set *admin_secure* to *true* if it is not already set to *true* (it defaults to *true* if your original HyperStore install was version 6.0.2 or newer, and defaults to *false* if your original HyperStore install was older than 6.0.2)
  - For the IAM Service, set *iam_secure* to *true* (it defaults to *false*)
  - For the CMC, set *cmc_web_secure* to *true* if it is not already set to *true* (it defaults to *true*)

  > **Note** The S3 Service does not have a system configuration setting for disabling the regular HTTP listener. Use a firewall to block regular HTTP access to the S3 Service, as stated above.

  After making any edits to *common.csv*, launch the installer and use the **b** "Cluster Management" menu to first push the configuration changes out to the cluster and then go to the "Manage Services" submenu to restart each of the services for which you changed the configuration. (To restart the Admin Service, restart the S3 Service -- this has the effect of restarting the Admin Service also.) Then exit the installer.

## 4.2.2. Server-Side Encryption

### 4.2.2.1. Server-Side Encryption Feature Overview

*Subjects covered in this section:*

- ***"Server-Side Encryption and Auto-Tiering"*** *(page 152)*
- ***"Server-Side Encryption and Cross-Region Replication"*** *(page 153)*

The HyperStore system supports server-side encryption (SSE) of data at rest. Several different methods of server-side encryption are supported:

- Encryption using **HyperStore system-generated encryption keys** (regular SSE)
- Encryption using encryption keys managed in part by a **Key Management Interoperability Protocol compliant Key Management System** (SSE-KMIP) (popular KMIP compliant systems include Fortanix, HashiCorp, and Thales)
- Encryption using encryption keys managed by the **Amazon Web Services Key Management Service** (AWS KMS)
- Encryption using **customer-provided encryption keys** (SSE-C)

The selection of whether to use server-side encryption, and which method to use, can be made at the **storage policy level** (so that a default encryption method is applied to all objects uploaded to any bucket that uses the storage policy), at the **bucket level** (so that a default encryption method is applied to all objects uploaded to the bucket), or at the **object level** (as specified by headers in the object upload request). However not all encryption types can be configured at every level: for example, SSE-KMIP can only be configured at the storage policy level and SSE-C can only be configured at the object level.

When the system is processing an object upload, the precedence ordering among these different configuration levels is as follows:

1. If a server-side encryption method is specified in the object upload request, the system uses that method. If not, then...
2. If a default server-side encryption method is specified in the configuration of the bucket to which the object is being uploaded, the system uses that method. If not, then...
3. If a default server-side encryption method is specified in the configuration of the storage policy used by the bucket to which the object is being uploaded, the system uses that method.

If no encryption method is specified in the object upload request, the configuration of the bucket into which the object is being uploaded, or the configuration of the storage policy used by the bucket, then no server-side encryption is applied to that object.

Note that:

- Encryption is applied at the HyperStore "coordinator node" (the node that happens to receive the incoming S3 PUT request) **before the object data is transmitted to the HyperStore endpoint nodes** where the object data will be stored.
- There is one type of deviation from the precedence ordering between the object, bucket, and storage policy levels described above: If regular SSE is specified in an object upload request or in a bucket configuration, then what is applied to that object or that bucket configuration depends on the parent storage policy. If the storage policy configuration is for regular SSE or if the storage policy has no SSE configuration, then regular SSE is used for the object or bucket. If the storage policy configuration is for SSE-KMIP, then SSE-KMIP is used for the object or bucket.

For more information about using the supported server-side encryption methods, see:

- **"Using Regular SSE"** (page 153)
- **"Using SSE-KMIP"** (page 155)
- **"Using AWS KMS"** (page 160)
- **"Using SSE-C"** (page 163)

### Server-Side Encryption Configuration Changes Do Not Apply Retroactively

Server-side encryption does not apply retroactively to objects that have already been uploaded to the system. For example, if you modify a bucket's configuration so that it includes server-side encryption, this will apply only to objects uploaded from that time forward -- not to objects that had been uploaded to the bucket previously. The same is true of adding server-side encryption to a storage policy's configuration: from that time forward, objects that get uploaded into buckets that use that storage policy will be encrypted, but objects that had already been uploaded previously will not be encrypted.

Conversely, if a bucket configuration or storage policy configuration uses server-side encryption and then you subsequently disable encryption for that bucket or storage policy, then from that time forward newly uploaded objects will not be encrypted -- but objects that had already been uploaded and encrypted prior to the configuration change will remain encrypted.

### Server-Side Encryption and Auto-Tiering

How the HyperStore system handles auto-tiering of server-side encrypted objects depends on the encryption method used and the tiering destination.

| Object's Encryption in Source HyperStore | Tiering to Amazon, Google, HyperStore, or Other Fully S3-Compliant Destinations | Tiering to Azure or Spectra BlackPearl |
|---|---|---|
| Regular SSE or SSE-KMIP | HyperStore decrypts the object, then tiers it to the destination system and includes an *x-amz-server-side-encryption: AES256* header in the upload request so that server-side encryption is implemented at the destination system. If the destination is a remote HyperStore system, the encryption applied there will be either regular SSE or SSE-KMIP depending on how the destination bucket's storage policy is configured. | HyperStore decrypts the object, then tiers it to the destination system in decrypted form. These destinations do not support a server-side encryption request header in object uploads. Azure always applies its own form of server-side encryption and Spectra does not support server-side encryption. |
| AWS KMS | HyperStore decrypts the object, then tiers it to the destination system and includes an *x-amz-server-side-encryption: aws:kms* header in the upload request so that AWS KMS based server-side encryption is implemented at the destination system. If the destination is a remote HyperStore system, AWS KMS based encryption will work only if the system is configured for AWS KMS integration. | |
| SSE-C | HyperStore tiers the encrypted object to the destination system, where it | |

| Object's Encryption in Source HyperStore | Tiering to Amazon, Google, HyperStore, or Other Fully S3-Compliant Destinations | Tiering to Azure or Spectra BlackPearl |
|---|---|---|
| | remains encrypted. To retrieve such a tiered object, client applications must first Restore the object into HyperStore, then GET the object (and include the SSE-C headers in the GET request). Streaming GETs of such tiered objects directly from the destination are not supported. | |

### Server-Side Encryption and Cross-Region Replication

When cross-region replication is configured for a source bucket:

- Objects encrypted by the regular SSE method **are** replicated to the destination bucket.
- Objects encrypted by the SSE-KMIP, AWS KMS, or SSE-C methods **are not** replicated to the destination bucket.

## 4.2.2.2. Using Regular SSE

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Preparing the Regular SSE Feature"** (page 153)*
- ***"Configuring Regular SSE as the Default for a Storage Policy (CMC Only)"** (page 154)*
- ***"Configuring Regular SSE as the Default for a Bucket (S3 API Only)"** (page 154)*
- ***"Requesting Regular SSE for Specific Objects (CMC or S3 API)"** (page 154)*

With **regular server-side encryption (regular SSE)**, HyperStore:

- Maintains a master encryption key, obfuscated in the source code
- Generates a unique per-object encryption key for each new object and uses the per-object key to encrypt the object
- Uses the master key to encrypt each per-object key and stores the encrypted per-object key within the object's metadata

Regular server-side encryption (SSE) can be configured as the default encryption type for a storage policy, as the default encryption type for a particular bucket, or as the encryption type for particular objects. For information about the precedence ordering among these configuration levels see **"Server-Side Encryption"** (page 150).

### Preparing the Regular SSE Feature

With regular SSE, the HyperStore system generates the per-object encryption keys using AES-128 by default. While AES-128 will work for regular SSE, and may be acceptable in a testing or evaluation environment, for greater security Cloudian recommends using AES-256. You can enable AES-256 in your HyperStore system as described in **"Enabling AES-256"** (page 164).

> **Note** Amazon uses AES-256 for its regular SSE implementation, and AES-256 is called for in Amazon's SSE specification.

### Configuring Regular SSE as the Default for a Storage Policy (CMC Only)

When you use the CMC to create or edit storage policies, one of the configurable policy attributes is server-side encryption. This sets a default server-side encryption type for all objects uploaded to all buckets that use the storage policy. To configure a storage policy to use regular SSE by default, in the "Server-Side Encryption" field of the storage policy configuration interface select "SSE".

For more information on storage policy creation and configuration, while on the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) click **Help**.

### Configuring Regular SSE as the Default for a Bucket (S3 API Only)

HyperStore's S3 API supports setting a default server-side encryption type for a bucket. This sets a default server-side encryption type for all objects uploaded to the bucket. The S3 API operations that set and manage a bucket default server-side encryption method are:

- PutBucketEncryption
- GetBucketEncryption
- DeleteBucketEncryption

To configure a bucket to apply regular SSE to uploaded objects by default:

- In the *PutBucketEncryption* request body set the *SSEAlgorithm* element to *AES256*.
- In the bucket's storage policy the server-side encryption configuration must be set to either "None" or "SSE". If in the *PutBucketEncryption* request the *SSEAlgorithm* element is set to *AES256* and the bucket's storage policy is configured to use "KMIP" for server-side encryption, then **SSE-KIMP** will be used as the bucket's default server-side encryption method rather than regular SSE.

For details of HyperStore's support for these S3 operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

Note that:

- If you have not **enabled AES-256** in your HyperStore system, the system will use AES-128 to generate per-object encryption keys even though the *SSEAlgorithm* element specifies *AES256*.
- The CMC does not support setting a bucket default server-side encryption method.
- If a bucket's storage policy is configured to use regular SSE by default, there is no need to explicitly configure regular SSE as the bucket default encryption method.

### Requesting Regular SSE for Specific Objects (CMC or S3 API)

Regular SSE can be requested for specific objects as those objects are uploaded to a bucket in the HyperStore system. This can be done either through the CMC or through a third party S3 client application that calls the HyperStore S3 API.

Note that:

- If regular SSE is configured as the default server-side encryption type for the bucket into which an object is being uploaded, or as the default server-side encryption type for the storage policy used by that bucket, there is no need to explicitly request regular SSE as the encryption type for that object.
- For the methods described below to result in an object being encrypted with regular SSE, the bucket's storage policy must be configured to use a default server-side encryption type of either "None" or "SSE". If either of the per-object methods described below are used and the bucket's storage policy is configured to use "KMIP" for server-side encryption, then **SSE-KIMP** will be used as the object's server-side encryption method rather than regular SSE.

#### Object-Level SSE Through the CMC

In the CMC's **Objects** page, where a user can upload objects into the HyperStore system, the interface displays a "Store Encrypted" checkbox. If the user selects this checkbox, the HyperStore system applies regular server-side encryption to the uploaded object(s).

#### Object-Level SSE Through the S3 API

HyperStore's S3 API supports regular server-side encryption by the inclusion of the *x-amz-server-side-encryption: AES256* request header in any of these operations on objects:

- PutObject
- CreateMultipartUpload
- UploadPart
- POST Object
- CopyObject

For details of HyperStore's support for these S3 operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

> **Note** If you have not **enabled AES-256** in your HyperStore system, the system will use AES-128 encryption even though the *x-amz-server-side-encryption* request header specifies *AES256*.

### 4.2.2.3. Using SSE-KMIP

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Preparing the SSE-KMIP Feature"*** *(page 156)*
- ***"Configuring SSE-KMIP as the Default for a Storage Policy (CMC Only)"*** *(page 159)*
- ***"Re-Keying a Bucket"*** *(page 159)*
- ***"Bucket Deletion"*** *(page 160)*

With **SSE-KMIP** -- server-side encryption utilizing a Key Management Interoperability Protocol (KMIP) compliant Key Management System (KMS) -- HyperStore triggers in the KMIP KMS the creation of one "master key" for each bucket that uses SSE-KMIP. Each bucket's master key is stored exclusively on the KMIP KMS, while in HyperStore a corresponding master key ID is stored in the bucket's metadata. When an object is uploaded to a bucket that uses SSE-KMIP, HyperStore generates a unique plain text data key and uses that data key to encrypt the object data. HyperStore then submits the plain text data key and the bucket's master key ID to the KMIP KMS, which uses the bucket master key to encrypt the data key and returns the encrypted data key back to HyperStore. HyperStore stores the encrypted data key in the object's metadata and discards the plain text data key.

Subsequently when a client downloads the object, HyperStore submits the encrypted data key and the bucket's master key ID to the KMIP KMS, which returns back the decrypted data key. HyperStore uses this plain text data key to decrypt the object and then deletes the plain text data key from memory.

SSE-KMIP can be configured at the storage policy level. It cannot be configured at the per-bucket level or the per-object level. For information about the precedence ordering among these levels see **"Server-Side Encryption"** (page 150).

Note that:

- In communicating with a KMIP KMS, HyperStore supports **KMIP version 2.1 and earlier**.
- The process described above for encrypting and decrypting per-object data keys is used when the KMS supports the KMIP *encrypt* and *decrypt* commands, as is typically the case. In the atypical case when the KMS does not support these commands (an example is older versions of HashiCorp Vault -- versions 1.10 and earlier), HyperStore will retrieve the bucket's master key from the KMS each time an object is uploaded or downloaded. HyperStore will use the bucket master key to perform the encryption or decryption of the object's data key and then delete the bucket master key from memory.
- When a new bucket is created that uses a storage policy that is configured for SSE-KMIP, HyperStore automatically has the KMIP KMS create a master key for the new bucket. However, **if you plan to add SSE-KMIP to an existing storage policy's configuration** and there are already existing buckets that use that storage policy, then **before** you enable SSE-KMIP on the storage policy you must manually initiate a HyperStore operation that will direct the KMIP KMS to create master keys for those buckets. Details are in the **"Preparing the SSE-KMIP Feature"** (page 156) section below.
- Although for simplicity the terms "encrypt" and "decrypt" are used throughout the SSE-KMIP introduction above, technically what is done in regard to the per-object data keys is NIST-compliant "wrapping" and "unwrapping".

## Preparing the SSE-KMIP Feature

This section describes the following SSE-KMIP preparation topics:



### Defining a Command-Restricted Role for HyperStore on the KMIP KMS

This is optional but recommended. As a security best practice, the KMIP KMS administrator should define a role for HyperStore that grants HyperStore permission to perform only the following KMIP operations (these are the only KMIP operations that HyperStore requires):

- DiscoverVersions
- Query
- Create
- Activate
- Get
- GetAttributes
- Locate
- Revoke
- Rekey
- Encrypt
- Decrypt

### Configuring HyperStore to Connect to the KMIP KMS

Before configuring HyperStore to connect to the KMIP KMS, note that:

- The use of TLS/SSL is required for connections between HyperStore and any KMIP KMS, and so TLS/SSL setup is part of the configuration procedure below. **Before starting the procedure below you must convert your KMIP KMS TLS/SSL client key(s)/certificate(s) to a PKCS12 formatted keystore file** to use in the procedure. (You can use OpenSSL or another TLS/SSL toolkit to perform the conversion. Each of your HyperStore nodes has OpenSSL on it, at */usr/bin/openssl*. For more information see for example **https://www.openssl.org/docs/man3.1/man1/openssl-pkcs12.html**.)
- HyperStore currently only supports connecting to **one KMIP KMS**. You cannot configure the system to connect to more than one KMIP KMS. If you have a multi-region HyperStore system, each of your service regions will communicate with this one KMIP KMS.

After converting your KMIP KMS TLS/SSL client key(s)/certificate(s) to a PKCS12 formatted keystore file, follow the steps below to configure HyperStore to connect to the KMIP KMS:

1. Copy your PKSC12-formatted KMIP KMS client keystore file into a working directory on the Configuration Master node (such as */tmp*).
2. On the Configuration Master node, import your KMIP KMS client keystore file by running this command:

   ```
   hsctl cert kmip import <filename-including-path>
   ```

3. Still on the Configuration Master node, open the **common.csv** file and edit these settings for the connection to the target KMIP KMS:

   - *kmip_host* (IP address, FQDN, or host name)
   - *kmip_port* (defaults to 5696; edit only if your KMS uses a different KMIP listening port)
   - *kmip_username* (optional, user name for connecting to the KMS)
   - *kmip_password* (optional, password for connecting to the KMS)
   - *kmip_keystore_password* (for the TLS/SSL keystore that you uploaded in Step 2).

   After making your edits save your changes and close the file.

   *If you are using the **HyperStore Shell***

   If you are using the HyperStore Shell (HSH) as a Trusted user, you can edit *common.csv* with this command:

   ```
   $ hspkg config -e common.csv
   ```

4. In the installation staging directory (*/opt/cloudian-staging/7.5.2*) launch the installer:

   ```
   ./cloudianInstall.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the HyperStore Shell (HSH) as a Trusted user, you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

5. Enter **2** for Cluster Management then **a** for Review Cluster Configuration, then enter **yes** when asked if you want to update the Puppet server with your changes.
6. Push your changes out to the cluster and restart the S3 Service. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

Next -- as the final step for setting up secure communications between HyperStore and the KMIP KMS -- **add the KMS server's CA certificate to HyperStore's truststore** by following the instructions in **"Adding a**

**Private Root CA's Certificate to HyperStore's Truststore"** (page 149). (Note that this step is necessary only if the KMS uses a private CA as its root CA. HyperStore's truststore includes all the major public root CAs by default.)

### Enabling AES-256 for Object Data Encryption

With SSE-KMIP, the HyperStore system by default uses AES-128 when encrypting object data. While AES-128 may be acceptable in a testing or evaluation environment, for greater security Cloudian recommends using AES-256. You can enable AES-256 in your HyperStore system as described in **"Enabling AES-256"** (page 164).

> **Note** When encrypting the **object data keys** supplied by HyperStore, the KMIP KMS will use AES-256 regardless of your HyperStore configuration.

### Generating Master Keys for Existing Buckets That Use a Storage Policy for Which You Plan to Enable SSE-KMIP

This task does not apply if you intend to only enable SSE-KMIP on new storage policies or on storage policies that are not yet being used by any buckets. In that case you can skip ahead to **"Configuring SSE-KMIP as the Default for a Storage Policy (CMC Only)"** (page 159).

If you plan to enable SSE-KMIP for **an existing storage policy that is already being used by existing buckets**, then **before** enabling SSE-KMIP for that storage policy you must do the following to trigger the creation of a bucket master key (on the KMIP KMS) for each of the existing buckets.

> **IMPORTANT !**
> * For this procedure to work you must have already completed the steps in **"Configuring HyperStore to Connect to the KMIP KMS"** (page 156) above.
> * If you enable SSE-KMIP on a storage policy used by an existing bucket without first triggering the creation of a master key for that bucket, new uploads of objects to that bucket will fail until you trigger the creation of a master key for that bucket.

Log into any HyperStore node. (If you have a multi-region HyperStore system, log into any node in the service region in which the storage policy is used.) Then, there are two options for how to create bucket master keys on the KMS for the buckets that use the storage policy for which you plan to enable SSE-KMIP.

To create a bucket master key for a specific bucket or for multiple buckets one by one, run the following command for each bucket:

```
curl -d "" 'http://localhost/.system/rekey?bucketName=<bucketname>'
```

For example:

```
curl -d "" 'http://localhost/.system/rekey?bucketName=bucket1'
```

To create a bucket master key for all buckets that use the storage policy for which you plan to enable SSE-KMIP, run the following command:

```
curl -d "" 'http://localhost/.system/rekey?storagePolicy=<storagePolicyId>'
```

where *<storagePolicyId>* is the system-generated unique ID for the storage policy (which you can get from the CMC's **Cluster -> Storage Policies** page or from the Admin API call *GET /bppolicy/listpolicy*).

For example:

```
curl -d "" 'http://localhost/.system/rekey?storagePolicy=958ee755f38e48839907b1576bd444e1'
```

### Configuring SSE-KMIP as the Default for a Storage Policy (CMC Only)

When you use the CMC to create or edit storage policies, one of the configurable policy attributes is server-side encryption. This sets a default server-side encryption type for all objects uploaded to all buckets that use the storage policy. To configure a storage policy to use SSE-KMIP by default, in the "Server-Side Encryption" field of the storage policy configuration interface select "KMIP".

Note that when SSE-KMIP is configured as the default for a storage policy:

- This will apply to the default configuration of buckets that use the storage policy, if the buckets either have no SSE configuration or are configured to use regular SSE. (By contrast, buckets configured to use AWS KMS will use AWS KMS).
- This will apply to objects uploaded to buckets that use the storage policy, if the objects either have no SSE specified in the upload request or have regular SSE specified in the upload request. (By contrast, objects that specify AWS KMS or SSE-C in the upload request will use AWS KMS or SSE-C).

For more information on storage policy configuration, while on the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) click **Help**.

### Re-Keying a Bucket

Once a bucket has been using SSE-KMIP -- because it uses a storage policy on which you've enabled SSE-KMIP -- you may wish to periodically generate a new master key for the bucket. This is known as "re-keying" the bucket. The bucket's new master key -- which will be created by and stored on the KMIP KMS along with the bucket's older master key(s) -- will apply only to the encryption of HyperStore-supplied data keys for objects uploaded **after** the re-keying occurs (and decryption of those data keys if those objects are subsequently downloaded). The older bucket master key(s) continue to be used for decrypting data keys for objects that existed in the bucket prior to the re-keying.

**IMPORTANT !** Some KMIP KMS products -- for example Fortanix -- allow you to rotate keys (re-key) on the KMS console. **Do not rotate HyperStore bucket keys directly on the KMS.** This will result in the KMS and HyperStore being out of sync in regard to which master key a bucket is currently using. Instead, follow the procedure below to re-key a bucket.

You can re-key a bucket or buckets **on demand** by logging into any HyperStore node and running whichever of the following commands best suits your purpose.

Re-key a specified bucket:

```
curl -d "" 'http://localhost/.system/rekey?bucketName=<bucketname>'
```

Re-key all buckets that use a specified storage policy:

```
curl -d "" 'http://localhost/.system/rekey?storagePolicy=<storagePolicyId>'
```

where <*storagePolicyId*> is the system-generated unique ID for the storage policy (which you can get from the CMC's **Cluster -> Storage Policies** page or from the Admin API call *GET /bppolicy/listpolicy*).

Re-key all buckets that use KMIP-enabled storage policies:

```
curl -d "" 'http://localhost/.system/rekey'
```

Note that in a multi-region HyperStore system:

- The commands for rekeying a specified bucket or a specified storage policy must be run from a HyperStore node within the region where the bucket or storage policy are based.

- The command for re-keying all buckets that use KMIP-enabled storage policies must be run from one node in each service region. For example if you have two regions "Region1" and "Region2", run the command from a node in Region1 and then run the command from a node in Region2.

Alternatively you can configure a **cron job** to regularly re-key all buckets that use KMIP-enabled storage policies:

1. Log into the HyperStore Configuration Master node.
2. Open the */etc/cloudian-7.5.2-puppet/modules/cloudians3/templates/cloudian-crontab.erb* file in a text editor.
3. At the bottom of the file:
   a. Locate the commented out sample line for triggering a re-key all buckets that use KMIP-enabled storage policies, once each month (shown here the line breaks, but it is one unbroken line in the actual *cloudian-crontab.erb* file):

```
# Re-key all KMIP-enabled buckets every month
# 0 0 1 * * <%= @cloudians3_template_run_user %> curl -d ""
http://localhost:<%= @cloudian_template_s3_port %>/.system/rekey > /dev/null 2>&1
```

   b. Uncomment the line and if desired edit the cron job schedule, using standard cron scheduling notation.
4. Push your changes out to the cluster and restart the S3 Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382). In a multi-region system, when doing the push be sure to push to all regions.

**Note** Do not have two processes re-keying the same bucket at the same time. For example, do not manually run re-key commands at the same time as a re-key cron job is running; and be sure not to have two different system administrators running re-key commands for the same bucket at the same time. This could result in the KMS and HyperStore being out of sync in regard to which master key a bucket is currently using.

### Bucket Deletion

When a bucket that has been using SSE-KMIP is deleted in HyperStore, HyperStore sends to the KMIP KMS a command to revoke the bucket's master key. If the bucket has had multiple master keys as a result of re-keying, HyperStore directs the KMIP KMS to revoke all of the bucket's master keys.

If you wish to permanently destroy the revoked keys, it must be done on the KMIP KMS. Check your KMS documentation for information about destroying revoked keys.

## 4.2.2.4. Using AWS KMS

*Subjects covered in this section:*

With **AWS KMS** encryption -- server-side encryption utilizing the Amazon Web Services Key Management Service -- HyperStore triggers in the AWS KMS the creation of one "customer master key" (CMK) per bucket. The CMK is stored exclusively in the AWS KMS. For each such CMK, HyperStore stores (in the Credentials DB) a CMK ID that allows HyperStore to tell AWS KMS which CMK to use for creating a unique encrypted "data key" for a given object (that is, the CMK for the bucket into which the object is being uploaded). AWS KMS returns to HyperStore both the encrypted data key and plain text version of the data key. After using the plain text data key to encrypt the object, HyperStore discards the plain text data key and stores the encrypted version of the data key within the object's metadata.

Subsequently when a client downloads the object, HyperStore submits the encrypted data key to the AWS KMS, which uses the CMK to decrypt the data key. AWS KMS returns the plain text data key to HyperStore, which uses it to decrypt the object and then deletes the plain text data key from memory.

AWS KMS based encryption can be configured as the default encryption type for a particular bucket or as the encryption type for particular objects. Setting the AWS KMS method as the default for a storage policy is not supported. For information about the precedence ordering between these configuration levels see **"Server-Side Encryption"** (page 150).

> **Note** In compliance with Amazon's implementation, all S3 requests involving AWS KMS encryption must use SSL and Signature Version 4. For example HyperStore will reject object upload requests that specify AWS KMS encryption, or download requests for AWS KMS encrypted objects, if such requests use Signature Version 2.

> **Note** For more information on the AWS KMS, in the AWS online documentation see **AWS Key Management Service (KMS)**.

### Preparing the AWS KMS Encryption Feature

To use the AWS KMS for HyperStore server-side encryption, you must have either or a combination of:

- **AWS account access credentials for each HyperStore user group** that will use the AWS KMS encryption feature. These group account credentials will be used by HyperStore to access the AWS KMS whenever a user within the group requests AWS KMS encryption for their bucket or for specific objects, or downloads AWS KMS encrypted objects.
- **Default AWS account access credentials** for your HyperStore service as a whole. These default AWS credentials will be used by HyperStore to access the AWS KMS on behalf of users who are in groups that do not have group account credentials for AWS.

To enable AWS KMS usage in your HyperStore system, complete the following system configuration steps:

1. On the Configuration Master node, open the following file in a text editor:

   ```
   /etc/cloudian-<version>-puppet/modules/cloudians3/files/awscredentials.properties
   ```

2. Edit the file to specify the system default AWS access credentials, and (optionally) any group-specific AWS access credentials. Create a separate block for each group that has its own AWS access credentials, using the formatting shown in the example below. In the example there are credentials for just one group, "CloudianTest1". HyperStore will use the CloudianTest1 group's AWS credentials when accessing the AWS KMS on behalf of users in that group. For users in any other group, HyperStore will use the system default AWS credentials when accessing the AWS KMS.

   ```
   [default]
   ```

```
aws_access_key_id = AKIAJKVELYABCCEIXXMA
aws_secret_access_key = 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

[CloudianTest1]
aws_access_key_id = ABCAJKVELY6YXCEIMAXX
aws_secret_access_key = abceikWvRR/7A8916x9vUDEhV+C+LIDmFCOE8MgC
```

Save your change and close the file.

3. Still on the Configuration Master node, open the following file in a text editor:

```
/etc/cloudian-<version>-puppet/modules/cloudians3/templates/mts.properties.erb
```

4. Find the property *util.awskmsutil.region* and set it to the AWS service region of the AWS KMS that you want HyperStore to use. The default is "us-east-1". Save your change and close the file.
5. Push your changes to the cluster and restart the S3 Service. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

## Configuring AWS KMS Encryption as the Default for a Bucket (S3 API Only)

HyperStore's S3 API supports setting a default server-side encryption type for a bucket. This sets a default server-side encryption type for all objects uploaded to the bucket. The S3 API operations that set and manage a bucket default server-side encryption method are:

- PutBucketEncryption
- GetBucketEncryption
- DeleteBucketEncryption

To configure a bucket for server-side encryption with AWS KMS, in the *PutBucketEncryption* request body set the *SSEAlgorithm* element to *aws:kms*.

For details of HyperStore's support for these S3 API operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

**Note** The CMC does not support setting a bucket default server-side encryption method.

## Requesting AWS KMS Encryption for Specific Objects (S3 API Only)

HyperStore's S3 API supports AWS KMS based server-side encryption by the inclusion of the *x-amz-server-side-encryption: aws:kms* request header in any of these operations on objects:

- PutObject
- CreateMultipartUpload
- UploadPart
- POST Object
- CopyObject

For details about HyperStore support of these operations see the S3 section in the ***Cloudian HyperStore AWS APIs Support Reference***.

Note that:

- If AWS KMS is configured as the default server-side encryption type for the bucket into which an object is being uploaded there is no need to explicitly request AWS KMS as the encryption type for that object.
- The HyperStore S3 Service does not support the *x-amz-server-side-encryption-aws-kms-key-id* or *x-amz-server-side-encryption-context* request headers.
- The CMC does not support requesting AWS KMS based encryption for objects as they are uploaded. It **does** support downloading objects that have been encrypted by the AWS KMS method.

### Deleting a Bucket That Uses AWS KMS Encryption

When you delete a HyperStore bucket that has used AWS KMS encryption -- either because AWS KMS encryption was the default encryption method for the bucket, or because certain objects within the bucket used AWS KMS encryption -- the bucket's "customer master key" (CMK) is scheduled for deletion from the remote AWS KMS system. The CMK deletion occurs 30 days after the deletion of the HyperStore bucket. During this 30 day period, if you do not wish the CMK to be deleted from the remote AWS KMS system you can execute a *CancelKeyDeletion* operation using the AWS Console or the AWS CLI.

## 4.2.2.5. Using SSE-C

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Preparing the SSE-C Feature"** (page 163)*
- ***"Requesting SSE-C for Specific Objects (S3 API Only)"** (page 164)*

When **server-side encryption with customer-provided encryption keys (SSE-C)** is used, the HyperStore system does not store the customer-provided encryption key itself but rather stores a hash of the key (for purposes of verifying the key if it's subsequently submitted in a GET Object request). The key hash is stored with the object metadata.

SSE-C based encryption can only be set at the per-object level. Setting SSE-C as the default encryption method for a bucket or a storage policy is not supported.

**IMPORTANT !** When SSE-C is used, **the user is responsible for managing the encryption key**. If an object is uploaded to HyperStore system and encrypted with a user-provided key, the user will need to provide that same key when later requesting to download the object. **If the user loses the key, the encrypted object will not be downloadable**. This is consistent with Amazon's implementation of the SSE-C feature. For more information on Amazon's SSE-C feature see **Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C)**.

### Preparing the SSE-C Feature

Before using SSE-C you must first:

- Enable AES-256 in your HyperStore system. Like Amazon S3, HyperStore's implementation of SSE-C requires AES-256 encryption of object data. For instructions see **"Enabling AES-256"** (page 164).
- Set up HTTPS for your S3 Service. Like Amazon S3, HyperStore's implementation of SSE-C requires that the relevant S3 API requests be transmitted over HTTPS rather than regular HTTP. For instructions see **"Security and Privacy Features"** (page 141).

**Note** HyperStore supports a configuration for allowing a regular HTTP connection between a load balancer and your S3 servers for transmission of SSE-C requests over your internal network, while client applications use HTTPS in the requests that come into the load balancer. See the configuration parameter *mts.properties.erb:* **"cloudian.s3.ssec.usessl"** (page 466).

### Requesting SSE-C for Specific Objects (S3 API Only)

HyperStore's S3 API supports server-side encryption with user-provided keys by the inclusion of the *x-amz-server-side-encryption-customer-** request headers in any of these operations on objects:

- PutObject
- CreateMultipartUpload
- UploadPart
- POST Object
- CopyObject (supporting also the *x-amz-copy-source-server-side-encryption-customer-** request headers)
- GetObject
- HeadObject

For the full list of supported *x-amz-server-side-encryption-customer-** request headers for each of these operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

**Note** The CMC does not support requesting SSE-C for objects as they are uploaded, and it does not support downloading objects that have been encrypted by the SSE-C method.

## 4.2.2.6. Enabling AES-256

In implementing **server-side encryption**, HyperStore by default uses AES-128 when encrypting object data. By default HyperStore does not use AES-256 for this purpose. AES-256 is the most secure form of the Advanced Encryption Standard.

You must enable AES-256 in your HyperStore system if you want to do either of the following:

- Have HyperStore use AES-256 to encrypt object data when implementing either **regular SSE** or **SSE-KMIP**.
- Use **SSE-C**. SSE-C requires that AES-256 is enabled for object data encryption and will not work if AES-256 is not enabled.

**Note** Enabling AES-256 is not necessary for -- and not relevant to -- server-side encryption using **AWS KMS**.

To enable AES-256 in your HyperStore system, do the following:

1. On the Configuration Master node, in the **common.csv** file, set *cloudian_s3_aes256encryption_enabled* to *true*. (By default it is set to *false*.)
2. Push your changes out to the cluster and restart the S3 Service. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

## 4.2.3. Setting Up User Password and Credential Controls

*Subjects covered in this section:*



### 4.2.3.1. CMC Passwords

System administrators can use the Cloudian Management Console (CMC) to monitor and administer the HyperStore system, and to provision and manage service users and user groups. Group administrators can use the CMC to provision and manage service users within their group, and to use the S3 storage service (creating and configuring buckets and uploading and downloading objects). Regular users can use the CMC to manage their own user accounts and to use the S3 storage service.

All types of users require a password to access the CMC. For the pre-configured system admin user named *admin*, the default password is *public* and you are required to change the password the first time you log into the CMC as the admin user (see **"Accessing the Cloudian Management Console"** (page 89)). For all other users, when you create new users in the CMC -- additional system admin users, or group admins, or regular users -- you specify an initial CMC password for each user. Users can subsequently change their own password through the CMC's **Security Credentials** page (accessible by hovering the cursor over the user name in the upper right corner of the CMC). And as a system admin you can change other users' passwords through the CMC's **Manage Users** page.

> **Note** The Admin API also supports provisioning users and creating user passwords. For more information see the "user" section of the ***Cloudian HyperStore Admin API Reference***.

As a system administrator you can configure several types of requirements for CMC passwords, as shown in the table below. These settings are all in the configuration file *common.csv*. Note that some of the requirements are disabled by default. For more detailed descriptions of these settings and how to change them (including enabling requirements that are disabled by default), see **"common.csv"** (page 388).

| Password Restriction | Configuration Setting | Default |
|---|---|---|
| Minimum number of characters in password | user_password_min_length | 9 |
| Number of days of password lifespan before the password expires and a new password must be created | user_password_rotation_expiration | Requirement disabled |
| Maximum allowed character redundancy between old password and new password | user_password_dup_char_ratio_limit | Requirement disabled |
| Minimum number of new passwords before an old password can be reused | user_password_unique_generations | Requirement disabled |
| Minimum days between password changes | user_password_rotation_graceperiod | Requirement disabled |
| Whether or not to lock out a user who submits an incorrect password or passwords multiple times | user_password_lock_enabled | Enabled if your original HyperStore version was |

| Password Restriction | Configuration Setting | Default |
|---|---|---|
| | | 7.3 or newer<br><br>Disabled if your original HyperStore version was older than 7.3 |
| Time interval of password lock-out in number of seconds (if lock-out feature is enabled) | user_password_lock_durationsec | 1800 |
| Number of failed attempts that triggers lock-out (if lock-out feature is enabled) | user_password_lock_maxfailedattempts | 6 |

#### 4.2.3.2. CMC Multi-Factor Authentication

The CMC supports multi-factor authentication (MFA), whereby a user is required to provide a time-based MFA code in order to log into the CMC (in addition to providing their user name and password). By default MFA is disabled for each user. Each user -- whether a system admin, a group admin, or a regular user -- has the option to enable MFA for their own CMC account. Users can enable MFA in the CMC's **Security Credentials** page (the same page in which they manage their CMC password).

For more information about how a user can enable MFA on their CMC account, while on the CMC's **Security Credentials** page click **Help**.

Optionally you can configure the system so that the CMC **requires** all users (including system administrators) to set up and use MFA for their CMC logins. For more information on this option see *common.csv:* **"mfa_enforced"** (page 408).

As a system administrator, you can disable MFA on the account of a user who currently has MFA enabled. You might need to do this if, for example, an MFA-enabled user is unable to log into the CMC because of a problem with their virtual MFA device (MFA application on their computer). If you have configured the system to require MFA for CMC logins, and you disable MFA for a particular user, the user will need to set up MFA again the next time they try to log into the CMC.

For more information about disabling MFA on a CMC user's account, while on the CMC's **Manage Users** page click **Help**.

#### 4.2.3.3. S3 Security Credentials

To use the HyperStore S3 Service -- the service through which storage buckets are created and configured and objects are uploaded and downloaded -- users must have S3 security credentials (a paired "access key" and "secret key"). These credentials are distinct from the CMC password, and are used specifically to interact with the HyperStore storage layer through the S3 API. When you create a new user in the CMC, the system automatically creates S3 security credentials for the user. Users can subsequently view or manage their S3 security credentials through the CMC's **Security Credentials** page. And as a system admin you can view or manage other users' S3 security credentials through the CMC's **Manage Users** page.

**Note** The Admin API also supports provisioning users and creating users' S3 security credentials. For more information see the "user" section of the ***Cloudian HyperStore Admin API Reference***.

If a user employs the CMC as their S3 client application, the CMC automatically includes the user's S3 security credentials when the CMC submits calls to the S3 Service. If a user employs a third party S3 client application to access the HyperStore S3 Service, the user will need to supply their S3 security credentials to the third party S3 client application (so the client application can include the credentials when submitting calls to the HyperStore S3 Service). The user can retrieve their S3 security credentials from the CMC's **Security Credentials** page, so that they can then supply the credentials to the third party S3 client application.

Users are allowed to have multiple S3 credentials (multiple access key / secret key pairs). They can create and delete credentials -- and make credentials active or inactive -- in the CMC's **Security Credentials** page. By default the system allows users to have a maximum of five total S3 security credentials (including inactive credentials as well as active credentials). This restriction is configurable by the *credentials.user.max* property in the configuration file *mts.properties.erb*.

## 4.2.4. Enabling Secure Delete

**IMPORTANT ! Using secure delete impacts system performance for delete operations**. Consult with your Cloudian representative if you are considering using secure delete.

*Subjects covered in this section:*



HyperStore supports a "secure delete" methodology for implementing object delete requests. **By default this feature is disabled.**

For background, note that object data is stored in an *ext4* file system on each HyperStore node, and the process of writing, reading, and deleting object data from the file system is managed by the HyperStore Service (for more information see **"HyperStore Service and the HSFS"** (page 63)). Also note that, depending on your storage policies, each object is either replicated or erasure coded, and the replicas or the erasure coded fragments are distributed across multiple nodes. Larger objects are broken into chunks first, before those chunks are then replicated or erasure coded.

When secure delete is enabled HyperStore implements the deletion of an object by first overwriting each byte of the object data three times, and then deleting the object. The three passes at overwriting each of the object's bytes are executed as:

- 1st pass: byte is overwritten as 00110101 (0x35)
- 2nd pass: byte is overwritten as 11001010 (0xCA)
- 3rd pass: byte is overwritten as 10010111 (0x97)

This overwriting occurs for **every byte of every replica or fragment of the object, on every node on which the object's data resides**. After the three overwriting passes complete, the object data is then deleted.

If you enable secure delete, then all deletes -- for any bucket, by any user -- are implemented as secure deletes. You cannot, for instance, apply this feature only to some buckets and not to others.

**Note**
* In the case of buckets that use versioning, to delete all versions of an object the S3 client application must explicitly delete each object version.
* Secure delete does not apply to object metadata stored in the Metadata DB (in Cassandra). When

objects are deleted, the system deletes the corresponding object metadata in the normal way -- with no overwriting passes -- regardless of whether or not you have the secure delete feature turned on. Therefore you should limit any user-defined object metadata created by your S3 client applications to information that does not require secure delete.

#### 4.2.4.1. Enabling Secure Delete

To enable the secure delete feature:

1. On the Configuration Master node, in the **hyperstore-server.properties.erb** file, set *secure.delete* to *true*. (By default it is set to *false*.)
2. Use the installer to push your configuration change to the cluster and to restart the HyperStore Service. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

#### 4.2.4.2. Secure Delete Logging

Secure delete activity is logged in *cloudian-hyperstore-request-info.log* on each node. The activity is logged only after completion of the third and final overwrite pass. The log entry indicates SECURE-DELETE as the operation type, and a 200 status code in the log entry indicates that the secure delete was successful. The log entry also includes the object name and the corresponding *ext4* file name.

For more information about *cloudian-hyperstore-request-info.log* see **"HyperStore Service Logs"** (page 354).

### 4.2.5. FIPS Support

*Subjects covered in this section:*



Federal Information Processing Standard (FIPS) **Publication 140-2** defines security requirements for cryptographic modules. HyperStore in some respects meets the FIPS 140-2 standard by default. In particular, HyperStore by default uses FIPS 140-2 prescribed cryptographic techniques for:

- **AES encryption** (used for HyperStore's **server-side encryption** feature)
- **SHA-256** (used to hash request payloads for some types of S3 requests)
- **HMAC** (used for Signature version 4 validation of S3 requests)

However:

- The cryptographic module that HyperStore uses by default -- the OpenSSL FIPS Object Module 2.0 -- entered "sunset" status in January 2022, and therefore is no longer strictly FIPS compliant.
- By default, the **SSH** service on HyperStore nodes -- OpenSSH -- is **not** FIPS 140-2 compliant because it supports ciphers that are not FIPS 140-2 approved as well as ciphers that are FIPS 140-2 approved.

Therefore, **if you need strict FIPS compliance you must enable HyperStore FIP compliance as described below**. Once you enable HyperStore FIPS compliance then:

- HyperStore will use the Bouncy Castle FIPS module rather than the sunsetted OpenSSL FIPS Object Module 2.0.
- Only FIPS 140-2 approved ciphers will be used for SSH connections to HyperStore nodes.

#### 4.2.5.1. Enabling HyperStore FIPS Compliance

To make HyperStore strictly FIPS compliant, follow these steps:

1. On the Configuration Master node, in the **common.csv** file, set *fips_enabled* to *true*. (By default it is set to *false*.) Save your change and close the file.

   **Note** Setting *fips_enabled* to *true* will work only if you leave the *sshdconfig_disable_override* setting (which is also in *common.csv*, directly below *fips_enabled*) at its default value of *false*.

2. Still on the Configuration Master node, run this command:

   ```
   hsctl config apply fips
   ```

3. Still on the Configuration Master node, use the installer to push your configuration change out to the cluster and to restart the S3 Service. If you need instructions for this step see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

#### 4.2.5.2. Limitations to FIPS Compliance for Server-Side Encryption

As described in **"Server-Side Encryption"** (page 150), HyperStore supports several types of server-side encryption. For Regular SSE, HyperStore generates a master encryption key and per-object encryption keys and implements the encryption and decryption of object data. Once you've enabled HyperStore FIPS support (as described above), this type of server-side encryption is fully FIPS 140-2 compliant.

However, HyperStore also supports methods of server-side encryption for which the encryption keys come from outside of HyperStore. With SSE-C, the keys are provided by the user. With server-side encryption using a KMIP KMS or AWS KMS, the keys are generated by an external key management system.

For these types of server-side encryption, although HyperStore executes the encryption and decryption of object data using FIPS-compliant AES, the generation of the encryption keys is outside HyperStore control. These types of server-side encryption are fully FIPS-compliant only if the keys are generated in a FIPS-compliant manner.

## 4.3. Auto-Tiering

### 4.3.1. Auto-Tiering Feature Overview

**Note** Auto-tiering is not supported in a HyperStore Single-Node system.

*Subjects covered in this section:*

The HyperStore system supports an "auto-tiering" feature whereby objects can be automatically moved from local HyperStore storage to a remote storage system on a defined schedule. HyperStore supports auto-tiering from a local HyperStore bucket to any of several types of destinations systems:

- S3-compliant systems: Amazon S3, Amazon Glacier, Google Storage Cloud, a HyperStore region or system, or a different S3-compliant system of your choosing
- Microsoft Azure
- Spectra Logic BlackPearl

Auto-tiering is configurable on a per-bucket basis. A bucket owner activating auto-tiering on a bucket can specify:

- The auto-tiering destination system
- Whether auto-tiering applies to all objects in the bucket, or only to objects that match against configurable filters for object name prefix, object size, and/or object tags

  **Note** The HyperStore S3 API supports auto-tiering filtering by prefix, size, and/or tags, but the CMC currently only supports filtering by prefix and/or size. The CMC does not support filtering by object tags.

- The tiering schedule, which can be implemented in any of these ways:
  - Move objects X number of days after they’re created
  - Move objects if they go X number of days without being accessed
  - Move objects on a fixed date — such as December 31, 2022
  - Move objects immediately after they're created ("Bridge Mode", also known as proxy tiering). Note that with Bridge Mode, filtering by prefix, size, or tags is not supported -- if Bridge Mode is used, this applies to all objects incoming to the bucket.

Although there can only be one auto-tiering destination for a given HyperStore bucket, bucket owners have the option of configuring different auto-tiering schedules for different sets of objects within the bucket, based on the object name prefix (unless Bridge Mode is being used, which does not support prefix filtering).

For more information about configuring auto-tiering in the system and on individual buckets, see **"Preparing the Auto-Tiering Feature"** (page 175) and **"Setting Up Auto-Tiering for a Bucket"** (page 178).

**Note Auto-tiering restrictions based on destination type:**
* Tiering to Azure, Google, or Spectra BlackPearl is not supported for source buckets that have

versioning enabled or that have had versioning enabled in the past.
* For tiering to Google to work, the destination bucket in Google must be configured for "fine-grained" access, not "uniform" access.
* When auto-tiering to Spectra BlackPearl is used for a bucket, objects in the bucket will not be auto-tiered unless they are larger than 5MB. Objects 5MB or smaller will remain in HyperStore. To change this limit, consult with Cloudian Support.

#### 4.3.1.1. Auto-Tiering Versus Cross-Region Replication

The key differences between auto-tiering and **cross-region replication** are:

- With cross-region replication each replicated object is stored in both the source bucket and the destination bucket. With auto-tiering, after an object is auto-tiered the object data is by default stored only in the destination bucket (although there is an option to retain a local copy for a configurable period of time).
- Cross-region replication replicates objects to the destination bucket immediately after they've been uploaded to the source bucket. With auto-tiering, typically the tiering of the object occurs on a user-defined schedule, after the objects have been in the source bucket for a period of time (although there is an option to tier immediately).
- Cross-region replication replicates nearly all object metadata and user-defined tags along with the object data, so that an object's full set of metadata resides both at the source and at the destination. With auto-tiering only basic metadata accompanies the tiered object, while the object's full metadata continues to be stored locally at the source.
- Cross-region replication requires that versioning be enabled on both the source bucket and the destination bucket. Auto-tiering does not have this versioning requirement.
- With cross-region replication the destination bucket is typically within the same HyperStore system as the source bucket (although replicating to an external system is supported). With auto-tiering the destination bucket is typically in an external system (although tiering to a bucket within the same HyperStore system is supported).

You cannot use cross-region replication and auto-tiering on the same source bucket.

#### 4.3.1.2. Tiering Destination Accounts, Credentials, and Buckets

Auto-tiering to Amazon or another destination system requires that there be an existing account that the HyperStore system can access at the destination system. There are two options for account access:

- Bucket owners can supply their own destination account credentials, on a per-bucket basis. In this way each bucket owner tiers to his or her own account at the destination system. This is the default method for auto-tiering.
- You can supply system default tiering credentials. This is the appropriate approach if all users will be tiering to the same account at the same tiering destination system. For more information see **"Preparing the Auto-Tiering Feature"** (page 175).

HyperStore encrypts supplied tiering account credentials and stores them in the Credentials DB, where they can be accessed by the system in order to implement auto-tiering operations.

Auto-tiering moves objects from a local HyperStore source bucket into a **tiering bucket** at the destination system. The source bucket owner when configuring auto-tiering can specify as the tiering bucket a bucket that

already exists in the destination system, or the source bucket owner can have HyperStore create a tiering bucket in the destination system. If having HyperStore create a tiering bucket, the source bucket owner can choose a tiering bucket name or have HyperStore automatically name the tiering bucket. When HyperStore automatically names the tiering bucket it uses this format:

*<origin-bucket-name-truncated-to-34-characters>-<random-string>*

The HyperStore system appends a 28-character random string to the origin bucket name to ensure that the resulting destination bucket name is unique within the destination system. If the origin bucket name exceeds 34 characters, in the destination bucket name the origin bucket name segment will be truncated to 34 characters.

After objects have been auto-tiered to the destination system they can be accessed directly through that system's interfaces (such as the Amazon S3 Console), by persons having the applicable credentials. Auto-tiered objects can also be accessed indirectly through the local HyperStore system interfaces. Tiered object access is described in more detail in Accessing Auto-Tiered Objects.

**Note** In the case of auto-tiering to Amazon Glacier, the HyperStore system creates a bucket in Amazon S3 and configures that remote bucket for immediate transitioning to Glacier. Objects are then auto-tiered from HyperStore to Amazon S3, where they are immediately subject to Amazon's automated mechanism for transitioning objects to Glacier.

#### 4.3.1.3. Bridge Mode (Proxy Tiering)

The HyperStore auto-tiering feature supports a "Bridge Mode" (also known as proxy tiering) whereby objects can be transitioned to a destination system immediately after they are uploaded to the HyperStore source bucket. As soon as such objects are successfully transitioned to the destination system, by default they are flagged for deletion from the local HyperStore system (the actual deletion will be executed afterwards, by a cron job that runs hourly). After deletion of the local copy of the objects, only object metadata remains (see **"How Auto-Tiering Impacts Usage Tracking, QoS, and Billing"** (page 174) for more information about this metadata).

With Bridge Mode, filtering by prefix or tags is not supported -- if Bridge Mode is used, this applies to **all** objects incoming to the bucket.

When the system implements Bridge Mode tiering, whichever S3 node processes the upload of a given object into the source bucket also initiates the immediate transmission of the object to the destination system. In this way, just as the workload for processing numerous incoming object uploads from S3 client applications is distributed across all the nodes in the cluster, so too the workload associated with Bridge Mode tiering is distributed across the cluster. If the initial attempt to transmit an object to the destination system fails with a temporary error, the same node that performed the initial attempt will retry once every hour until either the object is successfully transmitted or a permanent error is encountered (an example permanent error would be if someone had deleted the tiering destination bucket). The local copy of the object will not be deleted until the object is successfully transmitted to the destination system, as indicated by the destination system returning a success status. All attempts are logged as described in **"Auto-Tiering Logging"** (page 173).

Users have the option of choosing Bridge Mode when they configure auto-tiering rules for a bucket.

**Note** Bridge mode is not supported for tiering to Amazon Glacier or Spectra BlackPearl.

#### 4.3.1.4. Option to Retain a Local Copy

By default the system deletes the local copy of an object as soon as the object is successfully transitioned to the tiering destination. However, HyperStore supports an option to retain a local copy of objects that have been auto-tiered, for a specified period of time. Locally retained objects will continue to be protected by the storage policy associated with the bucket in which the objects reside (whether replication or erasure coding). After the specified retention period the system will automatically delete the local copy.

Users have the option of specifying a local retention period when they configure auto-tiering rules for a bucket. This option to retain a local copy is supported for Bridge Mode (Proxy) tiering as well as for regular, schedule-based tiering.

#### 4.3.1.5. Bi-Directional Auto-Tiering

Bi-directional auto-tiering between two buckets within the same HyperStore system -- so that HyperStore bucket1 tiers to HyperStore bucket2 and HyperStore bucket2 tiers to HyperStore bucket1 -- is prohibited in HyperStore version 7.5 and later.

#### 4.3.1.6. Auto-Tiering Logging

As the HyperStore system auto-tiers objects from local storage to the tiering destination, it records information about the tiering transactions (including source bucket name, object name, and destination bucket name) in the tiering request log **cloudian-tiering-request-info.log**. For regular auto-tiering that occurs on a defined schedule, the auto-tiering processing associated with a given bucket -- and the logging of that auto-tiering processing -- is spread out across all the nodes that participate in the storage policy used by that bucket. For example, in a HyperStore system with two data centers, if a bucket uses a storage policy that stores data only in one of the two data centers, the processing workload of that bucket's auto-tiering will be spread among the nodes in that one data center -- and so will the logging entries for that bucket's auto-tiering activity. By contrast, for a bucket that uses a storage policy that stores data in both data centers, the processing workload (and log entries) associated with that bucket's auto-tiering will be spread among all nodes in both data centers.

In the special case of "Bridge Mode" auto-tiering, whichever S3 node processes the upload of a given object into the source bucket also initiates the immediate auto-tiering of the object to the destination system, and the tiering request log entry for that is written locally on that node.

#### 4.3.1.7. Auto-Tiering Licensing

Your HyperStore license may impose a limit on how much tiered data you can have stored in third party external systems (destinations other than an external HyperStore system). For details see **"Licensed Maximum Tiered Storage Usage"** (page 36).

#### 4.3.1.8. Auto-Tiering of Encrypted Objects

For information about how auto-tiering works together with the server-side encryption feature, see **"Server-Side Encryption and Auto-Tiering"** (page 152). As detailed in that section, encrypted objects may or may not be eligible for auto-tiering depending on the encryption method and the particular tiering destination.

### 4.3.1.9. Auto-Tiering and User-Defined Object Metadata

For information about how auto-tiering impacts user-defined object metadata, see **"Cross-Region Replication and Auto-Tiering of Object Metadata and Tags"** (page 200).

### 4.3.1.10. How Auto-Tiering Impacts Usage Tracking, QoS, and Billing

When an object is auto-tiered to a destination system and deleted from local storage the size of the tiered object is subtracted from the bucket owner's HyperStore **usage count** for Storage Bytes. At the same time, a reference to the auto-tiered object is created and the size of this reference — 8KB, regardless of the transitioned object size — is added to the Storage Bytes count. Meanwhile, auto-tiering does not impact the Storage Objects count. An object counts toward the number of Storage Objects regardless of whether or not it is auto-tiered.

For example, if a 1MB object has been auto-tiered to Amazon then that object would count as:

- 8KB toward the bucket owner's HyperStore count for Storage Bytes.
- 1 toward the bucket owner's HyperStore count for Storage Objects.

If an auto-tiered object is temporarily restored to HyperStore storage, then while the object is restored the object's size is added back to the Storage Bytes count and the 8KB for the reference is subtracted from the count. After the restore interval ends and the restored object instance is automatically deleted, the object size is once again subtracted from Storage Bytes and the 8KB for the reference is added back. For more on temporary restoration of tiered objects see Accessing Auto-Tiered Objects.

Auto-tiering does not impact HyperStore usage counts for Bytes-IN or Bytes-OUT.

> **Note** If users select the **Retain Local Copy** option when configuring their buckets for auto-tiering, objects that are temporarily retained in HyperStore after they've been auto-tiered will continue to count toward the local Storage Bytes count until the local copy is deleted.

HyperStore's Quality of Service (QoS) and billing features make use of Storage Bytes and Storage Object counts. So the impact of auto-tiering on QoS and billing is as described above. For example in the case of the QoS restrictions applied to users, if a bucket owner's 1MB object has been auto-tiered to Amazon, then that object counts as 8KB toward that user's QoS limit for Storage Bytes; and as 1 toward that user's QoS limit for Storage Objects.

For more information on the QoS and billing features, see **"Quality of Service (QoS) Feature Overview"** (page 213) and **"Billing Feature Overview"** (page 225).

### 4.3.1.11. Tiering Statistics for Spectra BlackPearl

If your HyperStore system is using auto-tiering to Spectra BlackPearl, you can run the following command on the HyperStore cron job master node to retrieve tiering statistics. This command is supported only for Spectra BlackPearl tiering.

```
# curl http://localhost:80/.system/stats/tiering/spectra
```

For example:

```
# curl http://localhost:80/.system/stats/tiering/spectra
{"totalInQueue":0,"totalTiered":51,"tieringFail":0,"restored":1,"restoreFail":0,
"bytesTiered":629145600,"bytesRestored":1073741824}
```

The returned statistics are:

- Number of objects in queue waiting to be tiered to Spectra BlackPearl
- Number of objects tiered to Spectra BlackPearl
- Number of objects for which tiering to Spectra BlackPearl failed
- Number of tiered objects restored to HyperStore from Spectra BlackPearl
- Number of objects for which restoring to HyperStore from Spectra BlackPearl failed
- Total number of bytes tiered to Spectra BlackPearl
- Total number of bytes restored to HyperStore from Spectra BlackPearl

**Note** If you're not sure which node is your cron job node, you can check this in the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

## 4.3.2. Preparing the Auto-Tiering Feature

This section covers the following auto-tiering system set-up topics:


- ***"Enable and Configure the Auto-Tiering Feature in the CMC"*** *(page 175)*
- ***"Change the Multipart Upload Size Threshold, If Tiering to Google"*** *(page 177)*
- ***"Specify a Different HyperStore System as a Tiering Destination, If Desired"*** *(page 177)*


### 4.3.2.1. Enable and Configure the Auto-Tiering Feature in the CMC

**Note** The configuration task described below is applicable only to using the auto-tiering feature through the CMC. It is not applicable to using a third party S3 client application to invoke the HyperStore S3 Service's auto-tiering feature.

By default auto-tiering functionality is **disabled** in the CMC, such that when CMC users are configuring bucket lifecycle properties they will not see an option for auto-tiering (they will only see an option for auto-expiration). If you want to enable the auto-tiering feature in the CMC -- so that CMC users can apply auto-tiering to their buckets -- do the following:

1. In the CMC's **Configuration Settings** page, open the **Auto-Tiering** panel.
2. Set "Enable Auto-Tiering" to Enabled.
3. Configure how you want auto-tiering options to display to CMC users as they configure their buckets for auto-tiering. The system supports three different approaches:
    - If you want all CMC users to auto-tier to a **single system-default tiering destination account for which you are providing the account security credentials**, set "Enable Per Bucket Credentials" to Disabled and then enter the "Default Tiering URL" and the account security credentials.
    - If you want CMC users to be able to choose from a **pre-configured list of tiering destinations** (AWS S3, AWS Glacier, Google, and Azure by default) and no other destinations, leave "Enable Per Bucket Credentials" at Enabled (the default) and leave "Enable Custom Endpoint" at

Disabled (the default). With this approach users provide their own account security credentials for the tiering destination. You can edit the list of tiering destinations that will display for users, and the endpoints for those destinations, as described in **"Configure Tiering Destinations for Display in the CMC"** (page 176) below.

- If you want CMC users to be able to choose from a **pre-configured list of tiering destinations and also give users the option to specify a custom S3 tiering endpoint**, leave "Enable Per Bucket Credentials" at Enabled (the default) and set "Enable Custom Endpoint" to Enabled. With this approach users provide their own account security credentials for the tiering destination. If users specify a custom tiering endpoint, it must be an S3-compliant system and it cannot be a Glacier, Azure, or Spectra BlackPearl endpoint.

4. After finishing your edits in the **Configuration Settings** page, click **Save** at the bottom of the page to save your changes. These changes are applied dynamically and no service restart is required.

### Configure Tiering Destinations for Display in the CMC

**Note** The configuration task described below is applicable only to using the auto-tiering feature through the CMC. It is not applicable to using a third party S3 client application to invoke the HyperStore S3 Service's auto-tiering feature.

Unless you configure the system so that all users tier to the same one default tiering endpoint (as described above in Step 3's first bullet point), users when they configure auto-tiering for their buckets in the CMC will be able to choose from a list of several common tiering destinations. By default the destinations are AWS S3, AWS Glacier, Google Cloud Storage, and Azure; and by default the endpoints for those destinations are as follows:

| Destination | Default Endpoint |
|---|---|
| AWS S3 | https://s3.amazonaws.com |
| AWS Glacier | https://s3.amazonaws.com |
| Google Cloud Storage | https://storage.googleapis.com |
| Azure | https://blob.core.windows.net |

**Note** If your original HyperStore version was older than 7.1.4, then after upgrade to 7.1.4 or later the default list of tiering destinations will also include Spectra BlackPearl with endpoint *https://b-plab.spectralogic.com*.

If you want to change this list -- by changing the endpoint for any of the destinations above, or by adding or removing destinations -- you can do so by editing the **"cmc_bucket_tiering_default_destination_list"** (page 427) setting in **common.csv**. The setting is formatted as a quote-enclosed list, with comma-separation between destination attributes and vertical bar separation between destinations, like this:

```
"<name>,<endpoint>,<protocol>|<name>,<endpoint>,<protocol>|..."
```

This can be multiple destinations (as it is by default), or you can edit the setting to have just one destination in the "list" if you want your users to only use that one destination.

The *<name>* will display in the CMC interface that bucket owners use to configure auto-tiering, as the auto-tiering destination name. The *<protocol>* must be one of the following:

- s3
- glacier

- azure
- spectra

If you wish you can include multiple destinations of the same type, if those destinations have different endpoints. For example, "Spectra 1,<endpoint1>,spectra|Spectra 2,<endpoint2>,spectra". Each such destination will then appear in the CMC interface for users configuring their buckets for auto-tiering.

If you make any changes to this setting, push your changes to the cluster and restart the CMC. If you need instructions for this see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

### 4.3.2.2. Change the Multipart Upload Size Threshold, If Tiering to Google

By default HyperStore uses the S3 multipart upload function when auto-tiering objects larger than 16MiB. However, Google Cloud Storage does not support the multipart upload function. Therefore if your users will be auto-tiering to Google Storage Cloud you should increase the size threshold that triggers HyperStore to use multipart upload when auto-tiering objects. By setting the size threshold to a value larger than the largest object that you expect your users to be tiering to Google, you can prevent HyperStore from trying to use multipart upload when tiering to Google. HyperStore will upload all objects that are at or below the size threshold or less as a single "part".

To increase the size threshold that triggers HyperStore to use multipart upload when tiering objects:

1. On the Configuration Master node, open the configuration file **mts.properties.erb**.
2. Add this property to the "Tiering" section of the file (the property won't be in the file; you need to add it):

   ```
   cloudian.s3.tiering.part.threshold=<size threshold in number of bytes>
   ```

   For example, to configure HyperStore's auto-tiering feature so that multipart upload is only used for objects larger than 200MiB:

   ```
   cloudian.s3.tiering.part.threshold=209715200
   ```

   > **Note** The multipart upload size threshold that you set will be used for all auto-tiering regardless of the destination. This setting is not specific to tiering to Google.

   Save your change and close the file.

3. Push your change to the cluster and then restart the S3 Service. If you need instructions for this see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

### 4.3.2.3. Specify a Different HyperStore System as a Tiering Destination, If Desired

If your users will be tiering to a region in a **different HyperStore system** (i.e. users will be tiering from HyperStore system A to a region in HyperStore system B), then for tiering to that region to work "out of the box" the endpoint for that region must be in this format:

*s3-<region>.<domain>*

For example:

*s3-boston.company.com*

where "boston" is the actual region name in the destination HyperStore system's own system configuration.

If the endpoint for the external HyperStore system is in any format other than the above -- if, for example, the endpoint is simply *boston.company.com* rather than *s3-boston.company.com* -- then for tiering to work you must first make the following configuration change in your local HyperStore system (the source system from which the tiering will originate):

1. On the Configuration Master node, open the following file in a text editor:

   ```
   /etc/cloudian-<version>-puppet/modules/cloudians3/templates/
   tiering-regions.xml.erb
   ```

2. Copy the sample "Region" block at the top of the *tiering-regions.xml.erb* file and paste it toward the end of the file after the existing "Region" blocks (but before the closing "</XML>" tag that's at the very end of the file).
3. Edit the block as follows:
   - Use the "Name" element to specify the region name of destination system region that you will tier to. Enter the region name exactly as it is defined in the destination HyperStore system.
   - Leave the "ServiceName", "Http", and "Https" elements at their default values.
   - Use the "Hostname" element to specify the service endpoint that you will tier to.

   For example, if the region name is "boston" and the service endpoint is "boston.company.com", then your edited Region block would look like this:

   ```
   <Region>
     <Name>boston</Name>
     <Endpoint>
       <ServiceName>s3</ServiceName>
       <Http>true</Http>
       <Https>true</Https>
       <Hostname>boston.company.com</Hostname>
     </Endpoint>
   </Region>
   ```

   **Note** Make sure that the service endpoint that you will tier to is resolvable in your DNS system.

   **Note** Do not have two instances of "s3" in the endpoint, like *s3-tokyo.s3.enterprise.com*. This may cause auto-tiering errors (and cross-region replication errors, if you use the cross-region replication feature).

4. Save your change and close the *tiering-regions.xml.erb* file.
5. Push your changes to the cluster and restart the S3 Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

## 4.3.3. Setting Up Auto-Tiering for a Bucket

*Subjects covered in this section:*



After you have prepared the auto-tiering feature in the system as described in **"Preparing the Auto-Tiering Feature"** (page 175), you can set up auto-tiering for individual buckets as described in the sections below.

#### 4.3.3.1. Setting Up Auto-Tiering for a Bucket (CMC)

HyperStore service users can configure auto-tiering for their own buckets through the CMC's **Bucket Properties** page (**Buckets & Objects -> Buckets -> Properties**). Here users can set auto-tiering attributes such as the tiering destination, the tiering schedule, tiering filters, and whether or not to temporarily retain a local copy of tiered objects.

Alternatively, as a system administrator you can set auto-tiering for a user's bucket by retrieving the user in the **Manage Users** page and then clicking "View User Data" to open a **Bucket Properties** page for the user's bucket. (This is supported only if you've configured the system to allow system administrators to view and manage users' stored data -- see **"Showing/Hiding CMC UI Functions"** (page 231)).

With either approach, the bucket first must be created in the usual way, and then the bucket can be configured for auto-tiering.

For details, while on the CMC's **Bucket Properties** page click **Help**. Then in the bucket properties Help, from the Supported Tasks list choose "Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration".

> **Note** Auto-tiering cannot be enabled for a bucket that has an underscore in its name. For this reason it's best not to use underscores when naming buckets in HyperStore.

#### 4.3.3.2. Setting Up Auto-Tiering for a Bucket (S3 API and Admin API)

To configure auto-tiering rules on a bucket by using the S3 API, your S3 client application will need to use HyperStore extensions to the S3 API method *PutBucketLifecycleConfiguration*. The extensions take the form of request headers to specify the bucket's auto-tiering attributes. For details about these API extensions see "PutBucketLifecycleConfiguration" in the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

If you plan to configure auto-tiering on a bucket by calling the *PutBucketLifecycleConfiguration*S3 API method, you will **first need to use the HyperStore Admin API to store into the system the tiering destination account security credentials** that HyperStore should use when auto-tiering objects from that bucket. For example, if you want to use the S3 API method *PutBucketLifecycleConfiguration* to configure HyperStore source bucket "my-bucket" to auto-tier to AWS S3, you (or your application) must first use the HyperStore Admin API to post the security credentials that HyperStore should use when accessing AWS S3 on behalf of source bucket "my-bucket". The same requirement applies to other destination types. For details about the relevant Admin API methods, see the "tiering" section of the ***Cloudian HyperStore Admin API Reference***.

## 4.4. Cross-Region Replication

### 4.4.1. Cross-Region Replication Feature Overview

*Subjects covered in this section:*

Like Amazon S3, HyperStore supports cross-region replication (CRR). This feature may be valuable if your HyperStore system consists of multiple **service regions**. With cross-region replication, a bucket in one service region can be configured so that all objects uploaded into the bucket are replicated to a chosen destination bucket in a different service region within the same HyperStore system. This feature enables a bucket owner to enhance the protection of data by having it stored in two geographically dispersed service regions. The feature is also useful in cases where a bucket owner wants to have the same set of data stored in two different regions in order to minimize read latency for users in those regions.

If you wish, you can also use the CRR feature within a single HyperStore service region, so that objects uploaded into one bucket are replicated to a different bucket in the same service region.

As is the case with Amazon S3's implementation of this feature, with HyperStore **both the source bucket and the destination bucket must have "versioning" enabled** in order to activate cross-region replication.

Object metadata — including any access permissions assigned to an object, and any **user-defined object metadata or object tags** — is replicated to the destination bucket along with the object data itself. (The exception is that if an object in the source bucket has an *x-amz-expiration* header, HyperStore does not replicate this header.)

As with Amazon S3, HyperStore's implementation of the cross-region replication feature **does not replicate**:

- Objects that were already in the source bucket before the bucket was configured for cross-region replication (except as noted in **"Handling of Pre-Existing Objects"** (page 184))
- Objects that are encrypted with user-managed encryption keys (SSE-C) or AWS KMS managed encryption keys
- Objects that are themselves replicas from other source buckets. If you configure "bucket1" to replicate to "bucket2", and you also configure "bucket2" to replicate to "bucket3", then an object that you upload to "bucket1" will get replicated to "bucket2" but will not get replicated from there on to "bucket3". Only objects that you directly upload into "bucket2" will get replicated to "bucket3".
- Deletions of specific object versions.
    - In the case of an object deletion request that **specifies the object version**, the object version is deleted from the source bucket but is **not** deleted from the destination bucket.
    - In the case of an object deletion request that **does not specify the object version**, the deletion marker that gets added to the source bucket is replicated to the destination bucket.

> **Note** HyperStore currently supports only Version 1 of the Amazon S3 specification for replication configuration XML, not Version 2. Those two versions differ in regard to whether or not deletion markers are replicated. The behavior described above is the Version 1 behavior, which is implemented by HyperStore.

#### 4.4.1.1. Cross-Region Replication Versus Auto-Tiering

The key differences between cross-region replication and **auto-tiering** are:

- With cross-region replication each replicated object is **stored in both the source bucket and the destination bucket**. With auto-tiering, after an object is auto-tiered the object data is by default stored only in the destination bucket (although there is an option to retain a local copy for a configurable period of time).
- Cross-region replication replicates objects to the destination bucket **immediately after they've been uploaded to the source bucket**. With auto-tiering, typically the tiering of the object occurs on a user-defined schedule, after the objects have been in the source bucket for a period of time (although there is an option to tier immediately).
- Cross-region replication replicates **nearly all object metadata and user-defined tags** along with the object data, so that an object's full set of metadata resides both at the source and at the destination. With auto-tiering only basic metadata accompanies the tiered object, while the object's full metadata continues to be stored locally at the source.
- Cross-region replication requires that **versioning** be enabled on both the source bucket and the destination bucket. Auto-tiering does not have this versioning requirement.
- With cross-region replication the destination bucket is typically within the **same HyperStore system** as the source bucket (although replicating to an external system is supported, as described in **"Cross-System Replication"** (page 182)). With auto-tiering the destination bucket is typically in an external system (although tiering to a bucket within the same HyperStore system is supported).

You cannot use cross-region replication and auto-tiering on the same source bucket.

#### 4.4.1.2. Bi-Directional Replication

HyperStore supports bi-directional replication, whereby you configure two buckets to replicate to each other. For example, objects directly uploaded into "bucket1" can be replicated to "bucket2", while objects directly uploaded into "bucket2" are replicated to "bucket1".

As with the HyperStore cross-region replication generally, objects are not replicated if they are themselves replicas from another source bucket. Only objects directly uploaded into a bucket by a client application will be replicated. In the context of bi-directional replication this means that objects that are uploaded directly into one bucket will be replicated to the other bucket, but they will not then be replicated back into the original bucket and so on in an endless loop.

#### 4.4.1.3. Cross-Region Replication Impact on Usage Tracking

If a user configures cross-region replication and consequently objects are replicated from a source bucket in one HyperStore region system to a destination bucket in a different HyperStore region, the replica data in the destination region will count toward the user's Stored Bytes count in the destination region as well as in the source region. For instance a 100MB object that gets replicated in this way would count as 100MB toward the user's Stored Bytes count in the source region and also as 100MB toward the user's Stored Bytes count in the destination region, for a total of 200MB across the system. (Also, for each replicated object there are about 50 bytes of object metadata associated with implementing this feature).

Note also that to enable cross-region replication the source bucket and destination bucket must both have "versioning" enabled. Once versioning is enabled, when objects are modified by users the system continues to store the older version(s) of the object as well as storing the new version. In a cross-region replication context, this means that over time multiple versions of an object may come to be stored in both the source bucket and the destination bucket, with each version counting toward Stored Bytes counts in both buckets.

### Impact on System-Wide Stored Byte Count and Licensed Max Storage Limit

When users use cross-region replication to replicate objects from one HyperStore bucket to another HyperStore bucket, the objects in the source bucket and the object replicas in the destination bucket **both** count toward your system's stored byte count -- the count that is used to determine whether you are in compliance with your licensed maximum storage limit. In this respect bucket-to-bucket cross-region replication is different than storage policy based replication or erasure coding of objects within a region, which is treated as overhead and not counted toward your stored byte count.

## 4.4.1.4. Cross-Region Replication of Encrypted Objects

When cross-region replication is configured for a source bucket:

- Objects encrypted by the regular SSE method **are** replicated to the destination bucket.
- Objects encrypted by the SSE-KMIP, AWS KMS, or SSE-C methods **are not** replicated to the destination bucket.

## 4.4.1.5. Cross-Region Replication and Object Lock (WORM)

HyperStore does not support applying cross-region replication on a **source bucket** that has Object Lock enabled. However, an Object Lock enabled bucket is allowed to be the **destination bucket** in a cross-region replication relationship.

## 4.4.1.6. Cross-System Replication

Conventional cross-region replication replicates data from a source bucket to a destination bucket in a different service region within the same HyperStore system. HyperStore also supports an extension to cross-region replication whereby all objects uploaded to a HyperStore source bucket are replicated to a destination bucket in an external S3 compatible system. This is known as **cross-system replication (CSR)**. The external system can be either of the following:

- A different, independent HyperStore system (with its own user base and service regions and management controls)
- An external third party system with native S3 API support, such as AWS S3

> **Note**
> * Cloudian, Inc. has tested and verified only HyperStore and AWS as cross-system replication destinations. CSR should work for other native S3 destinations, as long as those destination systems 1) fully support the S3 API with respect to putting, listing, and getting versioned objects; and 2) base their ordering of object versions on write-time (exact time at which the object version is put to the system's S3 service by an external client). Any departures from these requirements may result in versioned objects being ordered differently in the replication destination bucket than they are in the source bucket in the local HyperStore system. This may make it difficult to tell, in the destination bucket, which of an object's versions is truly the current version, which is the next-most-current version, and so on. If you want to use CSR with a native S3 destination system other than HyperStore or AWS, Cloudian recommends that you first use CSR for just a small number of objects, and then confirm in the destination bucket that the object version ordering matches the ordering in the source bucket. If you encounter discrepancies, contact Cloudian Support.

> * Neither Microsoft Azure nor Google Cloud Storage are native S3 and so those systems are not supported for cross-system replication.

In most respects cross-system replication works just like cross-region replication, with the same behaviors and limitations described in all the preceding sections of the **Cross-Region Replication Feature Overview**. However, there are additional limitations and caveats specific to cross-system replication:

- For the most part, **cross-system replication does not replicate object ACLs**. This is because with cross-system replication the source system and destination system have different user bases, and so grants of permissions to specific users or groups in the source system cannot be meaningfully applied to the destination system. The **exception** is that if the destination is an **external HyperStore system**, cross-system replication will replicate:
  - Object permissions granted to S3 "predefined groups" (the 'Authenticated Users' group or the 'All Users' group)
  - Standard S3 "canned ACLs" (such as 'authenticated-read' or 'public-read')
- With cross-system replication, **bi-directional replication is not supported**. Do not replicate data from (for example) *bucket1* in your local HyperStore system to *bucket2* in an external system, while data in *bucket2* is also replicated to *bucket1*.
- With cross-system replication, **replication from bucket-to-bucket-to-bucket is not supported**. Do not replicate data from (for example) *bucket1* in your local HyperStore system to *bucket2* in an external system, while data in *bucket2* is also replicated to *bucket3* in any system. Combining this CSR prohibition with the aforementioned prohibition against bi-directional CSR, it can be said that if you set up CSR from a source bucket to a destination bucket in an external system, **the destination bucket should not use CRR or CSR to replicate its data to any bucket**.

In the CMC, cross-system replication is **disabled by default**, such that CMC users who are configuring bucket replication will not be presented with the additional fields necessary to configure cross-system replication (such as the destination endpoint). For information about enabling cross-system replication support in the CMC, see **"Preparing the Cross-Region Replication Feature"** (page 184).

### 4.4.1.7. Replication Error Handling

Objects are replicated to the destination bucket immediately after they are uploaded to the source bucket. The replication request is initiated by whichever S3 Service node processes the upload request for the original object. Errors in replicating an object to the destination bucket are handled as follows:

- If the destination system returns an HTTP 403 or 404 error when HyperStore tries to replicate an object to the destination, this is treated as a permanent error. In the **"Cross-Region Replication request log (cloudian-crr-request-info.log)"** (page 369) on the node that initiates the replication request, an entry for the object replication attempt is written with status FAILED. The system does not retry replicating the object. Examples of scenarios that could result in permanent errors like this are if the destination bucket has been deleted, or if versioning has been disabled on the destination bucket, or if the source bucket owner no longer has write permissions on the destination bucket.
- Any other type of error -- such as the destination region being unreachable due to a network partition, or the destination region returning an HTTP 5xx error -- is treated as temporary. In the Cross-Region Replication request log on the node that initiates the replication request, an entry for the object replication attempt is written with status PENDING. The object replication job will be queued and retried. Retries of object replication jobs with PENDING status are executed by a **system cron job** that runs once every four hours. These retries are executed by the **cron job node**. (To see which node this is in

your cluster, in the CMC's **Cluster Information** page [**Cluster -> Cluster Config -> Cluster Information**] look to see the "System Monitoring / Cronjob Primary Host"). The retries for an object will recur once every four hours until either the object is successfully replicated to the destination bucket or a permanent error is encountered. Each retry attempt results in a new entry in the Cross-Region Replication request log on the cron job node, with a status of either COMPLETED, PENDING, or FAILED.

**IMPORTANT !** If some objects uploaded to a source bucket encounter temporary replication errors, such that those objects are subject to the replication retry cron job, that cron job will also trigger an assessment of the replication status of **all objects in that bucket**. Then, any objects that are not yet replicated will be replicated to the destination bucket -- so that the destination bucket and the source bucket are fully "synced".

A permanent failure for replication of an object applies **only to that object** and does not impact the processing of other objects subsequently uploaded into the same source bucket. The system will continue to replicate -- or attempt to replicate -- other objects that subsequently get uploaded into the bucket. Those objects may also encountered permanent errors, but the system will continue to try to replicate newly uploaded objects unless you disable cross-region replication on the source bucket.

There is no limit on the number of replication retries for a given object or on the number of objects that are queued for retry.

**Note** If an attempt to replicate an object to the destination -- either the original attempt or a retry attempt -- results in a FAILED status in the Cross-Region Replication request log, so that there will be no further retries, this triggers an Alert in the CMC's **Alerts** page. This type of alert falls within the alert rules for S3 service errors (there is not a separate alert rule category for CRR replication failures).

#### 4.4.1.8. Handling of Pre-Existing Objects

Enabling cross-region replication or cross-system replication on a bucket does not cause the replication of objects that were already in the source bucket when cross-region replication or cross-system replication was enabled. However, if you wish you can run a command to trigger the replication of a bucket's pre-existing objects. For details see **"Triggering the Replication of a Bucket's Pre-Existing Objects"** (page 187).

### 4.4.2. Preparing the Cross-Region Replication Feature

HyperStore's cross-region replication feature is enabled by default. Optionally you can modify certain aspects of the feature's implementation as described below.



#### 4.4.2.1. Enabling CMC Support for Cross-System Replication

By default CMC users can configure replication from a source bucket to a destination bucket that's in the same HyperStore system as the source bucket. If you wish you can also give CMC users the option to configure replication from a HyperStore source bucket to a destination bucket in an external system that has native S3 API support (cross-system replication).

**Note** Before enabling CMC support for cross-system replication you should review **"Cross-System Replication"** (page 182) including the limitations and caveats.

To enable CMC support for cross-system replication:

1. Log into the Configuration Master node and in a text editor open the configuration file **common.csv**.
2. Set *cmc_crr_external_enabled* to *true*, then save your change and close the file.
3. Still on the Configuration Master node, use the installer to push your change out to the cluster and restart the CMC service. If you need instructions for this step see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

Once you've made this change, when CMC users **configure cross-region replication for a source bucket** they will be presented with additional fields that allow for replicating to a destination bucket in an external system.

### 4.4.2.2. Setting the Scope of the Replication Retry Cron Job

Once every four hours a HyperStore **cron job** processes any pending cross-region replication jobs. Typically with cross-region replication, objects are replicated to the destination bucket as soon as they are uploaded to the source bucket. Pending cross-region replication tasks result when an attempt to replicate an object from the source bucket to the destination bucket returns a temporary error such as a connection failure or an HTTP 5xx error. The cron job retries the replication of such objects.

By default the cron job simply processes the queue of replication retries, from recent replication tasks that encountered temporary errors. However, if you wish you can configure the system so that each time the cron job runs it also performs a complete sync-up of each bucket for which there are replication tasks in the retry queue. With this complete sync-up, any objects that are in these source buckets but not in their corresponding destination buckets will be replicated to the destination buckets. This includes objects that were in the source buckets before cross-region replication was enabled on the buckets, if any. (The exceptions are objects that were in the source buckets before versioning was enabled on the bucket, if any; and objects that are **encrypted by the SSE-C or AWS-KMS methods**, if any. Such objects will not be replicated.)

If you want the cron job to perform a complete source bucket to destination bucket sync-up for source buckets for which there are pending replication tasks -- rather than simply retrying the pending replication tasks -- do the following:

1. Log into the Configuration Master node and in a text editor open the configuration file **mts.properties.erb**.
2. Set *cloudian.s3.crr.syncbucket* to *true*, then save your change and close the file. (By default this setting is set to *false*.)
3. Still on the Configuration Master node, use the installer to push your change out to the cluster and restart the S3 service. If you need instructions for this step see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**Note** This change in the cron job behavior will not affect replication processing for source buckets for which there are not any pending replication tasks in the retry queue. It applies only to buckets for which there are pending replication tasks in the retry queue. If you want to trigger a replication sync-up for a particular source bucket you can do so as described in **"Triggering the Replication of a Bucket's Pre-Existing Objects"** (page 187).

## 4.4.3. Setting Up Cross-Region Replication for a Bucket

Bucket owners can configure replication from a source bucket to destination bucket. This can be done either through the CMC or through a different S3 client application that invokes the HyperStore implementation of the S3 API.


- **"Setting Up Cross-Region Replication for a Bucket (CMC)"** (page 186)
- **"Setting Up Cross-Region Replication for a Bucket (S3 API)"** (page 186)
- **"Triggering the Replication of a Bucket's Pre-Existing Objects"** (page 187)


### 4.4.3.1. Setting Up Cross-Region Replication for a Bucket (CMC)

**Note** Before configuring cross-region replication on a source bucket, be sure that both the source bucket and the destination bucket have **versioning enabled**. This is required in order to use cross-region replication. For information about enabling versioning on a bucket in HyperStore, while on the CMC's **Buckets Properties** page (**Buckets & Objects -> Buckets -> Bucket Properties**) click **Help**.

In the CMC, bucket owners can enable and configure cross-region replication through the **Buckets & Objects** page. For a given source bucket, cross-region replication is among the options that can be configured as bucket properties. For detail, while on the CMC's **Buckets Properties** page click **Help**. Then in the bucket properties Help, from the Supported Tasks list choose "Configure Cross-Region Replication for a Bucket".Along with specifying a destination bucket to which objects will be replicated from the source bucket, bucket owners can indicate whether they want all newly uploaded objects to be replicated or only objects for which the full object name starts with a particular prefix (such as */profile/images*). If **cross-system replication** is enabled in your system, bucket owners can also specify the destination system endpoint and their security credentials for the destination system.

Alternatively, as a system administrator you can configure cross-region replication for a user's bucket by retrieving the user in the **Manage Users** page and then clicking "View User Data" to open a **Bucket Properties** page for the user's bucket. (This is supported only if you've configured the system to allow system administrators to view and manage users' stored data -- see **"Showing/Hiding CMC UI Functions"** (page 231)).

### 4.4.3.2. Setting Up Cross-Region Replication for a Bucket (S3 API)

S3 applications can call the HyperStore S3 API to configure a source bucket for cross-region replication. For detail see "PutBucketReplication" in the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

Along with supporting conventional cross-region replication, HyperStore's implementation of the *PutBucketReplication* call includes support for extension headers that can be used for cross-system replication. Before using this API extension you should review **"Cross-System Replication"** (page 182) including the limitations and caveats noted in that section.

As with Amazon S3, the HyperStore implementation of the *PutBucketReplication* call requires that **versioning be enabled** on both the source and the destination bucket before cross-region replication can be applied. HyperStore supports the standard S3 *PutBucketVersioning* operation for enabling versioning on a bucket.

HyperStore also supports the *GetBucketReplication* method for retrieving a bucket's current CRR configuration, and the *DeleteBucketReplication* method for disabling CRR on a source bucket for which it is currently enabled. For detail see the S3 API section of the ***Cloudian HyperStore AWS APIs Support Reference***.

#### 4.4.3.3. Triggering the Replication of a Bucket's Pre-Existing Objects

> **Note** If you want to trigger the replication of a bucket's pre-existing objects it's best to do so immediately after cross-region replication has been enabled on the bucket (and before additional objects are uploaded to the bucket and automatically replicated). Triggering sync-up for a bucket after newly uploaded objects have already been replicated to the destination bucket can result in object version sequencing mis-matches between the source bucket and the destination bucket.

Enabling cross-region replication or cross-system replication on a bucket does not cause the replication of objects that were already in the source bucket when cross-region replication or cross-system replication was enabled. However, if you wish **you can trigger replication of the pre-existing objects** in the source bucket by logging into any HyperStore node and running this command:

```
curl -d "" http://localhost:80/.system/syncbucket?bucketName=string
```

For example:

```
curl -d "" http://localhost:80/.system/syncbucket?bucketName=my.bucket
```

This will replicate to the destination bucket any objects in the source bucket that have not already been successfully replicated, including objects that were in the source bucket before you enabled cross-region replication. (The exceptions are objects that were in the source bucket before versioning was enabled on the bucket, if any; and objects that are **encrypted by the SSE-KMIP, AWS KMS, or SSE-C methods**, if any. Such objects will not be replicated.)

# 4.5. Object Lock

## 4.5.1. Object Lock Feature Overview

*Subjects covered in this section:*



HyperStore can implement WORM (Write Once Read Many) protection for stored objects by supporting the standard AWS S3 "Object Lock" functionality. To use the Object Lock feature you must have a HyperStore license that activates this feature. Two different types of HyperStore licensing are available for Object Lock functionality:

- ***Certified* Object Lock**. With this type of Object Lock:
  - HyperStore is fully compliant with the AWS S3 APIs relating to Object Lock, and the safeguards they provide against deletion or modification of locked objects through the S3 API.
  - HyperStore also implements safeguards against deletion or modification of locked objects through means outside of the S3 API. Specifically:

        - To use the Certified Object Lock feature you must disable password-based *root* access to HyperStore nodes and enable the HyperStore Shell (HSH). The HSH allows administrators to log into HyperStore nodesin a manner that is much more restrictive than *root* access.
        - With Certified Object Lock there is no mechanism to delete or modify locked objects through HyperStore administrative commands or the HyperStore Admin API.

    Certified Object Lock is appropriate for HyperStore customers who are subject to the data protection mandates of U.S. SEC-17a or a comparable regulatory regime.

- ***Compatible* Object Lock**. With this type of Object Lock:
    - HyperStore is fully compliant with the AWS S3 APIs relating to Object Lock, and the safeguards they provide against deletion or modification of locked objects through the S3 API. In this respect Compatible Object Lock operates like Certified Object Lock.
    - Unlike Certified Object Lock, with Compatible Object Lock no additional data protection measures are enforced beyond those enforced by the S3 API. Specifically:
        - With Compatible Object Lock you are not required to disable password-based *root* access toHyperStore nodesand you are not required to enable the restrictive HyperStore Shell.
        - With Compatible Object Lock you can use a HyperStore Admin API call to delete all objects from an Object Lock enabled bucket -- including objects that are still within their lock retention period -- and (optionally) to delete the bucket itself. To use this Admin API call on an Object Lock enabled bucket you must obtain a temporary security token from Cloudian Support.

    Compatible Object Lock is appropriate for HyperStore customers who want to utilize the WORM functionality provided by the AWS S3 Object Lock APIs, but who are not subject to the data protection mandates of U.S. SEC-17a or a comparable regulatory regime.

**Note** If in an earlier version of HyperStore you were licensed for Object Lock, upon upgrade to HyperStore version 7.4 you are now licensed for Certified Object Lock (which behaves the same as Object Lock from pre-7.4 versions). If you wish to change your licensed Object Lock type to Compatible Object Lock, contact Cloudian Support.

For more information on applying Object Lock to buckets and objects in HyperStore, see **"Setting Up Object Lock for a Bucket"** (page 190).

For a summary of protections that the S3 API provides to locked objects, see **"Protections for Locked Objects"** (page 195).

For more information on deleting all objects from a locked bucket -- if your licensed Object Lock type is Compatible Object Lock -- see the "bucketops" section in the ***Cloudian HyperStore Admin API Reference***.

#### 4.5.1.1. Object Lock Audit Logging

All S3 client activity pertaining to setting Object Lock attributes on a bucket or on individual objects, and all S3 client attempts to delete locked objects, are logged in an audit log named *s3-worm.log*. For more information about this log see **"S3 Service Logs (including Auto-Tiering, CRR, and WORM)"** (page 365).

If your licensed Object Lock type is Compatible Object Lock and you use the Admin API to purge the content of a locked bucket, log messages regarding that purging operation are written to the Admin Service application

log (*cloudian-admin.log*) and the Admin Service request log (*cloudian-admin-request-info.log*). For more information about these logs see **"Admin Service Logs"** (page 349).

### 4.5.1.2. Object Lock and Bucket Lifecycle Policies

HyperStore does not support applying bucket lifecycle policies for auto-tiering on a **source bucket** that has Object Lock enabled. However, the system does support auto-tiering to a **destination bucket** that has Object Lock enabled.

Also, the system does support applying bucket lifecycle policies for auto-expiration on a source bucket that has Object Lock enabled:

- For a lifecycle policy that specifies an expiration schedule for current version of objects, Object Lock is irrelevant because for any bucket with versioning this type of expiration action only results in creation of a delete marker (and does not actually delete any object data from storage).
- For a lifecycle policy that specifies an expiration schedule non-current object versions, when a non-current object version reaches its expiration date the system checks for Object Lock and does not delete the non-current object version if it is still within its lock retention period. When the non-current object version's lock retention period ends, then the system deletes it at the next running of the auto-expiration job.

### 4.5.1.3. Object Lock and Cross-Region Replication

HyperStore does not support applying cross-region replication on a **source bucket** that has Object Lock enabled. However, an Object Lock enabled bucket is allowed to be the **destination bucket** in a cross-region replication relationship.

### 4.5.1.4. Object Lock and S3 Notifications

In the current release, HyperStore does not support S3 Notifications in regard to Object Lock related events.

### 4.5.1.5. Object Lock and User Deletion

The system will reject (with an HTTP 412 error response) any attempt to delete a user who currently owns a bucket that has Object Lock enabled:

- Before such a user can be deleted, the user's Object Lock enabled bucket(s) must be deleted.
- Before the user's Object Lock enabled bucket(s) can be deleted, all objects in those bucket(s) must be deleted. Depending on the type of Object Lock configuration used on those buckets and objects, it may be that the objects cannot be deleted until after the end of their retention period. For a summary of restrictions on deletion of "locked" objects through the S3 API, see **"Protections for Locked Objects"** (page 195).

  **Note** If your licensed Object Lock type is Compatible Object Lock, you can delete a locked bucket and its contents through the Admin API after obtaining a temporary security token from Cloudian Support. See the "bucketops" section of the ***Cloudian HyperStore Admin API Reference***.

## 4.5.2. Preparing the Object Lock Feature

Before Object Lock can be set up on buckets and objects your HyperStore system must meet these pre-requisites:

- **You must have a HyperStore license that supports the Object Lock feature**. To check whether your license supports this feature, in the CMC's **Cluster Information**page see the "Object Lock License" field: it will indicate either "Certified" (if your licensed Object Lock type is Certified Object Lock) or "Compatible" (if your licensed Object Lock type is Compatible Object Lock) or "Disabled" (if your license does not support Object Lock). If you want to use the Object Lock feature but it is disabled by your current license, contact your Cloudian representative to inquire about obtaining a different license. For a description of how Certified Object Lock differs from Compatible Object Lock, see **"Object Lock Feature Overview"** (page 187).
- **If your licensed Object Lock type is Certified Object Lock, you must enable the HyperStore Shell and disable the *root* account password** on your HyperStore hosts. For instructions see:
  - **"Enabling the HSH"** (page 96)
  - **"Disabling the root Password"** (page 99)

  These actions must be completed before Object Lock enabled buckets can be created. If your licensed Object Lock type is Certified Object Lock and you have not yet enabled the HyperStore Shell and disabled the root account password:
  - A warning will display in the "Object Lock License" field in the CMC's **Cluster Information**page
  - WARNING level log messages will be written to the Admin Service log *cloudian-admin.log*
  - Any attempts to create an Object Lock enabled bucket through the CMC or directly through the S3 API will fail and return an error response.

  **Note** If your licensed Object Lock type is **Compatible Object Lock**, the requirement to enable the HyperStore Shell and disable the *root* account password does not apply.

## 4.5.3. Setting Up Object Lock for a Bucket

*Subjects covered in this section:*



### 4.5.3.1. Setting Up Object Lock on a Bucket (S3 API or CMC)

With a third party S3 application that supports the S3 APIs for object locking, or with the CMC, HyperStore users with appropriate permissions can:

1. Create a new bucket with Object Lock enabled.
2. Optionally, set a default Object Lock configuration for that bucket.
3. Check a bucket's Object Lock configuration status.

The sections that follow provide more information about each of these steps.

**Note** Object Lock can only be enabled on a new bucket, as the bucket is created. **Object Lock cannot be enabled on an already existing bucket.**

### Creating a New Bucket with Object Lock Enabled

To create a new bucket with Object Lock enabled:

- **With a third party S3 client application**, the client application submits a ***CreateBucket*** request that includes the request header ***x-amz-object-lock-enabled: true***. For HyperStore support of the S3 *CreateBucket* operation, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.
- **With the CMC**, as a user creates a new bucket in the CMC's **Add Bucket** interface she can select a checkbox to enable Object Lock for the bucket. For more information, while on the CMC's **Buckets** page (**Buckets & Objects -> Buckets**) click **Help**.

Note that:

- Enabling Object Lock on a new bucket (either through a third party S3 client or through the CMC) **automatically enables Versioning** on that bucket as well. Object Lock can only be used in combination with Versioning (which protects objects from being overwritten).
- If you create a bucket with Object Lock enabled, you **cannot disable Object Lock or suspend versioning** for the bucket (even if you never apply a default Object Lock configuration to the bucket).
- Enabling Object Lock on a new bucket **does not by itself have the effect of locking objects that are subsequently uploaded** into that bucket. It only makes it possible to lock such objects, using the methods described below.

### Setting a Default Object Lock Configuration for the Bucket (Optional)

Once a new bucket has been created with Object Lock enabled, a default Object Lock configuration can optionally be set on the bucket. This Object Lock configuration will then by default be applied to all objects that are subsequently added to the bucket (it does **not** apply to any objects that were already in the bucket at the time that the default Object Lock configuration was set for the bucket). The default configuration can be overridden on a per-object basis, as described in **"Setting Object Lock Attributes on Individual Objects in a Bucket (S3 API or CMC)"** (page 192). If a default Object Lock configuration is not set on the bucket, then objects uploaded to the bucket will not be locked unless they have Object Lock attributes set on them on a per-object basis.

To set a default Object Lock configuration on the bucket:

- **With a third party S3 client application**, the bucket owner (or a user with *s3:PutBucketObjectLockConfiguration* permission on the bucket) submits a ***PutObjectLockConfiguration*** request to the HyperStore S3 Service. For HyperStore support of this S3 API operation see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***. Along with specifying a retention period in days or years, the request specifies whether the bucket's default Object Lock implementation is to be in **Governance** mode (which allows users who have the applicable permissions to change the object retention period or delete objects through the S3 API before their retention period completes) or **Compliance** mode (which does not allow any user to change the object retention period or delete objects through the S3 API before their retention period completes).
- **With the CMC**, the bucket owner can use the Object Lock tab of the **Bucket Properties** interface to set a default Object Lock configuration on the bucket. For more information, while on the CMC's **Bucket Properties** page (**Buckets & Objects -> Buckets -> Properties**) click **Help**.

With a default Object Lock configuration set on the bucket, **from that point forward whenever an object is uploaded to the bucket -- and lacks object-specific lock attributes -- the default retention period is applied to that object**. For example with a 90 day default retention period, every object is locked for 90 days starting from the time of object upload. For objects with multiple versions, in this example each object version is locked for 90 days from time of object version upload.

When an object version is locked, HyperStore rejects an S3 *DeleteObject* request for that object version with an HTTP 403 Forbidden response. The exception is if Governance mode is being used and the requesting user is a user who has the applicable permissions; for more information see **"Protections for Locked Objects"** (page 195).

A third party S3 client application can **change or remove the default Object Lock configuration** on a bucket by submitting another *PutObjectLockConfiguration* request; or in the CMC a bucket's default Object Lock configuration can be changed or removed through the **Bucket Properties** interface. However, the change applies only from that time forward, to objects that are subsequently added to the bucket. Locks that are already in place on existing objects -- in accordance with the prior default configuration on the bucket -- are not impacted.

**Note**

- Object Lock protects each version of an object individually and does not prevent the creation of new versions of the object.
- For a locked object, an S3 *DeleteObject* request that does not specify an object version is allowed and only results in the creation of a delete marker for that object. No object data is actually deleted from storage.
- Once a bucket is assigned a default lock configuration, any S3 requests for uploading objects to that bucket must use Signature Version 4 request authentication. This is consistent with Amazon's Object Lock requirements.

### Checking a Bucket's Object Lock Status

To check a bucket's current Object Lock status:

- **With a third party S3 client application**, the bucket owner (or a user with *s3:GetBucketObjectLockConfiguration* permission on the bucket) can submit a ***GetObjectLockConfiguration*** request to the HyperStore S3 Service. For HyperStore support of this S3 operation see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***. The request response will indicate whether Object Lock is enabled on the bucket, and what the default Object Lock configuration is for the bucket (if any).
- **With the CMC**, in the bucket owner can view the list of buckets she owns and any buckets for which Object Lock is enabled will have a padlock icon beside the bucket name. The bucket owner can then view the **Bucket Properties** interface for the bucket and check the Object Lock tab to see what the default Object Lock configuration is for the bucket (if any).

## 4.5.3.2. Setting Object Lock Attributes on Individual Objects in a Bucket (S3 API or CMC)

In a **bucket that has Object Lock enabled**, there is the option to set Object Lock attributes on individual objects. If the bucket has a default Object Lock configuration, then setting Object Lock attributes on individual objects will -- for those objects only -- override the bucket's default Object Lock configuration. If the bucket does not have a default Object Lock configuration, then setting Object Lock attributes on individual objects is the only way that objects will become locked.

**With a third party S3 application**, HyperStore users with appropriate permissions can:

- Set Object Lock attributes on objects as they are uploaded to the bucket.
- Set Object Lock attributes on existing objects that are already in the bucket.
- Check an object's lock status.

**With the CMC**, the bucket owner can:

- Set Object Lock attributes on existing objects that are already in the bucket.
- Check an object's lock status.

> **Note** The CMC does not currently support setting Object Lock attributes on objects as they are uploaded to the bucket. Instead the user can upload the object to the bucket, and then set Object Lock attributes on the object.

The sections that follow provide more information about these options.

### Setting Object Lock Attributes on Objects as They Are Uploaded

**With a third party S3 application**, objects being uploaded to on Object Lock enabled bucket can be assigned Object Lock attributes by the inclusion of the request headers ***x-amz-object-lock-mode***, ***x-amz-object-lock-retain-until-date***, and/or ***x-amz-object-lock-legal-hold*** with any of the standard S3 API requests for uploading objects:

- *PutObject*
- *CopyObject*
- *POST Object*
- *CreateMultipartUpload*

For HyperStore support of these S3 operations see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

Similarly to the bucket default configuration options, the individual Object Lock configuration options include the choice between **Governance** retention mode and **Compliance** retention mode (as set by the *x-amz-object-lock-mode* request header).

Unlike the bucket default configuration options, the individual object configuration attributes also include an option for **Legal Hold** (as optionally set by the *x-amz-object-lock-legal-hold* request header). Legal Hold prevents the object from being deleted by any user through the S3 API, for an indefinite period of time until the Legal Hold is explicitly removed from the object (by a user who has the applicable permissions). Legal Hold can be used instead of having a defined retention date for the object, or in combination with having a defined retention date for the object. For example, if both a Governance retention date and a Legal Hold are set on an object, and then the Legal Hold is removed before the retention date, the object will continue to be protected by Governance mode retention until its retention date is reached. For a second example, if both a Compliance retention date and a Legal Hold are placed on an object, and the Compliance retention date is reached while the Legal Hold is still in place, the object continues to be protected by the Legal Hold until the Legal Hold is explicitly removed.

> **Note**
> * Setting retention attributes on an object as the object is uploaded can be done by the bucket owner, or by a user who has *s3:PutObject* and *s3:PutObjectRetention* permissions on the bucket. Setting a legal hold on an object as the object is uploaded can be done by the bucket owner or by a user who

has *s3:PutObject* and *s3:PutObjectLegalHold* permissions on the bucket.
* Object upload requests that include Object Lock headers must use Signature Version 4 request authentication. This is consistent with Amazon's Object Lock requirements.
* The CMC does not support setting Object Lock attributes on objects as they are uploaded.

### Setting Object Lock Attributes on Existing Objects

- **With a third party S3 application**, existing objects in an Object Lock enabled bucket can be assigned Object Lock attributes by using the standard S3 requests ***PutObjectRetention*** and/or ***PutObjectLegalHold***.For HyperStore support of these S3 operations see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

  **Note** Setting retention attributes on an existing object can be done by the bucket owner or by a user who has *s3:PutObjectRetention* permission on the object. Setting or removing a legal hold on an existing object can be done by the bucket owner or by a user who has *s3:PutObjectLegalHold* permission on the object.

  For an existing object that already has retention attributes, the *PutObjectRetention* request can be used to increase the existing retention period on the object but not to reduce the existing retention period (unless Governance mode retention is being used and the requesting user is a user who has the applicable permissions, as described in **"Protections for Locked Objects"** (page 195)).

- **With the CMC**, the bucket owner can assign Object Lock attributes to an existing object in the bucket by accessing the object's **Object Properties** interface and using the Object Lock tab. For more information, while logged into the CMC's **Bucket Properties** page (**Buckets & Objects -> Buckets -> Properties**) click **Help**.

### Checking an Object's Lock Status

- **With a third party S3 application**, checking an object's lock status can be done by using the standard S3 requests ***GetObject***, ***HeadObject***, ***GetObjectRetention***, and/or ***GetObjectLegalHold***. For HyperStore support of these S3 operations see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

  An object's lock status will reflect the Object Lock attributes that were set directly on that individual object (if any), or otherwise will reflect the object's inheriting of the bucket's default Object Lock configuration (if any).

  **Note** Checking an object's lock status can be done by the bucket owner or by a user who has *s3:GetObject*, *s3:GetObjectVersion*, *s3:GetObjectRetention* (for retention status), and *s3:GetObjectLegalHold* (for legal hold status) permissions on the object.

- **With the CMC**, the bucket owner can view the list of objects in her bucket, and locked objects will have a padlock icon beside the version ID of each version of the object. The bucket owner can then view an object version's **Object Properties** interface and check the Object Lock tab to see what the lock attributes are for that object version.

### 4.5.4. Protections for Locked Objects

The table below shows the key differences between Governance mode retention, Compliance mode retention, and Legal Hold in terms of how they **protect locked objects against premature deletion through the S3 API**. Recall that for all forms of Object Lock, the lock is applied to each individual version of an object.

| | Governance Mode Retention | Compliance Mode Retention | Legal Hold |
|---|---|---|---|
| Delete a locked object version? | The bucket owner and users who have been granted both *s3:DeleteObjectVersion* and *s3:BypassGovernanceRetention* permission can use the *DeleteObject* request with an *x-amz-bypass-governance-retention: true* request header to delete a locked object version. | No user can delete a locked object version | No user can delete a locked object version |
| Remove the lock on an object version? | The bucket owner and users who have been granted both *s3:PutObjectRetention* and *s3:BypassGovernanceRetention* permission can use the *PutObjectRetention* request with an *x-amz-bypass-governance-retention: true* request header to remove the retention lock on a object version. | No user can remove the retention lock on an object version | The bucket owner and users who have been granted *s3:PutObjectLegalHold* permission can use the *PutObjectLegalHold* request to remove the legal hold on an object version. |
| Reduce the retention period for an object version? | The bucket owner and users who have been granted both *s3:PutObjectRetention* and *s3:BypassGovernanceRetention* permission can use the *PutObjectRetention* request with an *x-amz-bypass-governance-retention: true* request header to reduce the retention period for a locked object version. | No user can reduce the retention period for an object version | The bucket owner and users who have been granted *s3:PutObjectLegalHold* permission can use the *PutObjectLegalHold* request to remove the legal hold on an object version. |

**Note** In the CMC, no users other than the bucket owner can access a bucket or the objects that it contains. So in the table above, the statements about users who have been granted various permissions relating to locked objects apply only to users who are using third party S3 client applications -- not the CMC.

#### 4.5.4.1. Using the Admin API to Purge All Objects from a Locked Bucket

If your licensed Object Lock type is **Compatible Object Lock**, you can use a HyperStore Admin API call to purge all of the objects in a locked bucket. Note that:

- This operation deletes **all** objects from the bucket, regardless of whether the objects are protected by Governance mode, Compliance mode, or Legal Hold.

- There is no option to selectively delete objects from the bucket -- the operation only supports deleting **all** versions of **all** objects.
- The operation allows you (optionally) to delete the bucket itself, as well as all of the objects in it.
- The operation requires that you obtain a temporary security token from Cloudian Support.
- The operation is not allowed if your licensed Object Lock type is Certified Object Lock.

For more information see the "bucketops" section in the Cloudian HyperStore Admin API Reference.

# 4.6. Object Metadata and Search

## 4.6.1. Object Metadata and Search Feature Overview

*Subjects covered in this section:*



One of the advantages of S3-compliant object storage is the ability to associate rich metadata with each object, including metadata defined by users. HyperStore supports the creation of user-defined object metadata through standard S3 API calls. HyperStore also supports the ability to search for objects based on their metadata.

### 4.6.1.1. System-Defined and User-Defined Object Metadata

There are two types of S3-compliant object metadata:

- System-defined object metadata -- The HyperStore system automatically defines and sets a variety of object metadata having to do with object attributes and status
- User-defined object metadata -- S3 application users can define and set object metadata by using *x-amz-meta-** request headers when objects are uploaded

#### System-Defined Object Metadata

For each object, HyperStore maintains object metadata including (but not limited to) the following:

- Creation time
- Last modified time
- Last accessed time
- Size
- ACL information
- Version, if applicable
- Public URL, if applicable
- Compression type, if applicable

- Encryption key, if applicable
- Auto-tiering state, if applicable

### User-Defined Object Metadata

S3 user applications can define and set object metadata by including *x-amz-meta-** request headers in the object upload request. The application user defines both the metadata key and its value. For example if metadata identifying a document's author and reviewer was desired, this could be achieved with request headers such as *x-amz-meta-author: hernandez* and *x-amz-meta-reviewer: schmidt*. This is a standard S3 API feature that is supported by HyperStore's S3 Service. For more information see **"Creating and Retrieving User-Defined Object Metadata and Tags"** (page 200).

The maximum size limit for *x-amz-meta-** based metadata -- the maximum allowed total size of all *x-amz-meta-** header names and header values combined -- is 2KB per object.

## 4.6.1.2. Object Metadata Based Search

A HyperStore system can optionally support the ability to search for objects based on the objects' metadata. This metadata search capability is provided by the HyperStore Search Service, an optional HyperStore system component. The HyperStore Search Service is designed to run on "auxiliary" nodes that are closely integrated with your HyperStore cluster, but that have lesser resource requirements than the rest of your HyperStore nodes. After you've installed a HyperStore cluster you can use the HyperStore installer to manage the installation of the HyperStore Search Service.

Once the HyperStore Search Service is installed and running, you can enable metadata search on a per storage policy basis. For each bucket that uses a search-enabled storage policy, HyperStore will trigger the creation of an object metadata index in the HyperStore Search Service. As the population of objects in a bucket changes -- as objects are uploaded, overwritten, or deleted -- HyperStore will update the bucket's object metadata index in the HyperStore Search Service. The creation and updating of object metadata indexes in the HyperStore Search Service is triggered by an hourly cron job that runs on the HyperStore cron job host. The cron job also triggers the deletion of an index if the corresponding bucket is deleted from the HyperStore system.

You can monitor the status of your HyperStore Search nodes in the CMC's **Data Centers** page, and optionally you can use the **Alert Rules** page to configure alerts based on HyperStore Search Service status.

An end user who owns a bucket that uses a search-enabled storage policy can use the CMC's **Search** page to search for objects in that bucket based on any of the following object metadata:

- Object name string
- Object size
- Object creation date
- User-defined object metadata (metadata set in *x-amz-meta-** headers during object upload)

In regard to HyperStore's transmission of object metadata to the HyperStore Search Service, note that:

- All object metadata continues to exist **in the HyperStore Metadata DB**, with the durability protections it affords through replication across the HyperStore cluster. When object metadata is transmitted to HyperStore Search nodes for indexing, it resides on the HyperStore Search nodes **in addition** to residing in the HyperStore Metadata DB.
- As activity within a bucket occurs -- such as object uploads, downloads, or deletions -- the bucket's object metadata in the HyperStore Search Service is **not updated in real time**. Instead, the updates are

executed by the hourly cron job. When a user uploads a new object, there will be a delay before the object becomes findable by a metadata search (the delay until the next run of the hourly cron job).

Note also that the HyperStore Search Service only supports finding objects by reference to their metadata (metadata search):

- It **does not support finding objects by reference to their** object tags
- It **does not support searching through the content of objects** (content search)

For more information on setting up the HyperStore Search Service, see **"Preparing the Metadata Search Feature"** (page 202).

### Options if You Have a Legacy Integration with Elasticsearch

If you have a legacy integration with Elasticsearch from an earlier version of HyperStore, after you upgrade to HyperStore 7.5.1 your integration with Elasticsearch will no longer work as is. You have three options:

- Stop using your legacy Elasticsearch system (and the metadata that's stored in it), and start a fresh metadata search system with the new HyperStore Search Service.
- Migrate your elastic Elasticsearch system (and the metadata that's stored in it) to a new HyperStore Search Service deployment. This requires that your Elasticsearch system is version 7.10.2.
- Continue to use Elasticsearch for a while longer. This requires making configuration changes to your HyperStore 7.5.1 system so that it will integrate with Elasticsearch. Note however that **support for Elasticsearch integration will end in HyperStore version 8.0.**

Consult with Cloudian Support about which option you prefer and how to implement that option.

## 4.6.1.3. Object Tags

Object tags -- user-defined key-value pairs that are associated with an object, such as *department=engineering* or *status=complete* -- are similar to object metadata in that they encapsulate information about an object. However, from the perspective of AWS S3, object tags are distinct from object metadata because they are created differently and typically serve different purposes.

Object tags can be attached to an object as it's being uploaded, by the use of a single single *x-amz-tagging* request header which supports having a multi-segment value. But more commonly -- and differently from object metadata -- object tags can be attached to an object that's already in storage, by the use of the S3 API call *PutObjectTagging* that creates a tagging sub-resource for a specified existing object. Further, an object's tags can be updated at any time by that same S3 API call. (By contrast, an object's *x-amz-meta-** based object metadata cannot be modified, except by making a copy of the object and specifying different metadata during the copy operation.)

Object tagging is a standard S3 API feature that is supported by HyperStore's S3 Service. For more information see **"Creating and Retrieving User-Defined Object Metadata and Tags"** (page 200).

Another thing that distinguishes object tags from *x-amz-meta-** based object metadata is that object tags can serve as conditions in a bucket permissions policy or IAM permissions policy, or as filtering criteria in a bucket lifecycle policy. The CMC does not currently support this capability, but HyperStore's S3 API and IAM API do.

Object tagging is subject to these restrictions:

- For each object, each tag key must be unique, and each tag key can have only one value.
- Maximum number of tags per object – 10

- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8

**Note** The HyperStore Search Service (described in **"Object Metadata Based Search"** (page 197) above) does not support finding objects based on object tags.

#### 4.6.1.4. Object Metadata Storage in the Metadata DB

##### Object Metadata and Storage Policies

HyperStore object metadata and object tags are stored in the Metadata DB, and are protected by replication. The degree of replication depends on the type of storage policy being used. For more information see **"Metadata Replication"** (page 132).

##### Object Metadata Structure in the Metadata DB

Object metadata is stored in two different types of records (rows) in the Metadata DB:

- One record for each object (called "skinny row" object metadata because along with the key the row has just one column, for that one object)
- For each bucket, multiple records that in sum include metadata for all the objects in the bucket (called "wide row" object metadata because along with the key the rows have potentially very many columns, with one column per object)

These two different types of object metadata records -- object-level records and bucket-level records -- are used for different purposes within HyperStore. For example, while processing an S3 *GetObject* request or *HeadObject* request the system will retrieve the object-level record. While processing a *ListObjects* request for a bucket the system will retrieve bucket-level records. Bucket-level records are also used during batch delete operations and during some *hsstool* operations.

##### Partitioning of a Bucket's Object Metadata Into Multiple Records

HyperStore's approach to creating and utilizing bucket-level metadata records ("wide row" object metadata) has evolved across HyperStore versions:

- For buckets created in HyperStore 7.0.x and older, for each bucket a single bucket-level metadata record was created. This resulted in *ListObjects* performance issues in certain circumstances, particularly in cases where a bucket contained a very large number of objects and was subject to large numbers of concurrent deletes.
- For buckets created in HyperStore 7.1.x, 7.2.x, or 7.3.x, for each bucket multiple bucket-level metadata records were created, based in a single common algorithm that determined how many records to create per bucket. This improved *ListObjects* performance in most instances, but still resulted in sub-optimal performance in some cases -- such as buckets in which very large numbers of objects were stored in a single "directory" or buckets that had a very large number of directories each of which stored only a small number of objects.
- For buckets created in HyperStore 7.4 and later, for each bucket multiple bucket-level metadata records are created, using varying algorithms that are tailored to the workload type associated with the bucket (such as workloads associated with popular backup applications, each of which produces a distinctive directory structure in its storage target). In HyperStore 7.4 and later, when you create a storage policy you select a "workload type" for the storage policy (for more information, while on the CMC's **Storage Policies** page [**Cluster -> Storage Policies**] click **Help**). And then when buckets are created that use

that storage policy, for those buckets HyperStore uses a bucket metadata partitioning scheme that is optimized for that workload type. By using different bucket metadata partitioning schemes for different workload types, HyperStore is able to optimize *ListObjects* performance across a range of bucket workload types and their associated directory structures.

By default, HyperStore's feature that tailors bucket metadata partitioning to a bucket's workload type works **only for buckets created in HyperStore 7.4 and later**. If you have a bucket created in HyperStore 7.3.x or earlier that is experiencing poor S3 *ListObjects* performance, contact Cloudian Support. Cloudian Support can work with you to upgrade that bucket to use the optimized bucket metadata partitioning feature.

#### 4.6.1.5. Cross-Region Replication and Auto-Tiering of Object Metadata and Tags

When the **cross-region replication** feature replicates objects from a local source bucket to a bucket in a remote HyperStore region or system -- or a different S3 compliant destination system -- user-defined metadata (in the form of *x-amz-meta-** headers) and object tags are replicated along with the object data.

When the **auto-tiering** feature transitions objects from a HyperStore bucket to an external destination system, user-defined metadata (in the form of *x-amz-meta-** headers) and object tags are sent along with the object data if the tiering destination is a native S3 system such as Amazon Web Services. Also, the user-defined object metadata and tags associated with auto-tiered objects continue to be retained in the local HyperStore system as well (and if the HyperStore bucket uses a **search-enabled storage policy**, object metadata is also retained in the HyperStore Search Service).

If the tiering destination is not S3-compliant, such as Microsoft Azure or Spectra Pearl Logic, then the user-defined object metadata and object tags are **not** sent along with the object data. The user-defined object metadata and object tags will exist only in the HyperStore system, even after the corresponding object data is transitioned to the destination system.

### 4.6.2. Creating and Retrieving User-Defined Object Metadata and Tags

*Subjects covered in this section:*



**Note** The CMC's built-in S3 client does not support creating or retrieving user-defined object metadata or object tags. To create or retrieve user-defined object metadata or object tags **you will need to use a third party S3 client application.**

#### 4.6.2.1. Creating User-Defined Object Metadata

When uploading a new object, S3 client applications can create user-defined object metadata by including one or more *x-amz-meta-** request headers along with the *PutObject* request. For example, *x-amz-meta-topic: merger* or *x-amz-meta-status: draft*.

The S3 API method *POST Object* — for uploading objects via HTML forms — also allows for the specification of user-defined metadata (through *x-amz-meta-** form fields), and HyperStore supports this method as well.

Once uploaded along with an object, user-defined object metadata in the form of *x-amz-meta-** headers cannot be changed. The only way to associate different user-defined metadata with an object that has already been uploaded is to create a new copy of the object through the S3 *CopyObject* operation, and in that operation specify the desired *x-amz-meta-** headers.

For HyperStore support of these S3 API operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

### 4.6.2.2. Retrieving Object Metadata

S3 client applications can retrieve user-defined *x-amz-meta-** based object metadata -- as well as system-defined object metadata -- by using either of two standard S3 API methods, both of which the HyperStore system supports:

- *GetObject* returns this type of object metadata as response headers, as well as returning the object itself.
- *HeadObject* returns this type of object metadata as response headers, without returning the object itself.

For HyperStore support of these S3 API operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

### 4.6.2.3. Creating Object Tags

When uploading a new object, S3 client applications can create one or more user-defined object tags by including a single *x-amz-tagging* request header in the *PutObject* request. For example, *x-amz-tagging: team-m=Marketing&project=SEO&status=Needs-Review*.

S3 clients can also create object tags when calling the *POST Object* method, by using the *tagging* form field.

The *CopyObject* method supports copying or replacing an object's tags as it is copied, by making use of the *x-amz-tagging-directive* and *x-amz-tagging* request headers.

S3 clients can also create or update object tags for an object that is already stored in the HyperStore system, by using the S3 API method *PutObjectTagging*. This method creates the object tags as a sub-resource of the object, and allows a maximum of 10 key-value pairs.

For a high-level view of object tagging usage and methods, in the Amazon S3 online documentation see **Object Tagging**.

For HyperStore support of these S3 API operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

Object tagging is subject to these restrictions:

- For each object, each tag key must be unique, and each tag key can have only one value.
- Maximum number of tags per object – 10
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8

### 4.6.2.4. Retrieving Object Tags

The S3 methods *GetObject* and *HeadObject* do not return object tags. They do however return a count of the object tags associated with the object (if any), in an *x-amz-tagging-count* response header.

To retrieve the object tags associated with an object, use the S3 API method *GetObjectTagging*.

For a high-level view of object tagging usage and methods, in the Amazon S3 online documentation see **Object Tagging**.

For HyperStore support of these S3 API operations, see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

## 4.6.3. Preparing the Metadata Search Feature

*Subjects covered in this section:*



This section outlines at a high level how to prepare the object metadata search feature so that it is available to your end users. For some preparation tasks, this section provides just an overview of the task, along with a reference to more detailed instructions elsewhere in the HyperStore documentation.

> **Note** For an introduction to the metadata search feature, see **"Object Metadata Based Search"** (page 197).

### 4.6.3.1. Installing the HyperStore Search Service

The HyperStore Search Service is an optional HyperStore component that you can add to an existing HyperStore system so that the system will support object metadata search. If you are installing a new HyperStore system and you know from the outset that you want to use the metadata search feature, you will complete the regular HyperStore installation first, and verify that the system is operating properly, and then add the HyperStore Search Service as a separate procedure. If you are already using a HyperStore system in production, you can add the HyperStore Search Service to the system without taking the system offline (except for a restart of the S3 Service to apply the change). In either scenario you will use the HyperStore installer -- on your HyperStore Configuration Master node -- to execute the HyperStore Search Service installation.

The HyperStore Search Service is designed to run on virtual machines (VMs) -- not on the hosts on which you run the rest of your HyperStore services. One host VM is sufficient for HyperStore Search testing or initial evaluation, but three host VMs are required for production environments.

HyperStore Search host requirements and detailed instructions for installing the HyperStore Search Service are provided in the *HyperStore Installation Guide*. See Chapter 5 "Installing Search and/or File Service Nodes".

### 4.6.3.2. Configuring Alerts Based on HyperStore Search Service Status

You can monitor the status of your HyperStore Search nodes in the CMC's **Data Centers** page, and optionally you can use the **Alert Rules** page to configure alerts based on HyperStore Search Service status. Alerts based on HyperStore Search Service status are not enabled by default.

For details about status monitoring, while on the CMC's **Data Centers** page (**Cluster -> Data Centers**) click **Help**.

For details about setting alert rules, while on the CMC's **Alert Rules** page (**Alerts -> Alert Rules**) click **Help**.

### 4.6.3.3. Enabling Metadata Search for a Storage Policy

Once the HyperStore Search Service is installed, you can use the CMC's **Storage Policies** page to enable metadata search for individual storage policies. You can enable metadata search for a new storage policy as you create the storage policy, or you can edit any existing storage policy to enable metadata search for it.

Note that:

- The "Enable Metadata Search" option for individual storage policies will display in the CMC **Storage Policies** page only after you've installed the HyperStore Search Service. If you have not installed the HyperStore Search Service, no such option will display in the CMC.
- Even after you've installed the HyperStore Search Service, **by default metadata search will be disabled on each storage policy**. The system does not support a single setting to enable metadata search for all of your storage policies. Instead you must explicitly enable metadata search for each individual storage policy that you want to support metadata search.
- When you enable metadata search on a storage policy, then all buckets that use that storage policy will support object metadata search.
- If you enable metadata search on a storage policy, make a note of the service region in which the storage policy resides (this information is displayed on the CMC's **Storage Policies** page). You will need to know this when you are **"Triggering the Indexing Of Already-Existing Buckets and Objects"** (page 203).
- If you do not enable metadata search on a given storage policy, then buckets that use that storage policy will not support object metadata search.

**IMPORTANT !** Once you enable search for a storage policy, the system will automatically (by an hourly cron job) index only buckets that are created **after** you've enabled search for the storage policy and only objects uploaded **after** you've enabled search for the storage policy. To index buckets and objects that already existed when you enabled search for the storage policy you must manually trigger the indexing as described in **"Triggering the Indexing Of Already-Existing Buckets and Objects"** (page 203) below.

For more information about creating and editing storage policies, while on the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) click **Help**.

### 4.6.3.4. Triggering the Indexing Of Already-Existing Buckets and Objects

After enabling metadata search on a storage policy, to index already-existing buckets that use that storage policy (and already-existing objects in those buckets) you must do the following:

1. As *root*, log into any HyperStore node in the service region in which the storage policy is based. Log into a HyperStore core node, not an auxiliary node on which the Search service runs).

   **Note** As an alternative to using root you can log in as a **HyperStore Shell** user.

2. Run this command:

```
search-mgr.sh sync-indices
```

This will index all existing objects in all existing buckets that use any search-enabled storage policy in the region.

Note that:

- If you enable search on multiple storage policies in the region at the same time, you only need to run the above command once and this will index all existing buckets and objects that use any of those storage policies.
- If you enable search on some storage policies and then run the above command, and then later you enable search on another storage policy, you will need to run the command again to index existing buckets and objects from that storage policy. **Any time you enable search on an additional storage policy, run the command** if you want to index existing buckets and objects from that storage policy.
- If you enable search on storage policies in multiple service regions, you must **run the command in each region**. For example if you enable search on a storage policy in Region1 and you also enable search on a storage policy in Region2, you must run the command from a HyperStore node in Region1 and then also run the command from a HyperStore node in Region2.

Your running of the *search-mgr.sh sync-indices* command is required only for indexing already-existing buckets and objects. Once you've enabled search on a storage policy, then subsequently the indexing of newly created buckets that use that storage policy, and the indexing of objects that are newly uploaded to buckets that use that storage policy, occurs automatically. This automatic indexing is implemented by an hourly cron job.

If you want to check whether a particular bucket has been indexed you can run the following command:

```
search-mgr.sh list-indices
```

This returns a list of all indexes for the local service region. There is one index per bucket, with this index naming format:

*cloudian-<search-prefix>++<bucketname>*

For example:

*cloudian-boston++pubs.bucket1*

The *<search-prefix>* is the search prefix for your HyperStore system, as you configured during HyperStore File Service installation.

**Note** The *search-mgr.sh list-indices* command also returns a list security audit logs -- this is expected behavior. For more information about the *search-mgr.sh* tool see **"Using the search-mgr.sh Tool For On-Demand Indexing"** (page 205).

### 4.6.3.5. How End Users Access the Search Feature

In the CMC, a user who owns a bucket that uses a search-enabled storage policy can use the **Search** page to search for objects based on any of the following object metadata:

- Object name string
- Object size
- Object creation date
- User-defined object metadata (metadata set in *x-amz-meta-** headers during object upload).

> **Note** As stated in **"Creating and Retrieving User-Defined Object Metadata and Tags"** (page 200), user-defined object metadata cannot be created in the CMC. A third party S3 client application must be used to create such metadata.

In the **Search** page a user will first select the bucket in which they want to search, and then enter the search criteria in a visual query editor (an option to enter queries in a JMX editor is also presented).

A CMC user can search only in buckets that they own, and not in any other users' buckets.

If none of a user's buckets use a storage policy for which you've enabled metadata search, and the user tries to use the CMC's **Search** page, a message will display indicating that the user cannot use the **Search** page because none of their buckets use a search-enabled storage policy.

Note also that:

- Search is not supported for buckets that have bucket names exceeding 50 characters.
- Search is not supported for objects that have object names (including full path) exceeding 200 characters.

For system administrators, note that:

- The CMC **Search** page does not display if you are logged into the CMC as a system administrator. The **Search** page displays only for group admins and regular users. Put differently, the **Search** page displays only for users who have S3 accounts.
- Even if you have *common.csv*: **"cmc_view_user_data"** (page 424), set to *true*, as a system administrator you cannot access the CMC's **Search** page to perform a metadata search on a user's data.

### 4.6.3.6. Using the search-mgr.sh Tool For On-Demand Indexing

Each hour a **HyperStore cron job** invokes the following actions in the HyperStore Search Service:

- Create new search indices for buckets that were created within the past hour and that use a **search-enabled storage policy**.
- Update search indices for all buckets that use a search-enabled storage policy, to reflect changes to bucket content during the past hour -- such as object uploads or object deletions.
- Delete search indices for buckets that were deleted during the past hour and that had used a search-enabled storage policy.

However, if you wish you can use the *search-mgr.sh* tool to trigger the immediate update of all indexes for all buckets that use search-enabled storage policies, or the immediate indexing of a specific bucket, object, or object version. You can do this if you don't want to wait for the next hourly cron job to run, for example. Another use case for this tool is if your HyperStore Search Service cluster has been down more than a few hours, in which case the queue of pending search index update tasks may become larger than can be completely processed by the hourly cron job.

To use the *search-mgr.sh* tool, log into any of your core HyperStore nodes -- not an auxiliary node -- as *root* (or as an HSH user) and run the command appropriate to your objective:

Fully index all buckets in the local service region that use search-enabled storage policies:

```
search-mgr.sh sync-indices
```

Index a specific bucket:

```
search-mgr.sh sync-indices bucket <bucketname>
```

```
#Example:
search-mgr.sh sync-indices bucket pubs.bucket1
```

Index a specific object:

```
search-mgr.sh sync-indices object <bucketname>/<objectname>
#Example (object name must include full path):
search-mgr.sh sync-indices object pubs.bucket1/images/HybridThresholdOptions.png
```

Index a specific object version:

```
search-mgr.sh sync-indices object <bucketname>/<objectname>?version=<versionID>
#Example (object name must include full path):
search-mgr.sh sync-indices object pubs.bucket2/images/AddNewUser.png?version=fe1213e2-8763-
d79f-a929-00505685c5f2
```

Note that the *search-mgr.sh sync-indices [options]* command is limited in scope to the local service region in which the command is run. Therefore:

- If you have a multi-region system and you want to index all buckets that use search-enabled storage policies throughout the system, run the *search-mgr.sh sync-indices* command once from within each region. For example if you have two regions "Region1" and "Region2", run the command on a HyperStore node within Region1 and then run the command from a HyperStore node within Region2.
- If you have a multi-region system and you want to index a specific bucket (or a specific object within a specific bucket), run the appropriate command from a HyperStore node in the service region in which that bucket resides.

The full list of options supported by the *search-mgr.sh* tool is shown in the table below. In each case the scope is limited to indices in the local service region.

| search-mgr.sh Option | Purpose |
|---|---|
| help | Display command line help |
| sync-indices | Fully index all buckets in the local service region that use search-enabled storage policies |
| sync-indices bucket <bucketname> | Index a specific bucket |
| sync-indices object <bucketname>/<objectname> | Index a specific object |
| sync-indices object <bucketname>/<objectname>?version=<versionID> | Index a specific object version |
| list-indices | List all search indices in the local service region. By design there is one index for each bucket that uses a search-enabled storage policy. |

| search-mgr.sh Option | Purpose |
|---|---|
| status | Show the status of the HyperStore Search Service Kubernetes pods and API |
| *Snapshot and restore related options are listed below. These options can be used if, for example, Cloudian Support is guiding you through a conversion from a HyperStore legacy integration with Elasticsearch to the new HyperStore Search Service.* | |
| create-snapshot <snapshotname> | Create a snapshot of all search indices in the local service region. Only alpha-numeric characters are supported for <snapshotname> -- no special char-acters. |
| list-shapshots | List all snapshots in the local service region. |
| delete-all-indices | Delete all existing indices in the local service region. Warn-ing: This cannot be undone without a snapshot to restore from. |
| restore-shapshot <snapshotname> | Restore all indices from the specified snapshot. Requires that you *delete-all-indices* first. |
| delete-snapshot <snapshotname> | Delete the specified snapshot |

# 4.7. File Services (SMB/NFS)

## 4.7.1. File Services Feature Overview

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Bi-Modal Access Is Not Supported"** (page 208)*

A HyperStore system can optionally support Server Message Block (SMB) and Network File System (NFS) based file services. This SMB and NFS support is provided by the HyperStore File Service, an optional

HyperStore system component. For information about HyperStore File Service pricing, contact your Cloudian representative.

The basic file services delivery model is that the HyperStore File Service runs on dedicated "auxiliary" nodes and provides SMB and NFS protocol support to client applications. The HyperStore File Service uses configurable caching to store some file data locally on the auxiliary nodes while also archiving all file data to the HyperStore object storage cluster for highly durable and scalable storage. Each SMB or NFS file share uses its own corresponding HyperStore bucket as its storage back-end. The CMC provides for unified management of the file services feature within the context of your HyperStore system. You can configure and monitor the HyperStore File Service and its functionality through the CMC just as you do for the rest for your HyperStore services and features.

Once you've set up file service functionality in your HyperStore system, you can select HyperStore users for whom the file services feature will be enabled. Those HyperStore users will be able to use the CMC to create and configure SMB and/or NFS file shares, for access by file services end users. The HyperStore users for whom you enable the file services feature -- HyperStore users who will be able to create file shares -- can be group administrators or regular users. System administrator accounts cannot create file shares.

For SMB and NFS versions, the HyperStore File Service supports only:

- SMB versions 2 and 3
- NFS versions 3 and 4

For information about how to set up file services for your HyperStore system, and how to enable file services for selected users, see **"Preparing the File Services Feature"** (page 208).

#### 4.7.1.1. Bi-Modal Access Is Not Supported

In the current release of HyperStore's file services feature, "bi-modal access" -- whereby content in a bucket can be written and read by file service clients and also by S3 clients, and each type of client can read content written by the other type of client -- is not supported. Instead, **a bucket used as the back-end storage for a file share should be used exclusively that purpose, and should not be accessed by S3 user applications**.

Although an S3 user with read permissions on such a bucket could read content archived to the bucket by the HyperStore File Service, if an S3 user were to write to or delete from the bucket those changes would not be indexed by the HyperStore File Service and would not be visible to SMB/NFS users of the file share. The file share's database would be out of alignment with the actual content in the bucket.

And so:

- Only a new bucket that has not yet had any content uploaded to it by S3 clients should be used for back-end storage for a file share.
- A bucket that is being used as back-end storage for a file share should not be written to (or deleted from) by S3 users.

### 4.7.2. Preparing the File Services Feature

*Subjects covered in this section:*

This section outlines at a high level how to prepare the file services feature so that it is available to your end users. For some preparation tasks, this section provides just an overview of the task, along with a reference to more detailed instructions elsewhere in the HyperStore documentation.

**Note** For an introduction to the file services feature, see **"File Services Feature Overview"** (page 207).

### 4.7.2.1. Installing the HyperStore File Service

The HyperStore File Service is an optional HyperStore component that can be added to an existing HyperStore system so that the system will support SMB and NFS file services.

In the current release of HyperStore, a Cloudian Professional Services engagement is recommended for installing the HyperStore File Service. For more information contact your Cloudian representative.

### 4.7.2.2. Configuring Alerts Based on HyperStore File Service Status

You can monitor the status of your HyperStore File nodes in the CMC's **Data Centers** page, and optionally you can use the **Alert Rules** page to configure alerts based on HyperStore File Service status. Alerts based on HyperStore File Service status are not enabled by default.

For details about status monitoring, while on the CMC's **Data Centers** page (**Cluster -> Data Centers**) click **Help**.

For details about setting alert rules, while on the CMC's **Alert Rules** page (**Alerts -> Alert Rules**) click **Help**.

### 4.7.2.3. Configuring System Level Settings for File Services

After the HyperStore File Service has been installed, you can configure system level settings for file services in the CMC page **Cluster -> Cluster Config -> File Service Settings**. Broadly the system level file service settings are of two types:

- Settings that can only be set at the system level. These are:
  - The username and password that the CMC will initially use to connect to the HyperStore File Service administrative API.
  - The HyperStore File Service caching "waterlevel", which determines how much file data will be cached locally on the HyperStore File Service nodes.
- Settings for which you can set default values at the system level, but which can also be customized at the user level (for individual users for whom you enable file services functionality) and, for some settings, at the file share level (for individual file shares). These are:
  - Settings pertaining to how file data is moved to and read back from back-end object storage buckets

- SMB settings
- NFS settings
- DNS settings for the HyperStore File Service to use when resolving domains on its front end (such as Active Directory domains)

The first time you access the CMC's **File Service Settings** page you will need to connect the CMC to your HyperStore File Service by opening the **File Connection** section of the page. In the **File Connection** interface enter *admin* as the user name and *public* as the initial password. Then click **Connect**. The interface will then require you to change the password to one of your choosing:

- Enter the current password (*public*) and your chosen new password
- The new password must:
  - Be at least 9 characters long and no more than 36 characters
  - Include at least one upper case letter, one number, and one special character

After connecting to the HyperStore File Service and resetting the admin password, you can review and customize the file services configuration settings in the **File Service Settings** tab.

For details about these settings, in the CMC go to **Cluster -> Cluster Config -> File Service Settings** and then click **Help**.

**IMPORTANT !** Although the CMC does not prevent having both SMB and NFS enabled for the same file share, the **HyperStore File Service does not support multi-protocol write access to the same file share** -- and potential conflicts or corruptions may occur if the same file share is concurrently written to by SMB clients and NFS clients. Therefore, if both SMB and NFS are enabled on the same file share, a safe access policy must be implemented such as SMB write access, NFS read-only.

## 4.7.2.4. Enabling and Configuring File Services for Selected HyperStore Users

Topics in this sub-section:



### What a File Services Enabled HyperStore User Can Do

In the HyperStore file services delivery model, only a HyperStore user with S3 access can create and manage SMB and NFS file shares, and each file share will use one of the HyperStore user's buckets for its back-end storage. HyperStore system administrator accounts do not have S3 access and therefore **as a system administrator you cannot create and manage file shares**.

For file shares to be created, you must carefully choose HyperStore group administrators or HyperStore regular users for whom to enable file services. Those group admins or regular users will then be able to use the CMC to create and configure file shares, and their buckets will be used as the storage repositories for those file shares. Because persons for whom you enable file services will be able to create and configure SMB and NFS file shares, you should enable this capability only for persons who have experience with managing SMB and NFS file shares. (Note that although you as a system administrator can configure suitable defaults for most file services settings, the CMC will allow file services enabled users to customize those settings for their own file shares.)

Alternatively, you could create a HyperStore group administrator or regular user account for yourself to use, and use that account to create buckets and then to create and configure file shares backed by those buckets. Note though that with this approach, the S3 service usage associated with this activity (particularly, storage consumption by the buckets) will be attributed to this user account.

When you enable file services functionality for a HyperStore user, that user will be able to do the following through the CMC:

- Configure 'global' settings to apply across all of their own file shares.
- Create one or more file shares, with each file share corresponding to one of the user's own buckets. To serve as the archiving destination for a file share, a **bucket must have versioning enabled and should be used exclusively as the file share's archiving destination** and not for any other type of traffic. The bucket **must not have Object Lock enabled**. The current release of the HyperStore File Service feature will not work properly with buckets that have Object Lock enabled.
- Configure settings for each file share.
- Open and close file shares.
- Monitor file share usage (basic statistics such as number of files and size of files).
- Delete file shares.

To perform these tasks the user will work in the CMC's **Buckets & Objects** section, using interface functions that display only for users for whom you've enabled file services. Users can click **Help** while in the CMC's **Buckets & Objects** section for detailed descriptions of these tasks.

**Note** Although versioning must be enabled on buckets that serve as archives for file shares, non-current objects cannot be accessed through the file services front-end (SMB or NFS). Versioning is required because in future HyperStore File Service releases, support for volume snapshots is planned and that feature (and other planned features) will leverage versioning in the buckets that serve as file share archives. **Volume snapshots are not supported in the current release**.

**Note** If you have the CMC '**View user data**' function enabled in your system, this does not give you visibility into or control over a user's file shares. It only gives you access to users' S3 buckets. **You as a system administrator cannot create, view, edit, open/close, or delete a user's file shares through the CMC.**

## Enabling File Services for a HyperStore User

Enabling file services functionality for a HyperStore user can only be done through the CMC. You cannot enable file services for HyperStore users through the Admin API.

**To enable file services functionality for a HyperStore user in the CMC**:

- For a new user -- As you create the user in the **Add New User** panel set "File Services" to Enabled.
- For an existing user -- Retrieve the user, select **Edit** for the user, then in the **Edit User** panel set "File Services" to Enabled and then **Save** the change.

For details, in the CMC go to **Users & Groups -> Manage Users** and then click **Help**.

### Configuring Default File Services Settings for a HyperStore User

Once you've enabled file services functionality for a HyperStore user, when you retrieve that user in the CMC you will see a **File Setting** option among your controls for that user. Clicking **File Setting** will open a panel where you can set default file services setting values for that user. These are mostly the same settings for which you configured system level defaults, but here you can assign custom values for this specific user (values which will override the system defaults, just for this user). Here you can also configure Active Directory settings for authenticating end users of this HyperStore user's file shares. Active Directory settings are not configurable at the system default level -- they are only configurable at the level of specific HyperStore users for whom file services are enabled.

Note that a user for whom you've enabled file services will be able to set customized configuration settings for their file shares if they wish, overriding the default settings that you establish for the user.

For details, in the CMC go to **Users & Groups -> Manage Users** and then click **Help**.

## 4.7.2.5. Disabling File Services for a HyperStore User

If you've enabled file services for a user, and then at a later point in time you wish to disable file services for that user, you can do so through the CMC's **Edit User** interface (**Manage Users** -> retrieve user -> **Edit**) . In the **Edit User** page, change the user's "File Services" setting from Enabled to Disabled and then **Save** the change.

> **Note** If the user for whom you want to disable file services **still has file shares**, the user will need to delete those file shares before you can disable file services for the user. The CMC will not allow you to disable file services for a user who still has file shares. In a scenario where an uncooperative user will not delete their file shares, contact Cloudian Support for information about how you can delete the file shares on your own. (You as a system administrator cannot delete a user's file shares through the CMC, but Cloudian Support can guide you through "back door" methods for doing so.)

## 4.7.2.6. Deleting a HyperStore User Who Has File Services Enabled

You can delete a HyperStore user through the CMC's **Manage Users** page (or through the Admin API's *DELETE /user* call).

When you delete a user who has no buckets, it does not matter if the user has file services enabled -- the user will be deleted from the system just like any other user you would delete.

If you want to delete a user who still owns buckets, then the system behavior depends on your configuration:

- If the configuration setting *common.csv*: **"allow_delete_users_with_buckets"** (page 420) is set to *true*, then a user who owns buckets can be deleted and the system will not only delete the user but will also automatically delete the user's buckets and all the data in those buckets. If file services are enabled for the user and the user owns file shares, the system will also delete the file share data and metadata from the HyperStore File node(s).

  > **Note** If the user owns **file shares that are still Open**, then the system will not be able to delete those file shares and you will not be able to delete the user. Have the user close their file shares before you delete the user. If the user is uncooperative or has departed your organization, contact Cloudian Support for information about how you can close the shares. (You as a system

administrator cannot close a user's file shares through the CMC, but Cloudian Support can guide you through "back door" methods for doing so.)

- If the configuration setting *common.csv*: **"allow_delete_users_with_buckets"** (page 420) is set to *false*, you cannot delete a user who owns buckets. If the user owns buckets and corresponding file shares, have the user close and delete their file shares, and delete their buckets. Once the user has taken these actions this you can delete the user. If the user has departed your organization or is uncooperative, you could set *allow_delete_users_with_buckets* to *true* (at least temporarily) and then delete the user as described in the bullet point above.

#### 4.7.2.7. Restoring a Single-Node HyperStore File Service System

If you have a single-node HyperStore File Service deployment (as versus a three-node HyperStore File Service cluster deployment), then HyperStore File Service metadata and Kubernetes resources are automatically backed up regularly to two dedicated HyperStore S3 buckets. This backup mechanism was configured during HyperStore File Service installation (in the file */srv/pillar/hf/backup.sls* on your single HyperStore File node). In the unlikely event that your HyperStore File node needs to be completely restored from the backup files in the S3 buckets, open a case with Cloudian Support. Cloudian Support will work with you to implement the HyperStore File node system restore.

## 4.8. Quality of Service Controls

### 4.8.1. Quality of Service (QoS) Feature Overview

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Service Usage Types Subject to QoS Controls"** (page 214)*
- ***"QoS Assignment Granularity"** (page 214)*

The Cloudian HyperStore system supports user-level and group-level Quality of Service (QoS) settings:

- **User QoS settings** place upper limits on service usage by individual users.
- **Group QoS settings** place upper limits on aggregate service usage by entire user groups.

When the QoS feature is enabled, the HyperStore system enforces QoS settings by rejecting S3 requests that would result in a user (or a user's group) exceeding the allowed service usage level.

**Note** **All traffic and storage consumption associated with a bucket is counted as service usage by the bucket owner**. This includes access to the bucket by other users to whom the bucket owner has granted permission. It also includes access by public, unauthenticated users if the bucket owner has granted public access to bucket content -- such as in the case of a bucket configured as a static website.

**Note** Bucket logging is excluded from quality of service controls. Even if QoS limits are reached for a user or group, bucket logging (if enabled for the bucket) can continue.

#### 4.8.1.1. Service Usage Types Subject to QoS Controls

Several types of service usage metrics can be configured for QoS controls:

- Storage quota, by number of KiBs.
- Storage quota, by number of objects.
- Peak HTTP request rate, in requests per minute. The user is not allowed more than this many requests in a 60 second interval.
- Peak data upload rate, in KiBs per minute.
- Peak data download rate, in KiBs per minute.

When configuring QoS controls, you have the option of limiting some of the usage types above while leaving others unrestricted. For example, if you wish you could limit per-user and/or per-group storage volume (by KBs), while placing no restrictions on number of stored objects. Similarly, if you wish you could cap data upload rate while placing no cap on data download rate.

When the system rejects a user request because of a storage quota, it returns an HTTP 403 response to the client application. When the system rejects a user request due to rate controls, it returns an HTTP 503 response to the client application. Any time a user request is rejected due to a QoS threshold, the system writes a WARN level message to the S3 Service application log.

For each type of QoS metric, the system also supports a configurable warning level -- which you can set to a lower threshold of usage than the threshold at which requests will be rejected. If a user's request results in the warning threshold being exceeded, the request will succeed but the system will write a WARN level message to the S3 Service application log. Note that the system does not inform the user that the warning threshold has been exceeded -- it only writes the aforementioned log message.

If you wish you can configure the system so that WARN level messages in the S3 Service application log trigger an alert. (By default configuration, alerts are only triggered by ERROR level messages in the application log.) For more information see **"Preparing the QoS Feature"** (page 215).

**Note** The **storage overhead associated with replication or erasure coding does not count toward a user's storage quota**. For example, a 1MiB object that is protected by 3X replication counts as only 1MiB toward the storage quota.

**Note** For information on how auto-tiering impacts the implementation QoS controls, see **"How Auto-Tiering Impacts Usage Tracking, QoS, and Billing"** (page 174).

#### 4.8.1.2. QoS Assignment Granularity

The system also provides you the flexibility to assign different QoS settings to different users and groups. In the case of **user QoS settings**, the system provides you three levels of granularity:

- For the system as a whole, you can configure a **system default for user QoS settings**, applicable to all users of your S3 service.
- For particular groups, you can configure a **group-specific default for user QoS settings**. If you do, then for users in that group, the group-specific user QoS defaults will override the system-wide user QoS defaults.

- For particular users, you can configure **user-specific QoS settings**. If you do, these settings will override any group-wide or system-wide defaults for user QoS settings.

For **group QoS** settings you have two levels of granularity:

- For the system as a whole, you can configure a **system default for group QoS settings**, applicable to all groups in your S3 service.
- For particular groups, you can configure **group-specific group QoS settings**. If you do, these settings will override the system-wide defaults.

*Note for multi-region systems*

If your HyperStore service has multiple service regions, QoS limits are applied separately to each region, and the system provides you the ability to configure different QoS settings for different regions. For example, you could configure default user QoS settings that allow users 20GiB of storage in your "North" service region and 30GiB in your "South" service region.

## 4.8.2. Preparing the QoS Feature

This section covers the following QoS system set-up topics:


- ***"Enabling QoS Enforcement"*** *(page 215)*
- ***"Enabling Alerts for QoS Events"*** *(page 216)*


### 4.8.2.1. Enabling QoS Enforcement

By default, the HyperStore system's QoS functionality is **disabled**. Before enabling the functionality, think first about whether you want to implement QoS controls based just on storage quotas, or if you also want to apply QoS controls for request rates (HTTP requests and upload/download bytes). This choice impacts the configuration changes that you will make. In general, for optimal system performance you should enable only the QoS functionality that you actually intend to apply to service users.

Note that in enabling QoS functionality as described below, you are merely "turning on" the S3 Service mechanisms that enforce whatever QoS restrictions you establish for users and groups. The creation of specific QoS restrictions is a separate administrative task as described in **"Applying QoS Controls to Users"** (page 217).

To enable HyperStore QoS enforcement, in the CMC go to the **Configuration Settings** page and open the **Quality of Service** panel. Then:

- To enable enforcement of **storage quotas only** (number of bytes and/or number of objects), set just the "QoS Limits" setting to "enabled".
- To enable enforcement of **storage quotas and also traffic rates** (number of HTTP requests per minute, or bytes uploaded or downloaded per minute), set both the "QoS Limits" setting and the "QoS Rate Limits" setting to "enabled".

After you **Save**, your configuration changes are applied to the system dynamically — no service restart is required.

**Note** Enforcing QoS for traffic rates but not for stored bytes and objects is not supported at the system configuration level. If you want to use QoS in this way, set both "QoS Limits" and "QoS Rate Limits" to

enabled, then when you're configuring QoS limits for groups and users set the stored bytes and stored objects controls to unlimited and the rate controls to your desired levels.

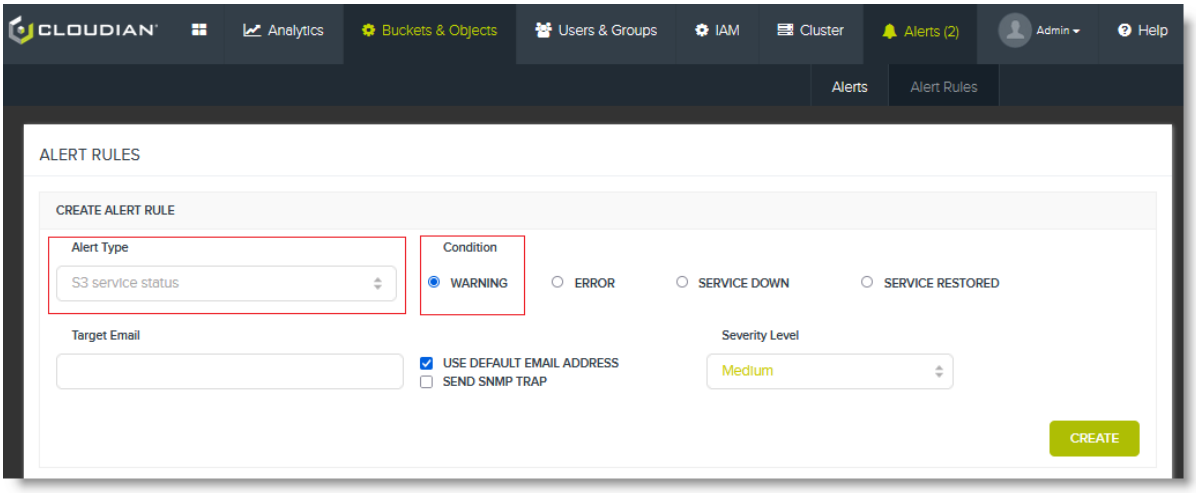
### 4.8.2.2. Enabling Alerts for QoS Events

If you've enabled QoS enforcement (as described above), then optionally you can also configure the system to generate alerts whenever either of these QoS-related event types occur:

- A user's S3 request has been fulfilled and as a result a user-level or group-level QoS warning threshold has been exceeded.
- A user's S3 request has been rejected because if the request was fulfilled a user-level or group-level QoS upper threshold would be exceeded.

When either of these event types occur the system writes a WARN level message to the S3 application log. By default WARN level messages to the S3 application log do not trigger alerts -- by default only ERROR level messages trigger alerts.

To enable alerting for WARN level messages in the S3 application log, in the CMC go to the **Alert Rules** page (**Alerts -> Alert Rules**) and create a new alert rule for the Alert Type "S3 service status" and the Condition "Warning".




Once you've created this alert rule, alerts will be generated whenever there are WARN level messages written to the S3 application log (this includes WARN level messages related to QoS as well as other WARN level messages.)

The alert text for QoS threshold related alerts will include the **canonical user ID** of the user whose fulfilled S3 request resulted in a warning level QoS threshold being crossed or whose S3 request was rejected because of an upper level QoS threshold.

To obtain the user name that corresponds to a canonical user ID you can use the Admin API call *GET /user-?canonicalUserId=string*. For more information on this API call see the "user" section of the *Cloudian HyperStore Admin API Reference*.

## 4.8.3. Applying QoS Controls to Users

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Setting System Default QoS Limits for Groups and Users (CMC)"** (page 217)*
- ***"Setting QoS Limits for Specific Groups and Users (CMC)"** (page 217)*
- ***"Setting QoS Limits for Groups and Users (Admin API)"** (page 218)*

By default, QoS controls for all service usage types — storage quota by bytes, storage quota by number of objects, HTTP requests per minute, upload bytes per minute, and download bytes per minute — are set to "unlimited". So when you **enable QoS enforcement by the system**, groups and users still are not subject to QoS controls until you set QoS limits for the various service usage types.

You can set system-wide defaults for group QoS limits and individual user QoS limits before provisioning any groups or users, if you wish. After **provisioning groups and users** you can set QoS limits specific to those groups and users, which would override the system-wide defaults.

**Note All traffic and storage consumption associated with a bucket is counted as service usage by the bucket owner**. This includes access to the bucket by other users to whom the bucket owner has granted permission. It also includes access by public, unauthenticated users if the bucket owner has granted public access to bucket content -- such as in the case of a bucket configured as a static web-site.

### 4.8.3.1. Setting System Default QoS Limits for Groups and Users (CMC)

HyperStore supports setting system default QoS limits that apply to all groups (default limits on the aggregate service usage per group) and system default QoS limits that apply to all users (default limits on service usage per user). If you wish you can set these system default QoS limits before you've provisioned groups and users into the system. You can also set or change these system default QoS limits at any time after you've provisioned groups and users.

To set default QoS limits for all groups in your HyperStore system go to the CMC's **Manage Groups** page and click **Group QoS Default** to open the **Group QoS Limits: Defaults** panel. In this panel you can set your desired default QoS limits for groups.

To set default QoS limits for all users in your HyperStore system go to the CMC's **Manage Users** page and click **User QoS Default** to open the **User QoS Limits: Defaults** panel. In this panel you can set your desired default QoS limits for users.

For details, while on the CMC's **Manage Groups** page or **Manage Users** page click Help. Then in the Help's "Supported tasks" list click "Set Quality of Service (QoS) Limits".

### 4.8.3.2. Setting QoS Limits for Specific Groups and Users (CMC)

Once you have provisioned groups and users into your HyperStore system you can set QoS limits for specific groups and users if you wish. If you do so, these group-specific and user-specific QoS limits will override the system default QoS limits, for those particular groups and users.

To set group QoS limits for a specific group, first retrieve the group in the **Manage Groups** page. Then click **Group QoS** for the group. This opens the **Group QoS Limits: Overrides** panel, where you can configure group QoS limits for that specific group.

To set default per-users QoS limits for all users who belong to a specific user group, first retrieve the group in the **Manage Groups** page. Then click **User QoS Group Default** for the group. This opens **User QoS Limits: Group Defaults** panel, where you can configure default user QoS limits for the group.

To set QoS limits for a specific individual user, first retrieve the user in the **Manage Users** page, then click **Set QoS** for the user. This opens the **User QoS Limits: Overrides** panel, where you can configure QoS limits for that specific user.

For details, while on the CMC's **Manage Groups** page or **Manage Users** page click Help. Then in the Help's "Supported tasks" list click "Set Quality of Service (QoS) Limits".

#### 4.8.3.3. Setting QoS Limits for Groups and Users (Admin API)

With the Admin API method *POST /qos/limits* you can set QoS limits for HyperStore groups and users. With the method's URI parameters you can specify whether you're setting system defaults for group QoS settings; or system defaults for user QoS settings; or QoS settings for a specific group or user. URI parameters also enable you to set the numerical limits for each service usage type — for example, a Storage Bytes limit of 100GB.

The Admin API also supports:

- A *GET /qos/limits* method, for retrieving QoS limits for groups or users
- A *DELETE /qos/limits* method, for deleting QoS limits for groups or users

For details about these API calls see the "qos" section of the ***Cloudian HyperStore Admin API Reference***.

## 4.9. Usage Reporting

### 4.9.1. Usage Reporting Feature Overview

*Subjects covered in this section:*



By default the HyperStore system keeps track of the following service usage metrics for each user group and each individual user:

- Number of Storage Bytes
- Number of Storage Objects

Optionally, you can configure the system to also track the following metrics for each group and user (these metrics are disabled by default):

- Number of HTTP Requests
- Number of Bytes IN (bytes uploaded into the system)
- Number of Bytes OUT (bytes downloaded from the system)

Like Amazon S3, the HyperStore system attributes all service usage to **bucket owners**. If a bucket owner grants permission (via ACL policies) for other users to use the bucket, the system attributes the service activity of the grantees to the bucket owner. For example, if grantees upload objects into the bucket, the associated Bytes IN activity and Storage Bytes impact is attributed to the bucket owner — not to the grantees.

If a HyperStore account root user creates IAM users, then storage activity associated with those IAM users' buckets counts toward the usage of the account root user.

The HyperStore system's tracking of service usage by groups and users serves two main purposes:

- **Usage reporting**. Based on service usage tracking data, you can generate service usage reports for individual users, for user groups, for a whole service region, or for your entire HyperStore system.
- **Billing**. Usage tracking provides the foundation for billing users or groups on the basis of their service usage level.

**Note** **Cloudian HyperIQ** is a product that provides advanced analytics and visualization of HyperStore S3 Service usage by users and groups, as well as HyperStore system monitoring and alerting. For more information about HyperIQ contact your Cloudian representative.

#### 4.9.1.1. Bucket Usage Statistics

Optionally, you can also configure the system to track usage statistics on a per-bucket basis. This may be useful if for example your HyperStore service is set up such that there is just single "user" for service access purposes, with that one user having multiple buckets each for a different purpose.

The bucket statistics features are disabled by default. For information on enabling these features see **"Preparing the Usage Reporting Feature"** (page 221).

#### 4.9.1.2. How Usage Data is Calculated, Tracked, and Processed

The table below provides a high level view of how the system generates and handles usage data for users and groups. Usage tracking for Storage Bytes and Storage Objects is handled somewhat differently than tracking for HTTP Requests and Bytes IN/OUT, including that the latter is disabled by default.

| Usage Data Types | System Handling |
|---|---|
| • Storage Bytes<br>• Storage Objects | For each S3 request that the S3 Service processes, the S3 Service updates per-user and per-group Storage Bytes and Storage Objects counters that are maintained in the Redis QoS database. In calculating the increment to Storage Bytes associated with a given S3 request, the system includes the object name size and object metadata size as well as the size of the object itself.<br><br>Note that storage **overhead associated with replication or erasure coding does not count** toward a user's Storage Bytes count. For example, a 1MB object that is protected by 3X replication or by 4+2 erasure coding counts as only 1MB toward the Storage Bytes count.<br><br>These per-user and per-group Storage Bytes and Storage Objects counters in Redis are used by the system to enforce storage quotas, if such quotas are part of your Quality of Service configurations.<br><br>Every five minutes, freshly updated Redis QoS counts for Storage Bytes |

| Usage Data Types | System Handling |
|---|---|
| | and Storage Objects are written to the Raw column family in Cassandra's Reports keyspace, where the data is subjected to additional processing in support of reporting and billing functionality. Each hour the Raw data is automatically processed to derive hourly roll-up data which is written to the RollupHour column family. The hourly roll-up data includes, for each user and each group, the hour's maximum value and weighted average value for Storage Bytes and for Storage Objects.<br><br>For example, if during a given hour User1 has 10MB of Storage Bytes for the first 20 minutes of the hour and then 15MB for the next 40 minutes of the hour, her weighted average Storage Bytes for the hour is:<br><br>`  10MB X 20/60 = 3.33MB`<br>`+ 15MB X 40/60 = 10 MB`<br><br>`=                13.33MB weighted average for hour`<br><br>This hourly data is in turn rolled up once each day to derive daily roll-up values (including, for each user and group, the day's maximum and day's average for Storage Bytes and Storage Objects); and the daily roll-up values are rolled up once each month to derive monthly roll-up values (including monthly maximums and averages for each user and group). This data is stored in the RollupDay column family and RollupMonth column family, respectively.<br><br>**Note** The writing of Storage Bytes and Storage Objects counters from Redis over to Cassandra, and the usage data roll-ups that take place within Cassandra, are triggered by **system cron jobs**. |
| • HTTP Requests<br>• Bytes IN<br>• Bytes OUT | By default system configuration, HTTP Requests, Bytes IN, and Bytes OUT are not tracked.<br><br>If you **enable usage tracking for HTTP Requests and Bytes IN/OUT**, then for each S3 request the S3 Service writes transactional metadata directly to the Raw column family in Cassandra's Reports keyspace. It does so asynchronously so that S3 request processing latency is not impacted.<br><br>In recording the Bytes IN and Bytes OUT impacts of a given S3 transaction, both the request and the response are counted, and HTTP header size is counted as well as body size. For example, a GET request/response pair will count as Bytes IN (typically very small) as well as Bytes OUT (varies with object size); and conversely a PUT request/response pair will count as Bytes OUT (typically very small) as well as Bytes IN (varies with object size).<br><br>Together with the Storage Bytes and Storage Objects data, the HTTP Request and Bytes IN/OUT data in the Raw column family is rolled up each hour. For each user and each group the roll-up calculates the hour's total for HTTP GETs, PUTs, and DELETEs, and for Bytes IN and Bytes OUT. For Bytes IN and Bytes OUT, maximums for the hour (largest single upload and largest single download) and averages for the hour |

| Usage Data Types | System Handling |
|---|---|
| | (average upload size and average download size) are derived for each user and group.<br><br>The hourly roll-up data is rolled up daily; and the daily roll-up data is rolled up monthly.<br><br>If you fully enable HyperStore Quality of Service functionality, then for each S3 request the S3 Service also updates HTTP Request and Bytes IN/OUT counters in the Redis QoS database, in support of QoS enforcement. |

#### 4.9.1.3. Protection of Usage Data Through Replication

All HyperStore service usage data is stored in Cassandra, in the Reports keyspace. This data is protected against loss or corruption by maintaining multiple copies of the data, with each copy stored on a different node. During system installation the install script prompts you to specify how many replicas to keep of service metadata, which includes usage data. If you are running HyperStore in multiple data centers, during installation you choose how many service metadata replicas to keep in each data center.

### 4.9.2. Preparing the Usage Reporting Feature

This section covers the following system set-up topics for usage reporting:

- ***"Enabling Additional Usage Reporting Features"*** *(page 221)*
- ***"Setting Usage Data Retention Periods"*** *(page 223)*

#### 4.9.2.1. Enabling Additional Usage Reporting Features

By default the HyperStore system keeps track of the number of stored bytes and the number of stored objects for each user group and each individual user. This topic describes how to enable additional HyperStore usage reporting features that are disabled by default:

- Per-user and per-group traffic rates (HTTP request rates and data transfer rates)
- Per-bucket usage tracking
- Per-bucket object and byte counts

**Per-User and Per-Group Traffic Rates**

By default the HyperStore system tracks Storage Bytes and Storage Objects for each user and each group. If you want the system to also track HTTP Requests, Bytes IN, and Bytes OUT for each user and group, log into the CMC and go to the **Configuration Settings** page, and then open the **Usage Tracking** panel. There, enable the "Track/Report Usage for Request Rates and Data Transfer Rates" setting and **Save** your change. Your change is applied to the system dynamically -- there is no need to restart services.

Once you enable this feature, this type of usage data will be tracked and available for reporting from that point in time forward. There will not be any per-user and per-group traffic rate data from prior to the time that you enabled this feature.

**Note** Enabling this feature results in additional data being stored in the Metadata DB, and additional work for the cron jobs that roll up usage data into hourly, daily, and monthly aggregates.

### Per-Bucket Usage Tracking

HyperStore supports usage tracking and reporting on a per-bucket basis, but this feature is disabled by default. To enable per-bucket usage statistics, in the configuration file **common.csv** set *bucketstats_enabled* to *true*. Then **push your change out to the cluster and restart the S3 Service**.

Once you enable this feature, bucket usage data for storage consumption and traffic rates will be tracked and available for reporting from that point in time forward, through the *usage* Admin API calls (for details see the "usage" section of the ***Cloudian HyperStore Admin API Reference***). There will not be any per-bucket usage data from prior to the time that you enabled this feature.

**Note** Enabling this feature results in additional metadata being stored in the Metadata DB, and additional work for the cron jobs that roll up usage data into hourly, daily, and monthly aggregates.

**Note** Once enabled, the per-bucket usage tracking feature allows a view into bucket activity across a specified period of time. By contrast, the per-bucket object and byte count feature -- described below -- allows just snapshots of current counts.

### Per-Bucket Object and Byte Counts

HyperStore supports keeping counts of the number of objects and bytes in each bucket, but this feature is disabled by default. To enable per-bucket object and byte counts:

1. In the configuration file **common.csv** set *s3_perbucket_qos_enabled* to *true*.
2. **Push your change out to the cluster and restart the S3 Service**
3. After restarting the S3 Service, run the Admin API call *POST /usage/repair?groupId=ALL* to bring the counters up to date for buckets that already have objects in them. For details see the "usage" section of the ***Cloudian HyperStore Admin API Reference***.

Subsequently, the counters will automatically be updated for each bucket each time there is an S3 transaction that impacts the byte count or object count for the bucket.

With this feature enabled, you can query the current stored-bytes and stored-objects counts for individual buckets by using the Admin API calls *GET /system/bytecount* and *GET /system/objectcount*. For details see the "system" section of the ***Cloudian HyperStore Admin API Reference***.

**Note** In some atypical circumstances the keeping of these per-bucket stored-bytes and stored-object counts, which will be updated after every S3 transaction, may cause a minor-to-moderate decline in S3 write performance. Example circumstances would be if there are an exceptionally large number of buckets in your HyperStore system, or if you have a multi-data center HyperStore deployment with more than usual latency between the data centers. If you want to enable per-bucket stored-bytes and stored-objects counters in your system but are concerned about potential performance impacts, consult with Cloudian Support.

#### 4.9.2.2. Setting Usage Data Retention Periods

Data retention periods are separately configurable for hourly roll-up, daily roll-up, and monthly roll-up usage data. After rolled up data has been stored for its configured retention period -- also known as its "time-to-live" or TTL -- it is automatically deleted from the system.

The relevant settings are in the configuration template **mts.properties.erb**:

- **"reports.rolluphour.ttl"** (page 456) (default = 5616000 seconds [65 days])

  **IMPORTANT !** The HyperStore system calculates monthly bills for service users by aggregating **hourly** roll-up data. Once hourly data is deleted, you will not be able to generate bills for the service period covered by that data. So be sure to have *reports.rolluphour.ttl* set to a value large enough to accommodate your billing routine.

- **"reports.rollupday.ttl"** (page 457) (default = 5616000 seconds [65 days])
- **"reports.rollupmonth.ttl"** (page 457) (default = 15552000 seconds [180 days])

If you edit any of these settings, **push your change out to the cluster and restart the S3 Service**.

### 4.9.3. Generating a Usage Report

You can generate usage report data through either the CMC or the Admin API.

#### 4.9.3.1. Generating a Usage Report (CMC)

In the CMC's **Usage By Users & Groups** page (**Analytics -> Usage By Users & Groups**) you can generate usage reports for a user, a group, a whole region, or your whole system. You can choose a report interval, a report granularity (raw, hourly, daily, or monthly roll-up), and the usage type to report on (stored bytes or stored object counts, or — if **request tracking is enabled** — statistics for HTTP requests and data transfer).

You can display reports in the CMC in tabular format or dynamic graph format, or download report data in comma-separated value (CSV) format.

The CMC does not support per-bucket usage reporting. For that you must use the Admin API.

#### 4.9.3.2. Generating a Usage Report (Admin API)

The Admin API method *GET /usage* returns usage data for a specified user, group, or bucket. You can choose a report interval, a report granularity (raw, hourly, daily, or monthly roll-up), and the usage type to report on (stored bytes or stored object counts, or — if **request tracking is enabled** — statistics for HTTP requests and data transfer).

**Note** To retrieve usage data for a whole region or the whole system, you must execute *GET /usage* separately for each group.

The Admin API also supports:

- *GET system/bytecount* and *GET system/objectcount* calls that return the current total bytes count and total objects count the system, a group, a user, or a bucket (if bucket counts are enabled)

- A *POST /usage/bucket* call that can retrieve raw usage data for multiple specified buckets at once (if bucket usage statistics are enabled)

For more detail on usage related Admin API calls see the "system" and "usage" sections of the ***Cloudian HyperStore Admin API Reference***.

## 4.9.4. Validating Storage Usage Data

As described in **"Usage Reporting Feature Overview"** (page 218), each time the S3 Service processes an S3 request it updates per-user and per-group counters for Storage Bytes and Storage Objects which are maintained in the Redis QoS database. These counts are regularly written over to the Cassandra Reports keyspace, where they are post-processed for reporting and for bill generation.

Because the Redis QoS counters for Storage Bytes and Storage Objects can impact your billing of service users (if you charge users based on volume of storage used), it's important that the counts be accurate.

Various types of errors and conditions can on occasion result in discrepancies between the Redis QoS counts and the actual stored bytes and objects owned by particular users. The HyperStore system provides mechanisms for detecting and correcting such discrepancies.

### 4.9.4.1. Routine Automated Validation

Twice a day, a **system cron job** executes the HyperStore Admin API method *POST /usage/repair/dirtyusers*. This operation randomly selects up to a configurable maximum number of users (*mts.properties.erb:* **"usage.repair.maxdirtyusers"** (page 458); default = 1000) for whom Storage Bytes and/or Storage Objects counts in Redis have changed since the last time a validation operation was run. For those users, the operation validates the Redis counters by comparing them to the information in the Cassandra "UserData_<policyId>" keyspaces' "CLOUDIAN_METADATA" column family, which stores metadata (including size) for every object belonging to each user of the HyperStore service. If any users' Redis counters are found to be out-of-sync with counts derived from the object metadata, the Redis counters are corrected.

In normal circumstances, this automated mechanism should suffice for maintaining the accuracy of usage data for Storage Bytes and Storage Objects.

### 4.9.4.2. Validation for Special Cases

If you have reason to question the accuracy of Storage Bytes and/or Storage Objects counts for a particular user — for example, if a user claims their usage report or their bill to be inaccurate — the Admin API supports a method for validating the counts for a specified user: *POST /usage/repair/user*.

The Admin API also supports a method for validating Storage Bytes and Storage Object counts for a whole user group, a whole service region, or the whole system: *POST /usage/repair*. Depending on how many users are in your system, this is potentially a resource-intensive operation. This operation should only be considered in unusual circumstances, such as if the Redis QoS database has been brought back online after being unavailable for some period of time.

For details on these API calls, see the "usage" section of the ***Cloudian HyperStore Admin API Reference***.

# 4.10. Billing

## 4.10.1. Billing Feature Overview

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Retention of Usage Data Used for Billing"** (page 225)*
- ***"Retrieving Bucket Tags"** (page 225)*

The HyperStore system maintains service usage data for each group and each user in the system. This usage data -- which is automatically generated by recurring **cron jobs** that call the Admin API -- serves as the foundation for HyperStore service billing functionality.

The system provides you the ability to create rating plans that specify charges for the various types of service usage activity, and to assign each group and each user a rating plan. You can then generate bills for a user or for a whole user group, for a selected service month. The CMC has a function for displaying a single user's bill report in a browser, but in the more typical use case you will use the HyperStore Admin API to generate user or group billing data that can be ingested a third party billing application.

HyperStore also allows for special treatment of designated source IP addresses, so that the billing mechanism does not apply any data transfer charges for data coming from or going to these "allowlisted" domains.

> **Note** For information on how auto-tiering impacts billing calculations, see **"How Auto-Tiering Impacts Usage Tracking, QoS, and Billing"** (page 174).

### 4.10.1.1. Retention of Usage Data Used for Billing

Billing calculation is derived from hourly rollup usage data that is automatically generated by system cron jobs. The retention period for this hourly rollup usage data is configurable and defaults to **65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.**

For more information about this configurable setting see **"Setting Usage Data Retention Periods"** (page 223)

### 4.10.1.2. Retrieving Bucket Tags

If your billing scheme makes use of bucket tags (as created by the S3 API method *PUT Bucket tagging*): The HyperStore Admin API supports a method for retrieving all the bucket tags for all users in a specified group. Because it is implemented through the Admin API, that method does not require the users' S3 access credentials. For more information see *GET /bucketops/gettags* in the "bucketops" section of the ***Cloudian HyperStore Admin API Reference***.

## 4.10.2. Preparing the Billing Feature

This section covers the following topics for setting up billing in the system:

- ***"Retention of Usage Data Used for Billing"** (page 226)*
- ***"Creating Rating Plans"** (page 226)*

- ***"Creating an "Allowlist" for Free Traffic (Optional)"*** *(page 228)*

### 4.10.2.1. Retention of Usage Data Used for Billing

Billing calculation is derived from hourly rollup usage data that is automatically generated by system cron jobs. The retention period for this hourly rollup usage data is configurable and defaults to **65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.**

For more information about this configurable setting see **"Setting Usage Data Retention Periods"** (page 223).

### 4.10.2.2. Creating Rating Plans

The HyperStore system supports the creation of billing rating plans in which charges can be specified for each of several different service activity types. The system supports rating plans that charge:

- Per GB of data in storage (based on a calculated average storage volume for the billing period)
- Per GB of data uploaded
- Per GB of data downloaded
- Per 10,000 HTTP GET or HEAD requests
- Per 10,000 HTTP PUT or POST requests
- Per 10,000 HTTP DELETE requests

A user's bill can then be calculated by applying the user's assigned rating plan to the user's activity levels for each of these activity types, and adding together the charges for each activity type to get a total charge for the billing period.

You can create multiple, named rating plans, each of which applies different charges to the various service activity types. Once you've created rating plans, those plans are then available for you to **assign to users**.

For example, you can create higher-priced and lower-priced rating plans and then assign different plans to different users based on the users' quality of service terms.

> **IMPORTANT !** If you want to bill for data upload or download volume, or for HTTP request volume, you must enable the "Track/Report Usage for Request Rates and Data Transfer Rates" setting in the CMC's **Configuration Settings** page, Usage Tracking section. By default this setting is disabled and the system does not maintain per-user HTTP request counts and data transfer byte counts.

You can create rating plans either through the CMC or through the HyperStore Admin API. The system also comes equipped with an editable default rating plan.

#### Creating Rating Plans for Billing (CMC)

To create rating plans through the CMC, use the **Rating Plan** page (**Users & Groups -> Rating Plan**). For each plan you create, you can select a currency and then specify the charges to apply per each of the chargeable service activity types (per GB of stored data, per GB of data uploaded, and so on). For each rating plan, the interface supports the creation of single-tier or multi-tier pricing schemes for each chargeable activity.

The **Rating Plan** page also supports viewing and editing existing plans, including the default rating plan that comes with the HyperStore system. For more information about the default rating plan, while on the CMC's **Rating Plan** page click **Help**.

### Creating Rating Plans for Billing (Admin API)

To create a rating plan through the Admin API, use the *PUT /ratingPlan* method.

The Admin API also supports:

- Retrieving a list of existing rating plans: *GET /ratingPlan/list*
- Retrieving a rating plan: *GET /ratingPlan*
- Editing a rating plan: *POST /ratingPlan*
- Deleting a rating plan: *DELETE /ratingPlan*

For details of these API calls see the "ratingPlan" section of the ***Cloudian HyperStore Admin API Reference***.

### Example of a Rating Plan Applied to Calculate a User's Monthly Bill

This example walks through a hypothetical rating plan and how it would apply to a user's service usage for a month.

### Storage Charges

- Types: One type only. The average number of GBs of data stored for the month.
- Example:
    - Unit: Dollars per GB-months.
    - Pricing: From 0-1 TB at $0.14 per GB-month, 1-10 TB at $0.12 per GB-month, 10+ TB at $0.10 per GB-month.
    - Usage: Store 0.1 TB for first 10 days of month, then 20 TB for remaining 21 days of month.
    - Sum up usage over month: 0.1TB X 240 hours + 20TB X 504 hours = 10,104 TB-hours.
    - Convert to GB-months: 10,104 TB-hours X (1024 GB/TB) X (1 month/744 hours) = 13,906.58 GB-months
    - Apply tiered pricing: ($0.14 X 1 X 1024GB) + ($0.12 X 9 X 1024GB) + ($0.10 X 3666.58GB) = $1615.94.

### Data Transfer Charges

- Types: 2 types. Data Transfer IN and Data Transfer OUT.
    - Each type (IN, OUT) has own cost.
- Example:
    - Unit: Dollars per GB.
    - Pricing: IN: 0GB+ at $0.10. OUT: 0-10GB at $0.20, 10GB+ at $0.10
    - Usage: Transfer-IN 300GB, Transfer-OUT 100GB.
    - (300 X $0.10) + (10 X $0.20 + 90 X $0.10) = $41.00.

### Request Charges

- Types: 3 types of requests are HTTP PUT/POST, HTTP GET/HEAD, and HTTP DELETE.
    - Each request type has own cost.

- Example:
  - Unit: Dollars per 10,000 requests.
  - Pricing: PUT/POST: $0.20 per 10,000 requests, GET/HEAD: $0.01 per 10,000 requests, DELETE: $0.00 per 10,000 requests.
  - Usage: For the month, 25,000 PUTs/POSTs, 300,000 GETs/HEADs, 1000 DELETEs.
  - ($0.20 X 25,000 / 10,000) + ($0.01 X 300,000 / 10,000) + ($0.00 X 1000 / 10,000) = $0.53.

### Total Bill for Month

- Storage charges: $1615.94
- Data transfer charges: $41.00
- Requests charges: $0.53
- TOTAL: $1657.47

## 4.10.2.3. Creating an "Allowlist" for Free Traffic (Optional)

There may be certain source domains from which you want users to be able to submit S3 requests to the HyperStore system without incurring any data transfer charges. The HyperStore system allows you to create such an "allowlist", consisting of IPv4 addresses and/or subnets. For traffic originating from these addresses or subnets, there is no charge for data IN or data OUT, nor is there any charge for HTTP requests — regardless of users' assigned rating plans.

Note that the allowlist does not have any impact on what users are charged for data **storage**. It allows only for free **traffic** from the specified origin domains. For data storage billing, a user's regular assigned rating plan pricing is used, even if all of the user's S3 requests originate from an allowlisted IP address.

You can creating a billing allowlist through either the CMC or the Admin API.

**IMPORTANT !** If your S3 Servers are behind a load balancer, the load balancer must be configured to pass through request source IP addresses in order for the allowlist feature to work. Also, note that when S3 requests are submitted via the CMC, the S3 Servers consider the CMC itself to be the source of the request. The CMC does not pass to the S3 Servers the IP addresses of CMC clients.

### Enabling the Allowlist Feature

To enable the HyperStore billing allowlist feature:

1. On your Configuration Master node, open the following configuration file in a text editor:

   ```
   /etc/cloudian-<version>-puppet/manifests/extdata/common.csv
   ```

2. Set *admin_whitelist_enabled* to *true*, then save your change. (If the setting is already set to *true*, close the file without changing the setting.)
3. If you made a change to the setting, use the installer to push your change to the cluster and to restart the S3 Service and the CMC. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

### Creating a Source IP Allowlist (CMC)

To create an allowlist through the CMC, use the **Allowlist** page (**Users & Groups -> Allowlist**). That page shows the ID and name of the default allowlist, which initially has no IP addresses in it. To create a functioning

allowlist, you edit the default allowlist by adding IP addresses or subnets to it.

Once created, the allowlist takes effect. From that time forward, the HyperStore billing system will no longer apply traffic charges to users' traffic originating from the allowlisted IP addresses and subnets.

**Note** If you want to see a particular user's allowlisted traffic volume during a given billing period, you can do so as part of the HyperStore system's functionality for .

### Creating a Source IP Allowlist (Admin API)

The HyperStore Admin API provides two different methods for creating an allowlist:

- Use the *POST /allowlist* method to post your allowlist as a JSON-encoded request payload.
- Use the *POST /allowlist/list* method to post your allowlist as a URI parameter.

The Admin API also supports a *GET /allowlist* method for retrieving the contents of your current allowlist.

For details on these API calls, see the "allowlist" section of the ***Cloudian HyperStore Admin API Reference***

## 4.10.3. Applying Billing to Users

This section covers the following topics for applying billing to users:


- ***"Assigning Rating Plans to Groups and Users"*** *(page 229)*
- ***"Generating Billing Data for a User or Group"*** *(page 230)*


### 4.10.3.1. Assigning Rating Plans to Groups and Users

When you create a new user group you can assign to the group a **rating plan** that by default will apply to each user in the group. Optionally you can assign a rating plan to specific users within the group, overriding the group's default rating plan.

In a multi-region HyperStore system, you have the option of assigning groups or individual users different rating plans for activities in different regions. For example, you might charge users more for stored data in buckets that they've created in your North region than for stored data in buckets created in your South region.

If you do not explicitly assign a rating plan to a user, the user is automatically assigned the rating plan that's assigned to the user's group. If you do not explicitly assign a rating plan to a group, then the system **default rating plan** is automatically used for that group.

You can assign rating plans to users through either the CMC or the Admin API.

### Assigning Rating Plans to Groups and Users (CMC)

To use the CMC to assign a rating plan to a group, use the **Manage Groups** page (**Users & Groups -> Manage Groups**). From here you can create a new group, and while doing so assign the group a rating plan from a drop-down list of rating plans that exist in the system (the system default plan plus any plans you've created). From here you can also retrieve an existing group and change the group's rating plan assignment. Whichever rating plan you associate with a group will be applied to each user in the group, except for any users to whom you explicitly assign a rating plan.

To use the CMC to assign a rating plan to an individual user, use the **Manage Users** page (**Users & Groups -> Manage Users**). From here you can create a new user, and while doing so assign the user a rating plan from

a drop-down list. By default the new user is assigned whichever plan is assigned to the user's group. From the **Manage Users** page you can also retrieve an existing user and change her rating plan assignment.

### Assigning Rating Plans to Groups and Users (Admin API)

To use the HyperStore Admin API to assign a rating plan to a group, you must have first created the group (with the *PUT /group* method). Once a group exists, you can assign the group a rating plan by using the *POST /group/ratingPlanId* method. You could subsequently use this same API method to change a group's rating plan assignment. The Admin API also supports a *GET /group/ratingPlanId* method, for retrieving a group's current rating plan identifier.

The approach is similar if you want to assign a rating plan to an individual user. The user must have already been created (with *PUT /user*), and then you can use *POST /user/ratingPlanId* to assign the user a rating plan (and to subsequently update that assignment). The method *GET /user/ratingPlanId* lets you retrieve the identifier of the rating plan currently assigned to a specified user, and there's also a *GET /user/ratingPlan* method for retrieving the user's rating plan in full.

For details of these API calls, see the "group" and "user" sections of the ***Cloudian HyperStore Admin API Reference***.

## 4.10.3.2. Generating Billing Data for a User or Group

The HyperStore system supports generating a bill for a specified user or for a whole user group. The system applies the user's or group's assigned rating plan to their service usage during the billing period. Bills can be generated for any **completed calendar month** of service usage.

With the CMC you can generate a billing report for an individual user (but not for a whole group). With the Admin API you can generate billing data for a specified user or for a whole user group.

> **IMPORTANT !** Billing calculation is derived from hourly rollup usage data. The retention period for hourly rollup usage data is configured by *mts.properties.erb*: **"reports.rolluphour.ttl"** (page 456). The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills. Therefore by default you cannot generate a bill for a month that started more than 65 days ago.

### Generating Billing Data for a User (CMC)

To generate billing data through the CMC, use the **Account Activity** page (**Users & Groups -> Account Activity**). In this page you can specify a user and select a billing period. The most recent period you can select is the most recently completed calendar month. You cannot generate a billing report for an in-progress month.

The billing data that you can generate from this page displays in the form of a printable billing document that includes the user's name and user-ID, their user group, the billing period, and the bill generation date. The document shows a summary of the user's rating plan, the user's service activity for the billing period, and the associated charges.

The **Account Activity** page also provides you an option to view a user's service traffic originating from allowlist domains, if any.

### Generating Billing Data for a User or Group (Admin API)

To generate user or group billing data through the HyperStore Admin API, use the *POST /billing* method. This triggers the calculation of the billing data for the specified calendar month, and returns the billing data as a

JSON-encoded response payload.

The Admin API method also supports a *GET /billing* method, which simply retrieves billing data that you've previously generated with the *POST /billing* method. Like the *POST /billing* method, the *GET /billing* method returns billing data as a JSON-encoded response payload.

For details on these API calls, see the "billing" section of the ***Cloudian HyperStore Admin API Reference***.

# 4.11. Customizing the CMC

## 4.11.1. Showing/Hiding CMC UI Functions

The CMC provides granular configuration control over which UI functions and sub-functions display for the three types of users that the CMC supports: system admins, group admins, and regular S3 service users. The table below summarizes the CMC's configurability for showing or hiding certain functionality. The third column indicates which user types have access to the functionality by default. The fourth column indicates the configuration setting that controls access to the functionality — all settings are in *mts-ui.properties.erb* on your Configuration Master node unless otherwise noted. You can edit these settings if you want the availability to be different than the default -- for example, any of the functionalities that by default display for system admins and group admins can be reconfigured to display only for system admins.

**Note** After making changes to a configuration file, use the installer to push the changes out to your cluster and restart the CMC service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

| Function Area | Functionality | Default Availability | Controlling Setting |
|---|---|---|---|
| Manage Users | Whole user management interface | System admins and group admins | **"admin.manage_users.enabled"** (page 476) |
| | Create users | System admins and group admins | admin.manage_users.create.enabled |
| | Edit users | System admins and group admins | admin.manage_users.edit.enabled |
| | Delete users | System admins and group admins | admin.manage_users.delete.enabled |
| | View and manage users' security credentials | System admins and group admins | **"admin.manage_users.edit.user_credentials.enabled"** (page 478) |
| | View and manage users' stored data | System admins and group admins | **"cmc_view_user_data"** (page 424)<br>in *common.csv* |
| | Edit users' Quality of Service settings | System admins and group admins | **"admin.manage_users.edit.user_qos.enabled"** (page 478) |
| Manage Groups | Whole group management interface | System admins and group admins | **"admin.manage_groups.enabled"** (page 478) |
| | Create groups | System admins | **"admin.manage_groups.create.enabled"** |

| Function Area | Functionality | Default Availability | Controlling Setting |
|---|---|---|---|
| | | | (page 479) |
| | Edit groups | System admins and group admins | **"admin.manage_groups.edit.enabled"** (page 479) |
| | Delete groups | System admins | **"admin.manage_groups.delete.enabled"** (page 479) |
| | Set default user QoS for a group | System admins and group admins | **"admin.manage_groups.user_qos_groups_default.enabled"** (page 480) |
| Billing whitelist | Set source IP addresses allowed free traffic | Hidden from all user types | **"admin_whitelist_enabled"** (page 420) in *common.csv* |
| User Account Self-Management | Edit own profile | System admins, group admins, and regular users | **"account.profile.writeable.enabled"** (page 480) |
| | Whole security credentials interface | System admins, group admins, and regular users | **"account.credentials.enabled"** (page 480) |
| | Manage own S3 credentials | Group admins and regular users | **"account.credentials.access.enabled"** (page 481) |
| | Change own CMC password | System admins, group admins, and regular users | **"account.credentials.signin.enabled"** (page 481) |
| Usage Reporting | Whole usage reporting interface | System admins, group admins, and regular users | **"usage.enabled"** (page 482) |
| | Reporting on HTTP request rates and byte transfer rates | Hidden from all user types | Track/Report Usage for Request Rates and Data Transfer Rates in the CMC **Configuration Settings** page |
| Auto-Tiering | Allow buckets to be configured for auto-tiering | Hidden from all user types | Enable Auto Tiering in the CMC **Configuration Settings** page |

**See also:**

- **"Configuring a Login Page Banner"** (page 232)
- **"Configuring a Login Page Acknowledgment Gate"** (page 234)
- **"Rebranding the CMC UI"** (page 236)
- **"Implementing Single Sign-On for the CMC"** (page 239)

## 4.11.2. Configuring a Login Page Banner

The CMC supports two types of customizations specifically for the login page:

- You can configure a text banner that displays at the top of the login page every time any user accesses the CMC.
- You can configure an acknowledgment gate that displays as an overlay in front of the login page every time any user accesses the CMC, and requires that the user acknowledge having read the gate text before they can log into the CMC.

By default the CMC login page has no banner and no acknowledgment gate. You can enable either one of these customizations, or enable both of them. This section describes how to configure a login page banner; for instructions for configuring an acknowledgment gate see **"Configuring a Login Page Acknowledgment Gate"** (page 234).

Here is an example in which a custom text banner has been added to the login page.


To configure a custom banner for the CMC login page:

1. Log in to the Configuration Master node as *root*.

   *If you are using the **HyperStore Shell***

   As an alternative to logging in as *root* you can log into the **HyperStore Shell** (HSH) on the Configuration Master node to perform this procedure, so long as you are an HSH **Trusted user**. In the steps below that call for using the *rebrand_cmc.sh* script, run the script simply as *rebrand_cmc.sh <command>* without specifying a path to the script.

2. Create a working directory, and change into that directory.
3. Run the following command:

   ```
   /opt/cloudian/tools/rebrand_cmc.sh  --backup
   ```

   This creates under your working directory a backup sub-directory named as *web_backup_<timestamp>*, with some relevant files in it.

4. Copy the *custom_banner.jsp* file from the backup sub-directory into the working directory.
5. In the working directory, use a text editor such as ***vi*** to make the following edits to the *custom_banner.jsp* file:
   a. Uncomment the starting *style* tag.

Default in *custom_banner.jsp* file (with starting *style* tag commented out):

```
<!-- style>
```

After uncommenting:

```
<style>
```

b. Replace *Title* with your desired banner title text. Do not alter the *h2* tags.

```
<h2>Title</h2>
```

c. Replace *Message* with your desired banner body text. Do not alter the *p* tags.

```
<p>Message</p>
```

6. After saving your changes and exiting the file, while still in the working directory run the following command:

```
/opt/cloudian/tools/rebrand_cmc.sh  --custom_banner
```

This copies your edited *custom_banner.jsp* file to the appropriate Puppet configuration directory.

7. Use the installer to push your changes out to the cluster and then restart the CMC. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

To verify that the banner is displaying as desired, go to the CMC in your browser.

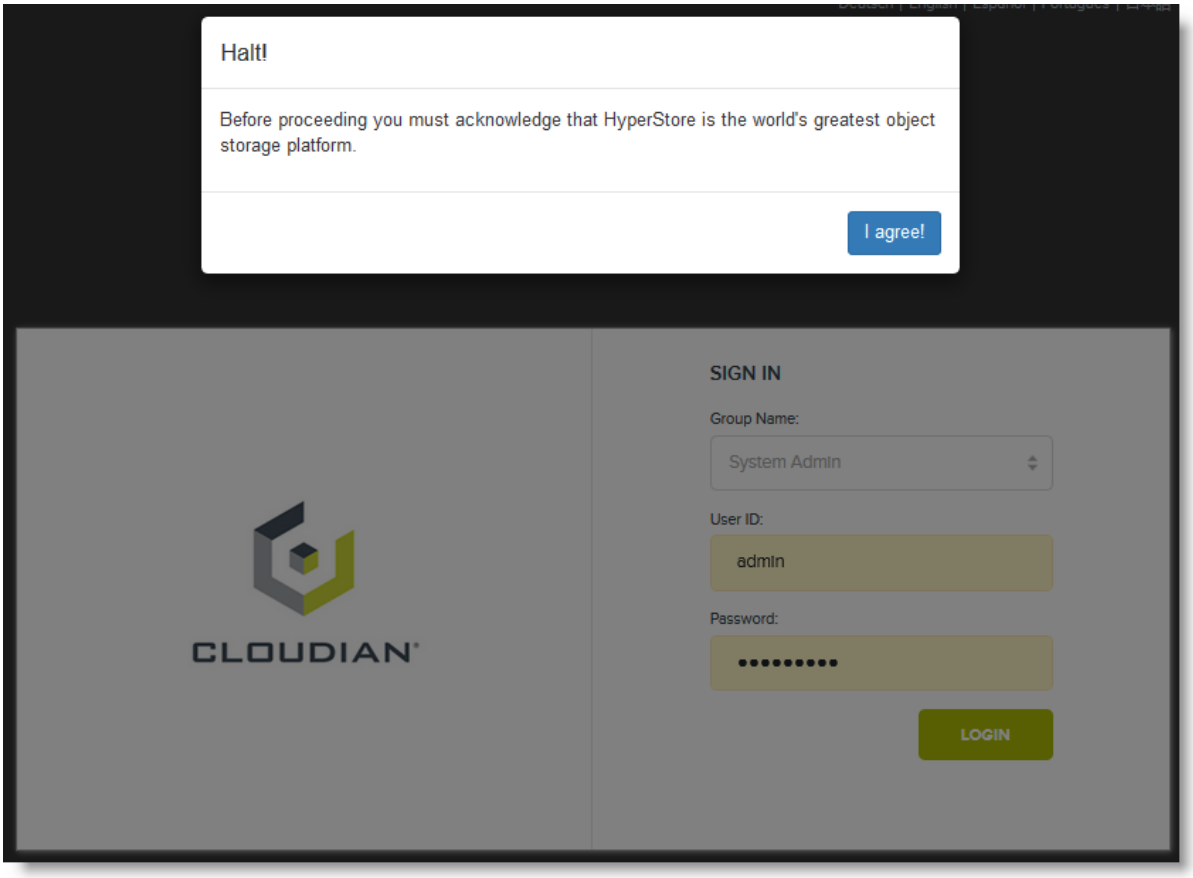
## 4.11.3. Configuring a Login Page Acknowledgment Gate

The CMC supports two types of customizations specifically for the login page:

- You can configure a text banner that displays at the top of the login page every time any user accesses the CMC.
- You can configure an acknowledgment gate that displays as an overlay in front of the login page every time any user accesses the CMC, and requires that the user acknowledge having read the gate text before they can log into the CMC.

By default the CMC login page has no banner and no acknowledgment gate. You can enable either one of these customizations, or enable both of them. This section describes how to configure an acknowledgment gate; for instructions for configuring a plain login page banner that does not require acknowledgment **"Configuring a Login Page Banner"** (page 232).

Here is an example in which an acknowledgment gate has been added to the login page. The CMC implements the acknowledgment gate as a modal dialog.




To configure a CMC login page acknowledgment gate:

1. Log into the Configuration Master node as *root*.

   *If you are using the **HyperStore Shell***

   As an alternative to logging in as *root* you can log into the **HyperStore Shell** (HSH) on the Configuration Master node to perform this procedure, so long as you are an HSH **Trusted user**. To open the *common.csv* file for editing as is required in this procedure you can use the command *hspkg config -e common.csv*. This command opens the file with the *vi* text editor.

2. Open the configuration file **common.csv** in a text editor.
3. Edit the following settings as desired:

   **Note** You do not need to enclose any of the setting text in quotes.

   - *cmc_login_banner_size* -- This setting controls the width of the acknowledgment dialog. The valid values are 0, 1, 2, or 3, with 0 being the narrowest and 3 the widest. The default is 1. In the screen shot above, the width is set to 1.
   - *cmc_login_banner_title* -- Title text of the acknowledgment dialog.
   - *cmc_login_banner_message* -- Body text of the acknowledgment dialog. If you want to apply any formatting to this text, use HTML format tags within the text. For example, *<br><br>* to start a second paragraph with a line break between paragraphs; or *text* to create bold text.
   - *cmc_login_banner_button_confirm* -- Text of the "confirm" button in the acknowledgment dialog. For example, you could use *Confirm* or *Acknowledge* or *Agree* or *OK* as the button text.

4. After saving your changes and exiting the file, push your changes to the cluster and then restart the CMC. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

To verify that the acknowledgment gate is working as desired, go to the CMC in your browser.

## 4.11.4. Rebranding the CMC UI

If you wish you can customize various aspects of the CMC interface to reflect your organization's own branding. The interface elements that you can customize are:

- Logo (default is the Cloudian company logo)
- Browser tab title text (default is "Cloudian® Management Console")
- Color scheme (default is the Cloudian color scheme with black, white, gray, and green)
- Application name in URLs (default is "Cloudian")

To rebrand any or all of these interface elements you will use a HyperStore tool that simplifies the process of putting the relevant files into the proper location.

Before starting, decide whether you want a change of logos to be part of your rebranding of the CMC UI. If so, you will need three images of your organization's logo, using the following file names and pixel sizes:

| Logo Image | Required Image File Name | Size in Pixels |
|---|---|---|
| Login screen logo | *logo_new.png* | 225 X 225 |
| Header logo | *logo_2.png* | 144 X 28 |
| Favicon for browser tab | *favicon.ico* | 16 X 16 |

Before starting the rebranding procedure below, you should have the three image files on your local machine (for instance a laptop computer from which you will connect to the Configuration Master node).

To rebrand the CMC UI, follow these steps:

1. Log into the Configuration Master node as *root*.

   *If you are using the **HyperStore Shell***

   As an alternative to logging in as *root* you can log into the **HyperStore Shell** (HSH) on the Configuration Master node to perform this procedure, so long as you are an HSH **Trusted user**. In the steps below that call for using the *rebrand_cmc.sh* script, run the script simply as *rebrand_cmc.sh <command>* without specifying a path to the script.

2. On the Configuration Master node create or choose a working directory from which you will manage the process of rebranding the CMC UI. Then change into the working directory.
3. Run the following script command to back up the current CMC files that are relevant to the UI's branding.

   ```
   /opt/cloudian/tools/rebrand_cmc.sh --backup
   ```

   This action creates a *web_backup_<timestamp>* sub-directory under your working directory on the Configuration Master node. In the backup directory are files copied from the CMC's Tomcat web server configuration, that affect the CMC's look and feel.

4. Make your desired changes to the CMC's branding

   *Replace logos*

a. Copy your organization's logo image files -- as specified in the introduction to this procedure -- from your local machine to the working directory on the Configuration Master node (by using *scp*, for example).

b. Change into working directory on the Configuration Master node, if you are not already there. Then run this script command:

```
/opt/cloudian/tools/rebrand_cmc.sh --images
```

This copies the image files from the working directory to the proper location within the Puppet configuration module for the CMC, so that you can subsequently push the files out to the whole cluster (as described in Step 6).

*Change browser tab title text*

By default the title text that displays at the top of a browser tab for the UI is "Cloudian® Management Console". To change this title, do the following:

a. On the Configuration Master node, copy the *resources.properties* file (for English language) and any of the *resources_<language-code>.properties* files (for other languages that the CMC supports, if applicable to your user population) from the backup sub-directory that you created in Step 3 into the working directory.

b. Use a text editor to edit the copy of *resources.properties* in the working directory, as follows:

   i. In the file find the following line:

   ```
   header.title=Cloudian&reg; Management Console
   ```

   ii. Change it to the desired browser tab name:

   ```
   header.title=<NEW TAB NAME>
   ```

c. Make the same change in the *resources_<language-code>.properties* files that you've copied into the working directory (if any).

d. After editing the file(s) and saving your changes, while still in the working directory run this script command:

```
/opt/cloudian/tools/rebrand_cmc.sh --resources
```

This copies the resource file(s) from the working directory to the proper location within the Puppet configuration module for the CMC, so that you can subsequently push the file(s) out to the whole cluster (as described in Step 6).

*Change UI color scheme*

To change the UI's color scheme -- as defined in its CSS file -- follow the steps below. (This assumes some familiarity with CSS files, and knowledge of your organization's preferred color scheme.)

a. On the Configuration Master node, copy the *master.css* file from the backup sub-directory that you created in Step 3 into the working directory.

b. Use a text editor to edit the copy of *master.css* file in the working directory, to replace all occurrences of the existing color codes with the desired new values.

c. After editing the file and saving your changes, while still in the working directory run this script command:

```
/opt/cloudian/tools/rebrand_cmc.sh --css
```

This copies the CSS file from the working directory to the proper location within the Puppet configuration module for the CMC, so that you can subsequently push the file out to the whole cluster (as described in Step 6).

*Change the application name in the CMC's URLs*

By default all CMC URLs include the application name "Cloudian" (for example *https://<host>:8443/Cloudian/dashboard.htm*). To change the application name, do the following:

a. On the Configuration Master node, use a text editor to open the configuration file */etc/cloudian-7.5.2-puppet/manifests/extdata/common.csv* and edit this setting:

```
cmc_application_name,Cloudian
```

Replace *Cloudian* with a different application name string of your choosing. Use only alphanumeric characters, with no spaces, dashes, or underscores.

b. Save the updated *common.csv* file.

5. To confirm that the rebranding script has successfully copied your customized image, resource, and/or CSS files to the Puppet configuration module, run this script command:

```
/opt/cloudian/tools/rebrand_cmc.sh --list
```

This should list all the image, resource, and/or CSS files that you worked with in Step 4. (It will not list the *common.csv* file.)

6. Use the installer to push your customized file(s) to the cluster and to restart the CMC to apply the changes. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**Note** If you customize the branding of the CMC, and then subsequently upgrade your HyperStore system to a newer version, only your customized logos and your customized application name will be retained after the upgrade. After the upgrade you will need to re-implement any changes that you had made to the browser tab title and/or the color scheme, by again following the instructions above.

**Note** The *rebrand_cmc.sh* script also supports adding a custom banner to the top of the CMC login page. For instructions see **"Configuring a Login Page Banner"** (page 232).

### 4.11.4.1. Rebranding the CMC Help

The CMC has three Help systems attached to it: one for system administrators, one for group administrators, and one for end users. The Help that displays for a logged-in CMC user depends on which of those three types of user he or she is.

Each of the three Help systems has a Cloudian logo. For each Help system the logo image file is named *CloudianLogoFull_2.png* and its size is 235 X 44 pixels.

To replace the Cloudian logo in the CMC Help with your organization's logo, replace these three instances of the logo file on each of your HyperStore nodes:

- /opt/cloudian-packages/apache-tomcat-7.0.103/webapps/Cloudian/help/HyperStoreHelp/Skins/Default/Stylesheets/Images/CloudianLogoFull_2.png

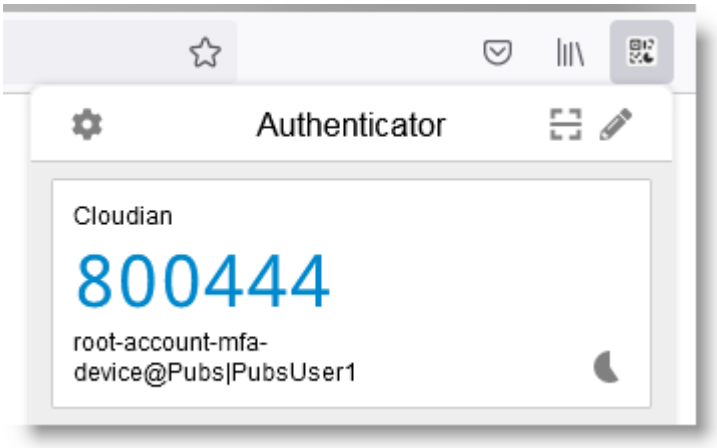
- /opt/cloudian-packages/apache-tomcat-7.0.103/webapps/Cloudian/help/HyperStoreHelpGroupAdmin/Skins/Default/Stylesheets/Images/CloudianLogoFull_2.png
- /opt/cloudian-packages/apache-tomcat-7.0.103/webapps/Cloudian/help/HyperStoreHelpEndUser/Skins/Default/Stylesheets/Images/CloudianLogoFull_2.png

Note that:

- Changing this image file is not supported by the *rebrand_cmc.sh* script, and this image file is not under Puppet control. To replace this image file you must do it manually on each of your HyperStore nodes.
- Changing this image file requires *root* access and cannot be performed through the HyperStore Shell.
- In the Help for group administrators and the Help for regular users, the text makes no reference to "Cloudian" or "HyperStore" or "the CMC". Instead the text uses generic, non-branded terminology such as "the storage system" and "the console".

### 4.11.4.2. Rebranding the MFA "Issuer"

Users can enable multi-factor authentication (MFA) on their CMC login accounts (see **"CMC Multi-Factor Authentication"** (page 166)). In the configuration file **"common.csv"** (page 388) the *mfa_totp_issuer* setting specifies the MFA TOTP (time-based one-time password) Issuer name that will display in a user's virtual MFA device if the user enables MFA on their CMC account. The example below is from the virtual MFA device Google Authenticator, with the Issuer at its default value "Cloudian".




If you wish you can change this setting's value in *common.csv* (so that your own organization's name is used rather than "Cloudian"). To apply your change do a Puppet push and then restart the IAM Service.

## 4.11.5. Implementing Single Sign-On for the CMC

To enable integration between a portal and the Cloudian Management Console, the Cloudian HyperStore system employs a one-way hash based Single Sign-On (SSO) solution. It allows for cross-domain sign-ons from the portal to CMC.

User provisioning is beyond the scope of the provided SSO solution. The HyperStore provides an Admin API for user provisioning (see the "user" section of the ***Cloudian HyperStore Admin API Reference***) but the implementation of user mapping is left to the portal application integrating with CMC.

### 4.11.5.1. How SSO for the CMC Works

Cloudian HyperStore SSO is ideal for sites that already have an authentication model in place using a browser-/login session and that want to incorporate the Cloudian Management Console into their web portal application.

The idea is that a portal application calculates an one-way hash (also known as a signature) based on Cloudian HyperStore user identification, timestamp and the shared key. Then the user's browser accesses *sso-securelogin.htm* with the signature. The CMC checks for this signature to determine whether a user is authenticated or not. If the signature is found valid, access to the CMC from the client will skip the login page and take the user directly to a CMC interior page such as the **Buckets & Objects** page.

**IMPORTANT !** To use the Cloudian HyperStore SSO feature, the following system configuration settings must be set to "true":

-- **"cmc_web_secure"** (page 421) in *common.csv* -- This is set to true by default. Leave it true if you want to use SSO.

-- **"cmc_sso_enabled"** (page 426) in *common.csv* -- This is set to false by default. Change it to true if you want to use SSO.

Also in *common.csv*, if you enable SSO functionality (by setting *cmc_sso_enabled* to true), then for security reasons you should set *cmc_sso_shared_key* and *cmc_sso_cookie_cipher_key* to custom values. **Do not use the default keys**.

### 4.11.5.2. CMC SSO Secure Login Using One-Way Hash

The single sign-on with one-way hash method relies on a one-way hash of query string parameters (also known as a signature).

The following HTTP API, using a signature, prompts the CMC to create an authenticated session for the client that submitted the request:

**Note** Submit this as a GET, not a POST. POST is not supported for CMC SSO login.

```
https://<cmc_FQDN>:<cmc_port>/Cloudian/ssosecurelogin.htm?user=USERID&group=GROUPID
&timestamp=TIMESTAMP&signature=SIG&redirect=RELATIVE_OR_ABSOLUTE_URL
```

- *user*: Cloudian HyperStore userId of the user
- *group*: Cloudian HyperStore groupId of the group to which the user belongs
- *timestamp*: Current Epoch time in milliseconds (eg. "1346881953440"). The timestamp is used to implement the configurable request expiration (mts-ui.properties: sso.tolerance.millis; expiration defaults to one hour).
- *signature*: This is the URI encoding of the base64 representation of the calculated signature. For further information see below.
- *redirect*: This optional parameter can be used to redirect the client to the given URL upon successful sign-in. It is typically set to a CMC interior page such as *bucket.htm*.

Each value must be URL-encoded by the client. Order of the parameters does not matter.

If the signature is found valid, the CMC creates an authenticated session for the HyperStore user, allowing the client to skip the login page and access to a CMC interior page.

### How to Create the Signature

The portal server can create the signature by the following steps.

1. Assemble the query string.
    - querystring = "user=USERID&group=GROUPID&timestamp=TIMESTAMP"

    > **Note** When using the querystring to create the signature, do not URL-encode the querystring. Also do not reorder the items. (By contrast, when the client subsequently submits the SSO secure login request to the CMC, it's desirable to URL encode the request querystring.)

2. Calculate one-way hash for the querystring using the standard HmacSHA1 and the CMC SSO shared key. The shared key is configured by *common.cscv: cmc_sso_shared_key*.
    - hashresult = HmacSHA1(querystring, sharedkey)
3. Base64 encode the resulting hash.
    - base64string = Base64Encode(hashresult)
4. URI encode the base64 encoded hash result.
    - signature = encodeURIComponent(base64string)

For a sample of a Python script that uses the one-way hash login API, see **"Cloudian HyperStore SSO Sample Script"** (page 242).

### Access to a CMC's Interior Page

After creating the signature, the portal server can return an HTML page with a hyperlink to the CMC SSO secure login API. The following example will display CMC's **Buckets & Objects** page (*bucket.htm*) embedded in the inline frame on the portal's page.

```
<iframe src="https://<cmc_FQDN>:<cmc_port>/Cloudian/ssosecurelogin.htm
?user=USERID&group=GROUPID&timestamp=TIMESTAMP&signature=SIG
&redirect=bucket.htm"></iframe>
```

### CMC SSO Secure Login HTTP Response

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is given, the CMC's SSO secure login API returns an HTTP redirect response.

- **If the request was successful**, the redirect response will take the client to the URL specified by *redirect*.
- **If the request failed**, the redirect response will take the client to the CMC's Login panel.

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is not given, the CMC's SSO secure login API returns an HTTP response with content-type "text/plain".

- **If the request was successful**, the HTTP response status is 200 OK.
- **If the request failed**, a 400 BAD REQUEST status is returned, along with a plain text status description. Possible reasons for failure include:

- Missing required parameters
- SSO token already exists (request is ignored)
- Timestamp in request is outside of configured tolerance range
- Invalid signature
- Invalid credentials (group ID and/or user ID is invalid)

### CMC Logout

This API method allows for immediately invalidating the CMC session.

```
https://<cmc_FQDN>:<cmc_port>/Cloudian/logout.htm&redirect=RELATIVE_OR_ABSOLUTE_URL
```

- *redirect*: This optional parameter can be used to redirect the client to the URL after logging out from the CMC. It is typically set to a portal page. The URL must be URL-encoded by the client.

### CMC Logout HTTP Response

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is given, the CMC's logout API returns an HTTP redirect response to take the client to the given URL after logging out from the CMC.

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is not given, the CMC's logout API returns an HTTP redirect response to take the client to the CMC's Login panel.

### Logging Out from the CMC and Portal at Once

You may want the logout link on the portal page to also trigger logout from the CMC. You can achieve this by using the *redirect* parameter.

For example, if you have the portal's logout link like this:

```
<a href="/auth/logout">Logout</a>
```

You can change it to the following:

```
<a href="https://<cmc_FQDN>:<cmc_port>/Cloudian/logout.htm
?redirect=https:%2F%2F<portal_FQDN>:<portal_port>%2Fauth%2Flogout">Logout</a>
```

- The redirect URL must be an absolute URL including the protocol (e.g. https://) and portal's FQDN.
- The redirect URL must be URL-encoded.

## 4.11.5.3. Cloudian HyperStore SSO Sample Script

Below is a sample Python script that outputs a HyperStore SSO secure login URL for use with the one-way hash method of having the CMC create a cookie. The script also creates an SSO logout URL.

```
#!/usr/bin/python

import time
import hmac
import hashlib
import base64
import urllib

# TODO: Move these config options to configuration file
SSO_DOMAIN = 'cmc.cloudian.com'
SSO_PORT = 8443
```

```
SSO_KEY = 'aa2gh3t7rx6d'

# TODO: Dynamically choose user/group based on the user
# and group you want to login using.
SSO_USER = 'sso@group'
SSO_GROUP = 'ssogroup'

# Do Not Change
SSO_PROTO = 'https://'
SSO_PATH = 'Cloudian/ssosecurelogin.htm'
SSO_LOGOUT_PATH = 'Cloudian/ssologout.htm'

def sso_sig(user, group, timestamp):
   # query string with no urlencoding for signature
   signme = 'user=%s&group=%s&timestamp=%s' % (user, group, timestamp)
   hmacsha1 = hmac.new(SSO_KEY, signme, hashlib.sha1).digest()
   return base64.b64encode(hmacsha1)

def sso_url(user, group):
   timestamp = int(time.time() * 1000)
   signature = sso_sig(user, group, timestamp)
   params = {'user': user,
      'group': group,
      'timestamp': timestamp,
      'signature': signature}
   query = urllib.urlencode(params)
   url = '%s%s:%d/%s?%s' % (SSO_PROTO, SSO_DOMAIN, SSO_PORT, SSO_PATH, query)
   return url

def sso_logout_url():
   url = '%s%s:%d/%s' % (SSO_PROTO, SSO_DOMAIN, SSO_PORT, SSO_LOGOUT_PATH)
   return url

print 'login: ' + sso_url(SSO_USER, SSO_GROUP)
print '\nlogout: ' + sso_logout_url()
```

**Note** The sample script hard-codes the SSO secret key, which is not advisable for actual practice. In practice, you should keep the secret key safely on the server side.

# 4.12. Provisioning Groups and Users

## 4.12.1. Group and User Provisioning Feature Overview

Through the HyperStore Admin API or through the CMC, you can provision the user groups and individual users who you want to authorize to use the HyperStore S3 service. **You will provision groups first, and then you can add individual users to each group.** All users must belong to a group.

As a system administrator you can act on groups in a variety of ways:

- Each group can be configured for integration with an external LDAP system as a means of authenticating users, if applicable to your environment. For more information see **"LDAP Integration"** (page 247).
- Each group can be assigned quality of service (QoS) limits that will enforce upper bounds on the service usage levels of the group as a whole. Each group can also be assigned default user-level QoS controls that will limit the service usage of individual users within the group. (Optionally, you can also assign per-user QoS limits that will supersede this default.)

  **Note** You can set system-wide defaults for group QoS limits and individual user QoS limits before provisioning any groups or users, if you wish. After provisioning groups and users you can set QoS limits specific to those groups and users, which would override the system-wide defaults. For more information see **"Quality of Service (QoS) Feature Overview"** (page 213).
- You can generate service usage reports for groups (and also for individual users).
- Each group can be assigned a default rating plan which will determine how users in that group will be charged for HyperStore service usage. (Optionally, you can also assign per-user rating plans that will supersede this default.)
- You can create one or more users who have group administrator privileges. By default system configuration, group administrators are able to perform the following operations through the CMC:
  - Create a user within the group
  - Edit a user's profile
  - Retrieve a list of users in the group
  - Assign user-specific QoS limits
  - Provide user support by accessing a user's data in the S3 object store
  - Delete a user
  - Generate a usage report for the group
  - Generate a usage report for an individual user in the group

  **Note** The set of privileges that you make available to group administrators is configurable in a granular way (in the mts-ui.properties.erb file, see the *admin.manage_users.enabled* property and those that follow it). Individual CMC UI functions and sub-functions can be displayed to or hidden from group administrators depending on your configuration settings.

## 4.12.2. Provisioning Groups

You can provision user groups through either the CMC or the Admin API.

**Note** Optionally, when creating a group you can enable LDAP-based authentication of group members. For more information see **"LDAP Integration"** (page 247).

#### 4.12.2.1. Provisioning Groups (CMC)

In the CMC's **Manage Groups** page (**Users & Groups -> Manage Groups**) you can perform group operations including:

- Add a group
- Set quality of service (QoS) for a group or its users
- Retrieve a group or a list of group
- Edit a group
- Delete a group

For details, while on the CMC's **Manage Groups** page click **Help**.

#### 4.12.2.2. Provisioning Groups (Admin API)

Through the HyperStore Admin API you can perform group operations including:

- Create a new group: *PUT /group*
- Assign a rating plan to a group: *POST /group/ratingPlanId*
- Assign QoS limits to a group: *POST /qos/limits*
- Retrieve a list of groups: *GET /group/list*
- Edit a group (including suspending a group): *POST /group*
- Delete a group: *DELETE /group*

For details on these API calls see the "group" section of the ***Cloudian HyperStore Admin API Reference***.

### 4.12.3. Provisioning Users

You can provision individual users through the CMC, or the Admin API, or by enabling LDAP integration. This section covers the following topics:



**Note** The HyperStore system does not support bulk provisioning of users. Users must be added one at a time.

#### 4.12.3.1. Provisioning Users (CMC)

In the CMC's **Manage Users** page (**Users & Groups -> Manage Users**) you can perform user operations including:

- Create a new user
- Assign a rating plan to a user

- Assign QoS limits to a user
- Retrieve a list of users
- Update a user's profile (including suspending a user)
- Delete a user

For more details, while on the **Manage Users** page click **Help**.

### 4.12.3.2. Provisioning Users (Admin API)

Through the HyperStore Admin API you can perform user operations including:

- Create a new user: *PUT /user*
- Assign a rating plan to a user: *POST /user/ratingPlanId*
- Assign QoS limits to a user: *POST /qos/limits*
- Retrieve a list of users: *GET /user/list*
- Update a user's profile (including suspending a user): *POST /user*
- Delete a user: *DELETE /user*

For details on these API calls see the "user" section and the "qos" section in the ***Cloudian HyperStore Admin API Reference***.

### 4.12.3.3. Provisioning Users (LDAP)

As an alternative to provisioning users through the CMC or the Admin API, on a per-group basis you can enable integration between the CMC and your Active Directory or other LDAP system, such that users will be automatically provisioned within HyperStore when they log into the CMC with their LDAP credentials. For more information see **"LDAP Integration"** (page 247).

### 4.12.3.4. Provisioning IAM Users

Users provisioned by any of the methods noted above are **HyperStore account root users**. They have full permissions for performing S3 actions such as creating buckets and uploading and downloading objects, and they can perform S3 actions either through the CMC (which has a built-in S3 client) or through a third party S3 client application. Like Amazon Web Services, HyperStore allows account root users to create **IAM users** under their root accounts. IAM (Identity and Access Management) users differ from HyperStore account root users in these important ways:

- IAM users have only the permissions granted to them by IAM policies created by the account root user.
- IAM users cannot log into the CMC. To perform S3 actions, IAM users must use a third party S3 client application to access the HyperStore S3 Service.
- HyperStore does not track service usage data specifically for IAM users. Instead, an IAM user's S3 storage activity counts toward the HyperStore usage data of the account root user who created the IAM user.

HyperStore account root users can create IAM groups, users, and policies in the CMC's **IAM** section.For more information, while in any of the pages in the CMC's **IAM** section click **Help**.

HyperStore account root users can also create IAM groups, users, and policies by using a third party IAM client application that calls the HyperStore IAM Service. The HyperStore IAM Service supports all the IAM API calls

associated with creating and managing IAM groups, users, and policies. For more information see the IAM section of the ***Cloudian HyperStore AWS APIs Support Reference***.

#### 4.12.3.5. SAML-Based Assumption of IAM Roles

HyperStore account root users can create **IAM roles**, either through the CMC or by using a third party IAM client application that calls the HyperStore IAM Service. Those IAM roles, and IAM policies governing the roles, can be set up in a way such that the roles can be temporarily assumed by federated users who have been authenticated by Security Assertion Markup Language (SAML 2.0) identity provider systems external to HyperStore.

SAML-authenticated external users who temporarily assume an IAM role:

- Have only the permissions granted to them by the IAM policies associated with the role.
- Cannot log into the CMC. To perform S3 actions, SAML users temporarily assuming an IAM role must use a third party S3 client application to access the HyperStore S3 Service.
- Do not have their own service usage counts. Instead the S3 storage activity of such users counts toward the HyperStore usage data of the account root user who created the IAM role.

For more information see "SAML Support" in the IAM section of the ***Cloudian HyperStore AWS APIs Support Reference***.

### 4.12.4. LDAP Integration

*Subjects covered in this section:*



HyperStore supports integrating with Active Directory or other types of LDAP systems so that users can log into the CMC with their LDAP-based login credentials. This feature is implemented on a per-group basis, so you have the option of creating some groups that are LDAP-enabled and others that are not. The system also supports having different groups use different Active Directory or LDAP servers for authentication, or having all LDAP-enabled groups use the same Active Directory or LDAP server.

Within an LDAP-enabled group, along with users who the CMC will authenticate against an Active Directory or other LDAP system you can optionally also have local users who the CMC will authenticate by use of a CMC-based password rather than LDAP.

**Note**
* Under no circumstances does the CMC try to write to your Active Directory or LDAP server — it only reads from it, for the purpose of authenticating users.
* Only system administrators can enable Active Directory or LDAP authentication for a group. Group administrators cannot enable Active Directory or LDAP authentication for their groups.
* For LDAP authentication of HyperStore Shell users to work (see **"LDAP Authentication of System Administrators and HSH Users"** (page 249)), your LDAP server must use either TLS or START_TLS.

### 4.12.4.1. Enabling LDAP Authentication for a Group

To use the CMC to enable LDAP authentication for a group, use the **Add Group** interface (to create a new group with LDAP authentication enabled) or the **Edit Group** interface (to enable LDAP authentication for an existing group). For details, while logged into the CMC's **Manage Groups** page (**Users & Groups -> Manage Groups**) click **Help**.When creating or editing the group select the "Enable LDAP Authentication" option and provide the required Active Directory or LDAP information.

To use the Admin API to enable LDAP authentication for a group, use the *PUT /group* method (to create a new group with LDAP authentication enabled) or the *POST /group* method (to enable LDAP authentication for an existing group). When creating or editing the group, in the *GroupInfo* object in the request body set the *ldapEnabled* attribute to true and also set the other LDAP-related attributes. For details on these API calls see the "group" section of the ***Cloudian HyperStore Admin API Reference***.

**Note** If you enable LDAP Authentication for an **existing group** to which you have already added users via the CMC's **Add User** function, those existing users will continue to be authenticated by reference to their CMC-based passwords -- not by reference to an LDAP server. LDAP Authentication will apply only for new users.

**Note** If you wish you can enable LDAP Authentication for the **System Admin group**, by editing the group either through the CMC or the Admin API. For further information specific to this group, see **"LDAP Authentication of System Administrators and HSH Users"** (page 249).

### 4.12.4.2. Provisioning of Users within LDAP-Enabled Groups

Within a HyperStore group that has LDAP authentication enabled you can have both LDAP-authenticated users and users who are authenticated by a CMC-based password rather than LDAP:

- **For users who you want to be authenticated by Active Directory or LDAP, do not manually create those users** through the CMC (or the Admin API). Instead, simply have those users log into the CMC using their LDAP credentials. If a user tries to log into the CMC as a member of an LDAP-authenticated group and the user is not already registered in HyperStore as a member of the group, the CMC will attempt to authenticate the user against the LDAP system. If the authentication succeeds, the CMC will **automatically provision** the user into HyperStore. This includes automatic creation of security keys for accessing the HyperStore S3 data store. Going forward whenever the user logs in the CMC will recognize the user as a registered HyperStore user, but will continue to authenticate the user against the LDAP system each time rather than by reference to a CMC-based password.

  Note that for such users to be successfully provisioned, the user names that they use when logging into the CMC must satisfy HyperStore restrictions for user names:

  - Must be unique within the group.
  - Only letters, numbers, dashes, and underscores are allowed. No spaces or special characters.
  - Maximum allowed length is 64 characters.
  - Must not be any of the following: "anonymous", "public", "null", "none", "admin", "0". These names are reserved for system use.

  **Note** If you want the group administrator to be authenticated by LDAP, have the user log into

the CMC using their LDAP credentials. Once this occurs and the CMC automatically provisions the user, you can subsequently edit the user's profile (using the CMC's **Edit User** function or the Admin API's *POST /user* method) to promote them to the group admin role.

- For users who you want to be authenticated by a CMC-based password rather than by the LDAP system, create those users through the CMC's **Add New User** interface (or the Admin API's *PUT /user* method). The CMC will not use LDAP-based authentication for users created through the **Add New User** interface or the *PUT /user* method.

### 4.12.4.3. Deleting Users from the CMC and/or LDAP

After you've enabled LDAP integration and have been using this feature, the HyperStore system behaves in the following way in respect to users being deleted from the CMC and/or LDAP:

- If you delete a user from the CMC but that user still exists in LDAP, the user will be able to log in to the CMC as if they were a first-time user and the CMC will auto-provision the user once again. If you want to prevent a user from accessing the CMC and HyperStore, but the user still exists in LDAP, the thing to do is to **deactivate** the user in the CMC (through the CMC's **Edit User** function), rather than deleting them. This will prevent the user from logging into the CMC or accessing HyperStore storage, even though they still exist in LDAP.
- If you delete a user from LDAP but do not delete them from the CMC, the user will not be able to log into the CMC. However, they still have valid S3 credentials and can access the HyperStore storage layer through a different S3 client. If you want a user who you've deleted from LDAP to not have access to the HyperStore S3 system, you should delete them from CMC also (which prevents access and also deletes the user's stored data) or else deactivate them in the CMC (which prevents access but leaves their stored data in place).

### 4.12.4.4. Disabling LDAP Authentication After Having Used It

If you have LDAP enabled for a particular group for some period of time, and during that time LDAP-based users from the group logged into the CMC with their LDAP credentials and were auto-provisioned into the HyperStore system, and then you subsequently disable LDAP for that group — those auto-provisioned users will no longer be able to log into the CMC.

### 4.12.4.5. LDAP Authentication of System Administrators and HSH Users

HyperStore supports enabling LDAP authentication for the System Admin group, which is a pre-existing group in any HyperStore system. You can do this through the CMC's **Edit Group** interface or the Admin API's *POST /group* method, just as you would for any group. Just as with any other group, any users who already exist in the System Admin group at the time that you enable LDAP authentication for the group will continue to be authenticated by reference to their CMC-based password. Just as with any other group, after enabling LDAP authentication for the System Admin group, if you want a new system admin user to be authenticated by LDAP **do not manually create that user** through the CMC or the Admin API -- instead, have that user log into the CMC with his or her LDAP credentials, and the user will then be automatically be provisioned into HyperStore (see **"Provisioning of Users within LDAP-Enabled Groups"** (page 248)).

When using LDAP authentication for the System Admin group:

- The default System Admin user -- with user ID *admin* -- is considered a pre-existing user and can only be authenticated by reference to the CMC-based password for that user. LDAP authentication is not supported for the *admin* user.
- To edit the System Admin group via the CMC or the Admin API you will need to know the group ID, which is "0" (the number zero).
- When a new system admin user is auto-provisioned into HyperStore as an LDAP-authenticated user (upon their first login to the CMC with their LDAP credentials), and the system automatically creates a corresponding HyperStore Shell user -- so that the new system admin user can log into and use the HyperStore Shell -- that user's **logins to the HyperStore Shell will also be LDAP-authenticated**. That is, when logging into the HyperStore Shell the user will supply their LDAP username and password, and the system will verify those credentials against your LDAP service.
    - For "local" system admin users -- who have been manually added through the CMC or the Admin API and who are not configured for LDAP authentication -- their HyperStore Shell user name is their CMC login user name prefixed by "sa_" (such as "sa_admin2"). By contrast, for LDAP-authenticated system admin users their HyperStore Shell user name is simply the same user name that they use to log into the CMC (without any prefix).
    - For LDAP authentication of HyperStore Shell users to work, along with enabling LDAP for the System Admin group in the CMC's **Edit Group** interface (or through the Admin API's *POST /group* method) you must perform this additional configuration step:
        1. Log in to the Configuration Master node (as root or as a locally authenticated HyperStore Shell user).
        2. Set the Distinguished Name for binding to your LDAP service, and the password:

```
hsctl config set hsh.ldap.bindDN=<bind Distinguished Name>
hsctl config set hsh.ldap.bindPassword=<bind password>
hsctl config apply hsh
```

For more information on the HyperStore Shell see:

- **"HyperStore Shell (HSH) Feature Overview"** (page 95)
- **"Enabling the HSH and Managing HSH Users"** (page 96)
- **"Using the HSH"** (page 101)

**Note** For LDAP authentication of HyperStore Shell users to work, your LDAP server must use either TLS or START_TLS.

**Note** LDAP authentication for HyperStore Shell users works only for system admin users created in HyperStore version **7.2.3 and later**.

# Chapter 5. Cluster and Node Operations

## 5.1. Starting and Stopping Services

*Subjects covered in this section:*



You can start, restart, or stop a service across all the nodes in your cluster by using the HyperStore installer. If instead you want to start, restart, or stop a service on just one particular node, you can do so through the CMC or by using a HyperStore service initialization script. Note also that HyperStore services are configured to start automatically when you reboot a node.

**Note** If you have the File Service and/or the Search Service running on auxiliary nodes, and you need to start or stop either of those services or their corresponding K3s services, contact Cloudian Support for guidance.

### 5.1.1. Start or Stop Services on All Nodes in the Cluster

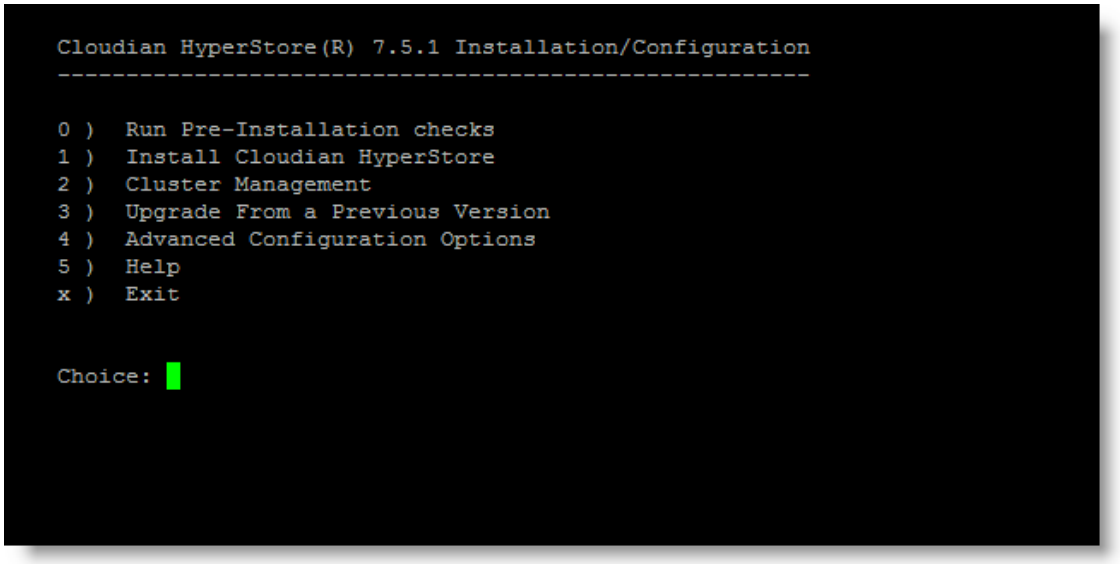
The interactive tool that you used to install the HyperStore system — *cloudianInstall.sh* — can also be used to manage HyperStore services. The tool enables you to start, restart, or start one or more services **on all nodes at once** (or to put it more precisely, on all nodes in rapid succession).

**Note** The installation tool does not support managing a service on just one particular node.

1. On your **Configuration Master node**, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

   This displays the top-level menu of options.



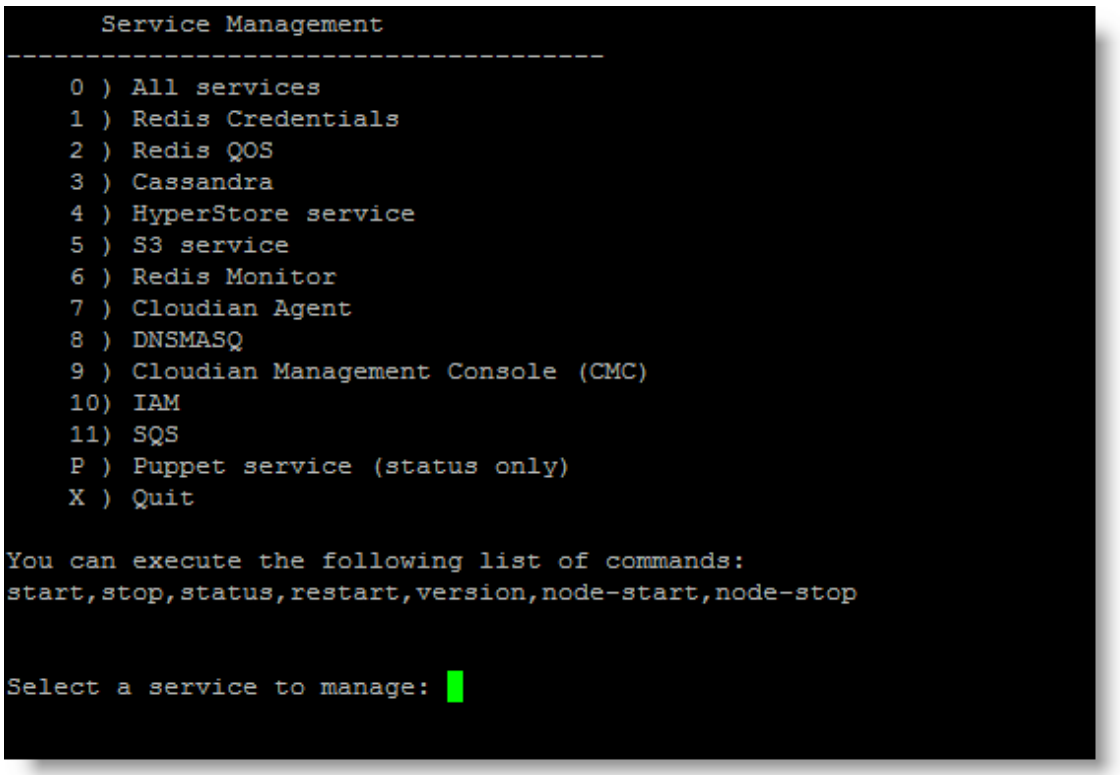

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

```
$ hspkg install
```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. Choose "Cluster Management", then from the sub-menu that displays choose "Manage Services". This displays the "Service Management" sub-menu:



> **Note** For future reference: As an alternative to launching *cloudianInstall.sh* and navigating to the "Service Management" menu, you can access this menu directly by launching the *cloudianService.sh* script in your installation staging directory.

a. At the prompt, enter a service number from the menu. To manage all services enter option (0).

> **Note** The Admin Service is bundled with the S3 Service. Any operation that you apply to the S3 Service (such as stopping or restarting) applies also to the Admin Service. Likewise, the STS Service is bundled together with the IAM Service, so any operation that you apply to the IAM Service also applies to the STS Service.

b. At the prompt that appears after you make your service selection, enter a service command: start, stop, status, restart, or version. (The "version" option is supported only for the S3 Service.)

The service command you enter will be applied to all nodes on which the service resides. For example, if you choose Cassandra (the Metadata DB) and then enter "start", this will start Cassandra on all nodes on which it is installed. Likewise if you choose S3 and then "status", this will return the status of the S3 Service on each node on which it is installed. And if you choose "All services" and then "stop", this will stop all services on all nodes.

> **Note** From the "Service Management" menu all you can do for the Puppet service (the Configuration Management service) is check its status. To stop or start the Puppet daemons (Configuration Agents), from the installer's main menu choose "Advanced Configuration Options". From the advanced sub-menu that displays you can stop or start the Puppet daemons.

## 5.1.2. Start or Stop Services on One Node

You can start, restart, or stop a service on just one particular node by using the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**).

As an alternative to using the CMC for this task, you can use the following *systemctl* commands. These commands can be run from any directory on the target node.

```
# systemctl start|restart|stop|is-active <servicename>
```

Example (restarting the S3 service):

```
# systemctl restart cloudian-s3
```

Example (checking the status of the S3 service):

```
# systemctl is-active cloudian-s3
active
```

*If you are using the **HyperStore Shell***

The **HyperStore Shell (HSH)** supports using *systemctl* commands such as those above.

The table below shows all HyperStore services and their corresponding *<servicename>*.

| Service | <servicename> |
|---|---|
| S3 Service and Admin Service. These two services start and stop together. | *cloudian-s3* |

| Service | <servicename> |
|---|---|
| IAM Service and STS Service. These two services start and stop together. | *cloudian-iam* |
| SQS Service | *cloudian-sqs*<br>(by default this service is disabled; see the introduction to the SQS section in the ***Cloudian HyperStore AWS APIs Support Reference***) |
| Cassandra Service (Metadata DB) | *cloudian-cassandra* |
| HyperStore Service | *cloudian-hyperstore* |
| Redis Credentials Service (Credentials DB) | *cloudian-redis-credentials* |
| Redis QoS Service (QoS DB) | *cloudian-redis-qos* |
| Redis Monitor | *cloudian-redismon* |
| Cloudian Management Console | *cloudian-cmc* |
| Cloudian Monitoring Agent | *cloudian-agent* |
| Dnsmasq | *cloudian-dnsmasq* |

## 5.1.3. Stop or Start All Services on One Node

**To stop all of the services on a single node,** stop each individual service (as described in **"Start or Stop Services on One Node"** (page 253)) in this order:

1. CMC
2. Cloudian Monitoring Agent
3. Redis Monitor
4. S3 Service
5. IAM Service
6. SQS Service (if being used)
7. HyperStore Service
8. Redis QoS
9. Redis Credentials
10. Cassandra
11. Dnsmasq

**To start all of the services on a single node**, start each individual service in this order:

1. Cassandra
2. Redis Credentials
3. Redis QoS
4. HyperStore Service
5. SQS Service (if being used)
6. IAM Service
7. S3 Service

8. Redis Monitor
9. Cloudian Monitoring Agent
10. CMC
11. Dnsmasq

### 5.1.4. Shutting Down or Rebooting a Node

You can shut down or reboot a HyperStore host machine by logging into the machine and using *systemctl* commands.

To power off:

```
# systemctl poweroff
```

To reboot:

```
# systemctl reboot
```

*If you are using the **HyperStore Shell***

The **HyperStore Shell (HSH)** supports using *systemtctl* commands such as those above.

> **IMPORTANT !** If you are rebooting multiple nodes, make sure that each node is back up for at least one second before moving on to reboot the next node.

### 5.1.5. Automatic Service Start on Node Boot-Up

By default all of the HyperStore services on a host are configured to automatically start when the host is booted up (the controlling configuration setting is *common.csv:* **"service_starts_on_boot"** (page 391), which defaults to "true"). This includes *dnsmasq* if it was included in your HyperStore software installation.

> **Note** *ntpd* is configured to automatically start on host boot-up. By design, the sequencing is such that *ntpd* starts up before major HyperStore services do.
>
> Cloudian Inc. recommends that after booting a HyperStore host, you verify that *ntpd* is running. You can do this with the *ntpq -p* command. If *ntpd* is running this command will return a list of connected time servers.

### 5.1.6. Automatic Service Restarts After a Crash (Disabled By Default)

Optionally, you can configure the system so that a service that crashes on a node is automatically restarted on that node. This feature is disabled by default. If you enable the feature it applies only to the S3 Service, Admin Service, HyperStore Service, and Cassandra Service (Metadata DB). For more information see **"service_restart_on_failure"** (page 391).

## 5.2. Upgrading Your HyperStore Software Version

*Subjects covered in this section:*

The instructions that follow are for upgrading to HyperStore version 7.5.2 **from HyperStore version 7.4 or newer**. This upgrade procedure does not require S3 service interruption.

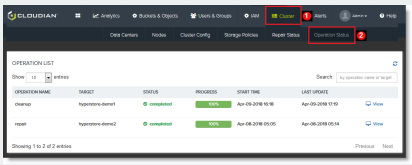
If you are already running HyperStore version 7.5.2 and are installing a patch release (with a 4-digit release number), you can jump directly to **"Installing a Patch"** (page 262).

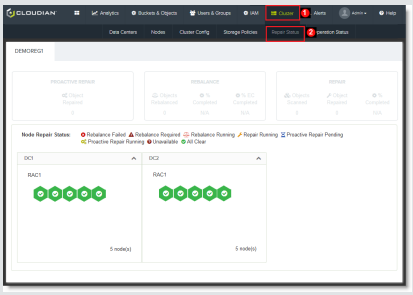
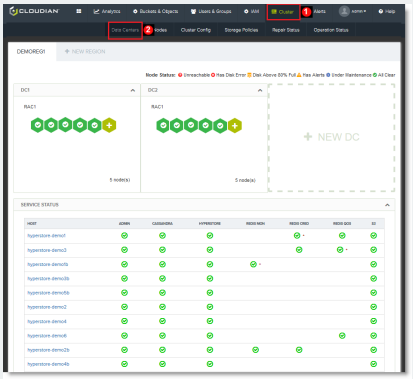
**IMPORTANT ! Contact Cloudian Support before upgrading if any of these conditions apply** to your existing HyperStore system:
* Your current HyperStore version is earlier than version 7.4.
* One or more of your existing HyperStore hosts has less than 128GB RAM.
* Your HyperStore hosts are using Xen or are in Amazon EC2.
* Your HyperStore data disks are using Logical Volume Manager (LVM).

**IMPORTANT !** HyperStore 7.5.1 introduces the HyperStore Search Service. If you have a legacy integration with Elasticsearch from an earlier version of HyperStore, after you upgrade to HyperStore 7.5.1 your integration with Elasticsearch will no longer work as is. For information about your options see **"Options if You Have a Legacy Integration with Elasticsearch"** (page 198).

## 5.2.1. Preparing to Upgrade Your System

Before upgrading your HyperStore system, log into the CMC and under the **Cluster** tab use the system status display pages to make sure that your system is in a fully healthy, unrestricted state and that no major operations are in progress.

| CMC Page | What to Look For |
|---|---|
| **Operation Status**<br>  | **Check whether there are any operations currently in progress** in any of your service regions.<br>• If any **repair** or **repairec** operation is in progress when you launch the upgrade script, the script will fail in the pre-check stage and will not make any changes to your system. So before launching the upgrade you must wait for any in-progress repair to complete (or alternatively you can stop the repair by using the "*-stop*" option that is supported by the **hsstool repair** and **hsstool repairec** commands).<br>• If any **cleanup, cleanupec, decommission,** or **rebalance** operation is in progress, Cloudian recommends that you wait until the operation completes before you perform the upgrade (or in the case of cleanup you can stop the cleanup by using the "*-stop*" option that is supported by the **hsstool cleanup** and **hsstool cleanupec** commands). |
| **Repair Status** | **Make sure there are no proactive repairs in progress**. Note that in- |

| CMC Page | What to Look For |
|---|---|
|  | progress proactive repairs will not display in the **Operation Status** page but will display in the **Repair Status** page.<br><br>If a proactive repair is in progress, either wait until it completes or alternatively you can stop the proactive repair using the **hsstool proactiverepairq** command with the "*-stop*" option. If a proactive repair is in progress, the upgrade script will fail in the pre-check stage and will not make any changes to your system. |
| **Data Centers**<br>  | **Make sure that all nodes and all services are up and that no node is in a restricted status**. If a service is down or a node is in one of the restricted statuses noted below, the upgrade script will fail in the pre-check stage and will not make any changes to your system.<br><br>• If any **services are stopped**, start them. For instructions see **"Starting and Stopping Services"** (page 251).<br>• If any node is in **Maintenance Mode**, take it out of Maintenance Mode. For instructions see Stopping Maintenance Mode.<br>• If any node is in **"stop-write" condition**, contact Cloudian Support before performing the upgrade.<br><br>Also, if the **Data Centers** page indicates that **any node has alerts**, go to the **Node Status** page and then select that node and review its alerts. Resolve any serious issues before proceeding with the upgrade. |

**Note** The upgrade script will automatically disable the auto-repair and proactive repair features -- so that no new repairs kick off during the upgrade -- and then after the upgrade completes the script will automatically re-enable the auto-repair and proactive repair features.

#### 5.2.1.1. Additional Upgrade Preparation If Your System Currently Has Failed Disks

By default the HyperStore upgrade script will abort if it detects that any disks on your HyperStore nodes are failed or disabled. If you want to perform a HyperStore version upgrade while there are failed or disabled disks in your current system, take the following preparation steps before doing the upgrade:

1. On the Configuration Master node, in the staging directory for your **current** HyperStore system, open this file in a text editor:

   ```
   CloudianInstallConfiguration.txt
   ```

2. In the file, change *INSTALL_SKIP_DRIVES_CHECK=false* to *INSTALL_SKIP_DRIVES_CHECK=true*. Then save and close the file.

#### 5.2.1.2. Additional Upgrade Preparation If You Are Upgrading From HyperStore 7.5.1

If you are upgrading to HyperStore 7.5.2 from HyperStore 7.5.1, follow the steps below before performing the upgrade. These steps are not applicable if you are upgrading to HyperStore 7.5.2 from HyperStore 7.5.0 or

7.4.x.

1. On the Puppet Master node, open this configuration file in a text editor:

   ```
   /etc/cloudian-7.5.1-puppet/modules/cloudians3/templates/hyperstore-server.properties.erb
   ```

2. Add the following line to the bottom of the configuration file:

   ```
   auto.repair.concurrent.enabled=false
   ```

   Then save your change and close the file.

3. Change into the installation staging directory (*/opt/cloudian-staging/7.5.1*) and launch the installer:

   ```
   # ./cloudianInstall.sh
   ```

4. Use the installer to push your configuration change out to the cluster: Option **2** "Cluster Management", then option **b** "Push Configuration Settings to Cluster" (all nodes).
5. Still on the "Cluster Management" menu, restart the HyperStore Service: Option **c** "Manage Services", then choose **4** "HyperStore service" and then type 'restart' at the command prompt.
6. After the HyperStore Service is restarted, exit the installer.

Now you can proceed with the upgrade as described below.

## 5.2.2. Upgrading Your System

To perform the upgrade to HyperStore version 7.5.2:

1. Download the HyperStore product package (*CloudianHyperStore-7.5.2.bin* file) from the Cloudian Support portal into a working directory on the Configuration Master node (such as */tmp* or your home directory -- **do not use the installation staging directory from your existing HyperStore system**). You can also download from the Support portal the signature file (*.sig* file) corresponding to the product package file -- you will need the signature file if you are using the HyperStore Shell to perform the upgrade.
2. Copy your **current Cloudian license file** into the same working directory as the product package and signature file. Your current license file is located in the */opt/cloudian/conf* directory and the file name ends with suffix *.lic*. If there are multiple *.lic* files in this directory, use the most recent one. Copy this file to the working directory in which you've placed the new HyperStore product package.

   > **Note** The license file must be your cluster-wide license that you obtained from Cloudian, not a license for an individual HyperStore Appliance machine (not a *cloudian_appliance.lic* file).

3. In the working directory run the commands below to unpack the HyperStore package:

   ```
   # chmod +x CloudianHyperStore-7.5.2.bin
   # ./CloudianHyperStore-7.5.2.bin <license-file-name>
   ```

   **This creates a new installation staging directory** named */opt/cloudian-staging/7.5.2*, and extracts the HyperStore package contents into the staging directory.

   ***If you are using the HyperStore Shell** to perform the upgrade*

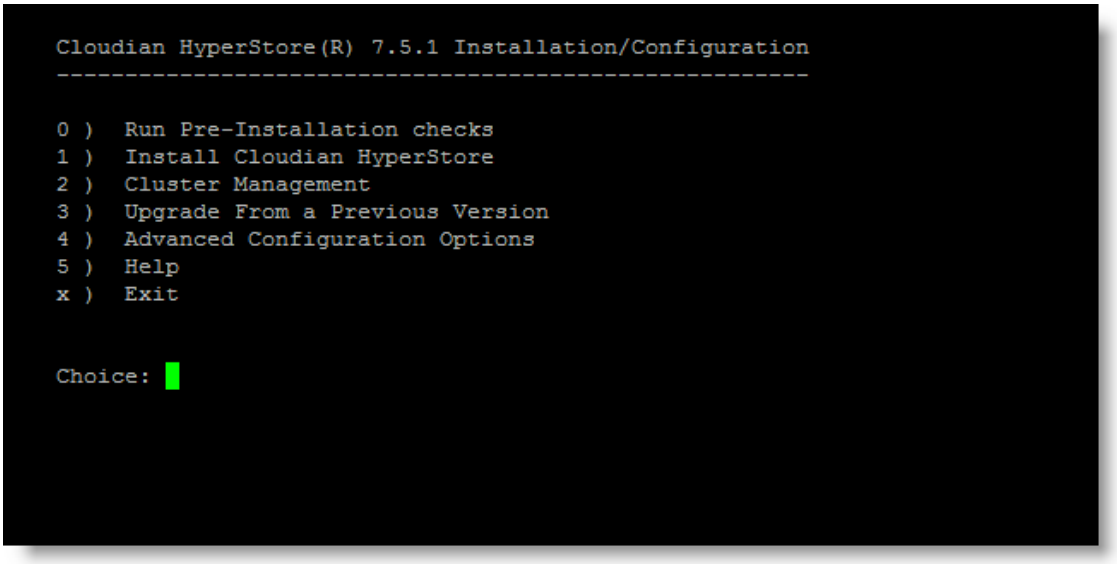
   If you are using the **HyperStore Shell (HSH)** to perform the upgrade, make sure you have in your working directory the signature file (*.sig* file) corresponding to the product package file, as well as the product package file and your current Cloudian license file (as described in Steps 1 and 2). Then from the shell, run these commands to unpack the HyperStore package (rather than the commands stated above):

```
$ chmod +x CloudianHyperStore-7.5.2.bin
$ hsrun --root CloudianHyperStore-7.5.2.bin <license-file-name>
```

**Note** To perform the upgrade using the HSH you must be an HSH **Trusted user**.

4. **Change into the new installation staging directory** and then launch the installer:

```
# cd /opt/cloudian-staging/7.5.2
# ./cloudianInstall.sh
```




*If you are using the HyperStore Shell to perform the upgrade*

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

```
$ hspkg install
```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

5. From the installer main menu enter "**3**" for "Upgrade from a Previous Version". Then at the prompt, confirm that you wish to continue with the upgrade.

The upgrade script will first check the configuration template files (**.erb* files) from your existing HyperStore system to determine whether you made any customizations to those settings (changes from the default values):

- If you have **not** made any such changes, the upgrade proceeds.
- If you **have** made such changes, then in the new installation staging directory the installer creates a text file that lists those changes -- a "diff" file -- and prompts you to review the file, before the upgrade proceeds:
    - Open a second terminal instance and in that terminal go to the new installation staging directory *(/opt/cloudian-staging/7.5.2*) and view the "diff" file that the installer created -- while the upgrade process remains paused in the first terminal instance. The "diff" file will be named as *cloudian-merge-conflicts-<timestamp>.txt*.
    - Then to carry forward your existing **.erb* file customizations that are identified in the diff file, in the second terminal instance manually make those same customizations to the new HyperStore

version's **.erb* files (under */etc/cloudian-7.5.2-puppet/modules*). For example, if in your existing HyperStore system you had set a custom value for a *hyperstore-server.properties.erb* setting, edit that same setting in */etc/cloudian-7.5.2-puppet/modules/cloudians3/templates/hyperstore-server.properties.erb*.

> **Note**
> * You do not need to manually carry forward customizations to the *sso.enabled*, *sso.shared.key*, and *sso.cookie.cipher.key* properties in *mts-ui.properties.erb*. If you've customized these properties, the upgrade will carry forward these customized property values automatically. Starting in HyperStore 7.5.2 these properties are controlled by corresponding settings in *common.csv*.
> * If you are upgrading to 7.5.2 from 7.5.1 and you had added *auto.repair.concurrent.enabled=false* to *hyperstore-server.properties.erb* (as described in the upgrade preparation procedure), you do not need to make this same change to the 7.5.2 files. This setting is not needed in 7.5.2.

- After saving your changes return to the original terminal instance in which you are running the upgrade, and at the installer prompt continue with the upgrade. Note that you do not need to do a configuration push, since the upgrade will apply your configuration edits.

***If you are using the HyperStore Shell** to perform the upgrade*

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, you can view the diff file in the installation staging directory using *cat* and then use the following command to edit HyperStore **.erb* configuration files as needed:

```
$ hspkg config -e <filename> --version 7.5.2
```

In the background this invokes the Linux text editor *vi* to display and modify the configuration file.

> **Note** Customizations that you have previously made to the configuration file *common.csv* are handled differently. The installer detects such customizations and automatically applies the same customizations to the new version's *common.csv* file, without you having to do anything.

**When the upgrade process proceeds it upgrades one node at a time** -- by shutting down the node, updating the packages on the node, and then restarting the node and the services on the node -- **until all nodes are upgraded**. Messages in the terminal will indicate the upgrade progress.

After the upgrade successfully completes, proceed to **"Verifying Your System Upgrade"** (page 261).

> **Note**
> * Once you've started the upgrade, you cannot *<ctrl>-c* out of it.
> * If you have initiated the upgrade through a remote terminal, and the connection between the terminal and the Configuration Master node is subsequently lost, the upgrade will continue.

> **IMPORTANT !** After the upgrade, do not delete the staging directory that was created when you unpacked the product package file (*/opt/cloudian-staging/7.5.2*). HyperStore will continue to require certain files in this directory throughout the time that you are using this HyperStore version.

#### 5.2.2.1. Upgrade Failure and Roll-Back

If the upgrade fails for certain nodes, those nodes are automatically rolled back to your previously existing HyperStore software version. The terminal will display basic information about the failure, and you can get more details from the *cloudian-installation.log* and the *cloudian-upgrade.log*, both of which are generated in the installation staging directory. Try to resolve the problem(s) on the node(s) that failed to upgrade, and then run the upgrade process again (action number **3** from the installer's main menu).

The upgrade process also generates an *upgrade-logNconfig*.tgz* "S.O.S" tar file (which packages together multiple upgrade-related files) in the staging directory that you can provide to Cloudian Support in the event that you need assistance in resolving any upgrade problems.

### 5.2.3. Verifying Your System Upgrade

After all HyperStore nodes have been upgraded, verify that all services are running and that the HyperStore version is now 7.5.2:

1. After the automated upgraded completes, you should be taken back to the main menu of the HyperStore installer. The first post-upgrade step is to confirm that all your HyperStore services are up and running:
    a. From the installer's main menu select "Cluster Management".
    b. From the Service Management sub-menu that displays select "Manage Services".
    c. At the "Select a service to manage:" prompt, select All Services.
    d. At the "Enter command" prompt, type *status*.

    All services on all nodes should then indicate that they are running.
2. Next, confirm that the HyperStore software version is correct:
    a. Still on the Service Management menu, at the "Select a service to manage:" prompt select the S3 Service.
    b. At the "Enter command" prompt, type *version*.

    On all nodes the S3 version should indicate version 7.5.2.

    After confirming the version you can exit the installer.
3. Use the CMC to check on your upgraded system:
    - On the **Node Advanced** page, select command type "Info" then execute the "repairqueue" command to **verify that auto-repair is enabled** for replica, EC, and Cassandra data. (The installer disables auto-repair prior to executing the upgrade and then re-enables auto-repair at the end of the upgrade process.)
    - On the **Manage Users** page, confirm that you can retrieve users.
    - Log out of the CMC as system admin and log back in as a regular user, and then confirm that you can successfully download and upload objects.
4. If prior to the upgrade you had made any customizations to the branding of the CMC interface, only your customized logos and customized application name will be retained after the upgrade. You will need to re-implement any changes that you had made to the browser tab title and/or the color scheme, by again following the instructions for **"Rebranding the CMC UI"** (page 236).
5. If you have been using ElasticSearch for search of HyperStore object metadata, you should have upgraded your ES cluster to version 7.7.x before upgrading HyperStore to version 7.5.2 (as noted in

**"Preparing to Upgrade Your System"** (page 256)). Now, after having upgraded HyperStore, run this command from any HyperStore node to verify that a sync-up of the object metadata in your ES cluster against the object metadata in HyperStore can still be performed without error:

```
# /opt/cloudian/bin/elasticsearchSync all
```

***If you are using the HyperStore Shell** to perform the upgrade*

If you are using the HyperStore Shell, you can run the ES sync tool as follows (with no path):

```
$ elasticsearchSync all
```

You are now done with upgrading to HyperStore 7.5.2.

**Note** If you disabled the failed disk check before performing your upgrade (as described in **"Additional Upgrade Preparation If Your System Currently Has Failed Disks"** (page 257)), note that after you've completed the upgrade, in the new instance of *CloudianInstallConfiguration.txt* in the staging directory for your new HyperStore version the *INSTALL_SKIP_DRIVES_CHECK* setting is set back to its default of *false*. So the next time you upgrade your HyperStore version the check for failed drives will be executed, unless you once again disable the check by changing that setting to *true*.

## 5.2.4. Installing a Patch

On occasion Cloudian may release a "patch" that enables customers to benefit from a recent HyperStore bug fix or fixes without having to wait for the next full release. A HyperStore patch release has a 4-digit release number, and can only be installed on systems running the preceding 3-digit release. For example:

| Hypothetical Patch Release Number | Can Only Be Installed On Systems Running |
|---|---|
| 7.4.0.1 | 7.4.0 (7.4) |
| 7.4.1.1 | 7.4.1 |
| 7.4.3.1 | 7.4.3 |

It may happen that there are multiple patches in between full releases -- for example, if you are running 7.4.1 it may be that there is a 7.4.1.1 patch release and then later there is a 7.4.1.2 patch release. In this case you can install each patch when it comes out, or alternatively if you miss the first patch for some reason, you can install just the second patch and the second patch will include the fixes from the first patch.

Cloudian Support will announce patch releases when they come out, and you can download the patch from the Support portal. A patch is released as a self-extracting binary file, named as *S3Patch-<version>.bin* (for example *S3Patch-7.4.3.1.bin*). You can also download from the Support portal the signature file (*.sig* file) corresponding to the patch -- you will need the signature file if you are using the HyperStore Shell to apply the patch.

**Before installing a patch**, check to confirm that:

- All services are up in your HyperStore system
- No repair, cleanup, or rebalance operations are currently running in your HyperStore system.

For more information on performing these checks, see **"Preparing to Upgrade Your System"** (page 256).

**To install a patch:**

1. Place the patch binary file in a working directory on your Configuration Master node (such as */tmp* or your home directory).

2. Change into that directory, and then run the following commands to run the patch file:

```
# chmod +x S3Patch-<version>.bin
# ./S3Patch-<version>.bin
```

***If you are using the HyperStore Shell** to apply the patch*

If you are using the **HyperStore Shell (HSH)** to apply the patch, make sure you have in your working directory the signature file (*.sig* file) corresponding to the patch file. Then from the shell, run these commands (rather than the commands stated above):

```
$ chmod +x S3Patch-<version>.bin
$ hsrun --root S3Patch-<version>.bin
```

When you run the patch file, you will be prompted to confirm that you want to install the patch -- enter **y** to do so. Then it automatically takes all the actions necessary to apply the patch to each of your HyperStore nodes. Specifically, the following actions are automatically executed:

- The *S3Patch-<version>.bin* file content -- including a patch installation script -- is extracted into a */s3patch/<patch-version>/* sub-directory under your HyperStore system's current installation staging directory
- The patch installation script is automatically launched. The script performs a non-disruptive, rolling install of the patch to each of your HyperStore nodes one at a time -- including automatically restarting the affected services on one node at a time.
- The status of the patch installation process is written to the console, and log messages pertaining to the patch installation are written to *<current-staging-directory>/installs3patch.log*. Also, a backup copy of the original, unpatched version of the main *.jar* file (Java archive file) from your existing HyperStore version is written to *<current-staging-directory>/s3patch/backup/*.

After a successful patch upgrade of all nodes, you can launch the main HyperStore installer (*./cloudianInstall.sh* in your current staging directory), go to the Manage Services menu, and for the S3 Service check the version. The results should show that on all nodes, the S3 Service now has the version number of the patch that you installed. You should also log into the CMC and check some of the main status reporting pages -- such as the **Data Centers** page and the **Alerts** page -- to confirm that your patched HyperStore system is healthy. You may also want to exercise the system by, for instance, uploading some objects into a bucket.

**Note** Do not delete the *S3Patch-<version>.bin* file from the working directory in which you placed it. You may need to use the file again, as described in the sections below.

### 5.2.4.1. Reapplying the Patch in the Case of Installation Errors

In some cases the patch installation script may write messages to the console indicating that the patch did not successfully install on a certain node or nodes. An example is if a node is down at the time that the patch installation script runs -- the script will not be able to install the patch to that node. In this scenario, first correct the underlying condition -- for example, bring a down node back up. Then on the Configuration Master node change into the working directory where the *S3Patch-<version>.bin* file is located and run the file again:

```
# ./S3Patch-<version>.bin
```

You will again be prompted to confirm that you want to install the patch -- enter **y** to do so. The patch script will then check each HyperStore node and install the patch only on nodes on which it has not already been successfully installed. The status will be written to the console, and also logged to *<current-staging-directory>/installs3patch.log*.

#### 5.2.4.2. Reverting a Patch

In the unlikely event that you need to revert a patch, the patch binary file supports doing so. You might need to revert a patch if, for example:

- You are unable to successfully install the patch to all nodes -- that is, you are in a condition where some nodes were successfully patched while errors prevented patching of the other nodes, and you are unable to correct the errors.
- You successfully patch all nodes, but subsequently you encounter negative behavior in your system that you had not encountered prior to the installation of the patch.

If you are reverting a patch, **contact Cloudian Support** (either before or after reverting the patch).

To revert a patch:

1. On the Configuration Master node, change into the working directory in which the *S3Patch-<version>.bin* file is located.
2. Run the patch bin file using the **-r** option:

```
# ./S3Patch-<version>.bin -r
```

You will be prompted to confirm that you want to revert the patch -- enter **y** to do so. The patch script will then revert your HyperStore system to the preceding **3-digit release version**. For example, reverting a 7.5.2.1 patch will revert your system to 7.5.2; and reverting a 7.5.2.2 patch will also revert your system to 7.5.2 (not to 7.5.2.1).

#### 5.2.4.3. Adding Nodes to a Patched System

If you patch your system and then you **add nodes** (or a **new data center** or **new region**) to your system before you have upgraded to a later full release version, the patch will automatically be applied to the newly added nodes.

## 5.3. Adding Nodes

This procedure is for **adding one or more nodes to an existing HyperStore data center**. During this procedure you will:

- Prep the new node(s)
- Verify that your existing cluster is in a proper condition to add nodes
- Add the new node(s) to the cluster
- Rebalance data within the cluster

**IMPORTANT !**
* Within a data center, if your HyperStore system is configured with multiple rack names and you are adding nodes, make sure to **position the new nodes so that when you are all done adding nodes, each rack has the same number of nodes**. Having an imbalance in the number of nodes per rack will result in data load imbalance, such that on racks with fewer nodes there will be more stored data per node than on racks with more nodes.
* Each node that you add must have the **same networking configuration as the existing nodes** in your HyperStore system -- for example, if the existing nodes each have two NICs, one for front-end

network and one for back-end network, then the new nodes must have the same configuration.
* Once you have added a node to the cluster **you cannot simply "undo" the process**. If after adding a new node to the cluster you were to change your mind about keeping the node in the cluster, you would still be required to rebalance data within the cluster and then afterwards you would need to decommission the node.
* If you want to **re-use a removed node's host machine** by adding it back to your HyperStore cluster as a new node, then before using the host machine as a new HyperStore node you must remove all data from the host and re-install the operating system.

## 5.3.1. Special Requirements if an Existing Node is Down

The CMC's **Add Node** function **will not work if an existing node in your cluster is down or unreachable**. If you have more nodes in your cluster than are required by your configured storage policies and one of those nodes is permanently down, follow the procedure for **"Removing a Node"** (page 289) to remove the dead node from the cluster. After you complete that procedure you can then follow this procedure for Adding Nodes. If your current cluster has only the minimum number of nodes required by your storage policies and one of those nodes is dead, contact Cloudian Support for guidance.

## 5.3.2. Preparing to Add Nodes

Before you add a node or nodes to your cluster, prepare by taking the actions below.

1. Make sure the new host(s) **meet requirements** for:
   - Hardware specs and operating system: See **"Host Hardware and OS Requirements"** (page 43).
   - Open listening ports: See **"HyperStore Listening Ports"** (page 54).
2. Make sure you have the **information you will need** to complete this procedure: each new node's hostname, IPv4 address, internal interface name (optional), and *root* user password (or *sa_admin* user password for a "Secure Appliance" with its HyperStore Shell enabled and root password disabled at the factory).
3. **Start the new host(s)**, if not already running.
4. From your Configuration Master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

   *More detail:*

   a. On your Configuration Master node change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the *system_setup.sh* tool.

   ```
   # ./system_setup.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the system setup tool with this command:

   ```
   $ hspkg setup
   ```

Once launched, the setup tool's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

b. In the tool's main menu select "**8**" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.

c. Use the node preparation task menu to prepare the node:

- Complete the configuration of network interfaces for the node, if you haven't already.
- Set the timezone for the node.
- Install and configure HyperStore prerequisites on the node.
- Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
- After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.

d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the *system_setup.sh* tool.

**IMPORTANT !** If the new node(s) are not HyperStore Appliances and **if you do not use *system_setup.sh* to format the data disks on the new node(s)**, then in the installation staging directory on your Configuration Master node you must for each new node create a text file named *<hostname>_fslist.txt* that specifies the new node's data mount points, in this format:

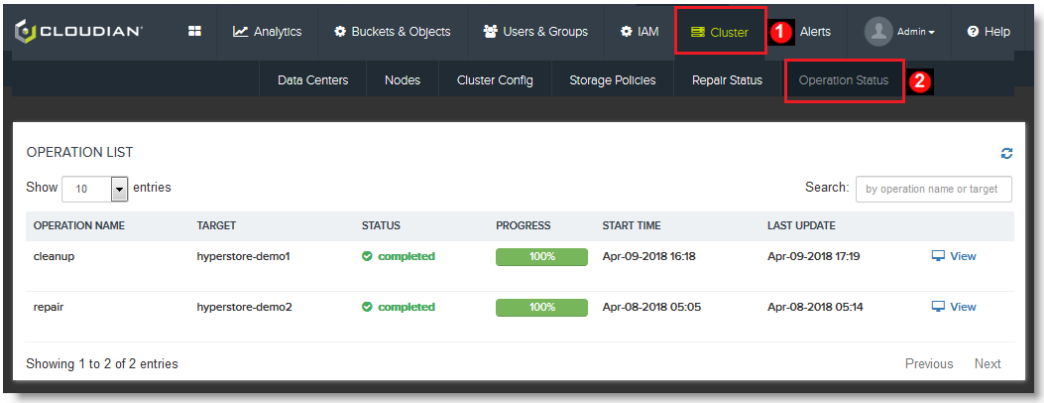
*<devicename> <mountpoint>*
*<devicename> <mountpoint>*
*etc...*

5. **Make sure the cluster is in proper condition** to add nodes:

a. If there are any operations in-progress in the CMC's **Operation Status** page, wait for them to finish (or otherwise, the **Add Node** operation will automatically stop them).

*More detail:*

When you initiate the **Add Node** operation in the CMC (as described in "Adding Nodes" below), the system will automatically stop any in-progress *repair*, *repairec*, *repaircassandra*, *cleanup*, or *cleanupec* operations in the service region. If there is an in-progress operation that you do not want the system to stop, wait until the operation completes before you add nodes to your cluster.

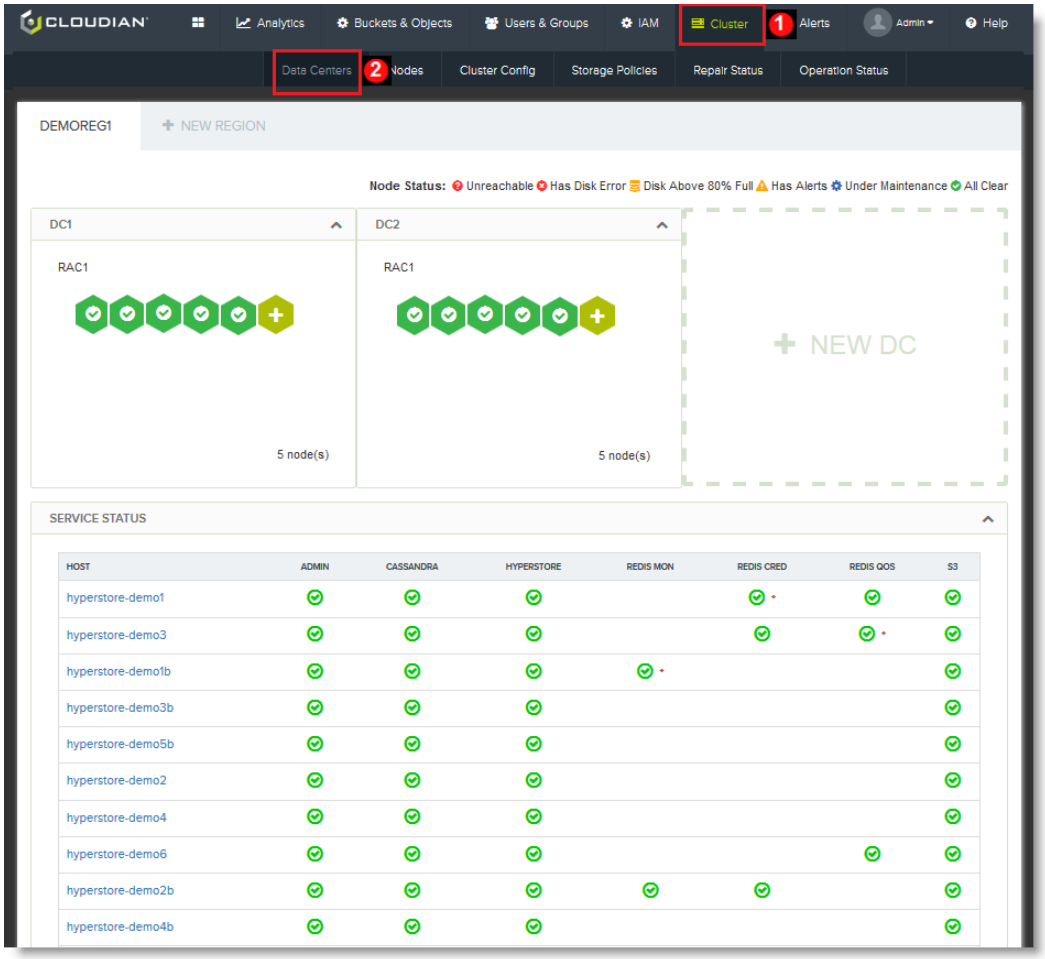



- If a repair is running on a node because you recently initiated a "**replacedisk**" operation on that node, Cloudian recommends **waiting until the repair completes** before you add a node to your cluster.
- If you proceed with adding a node at a time when a repair is in progress, and the system automatically stops the repair, do not run the "*-resume*" option on that repair after you've added the node. Adding a node affects the token range distribution within the cluster, so resuming an interrupted repair operation afterward is not supported. If you want to repair a node on which repair was interrupted, wait until after you've completed the rebalance operation (as part of the Adding Nodes procedure), and then run a fresh repair operation on the node for which repair was interrupted.

**Note** When you in initiate the **Add Node** operation in the CMC, the system will also automatically disable the auto-repair feature and the proactive repair feature -- so that no new repairs kick off while you're expanding your cluster -- and then after you've completed the rebalance operations for all new nodes the system will automatically re-enable auto-repair and proactive repair.

b. In the **Data Centers** page, make sure that all the existing nodes and services are up and running in the service region.

*More detail:*

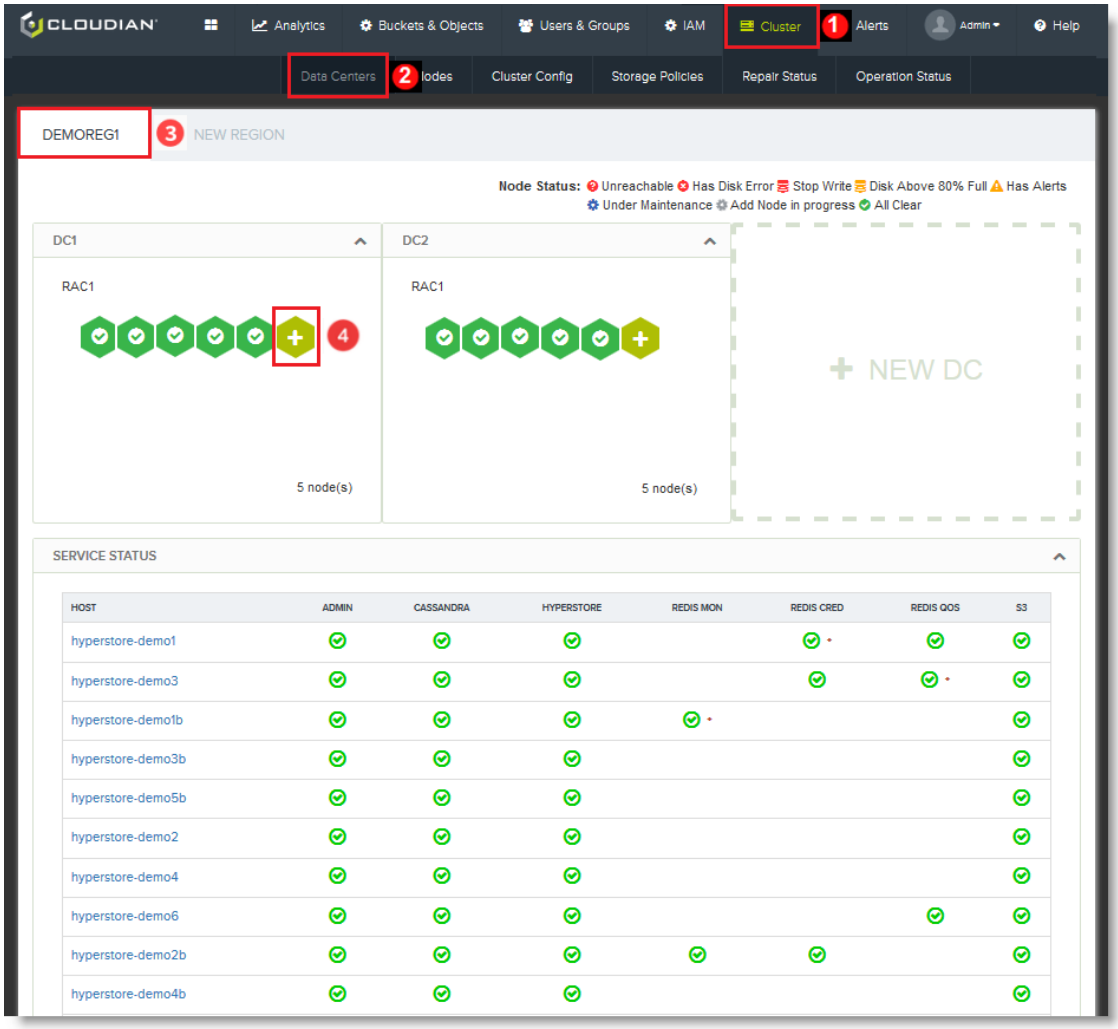



The **Add Node** function will not let you add nodes if any existing nodes or services in the region are down.

## 5.3.3. Adding Nodes

1. In the CMC's **Data Centers** page, in the display for the data center and rack in which you want to add a node or nodes, click the light green cube icon that has a plus sign on it.

   *More detail:*

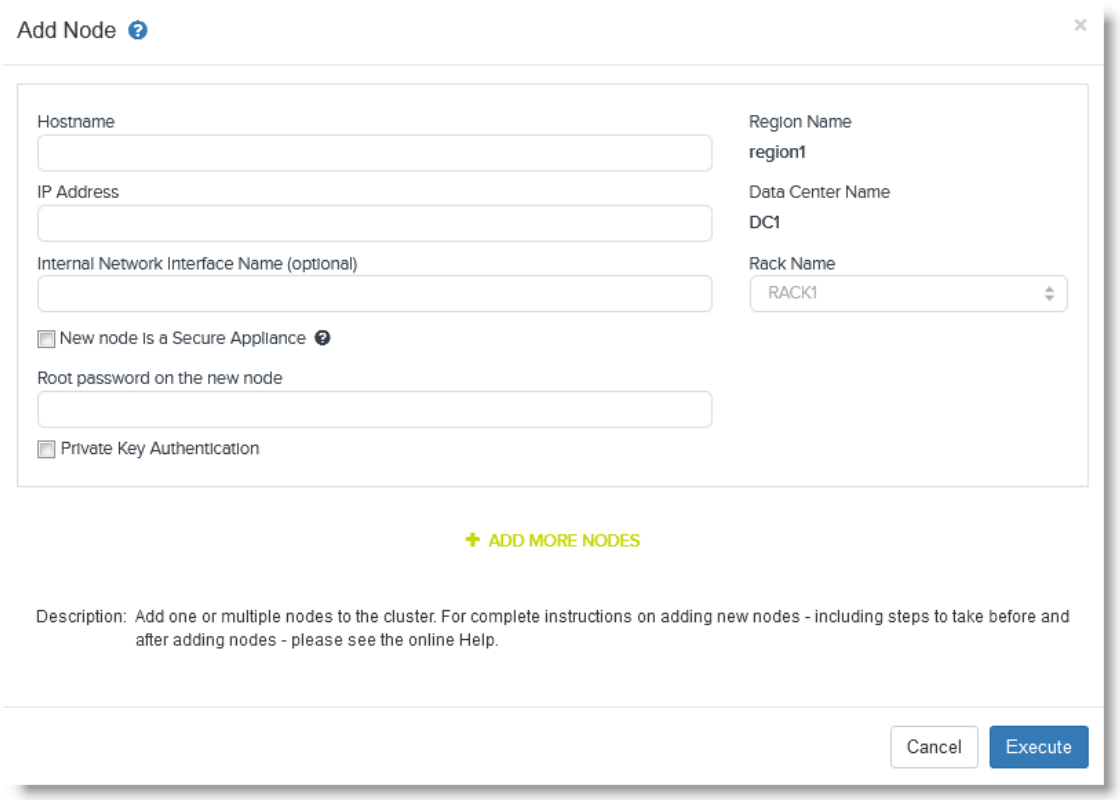



Clicking the light green cube opens the **Add Node** interface.

> **Note** If the data center has multiple racks for HyperStore -- which is possible only if your original HyperStore version installed was earlier than version 7.2 -- and if you want the new node(s) to be assigned to a new rack rather than one of the existing racks, just click on the light green cube icon for any rack in the display for the correct data center. If (and only if) the data center has multiple racks for your existing HyperStore nodes, you will have an opportunity to specify a new rack name in the next step below.

2. In the **Add Node** dialog that displays, complete the node information fields for the new node(s). After completing all the fields for one new node, click **Add More Nodes** and complete the fields for another new node; and repeat this process until you've completed the fields for all the new nodes that you want to add to the data center.

   *More detail:*




*Hostname (required)*

Hostname of the new node. This must be just a hostname, not an FQDN. Do not use the same hostname for more than one node in your entire HyperStore system. Each node must have a unique hostname, even if the nodes are in different domains.

*IP Address (required)*

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

*Internal Network Interface Name (optional)*

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

*Rack Name (required)*

The behavior of this field depends on your particular HyperStore system and existing set-up:

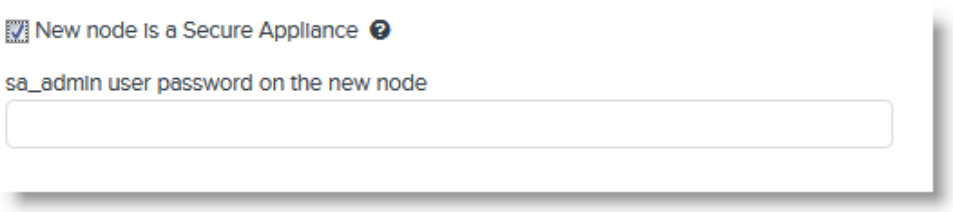
- If your original HyperStore system is version 7.2 or later, or if your original HyperStore system was earlier than 7.2 and you have only been using one rack name for your existing nodes in the data center, this field value is fixed to the rack name that you have been using for your existing nodes. You cannot edit this field.
- If your original HyperStore system was earlier than 7.2 and you have been using multiple rack names for your existing nodes in the data center, you can select any of those rack names from the drop-down list or create a new rack name.

> **IMPORTANT !** If your system is configured with multiple rack names and you are adding nodes, make sure to position the new nodes so that when you are all done adding nodes, each rack has the same number of nodes (for example, three nodes on each rack). Having an imbalance in the number of nodes per rack will result in data load imbalance, such that on racks with fewer nodes there will be more stored data per node than on racks with more nodes.

*Authentication fields (based on new node type)*

During the **Add Node** operation, the HyperStore installer on your Configuration Master node needs to securely connect to the new node. The authentication options for doing so depend on the new node type:

- **If the new node is a "Secure Appliance"** (a HyperStore Appliance for which the HyperStore Shell [HSH] was enabled and the root password disabled at the factory), select the "New node is a Secure Appliance" checkbox. Then enter the *sa_admin* user's password for the new node.




> **Note** Before the new node is added to your HyperStore cluster, the *sa_admin* user's password on the new node may be different than the *sa_admin* user's password in your existing cluster. If so, after the new node is added to the cluster, the *sa_admin* user's password on the new node will be automatically changed to match the *sa_admin* user's password in the cluster.

- **If the new node is not a "Secure Appliance"** -- that is, if the new node is a standard Appliance or a software-only node on commodity hardware -- then you can use either one of these authentication methods (use one method or the other -- not both):
  - Enter the *root* user's password for the new node.

    OR

  - Select the "Private Key Authentication" checkbox. In this case the installer will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:
    - If during installation of the cluster you let the installer generate an SSH key pair for you, or you used your own existing SSH key pair and you copied both the private and the public key into the installation staging directory on the Configuration Master, then distribution of the public key to the new node will be taken care of automatically by the installer.
    - If during installation of the cluster you used your own existing SSH key pair and you copied only the private key into the installation directory -- and you

copied the public key to the target installation nodes manually -- **then you must also copy the public key to the new node manually**, before executing the **Add Node** operation.

3. Click **Execute**. The system will first run a pre-check to confirm that each of your specified new nodes can be reached through the network. The system then performs a **simulation that shows how token ranges and data will be distributed across your cluster** if you proceed with adding the new nodes. The simulation results are displayed in an overlay over the **Add Node** dialog, including summary assessments of whether the data associated with each of your existing storage policies will be well-balanced across the cluster if you add the new nodes. Scroll through the simulation results and carefully review the summary assessments. Typically the assessments will indicate a good data distribution balance, and you can then proceed with adding the new nodes by clicking **Install**.

   **Note** If the assessments project an **imbalance** for a storage policy that's being used extensively in your production service, close out of the simulation overlay and the **Add Node** dialog and contact Cloudian Support for assistance.

4. When you click **Install** in Step 3 the system then adds the new node(s) to your cluster, one node at a time. As part of adding a new node to the cluster, the system automatically streams to the new node its proper share of system and object metadata. Depending on how many nodes you are adding and how much data is in your system, this process can take anywhere from several hours up to several days or more. When this streaming process completes successfully for all the new nodes, for each new node an orange node icon with a gear inside of it should display in the **Data Centers** page. 

   *More detail:*

   The **Add Node** operation entails verifying that the new host meets HyperStore requirements, installing software, updating system configuration, starting services, joining the new node into the cluster, and streaming system and object metadata to the new node (in the Cassandra database).

   There are two CMC locations where you can monitor the **Add Node** operation progress:

   - The **Data Centers** page. As each node is added to the cluster a new node icon appears, with color-coding to indicate the node's status. You can hold your cursor over the node icon for a text description of the status.

     Grey node with gear -- The node has been added to the cluster and Cassandra repair is in progress (which streams system and object metadata to the new node).

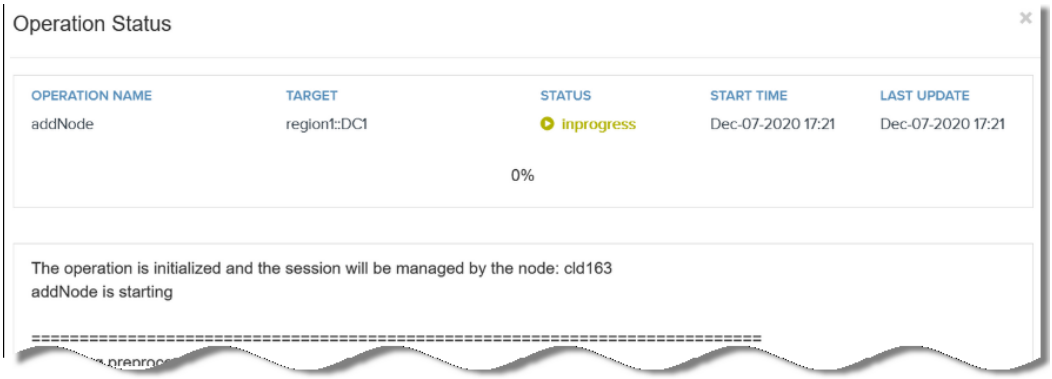
     Orange node with gear -- Cassandra repair has completed successfully, and the node is ready for you to run a "rebalance" operation to stream object data to it (as described later in this procedure). **If all the new nodes show this status** you can proceed to Steps 5 and 6.

     Red node with gear -- Cassandra repair has failed for the node.

   - The **Operation Status** page. In this page you will see a one-line status summary for the **addNode** operation (even if multiple nodes are being added there is just one summary status

line for the operation as a whole). If you click **View** to the right of the summary status line you can display detailed progress information.




**Handling Failures:**

- *Failure in adding the node to the cluster*: If in the **Data Centers** page the grey node icon (indicating that the node has been added and Cassandra repair is underway) never appears for one or more nodes, go the **Operation Status** page and view the status detail. Try to resolve the problem reported in the status detail -- in some cases a reboot of the new node may suffice. Then go back to the CMC's **Data Centers** page and again initiate the Add Node operation. This time enter the node information **only for the node(s) that failed to be added** to the cluster, then Execute the operation. Be sure to include only the node(s) that failed to be added (do not include the successfully added nodes, and do not add entirely new nodes that you haven't tried to add yet -- this will cause the operation to fail.) Note that you will not be shown a data distribution simulation again on this second attempt at adding the nodes.

  **Note** Do not change the disk configuration of the new nodes before trying again to add the new nodes (since this could result in sub-optimal token distribution, and consequently a sub-optimal data distribution). If for some reason you must change the disk configuration before trying again to add the new nodes, contact Cloudian Support.

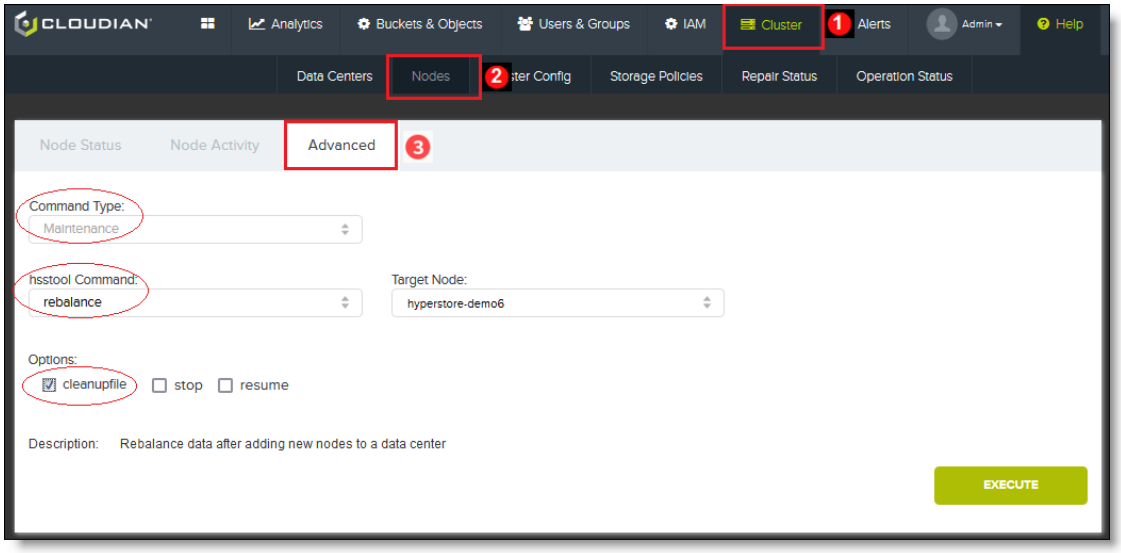
- *Failure in Cassandra repair* (as indicated by red node icon with gear): If this status occurs, first check the **Data Center** page's Service Status section to make sure that Cassandra is up and running on all nodes (if it's down on any node, go to the **Node Status** page for that node and start Cassandra). After making sure Cassandra is up on all nodes go to the **Node Advanced** page, and from the Maintenance command type menu run *repaircassandra* on the new node. Periodically check the progress. In the **Data Centers** page the new node's icon should turn orange when the repair completes successfully.

If these corrective measures fail, contact Cloudian Support.

5. After the **Add Node** operation has completed successfully for all of the new nodes -- as indicated by there being an orange node icon for each new node in the **Data Centers** page -- **update your DNS and/or load balancer** configurations to include the new nodes, so that the new nodes can participate in servicing S3 user request traffic. For more information see **"DNS Set-Up"** (page 50) and **"Load Balancing"** (page 53).
6. In the **Node Advanced** page, from the Maintenance command type group, **execute *hsstool rebalance* on each new node**. When launching *rebalance* on each new node, use the **cleanupfile** option. Rebalance is a long-running background operation that you can run concurrently on multiple new nodes that

you've added (but check the details below for important limitations on concurrency). When all new nodes have completed rebalancing, for each node a green check-marked cube icon will display in the **Data Centers** page.

*More detail:*




The rebalance operation populates the new node(s) with their appropriate share of S3 object data. The rebalance is a background operation that may take up to several days or more for each new node to complete, depending on factors such as data volume and network bandwidth. When rebalance is performed with the **cleanupfile** option, as soon as a replica or fragment is successfully copied to the new node, it is deleted from the older node on which it no longer belongs -- thereby freeing up storage space on the older nodes as the rebalance operation progresses. For more information see **"hsstool rebalance"** (page 543).

You can run *rebalance* on multiple new nodes concurrently. However, **do not run *rebalance* concurrently on a number of new nodes that exceeds one-third the number of your already existing nodes** in that same data center. For example, if you have 12 existing nodes in a DC and you are adding 6 new nodes to that DC, do not run *rebalance* on more than 4 of the new nodes concurrently (one-third of 12). In this scenario you would run *rebalance* on 4 of the new nodes concurrently, and then when the *rebalance* operation completes on at least 2 of those nodes, you can start *rebalance* on the other 2 new nodes. In this way *rebalance* is never running on more than 4 nodes at a time (never running on more than one-third the number of the pre-existing nodes in the DC).

**Note** During in-progress rebalance operations the affected data remains readable by S3 client applications. Meanwhile the new nodes are immediately available to support writes of new data, even before any rebalancing occurs.

**Note** When you have rebalance running on a new node or multiple new nodes, you cannot add any additional nodes to the service region (using the CMC's **Add Node** feature) until rebalance has competed successfully on all of the current new nodes.

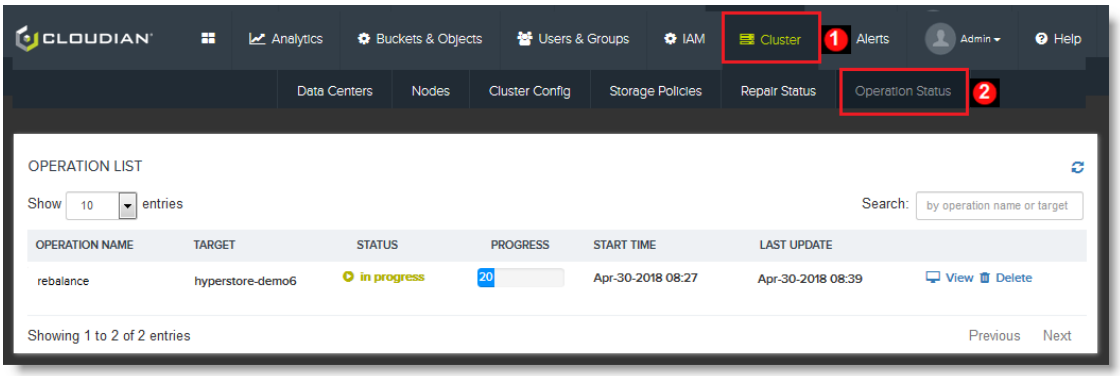
Use the **Data Centers** page to periodically **check the progress** of the rebalance operation on each of the new nodes. The icon for each of the new nodes will be color-coded, and you can hold your cursor over the icon for a text description of the status.

Grey node with gear -- The rebalance operation is in progress. If you have added multiple nodes and have started rebalance operations on the nodes, each node's status icon will remain in this status until rebalance completes on all of the nodes in the batch.

Red node with gear -- The rebalance operation has failed for one or more token ranges. If this status displays, go to the **Nodes Advanced** page and run rebalance on the node again, using the **retry** option this time (select the retry checkbox when you run rebalance on the node), as well as the **cleanupfile** option. This will try the rebalance again, just for the failed token range(s).

Green node with check mark -- The rebalance operation has completed successfully.

Another source of status information for the rebalance operation is the **Operation Status** page.




For status detail click **View** to the right of the summary status line.

If for some reason you need to stop -- and then later resume -- a rebalance operation on a node, you can do so as described in **"hsstool rebalance"** (page 543).

This completes the procedure for adding nodes to your cluster.

Note: **If you removed a dead node prior to performing the Adding Nodes procedure** and deferred the node repair operations that are necessary after you remove a dead node, perform those repairs now.

*More detail:*

a. From the **Node Advanced** page, run **hsstool repair** on each node in the service region except for the new node(s), just one node at a time. When repairing each node, use the **allkeyspaces** option and also the **-pr** option. Leave the **-l** and **-m** options selected, as they are by default. Use the **Operation Status** page to track the progress of each repair. After repair of a node is complete, repair another node -- until all nodes except for the new node(s) have been successfully repaired.

b. If you have erasure coded object data in your system, from the **Node Advanced** page run **hsstool repairec** on one node in each HyperStore data center in the region. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region. Use the **Operation Status** page to track repair progress.

# 5.4. Adding a Data Center

This procedure is for **adding a new data center to an existing HyperStore service region**. During this procedure you will:

- Prep the nodes in the new data center
- Verify that your existing cluster is in a proper condition to successfully add a data center
- Add the new data center's nodes to the cluster
- Take initial steps to start utilizing the new data center

**Note** Each node that you add must have the same networking configuration as the existing nodes in your HyperStore system -- for example, if the existing nodes each have two NICs, one for front-end network and one for back-end network, then the new nodes must have the same configuration.

## 5.4.1. Special Requirements if an Existing Node is Down or Unreachable

The CMC's **Add DC** function **will not work if an existing node in your cluster is down or unreachable**. If you have more nodes in your cluster than are required by your configured storage policies and one of those nodes is permanently down, follow the procedure for **"Removing a Node"** (page 289) to remove the dead node from the cluster. After you complete that procedure you can then follow this procedure for Adding a Data Center. If your current cluster has **only the minimum number of nodes required by your storage policies and one of those nodes is dead**, contact Cloudian Support for guidance.

## 5.4.2. Preparing to Add a Data Center

Before you add a new data center to a region, prepare by taking the actions below.

1. The HyperStore nodes in each data center will need to be able to communicate with the HyperStore nodes in the other data center(s). This includes HyperStore services that listen on the internal interface. Therefore, if you haven't already done so you must **configure your inter-DC networking so that the DCs' internal networks are connected to each other** (for example, by using a VPN).
2. Make sure the new host(s) **meet requirements** for:
    - Hardware specs and operating system: See **"Host Hardware and OS Requirements"** (page 43).
    - Open listening ports: See **"HyperStore Listening Ports"** (page 54).
3. Make sure you have the **information you will need** to complete this procedure: the new data center's name, and each new node's hostname, IPv4 address, internal interface name (optional), and root password (or *sa_admin* user password for a "Secure Appliance" with its HyperStore Shell enabled and root password disabled at the factory). For data center names only alphanumeric characters and dashes are supported.
4. **Start the new host(s)**, if not already running.
5. From your Configuration Master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

    *More detail:*

a. In your existing cluster, on your Configuration Master node change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the *system_setup.sh* tool.

```
# ./system_setup.sh
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the system setup tool with this command:

```
$ hspkg setup
```

Once launched, the setup tool's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

b. In the tool's main menu select "**8**" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.

c. Use the node preparation task menu to prepare the node:

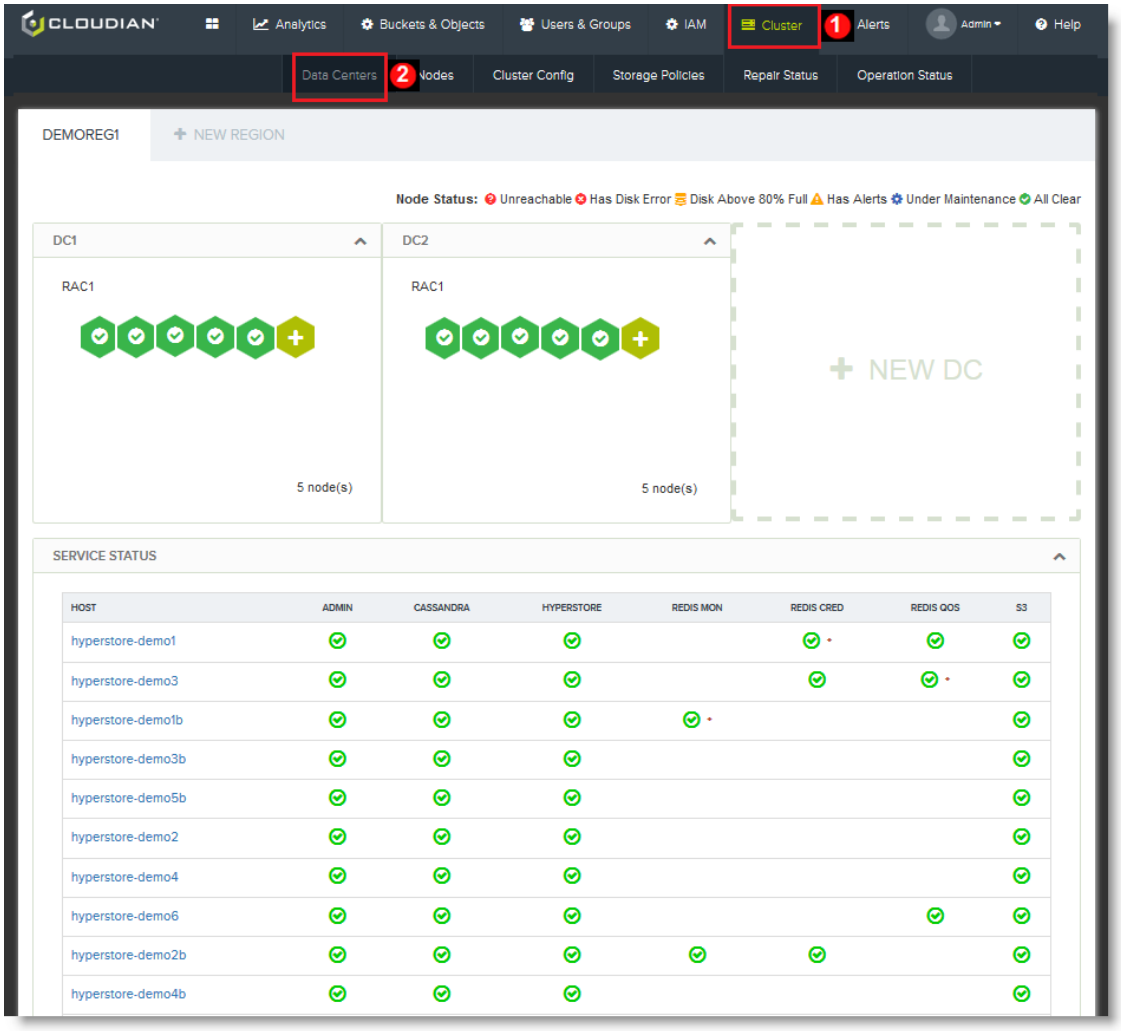
- Complete the configuration of network interfaces for the node, if you haven't already.
- Set the timezone for the node.
- Install and configure HyperStore prerequisites on the node.
- Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
- After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.

d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the *system_setup.sh* tool.

**IMPORTANT !** If the new node(s) are not HyperStore Appliances and **if you do not use *system_setup.sh* to format the data disks on the new node(s)**, then in the installation staging directory on your Configuration Master node you must for each new node create a text file named *<hostname>_fslist.txt* that specifies the new node's data mount points, in this format:

*<devicename> <mountpoint>*
*<devicename> <mountpoint>*
*etc...*

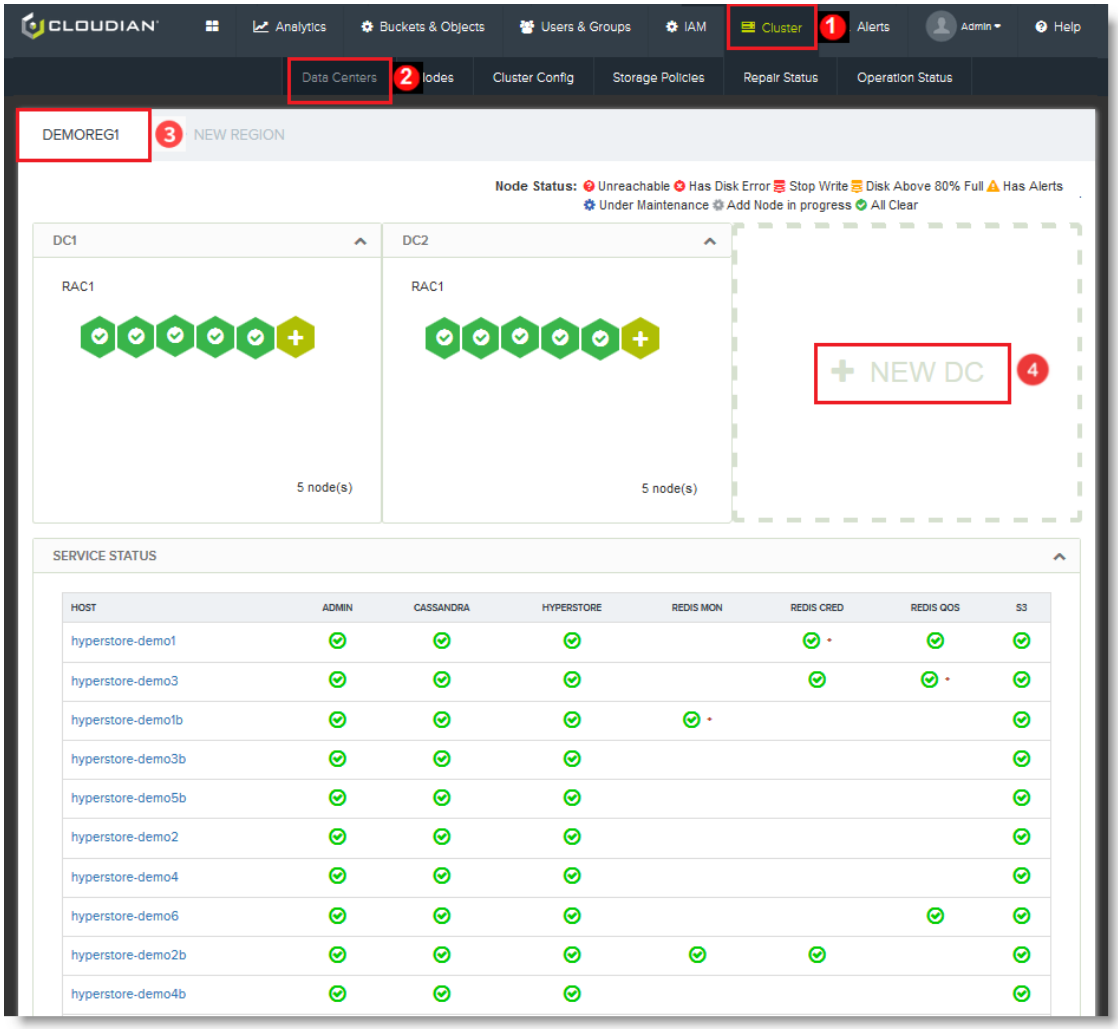
6. In the CMC's **Data Centers** page, **make sure that all the existing nodes and services are up and running** in the region in which you are adding a data center. The **Add DC** function will not let you add nodes if any existing nodes or services in the region are down.




**Note** When you initiate the **Add DC** function (as described below) the system automatically disables Cassandra auto-repair throughout the service region, and then when the **Add DC** operation has completed the system automatically re-enables Cassandra auto-repair throughout the service region.

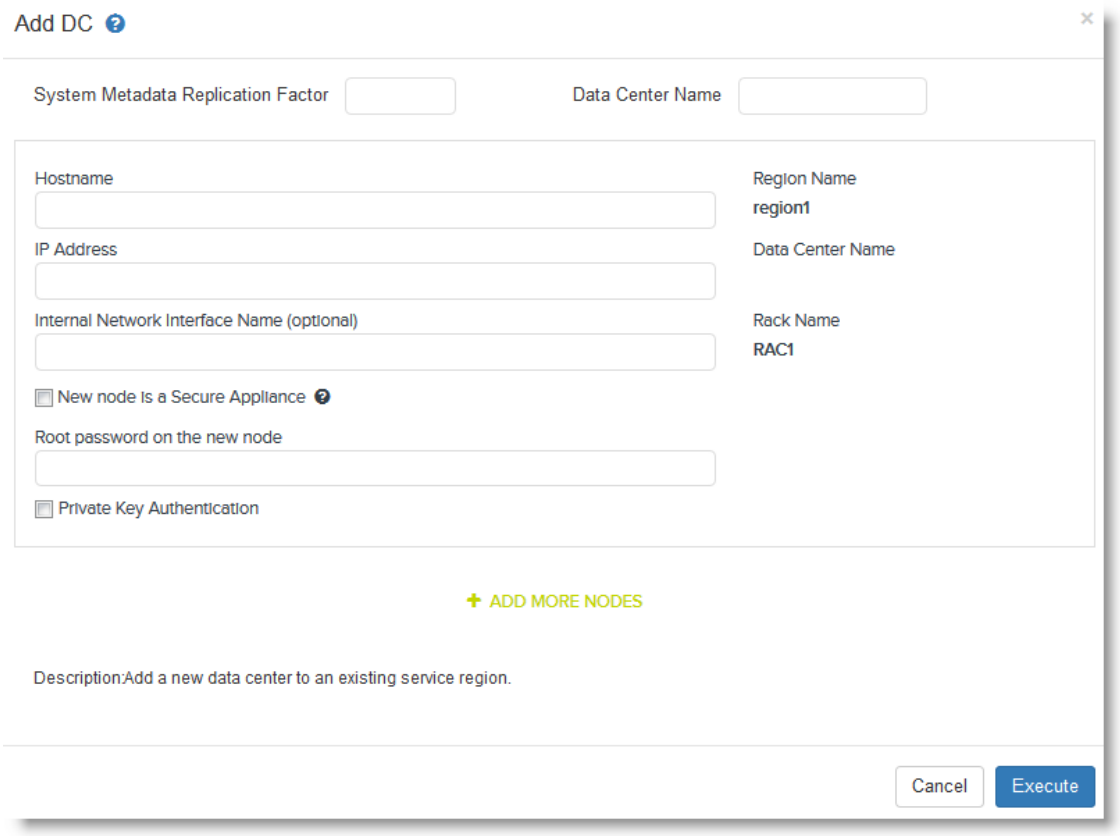
## 5.4.3. Adding a Data Center

1. In the CMC's **Data Centers** page, with the correct service region tab selected, click the large box that says **+NEW DC**.




2. In the **Add DC** interface that displays, complete the top two fields for the new data center as a whole, then complete the remaining fields for each node in the data center, clicking **Add More Nodes** to display fields for additional nodes as needed.

   *More detail:*




*System Metadata Replication Factor (required)*

In this field indicate how many replicas of system metadata (such as usage reporting data, user account information, and system monitoring data) you want to store in the new DC. For each metadata item, each replica will be stored on a different node, to protect this metadata against loss or corruption. It's recommended that you set this replication factor to 3 if you are going to have three or more nodes in the new DC. If there will be only two nodes in the new DC, set this to 2; if there will be only one node, set this to 1. You cannot set a metadata replication factor higher than the number of nodes in the new DC.

*Data Center Name (required)*

Name of the new data center. Maximum 256 characters. Only ASCII alphanumerical characters and dashes are allowed.

*Hostname (required; must be unique within system)*

Hostname of the new node.

**Note**
* This must be just a hostname, not an FQDN.
* Do not use the same hostname for more than one node in your entire HyperStore system. Each node must have a unique hostname within your entire HyperStore system, even in the case of nodes that are in different domains.

*IP Address (required)*

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

*Internal Network Interface Name (optional)*

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

*Rack Name (fixed value "RAC1")*

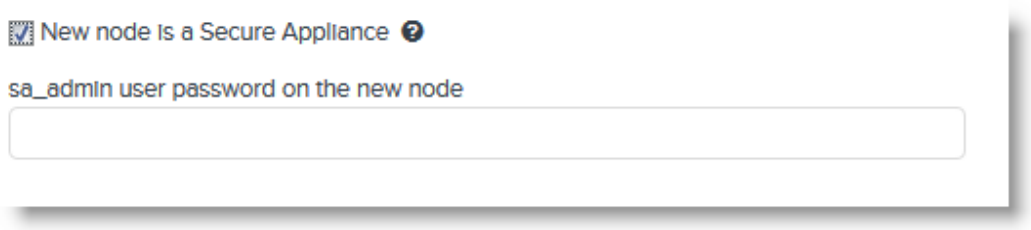
The "Rack Name" value is automatically fixed to "RAC1" for all nodes in the new data center. You cannot edit this value.

> **Note** This is an internal value used by HyperStore. It does not need to correspond to any actual rack name in your data center.

*Authentication fields (based on new node type)*

During the **Add DC** operation, the HyperStore installer on your Configuration Master node needs to securely connect to each new node. The authentication options for doing so depend on the new node type:

- **If the new node is a "Secure Appliance"** (a HyperStore Appliance for which the HyperStore Shell [HSH] was enabled and the root password disabled at the factory), select the "New node is a Secure Appliance" checkbox. Then enter the *sa_admin* user's password for the new node.




> **Note** Before the new node is added to your HyperStore cluster, the *sa_admin* user's password on the new node may be different than the *sa_admin* user's password in your existing cluster. If so, after the new node is added to the cluster, the *sa_admin* user's password on the new node will be automatically changed to match the *sa_admin* user's password in the cluster.

- **If the new node is not a "Secure Appliance"** -- that is, if the new node is a standard Appliance or a software-only node on commodity hardware -- then you can use either one of these authentication methods (use one method or the other -- not both):
  - Enter the *root* user's password for the new node.

    OR
  - Select the "Private Key Authentication" checkbox. In this case the installer will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:
    - If during installation of the cluster you let the installer generate an SSH key pair for

you, or you used your own existing SSH key pair and you copied both the private and the public key into the installation staging directory on the Configuration Master, then distribution of the public key to the new node will be taken care of automatically by the installer.

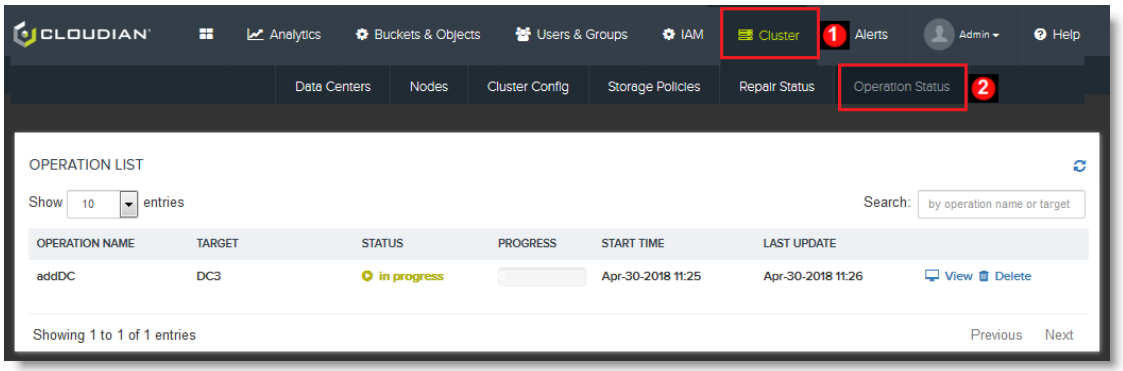
- If during installation of the cluster you used your own existing SSH key pair and you copied only the private key into the installation directory -- and you copied the public key to the target installation nodes manually -- **then you must also copy the public key to the new node manually**, before executing the **Add DC** operation.

3. Click **Execute**. This initiates a background operation that will take anywhere from several minutes to several hours to complete depending on your environment. When it completes successfully the **Data Centers** page will display an additional block representing the newly added DC, with a green, check-marked cube icon for each of the new DC's nodes.

   *More detail:*

   The **Add DC** operation entails verifying that the new hosts meet HyperStore requirements, installing software, updating system configuration, starting services, joining the new nodes into the cluster, and streaming system metadata to the new nodes (in Cassandra).

   You can use the **Operation Status** page to monitor progress of the operation.




   For status detail click **View** to the right of the summary status line.

   When the operation is complete, to see the new nodes in the **Data Centers** page you may need to refresh the page in your browser.

   If you hold your cursor over each cube in the new data center the node host names will display.

   **Note** If the **Operation Status** page indicates that the **Add DC** operation has failed, click "View" for detail. Then for more information to support troubleshooting efforts, *grep* for "ERROR" level messages in the *cloudian-installation.log* file under the installation staging directory on your Configuration Master node.

4. **Update your DNS and load balancing configurations** to include the new data center and its nodes, if you have not already done so. For more information see **"DNS Set-Up"** (page 50) and **"Load Balancing"** (page 53).

5. Go to the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) and **create one or more storage policies that utilize the new DC**. Until you create storage policies that use the new DC and users

subsequently create buckets that use those storage policies, no S3 object data will be stored in the new DC.

> **Note** Because your previously existing storage policies do not include the new DC, none of the data already stored in your system in association with those storage policies will be migrated into the new DC. Accordingly, there is no need for you to run a *rebalance* operation after adding a new DC.

This completes the procedure for adding a data center to your cluster.

NOTE: **If you removed a dead node prior to performing the Adding a Data Center procedure** and deferred the node repair operations that are necessary after you remove a dead node, perform those repairs now.

*More detail:*

a. From the **Node Advanced** page, run **hsstool repair** on each node in the service region except for the new nodes, just one node at a time. When repairing each node, use the **allkeyspaces** option and also the **-pr** option. Leave the **-l** and **-m** options selected, as they are by default. Use the **Operation Status** page to track the progress of each repair. After repair of a node is complete, repair another node -- until all nodes except for the nodes in the new data center have been successfully repaired.

b. If you have erasure coded object data in your system, from the **Node Advanced** page run **hsstool repairec** on one node in each HyperStore data center in the region except for the new data center. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region, except for the new DC. Use the **Operation Status** page to track repair progress.

# 5.5. Adding a Region

This procedure is for **adding a new service region to your HyperStore system**. During this procedure you will:

- Prep the nodes in the new service region
- Add the new region's nodes to the system
- Take initial steps to start utilizing the new region

> **Note** Each node that you add must have the same networking configuration as the existing nodes in your HyperStore system -- for example, if the existing nodes each have two NICs, one for front-end network and one for back-end network, then the new nodes must have the same configuration.

## 5.5.1. Preparing to Add a Region

Before you add a new region to your system, prepare by taking the actions below.

1. The HyperStore nodes in each region will need to be able to communicate with the HyperStore nodes in the other region(s). This includes HyperStore services that listen on the internal interface. Therefore, if you haven't already done so you must **configure your networking so that the internal networks of all of your data centers in all of your regions are connected to each other** (for example, by using a VPN).

2. Make sure the new host(s) **meet requirements** for:
   - Hardware specs and operating system: See **"Host Hardware and OS Requirements"** (page 43).
   - Open listening ports: See **"HyperStore Listening Ports"** (page 54).
3. Make sure you have the **information you will need** to complete this procedure: the name of the new region, the name(s) of the data center(s) that will comprise the region, and each new node's hostname, IPv4 address, internal interface name (optional), rack name, and root password (or *sa_admin* user password for a "Secure Appliance" with its HyperStore Shell enabled and root password disabled at the factory). For region, DC, and rack names only alphanumeric characters and dashes are supported.
4. **Start the new host(s)**, if not already running.
5. From your Configuration Master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

   *More detail:*

   a. In your existing cluster, on your Configuration Master node change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the *system_setup.sh* tool.

   ```
   # ./system_setup.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the system setup tool with this command:

   ```
   $ hspkg setup
   ```

   Once launched, the setup tool's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

   b. In the tool's main menu select "**8**" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.

   c. Use the node preparation task menu to prepare the node:
      - Complete the configuration of network interfaces for the node, if you haven't already.
      - Set the timezone for the node.
      - Install and configure HyperStore prerequisites on the node.
      - Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
      - After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.

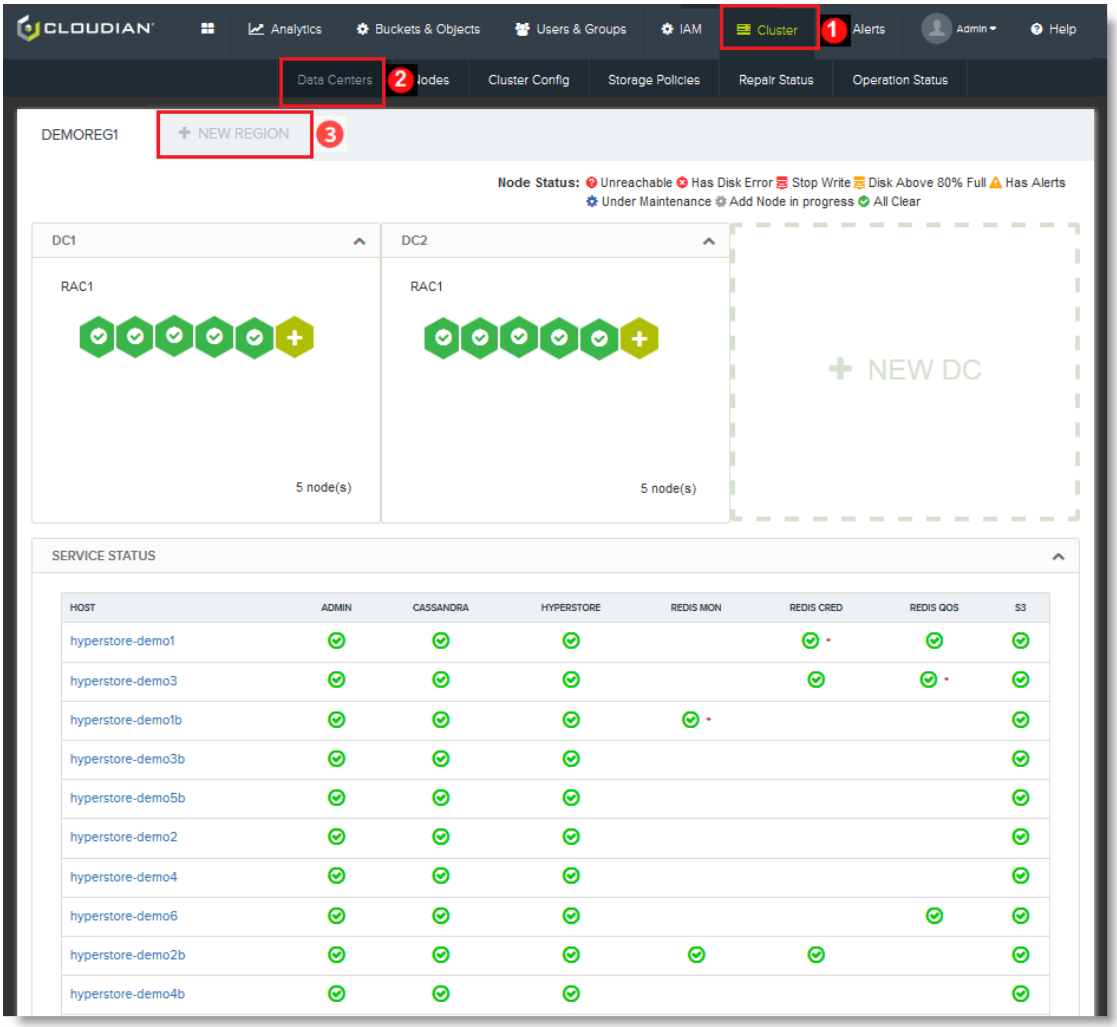
   d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the *system_setup.sh* tool.

**IMPORTANT !** If the new node(s) are not HyperStore Appliances and **if you do not use *system_setup.sh* to format the data disks on the new node(s)**, then in the installation staging directory on your Configuration Master node you must for each new node create a text file named *<hostname>_fslist.txt* that specifies the new node's data mount points, in this format:

```
<devicename> <mountpoint>
<devicename> <mountpoint>
etc...
```

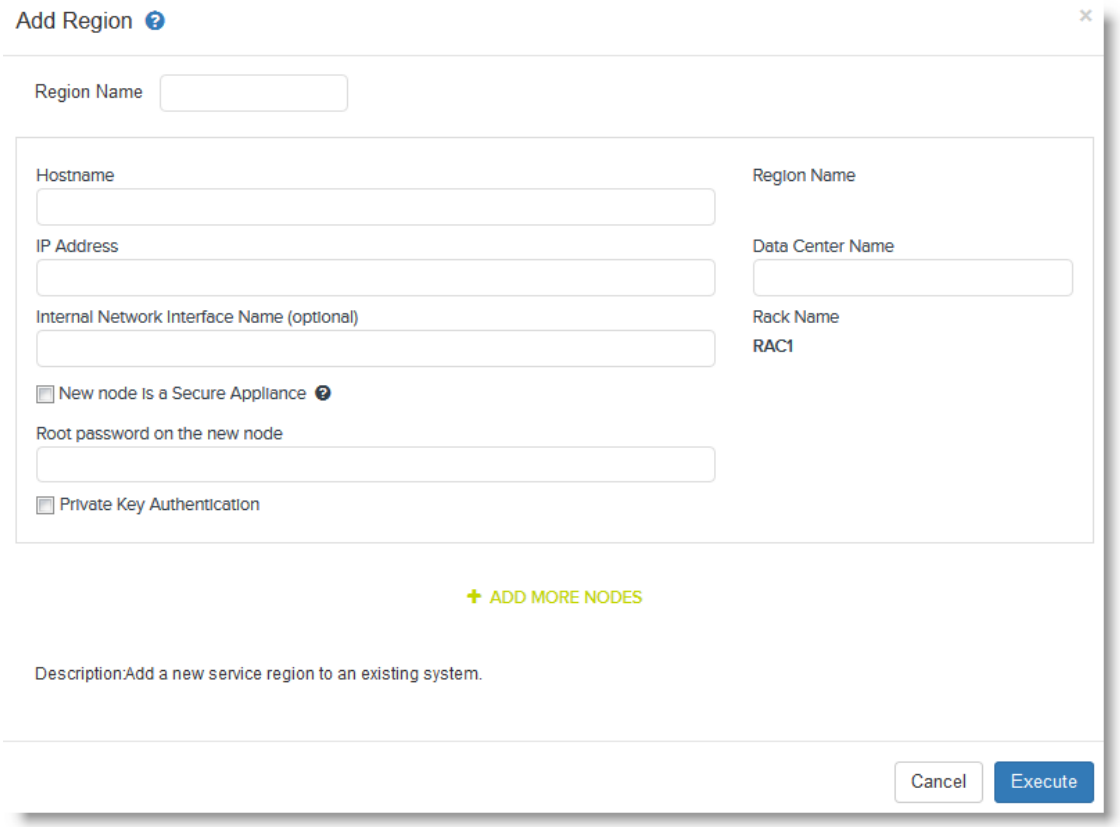
## 5.5.2. Adding a Region

1. In the CMC's **Data Centers** page, click the tab that says **+NEW REGION**.




2. In the **Add Region** interface that displays, enter the name of the new region then complete the fields for each node in the data center, clicking **Add More Nodes** to display fields for additional nodes as needed.

   *More detail:*




*Region Name (required)*

Name of the new region. Maximum 52 characters. Only ASCII alphanumerical characters and dashes are allowed. Letters must be lower case.

*Hostname (required; must be unique within system)*

Hostname of the new node.

**Note**
* This must be just a hostname, not an FQDN.
* Do not use the same hostname for more than one node in your entire HyperStore system. Each node must have a unique hostname within your entire HyperStore system, even in the case of nodes that are in different domains.

*IP Address (required)*

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

*Data Center Name (required)*

Name of the data center in which the new node is located. Maximum 256 characters. Only ASCII alpha-numerical characters and dashes are allowed.

*Internal Network Interface Name (optional)*

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your

cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

*Rack Name (fixed value "RAC1")*

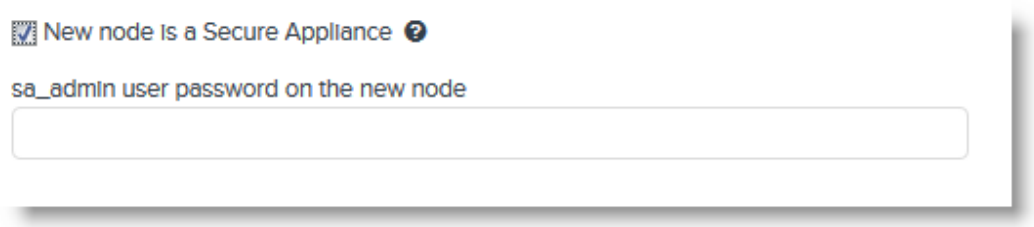
The "Rack Name" value is automatically fixed to "RAC1" for all nodes in the new region. You cannot edit this value.

> **Note** This is an internal value used by HyperStore. It does not need to correspond to any actual rack name in your data center(s).

*Authentication fields (based on new node type)*

During the **Add Region** operation, the HyperStore installer on your Configuration Master node needs to securely connect to each new node. The authentication options for doing so depend on the new node type:

- **If the new node is a "Secure Appliance"** (a HyperStore Appliance for which the HyperStore Shell [HSH] was enabled and the root password disabled at the factory), select the "New node is a Secure Appliance" checkbox. Then enter the *sa_admin* user's password for the new node.




> **Note** Before the new node is added to your HyperStore cluster, the *sa_admin* user's password on the new node may be different than the *sa_admin* user's password in your existing cluster. If so, after the new node is added to the cluster, the *sa_admin* user's password on the new node will be automatically changed to match the *sa_admin* user's password in the cluster.

- **If the new node is not a "Secure Appliance"** -- that is, if the new node is a standard Appliance or a software-only node on commodity hardware -- then you can use either one of these authentication methods (use one method or the other -- not both):
  - Enter the *root* user's password for the new node.

    OR

  - Select the "Private Key Authentication" checkbox. In this case the installer will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:
    - If during installation of the cluster you let the installer generate an SSH key pair for you, or you used your own existing SSH key pair and you copied both the private and the public key into the installation staging directory on the Configuration Master, then distribution of the public key to the new node will be taken care of

automatically by the installer.

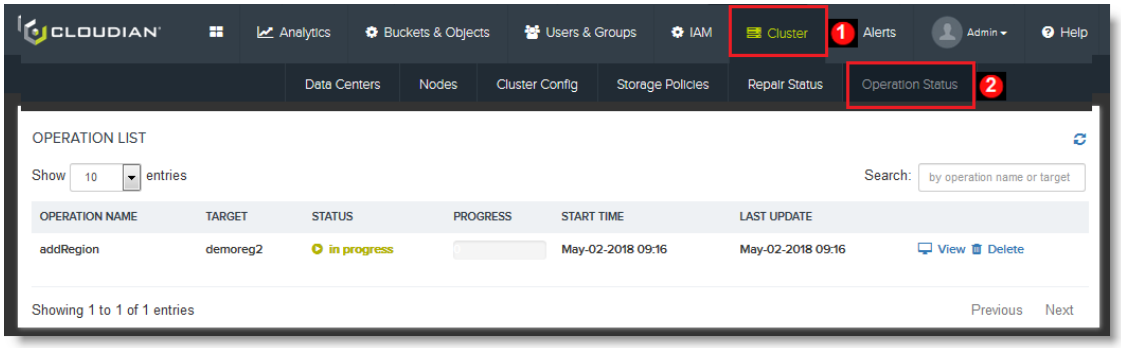
- If during installation of the cluster you used your own existing SSH key pair and you copied only the private key into the installation directory -- and you copied the public key to the target installation nodes manually -- **then you must also copy the public key to the new node manually**, before executing the **Add Region** operation.

3. Click **Execute**. This initiates a background operation that will take anywhere from several minutes up to an hour to complete depending on your environment. When it completes successfully the **Data Centers** page will display an additional tab representing the newly added region, with a green, check-marked cube icon for each of the new region's nodes.

   *More detail:*

   The **Add Region** operation entails verifying that the new hosts meet HyperStore requirements, installing software, updating system configuration, starting services, and joining the new nodes into a new Cassandra ring.

   Use the **Operation Status** page to monitor progress of the operation.




   For status detail click **View** to the right of the summary status line. (The page may at one point indicate that it cannot retrieve status information -- this is due to an S3 Server / Admin Server restart which is an automatic part of the **Add Region** operation. Wait a few minutes then refresh the page.)

   When the operation is complete, to see the new nodes in the **Data Centers** page you may need to refresh the page in your browser.

   If you hold your cursor over each cube in the new region the node host names will display.

   > **Note** If the **Operation Status** page indicates that the **Add Region** operation has failed, click "View" for detail. Then for more information to support troubleshooting efforts, *grep* for "ERROR" level messages in the *cloudian-installation.log* file under the installation staging directory on your Configuration Master node.

4. **Update your DNS and load balancing** configurations to include the new service region and its nodes, if you have not already done so. Note that name server configurations in each of your existing region(s) and the new region must have entries for **all** your regions' S3 service endpoints, as well as for the global Admin service and CMC service endpoints. For more information see **"DNS Set-Up"** (page 50) and **"Load Balancing"** (page 53).

5. Go to the **Storage Policies** page (**Cluster -> Storage Policies**) and select the new region. Then **create a storage policy for the new region**. This first storage policy will become the default storage policy for

the region. Later if you've created more than one storage policy in the region you can change which policy is the default policy if you wish. Until you create one or more storage policies in the new region and users subsequently create buckets that use those storage policies, no S3 object data will be stored in the new region.

6. Optionally, set Quality of Service (QoS) limits for users' and groups' activity in the new service region. Each service region has its own QoS configuration. By default, in each region no QoS limits are enforced. For more information, while logged into the CMC's **Manage Groups** or **Manage Users** page click **Help**. When using the QoS configuration interfaces be sure to select the new service region from the interfaces' drop-down list of regions.

This completes the procedure for adding a service region to your system.

## 5.6. Removing a Node

This procedure is for **permanently removing a node from your cluster** so that the data storage responsibilities of that node are redistributed to the remaining nodes in the cluster. During this procedure you will:

- Verify that your existing system is in a proper condition to successfully support the removal of a node
- Remove the node
- If the node you removed was "dead" -- the Cassandra Service on the node is down or unreachable when you remove the node -- repair the remaining nodes in your cluster (this is not necessary if the node you removed was live)

**Note** **If you are removing a node after having added a new node** to your cluster, you must complete the rebalance operation for the new node before removing a node. For more information on rebalance see **"Adding Nodes"** (page 264).

**IMPORTANT !** Removing a node should be something that you do **only if absolutely necessary**. When you remove a live node from your cluster, during the decommissioning process the data replicas and erasure coded fragments on that node are unavailable to help meet your configured read consistency requirements, when servicing read requests from client applications. Once data has been streamed from the decommissioning node to the other nodes that data is again available to contribute to meeting read consistency requirements. Depending on the data volume and network bandwidth, it may take several days or more until the decommissioning process has completed for all of the node's data.

**Note** For instructions on **uninstalling an entire HyperStore system**, from all nodes, including deleting all data, see "cloudianInstall.sh Command Line Options" in the ***Cloudian HyperStore Installation Guide***.

### 5.6.1. Preparing to Remove a Node

Take these actions to prepare to remove a node from your cluster:

1. **Make sure that removing the node won't leave your cluster with fewer nodes than your configured storage policies require.**

*More detail:*

For example if 4+2 erasure coding is being used in your system you cannot reduce your cluster size to fewer than 6 nodes, even temporarily. Or for another example if you have a storage policy that for each object places 3 replicas in DC1 and 3 replicas in DC2, do not reduce the number of nodes in either data center to fewer than 3.

If you're not certain what storage policies currently exist in your system, check the CMC's **Storage Policies** page (**Cluster -> Storage Policies**).

The CMC's **Uninstall Node** function checks and enforces this requirement and will not let you remove a node if doing so would leave fewer than the required number of nodes in your cluster. If your cluster is currently at the minimum size required by your storage policies and you want to remove a node:

- If the node you want to remove is **live** you can first add a new node to your cluster by following the complete procedure for **"Adding Nodes"** (page 264) (including rebalancing); and then after that you will be able to remove the node that you want to remove.
- If the node you want to remove is **dead** contact Cloudian Support for guidance.

2. **Make sure that you have sufficient available storage capacity on the other nodes in your cluster**. The data from the removed node will be redistributed to the remaining nodes that are encompassed by the same storage policies as the removed node.

   *More detail:*

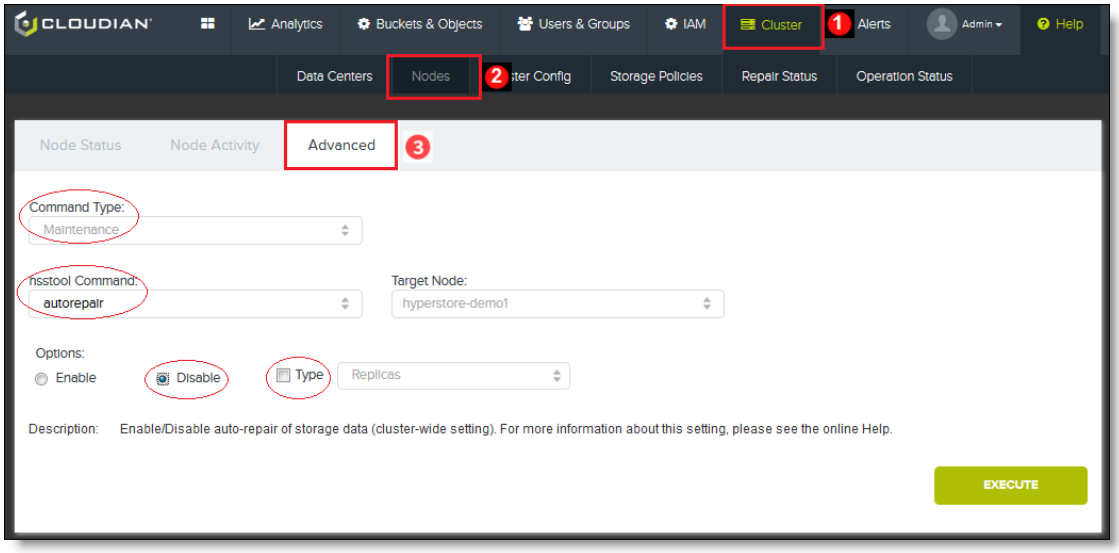
   Each remaining node must have available storage capacity to take on some of the data from the removed node. You can review your cluster and per-node storage space availability in the CMC's **Capacity Explorer** page (**Analytics -> Capacity Explorer**).

   > **Note** If any of the other nodes are in the **"stop-write" condition** -- or if any disks on a node are in stop-write condition -- at a time when you decommission a different node from your system, the decommissioning process overrides the stop-write restriction on the node(s) or disk(s) where it exists in order to stream to **all** nodes and disks in the cluster their share of data from the decommissioned node. Consequently, disks that were nearly full before a decommissioning operation may become completely full during the decommission operation, resulting in stream job failures once the disks can accept no more data.
   >
   > To avoid this, before removing a node from your cluster make sure there is plenty of space on all the remaining nodes and disks to absorb the data from the node you intend to remove. **If you want to remove a node from your cluster at a time when some disks on the other nodes are nearly full, consult with Cloudian Support**.

3. In the **Node Advanced** page, from the Maintenance command type group, execute the *autorepair* command with the Disable option to **temporarily disable the automated repair feature** in the service region in which you are removing a node.

   *More detail:*

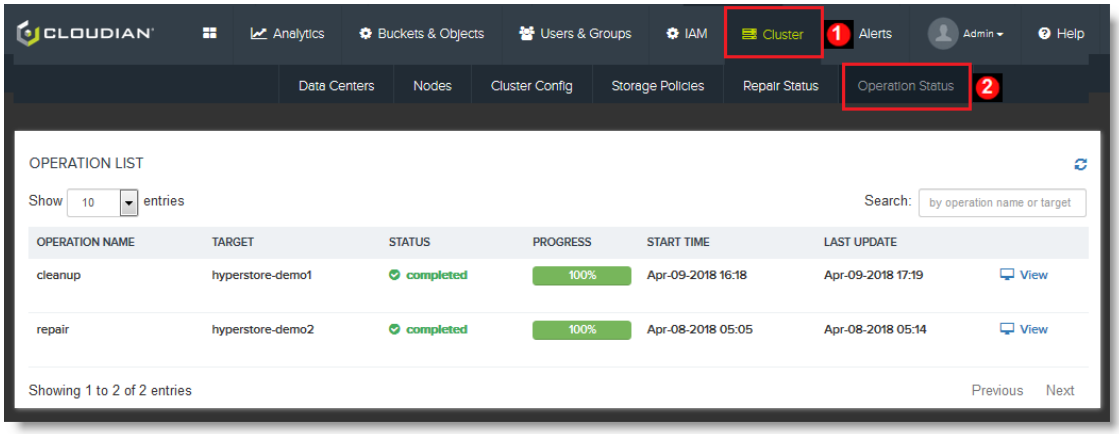
   The target node for the command can be any node in the region. Leave the "Type" option unselected so that all automated repair types are disabled. This will prevent any new schedule-based auto-repairs or proactive repairs from launching during the node removal process.




4. In the **Operation Status** page, **make sure there are no rebalance operations or other operations currently in progress** in the service region.

   *More detail:*

   If any operations are in progress, wait until the operations complete before removing a node from your cluster. The CMC's **Uninstall Node** function will not let you remove a node if a major operation such as rebalance, repair, or cleanup is running in the service region.



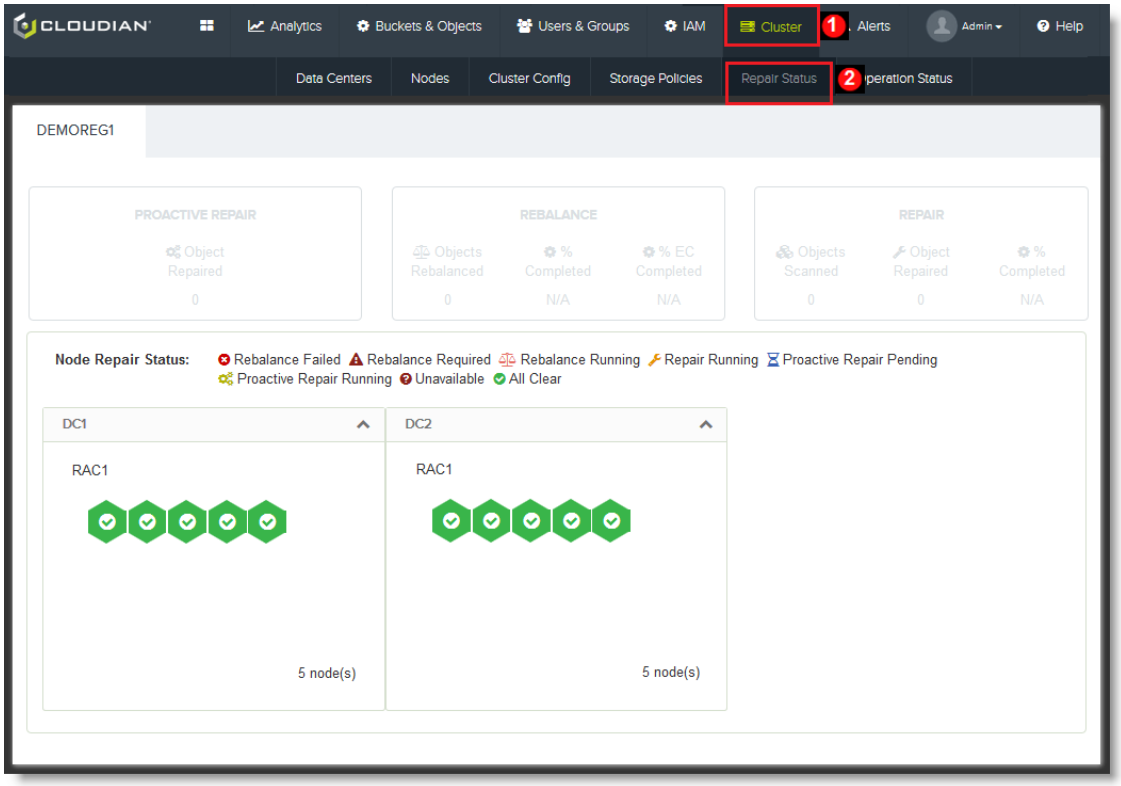

> **Note** If you don't want to wait for an in-progress repair or cleanup of a node to complete you have the option of terminating the operation. To do so, go to the **Node Advanced** page and for that node execute **hsstool repair** or **hsstool repairec** or **hsstool cleanup** or **hsstool cleanupec** with the "stop" option selected. Note that an *hsstool rebalance* operation does not support a stop option and cannot be terminated while in progress.

5. In the **Repair Status** page, **make sure there are no proactive repairs currently in progress** in the service region.

   *More detail:*

   If any proactive repairs are in progress, wait until they complete before removing a node from your cluster. The CMC's **Uninstall Node** function will not let you remove a node if a proactive repair is

running in the service region.

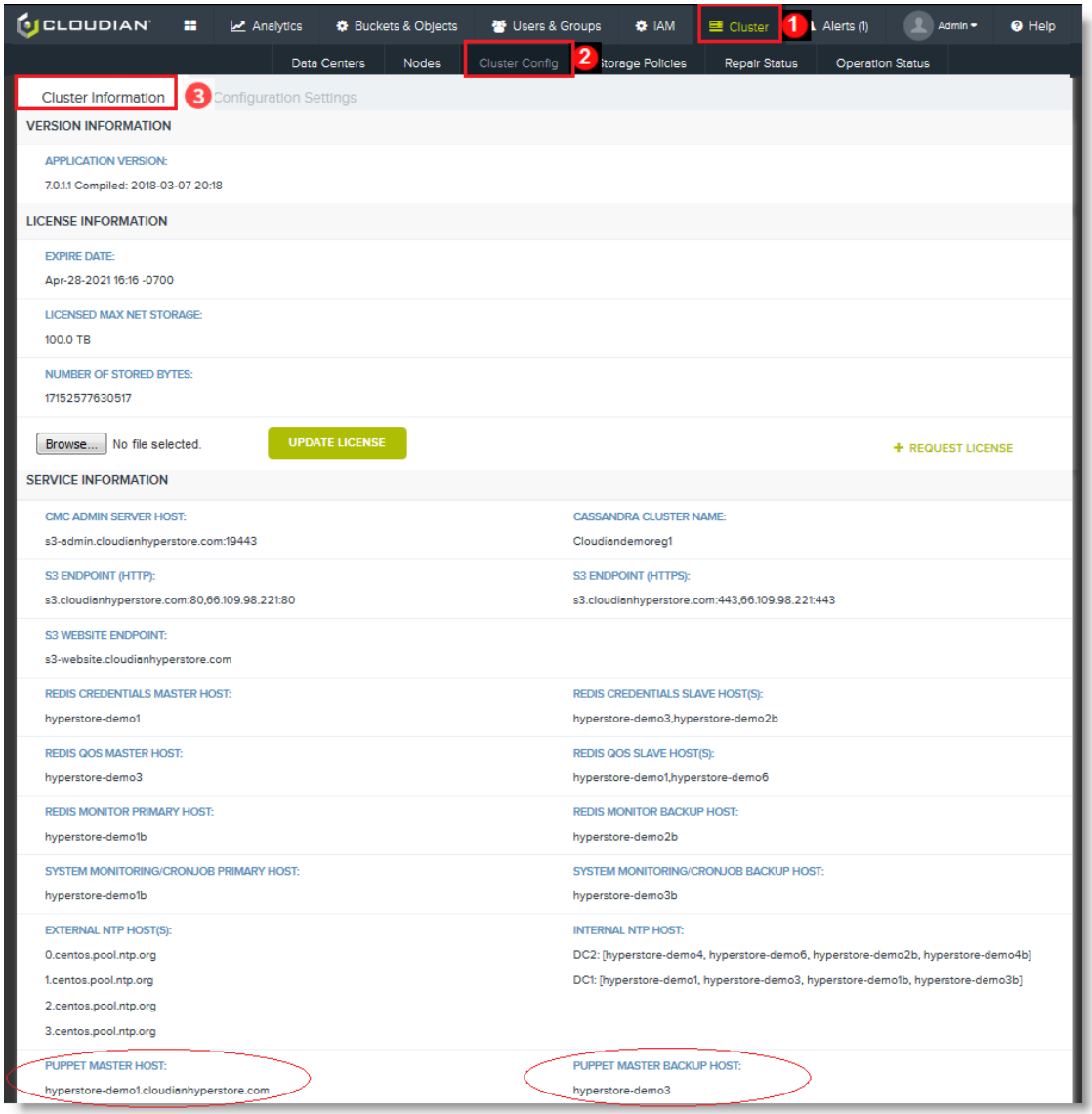



**Note** If you don't want to wait for an in-progress proactive repair of a node to complete you have the option of terminating the repair. To do so, go to the **Node Advanced** page and for that node execute **hsstool proactiverepairq** with the "stop" option selected.

6. **If the node you want to remove is the active Configuration Master node**, manually fail over the Configuration Master role from the primary instance to the backup instance.

   *More detail:*

   If you're not sure whether the node you want to remove is the Configuration Master (Puppet master) host, check the CMC's **Cluster Information** page. That page also shows which host is the Configuration Master backup host.



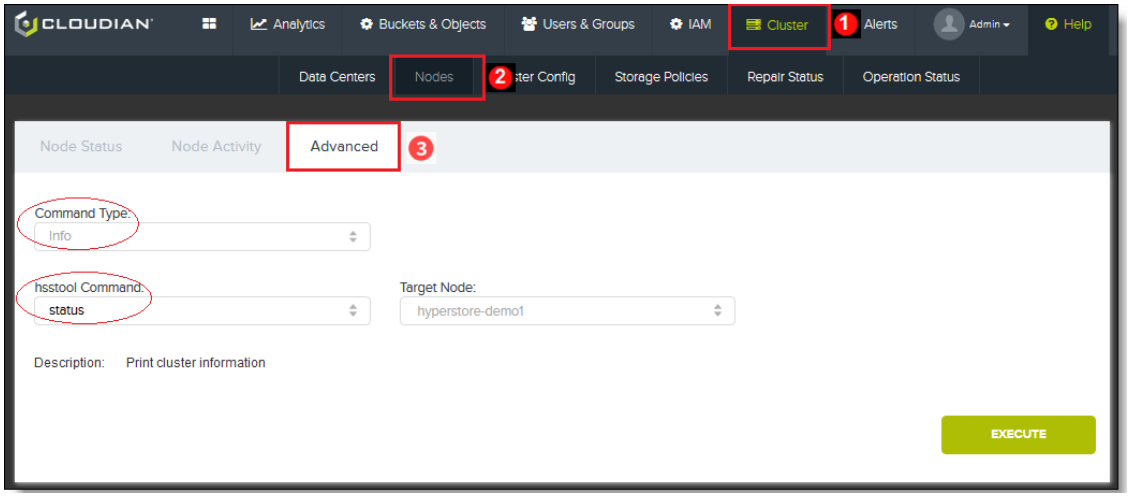

For instructions on failing over the Configuration Master from primary to backup, see **"Manually Fail Over the Configuration Master Role from the Primary to the Backup"** (page 315).

> **Note** Any other specialized services on the node -- such as Redis or Redis Monitor -- will be automatically moved to other nodes in the cluster by the CMC's **Uninstall Node** function.

7. In the **Node Advanced** page, from the Info command type group, submit the *status* command to any healthy node to retrieve status information for all nodes in the region. The **HyperStore Service must be up on all nodes** including the node that you are going to remove. The **Cassandra Service must be up on all nodes other than the one you are going to remove**. If Cassandra is down or unreachable on the node you are removing -- if the node is "dead" in terms of its relation to the Cassandra cluster -- you can still proceed with removing the node, but an extra step will be required toward the end of the node removal procedure (as described in **"Removing a Node"** (page 294)).

   *More detail:*

   The target node for the command can be any healthy node in the same service region as the node you want to remove -- the command returns status information for the cluster as a whole.




a. In the command response, for the **node that you want to remove** check the status of the Cassandra Service (in the "Cassandra" column) and the HyperStore Service (in the "Hss" column).

- **If the Cassandra status and the HyperStore Service status are both Up**, the data redistribution that will be required in the cluster when you remove the node will be executed by a decommissioning process that will be automatically invoked by the CMC's **Uninstall Node** function.
- **If the Cassandra status is Down or unreachable ("?")**, data redistribution by decommissioning is not supported. Instead, at the end of the remove node procedure you will need to run repair on all the remaining nodes in order to redistribute data in the cluster. Details are in the procedure below.

> **Note** If possible, start Cassandra on the node that you want to remove, so that the **Uninstall Node** function can automatically implement a decommissioning process and you won't have to perform repairs.

- **If Cassandra is Up but HyperStore Service is Down or unreachable**, the CMC's **Uninstall Node** function will abort if you try to run it. Before trying to remove the node, start the HyperStore Service on the node (or resolve the network access problem if there is one).

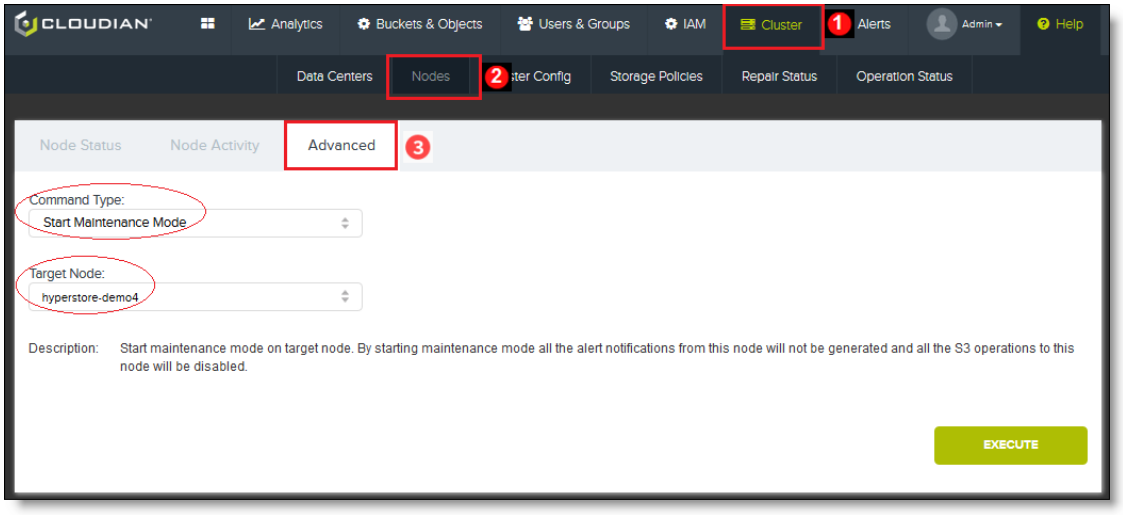
b. For **all the other nodes** confirm that Cassandra and the HyperStore Service are Up. The CMC's **Uninstall Node** function will abort if Cassandra or the HyperStore Service are Down or unreachable on any other node in the cluster. If those services are down on any of the other nodes, start the services (or resolve the network access problem if there is one) before trying to remove a node.

## 5.6.2. Removing a Node

1. If you have not already done so, log in to a CMC instance on a node **other than the node that you are going to remove**, but in the same service region as the node that you are going to remove.

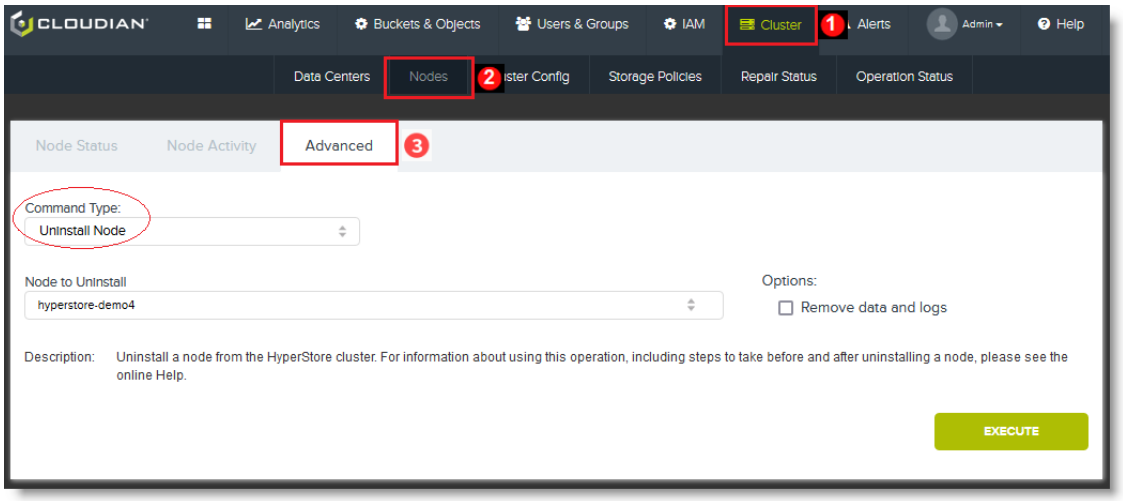
   *https://<IP_address_of_node_other_than_removal_node>:8443/Cloudian*

2. In the CMC's **Node Advanced** page, from Command Type drop-down list select **Start Maintenance Mode**. For the target node select the node that you want to remove from the cluster.



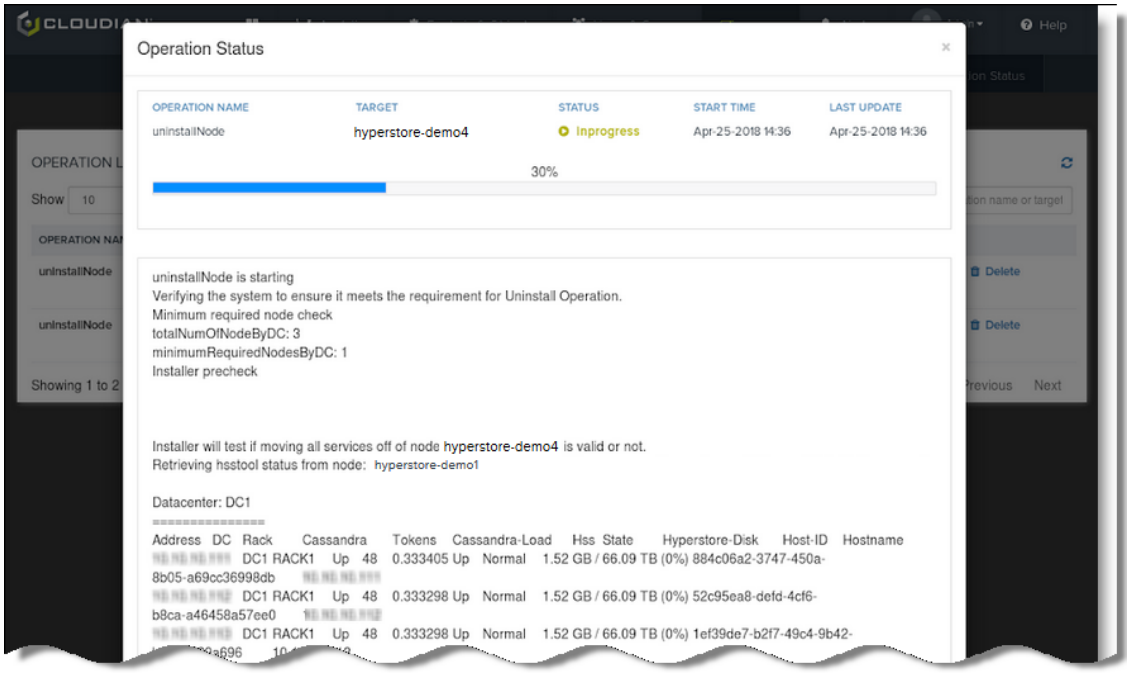

Then click **Execute**. This directs the rest of the cluster to stop sending S3 requests to the specified node.

3. In the **Node Advanced** page, from Command Type drop-down list select **Uninstall Node**. For the "Node to Uninstall" list select the node that you want to remove from the cluster. This operation will recreate the selected node's data on to other nodes remaining in the cluster (if the node you are removing is a "live" node), remove the node from the cluster, and remove HyperStore software from the node. If you want also for the HyperStore data and logs to be removed from the node, select that option.




Then click **Execute**. After you confirm that you want to proceed, the operation is initiated.

4. Use the **Operation Status** page to periodically check on the progress of the **Uninstall Node** operation. To pop up a detailed status report click **View** next to the summary status line.




If the node you are removing is live, the **Uninstall Node** operation will include decommissioning the node -- streaming copies of its data to the remaining nodes in the cluster -- and this may take up to several days or more to complete. If the node you are removing is dead, the **Uninstall Node** operation will be much briefer.

> **Note** If you want to remove multiple nodes, wait until the **Uninstall Node** operation completes for one node before you start to uninstall the next node. The system does not support uninstalling multiple nodes concurrently.

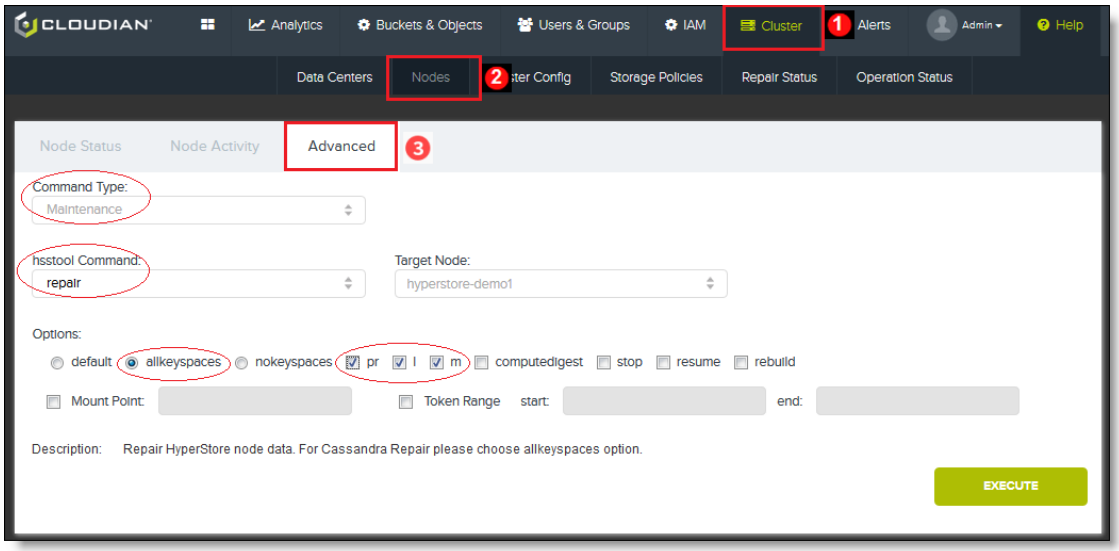
5. After the **Operation Status** page shows that the status of the **Uninstall Node** operation is Completed, go to the **Node Status** page (**Cluster -> Nodes -> Node Status**) and confirm that the removed node no longer appears in the "Host" drop-down list.
6. **If the node you removed was "dead"** -- meaning that the node's Cassandra Service was down or unreachable when you checked it (as described in **"Preparing to Remove a Node"** (page 289)) -- you must repair each of the remaining nodes in the region.

   *More detail:*

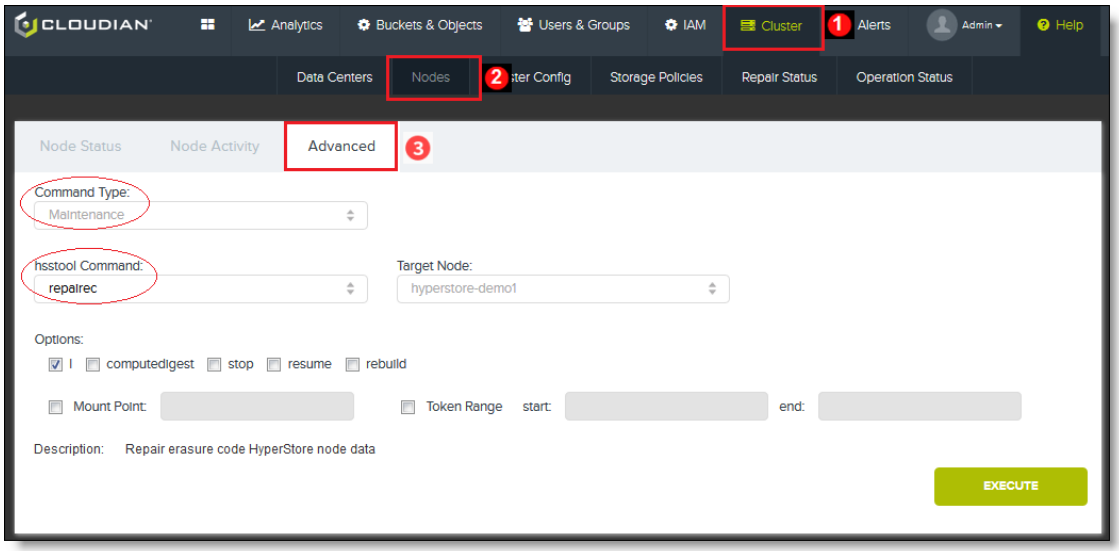
   If the node you removed was "dead", take the following actions to recreate the removed node's data on the remaining nodes in the cluster. (If the node you removed was "live" -- in which case the **Uninstall Node** operation executed a decommissioning process -- these actions are not needed and you can jump down to Step 7.)

   > **Note** If you have removed a dead node from your cluster as a precursor to adding a new node to your cluster -- if you wish you can defer the time-consuming repair operations called for below until after you have added the new node(s). If that's the case, you can now perform the **"Adding Nodes"** (page 264) procedure, and that procedure indicates the point at which you should initiate the deferred repairs. If you have removed a dead node and are **not** adding a new node, perform the repair now as described below.

From the **Node Advanced** page, run **hsstool repair** on each of the remaining nodes in the service region. When repairing each node, use the **allkeyspaces** option (so as to repair Cassandra metadata as well as S3 object data) and also the **-pr** option (this makes for more efficient repairs when repairing multiple nodes). Leave the **-l** and **-m** options selected, as they are by default. Since you are using the **-pr** option you can run repair on multiple nodes concurrently (in the GUI you will have to execute the repair command for each node individually, but you do not need to wait for the repair operation on one node to complete before you execute the command on the next node).



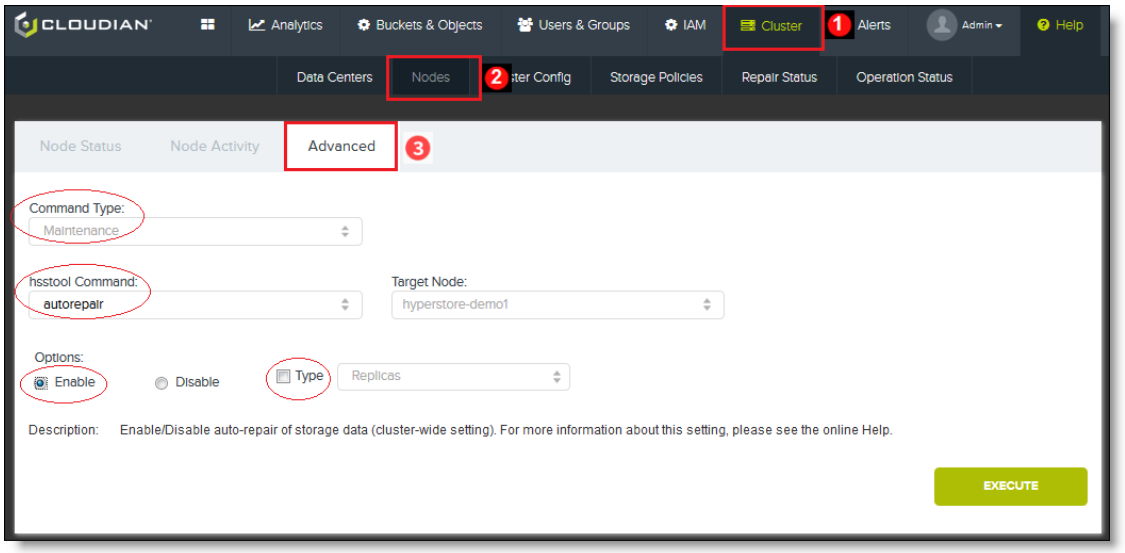

Also, if you have any erasure coded object data in your system, from the **Node Advanced** page run **hsstool repairec** on one node in each HyperStore data center in the region. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region. (Note that you do not need to wait for the in-progress *hsstool repair* operations to finish before launching *hsstool repairec* -- it's OK to run *hsstool repairec* and *hsstool repair* concurrently.)




Use the **Operation Status** page to track the progress of all repairs. After **all repairs have completed** proceed to Step 7.

> **Note** Because the *repairec* operation repairs erasure coded data on all hosts in a data center, it's potentially a very long running operation. In a large cluster with high data volume it may take multiple weeks to complete.

7. In the **Node Advanced** page, from the Maintenance command group execute the *autorepair* command with the Enable option to **re-enable the HyperStore automated repair features** in the service region. The target node can be any node in the region. Leave the "Type" option unselected so that all repair types are enabled. This also re-enables proactive repair.




This completes the procedure for removing a node from your cluster.

Note that the CMC's **Uninstall Node** function deletes the node from the HyperStore cluster configuration and removes all HyperStore software from the node. It does not delete the Cassandra metadata or HyperStore object data from the node.

> **IMPORTANT !** If the node was "live" when you removed it -- so that the **Uninstall Node** operation included a decommissioning process -- make sure that in the **Operation Status** page the **Uninstall Node** operation shows as having completed successfully with no errors, before you consider manually deleting the data that remains on the removed node. If the node was "dead" when you removed it -- so that you had to subsequently run repair operations on the remaining nodes in the cluster -- make sure that in the **Operation Status** page the **Uninstall Node** operation and also all those repair operations show as having completed successfully with no errors, before you consider manually deleting the data that remains on the removed node.

> **IMPORTANT ! If you want to re-use a removed node's host machine by adding it back to your HyperStore cluster as a new node**, then before using the host machine as a new HyperStore node you must remove all data from the host and re-install the operating system.

# 5.7. Replacing a Node

If you want to replace a **dead node** (for example a node that has failed due to hardware problems) with a new node:

- First perform the procedure for **"Removing a Node"** (page 289) (for the dead node).

  **Note** If your current number of nodes is at the minimum required by your storage policies, the system will not allow you to remove the node in the standard way. If this is your circumstance -- you have the minimum number of nodes required by your storage policies and one of those nodes is dead -- please contact Cloudian Support for guidance.

- Then perform the procedure for **"Adding Nodes"** (page 264) (for the new node).

If you want to replace a **live node** (a node on which the Cassandra Service and HyperStore Service are running and reachable) with a new node:

- First perform the procedure for **"Adding Nodes"** (page 264) (for the new node).
- Then perform the procedure for **"Removing a Node"** (page 289) (for the live node that you want to remove).

# 5.8. Changing a Node's IP Address

Changing the IP address of an existing installed HyperStore node is not recommended. There is no standardized, supported way of changing a HyperStore node's IP address through the CMC or through command line tools. For more detail on this topic contact Cloudian Support.

# 5.9. Restoring a Node That Has Been Offline

- **"Restoring a Node That Has Been Down for Less Than Four Hours"** (page 299)
- **"Restoring a Node That Has Been Down for More Than Four Hours"** (page 301)
- **"Restoring a Node That Has Been Down for More Than a Few Days"** (page 301)

## 5.9.1. Restoring a Node That Has Been Down for Less Than Four Hours

When a node has been down or inaccessible for just a short while -- less than four hours -- and then you bring it back online, the HyperStore system **automatically** performs the most important actions necessary for restoring the node to a correct and up-to-date condition:

- HyperStore will use **proactive repair** to automatically populate the node with any data that the node is responsible for storing but that is missing due to the node having been offline.

  **IMPORTANT !** The longest node outage that a proactive repair can cover for is four hours (by default configuration). **If a node is down for more than four hours you need to take manual steps to fully repair it.** For detail see **"Restoring a Node That Has Been Down for More Than Four Hours"** (page 301) or (if applicable to your circumstances) **"Restoring a Node That Has Been Down for More Than a Few Days"** (page 301).

- If the node that you are restoring is a Redis slave node (for either the Redis Credentials DB or the Redis QoS DB), when you bring the node back online it will automatically sync with the Redis master node to get the most current data.

**If you made any configuration changes to your cluster while the node was down**, from the Configuration Master node **push the changes out to the cluster** after the node is back up.

**Optionally, you can run a cleanup on the node** in order to remove from the node any data that should no longer be there. This would apply, for example, if service users deleted some of their stored objects from the system while the node was down. In this case after being brought back into service the node will still be storing replicas or erasure coded fragments of those objects, resulting in wasted use of disk space. Cleaning the node removes this "garbage" data. For cleanup command details see **"hsstool cleanup"** (page 504) (if you have erasure coded data in your system use the command's *-a* option so that it cleans erasure coded data as well as replica data).

> **Note** Before cleaning a node you should wait until any proactive repair that's automatically run on the node has completed. You can check this on the CMC's **Repair Status** page (**Cluster -> Repair Status**). Wait until the node's status displays as "All Clear", and then you can clean the node.

**Also optionally, you can return to the node any specialized service role that the node was playing before it went down**. If the node had been acting as a "master" or "primary" within one of the HyperStore system's specialized services, then when the node went offline that role would have failed over to a different node. If you want you can return the master or primary role to the restored node after it's back online — though it is not necessary to do so.

*How going down and then being brought back up affects a node's specialized service roles...*

The table below shows how having been down impacts a node's specialized service role(s).

| If before going down the node was… | Then while the node was down that role… | And when brought back online the node is now… |
|---|---|---|
| Credentials DB master | Automatically failed over to a Redis Credentials slave | Redis Credentials slave |
| QoS DB master | Automatically failed over to a Redis QoS slave | Redis QoS slave |
| Redis Monitor primary | Automatically failed over to the Redis Monitor backup | Redis Monitor backup |
| Cron job primary | Automatically failed over to the Cron job backup | Cron job backup |
| Configuration Master primary | May or may not have been manually failed over to the Configuration Master backup by an administrator | Still Configuration Master primary, or else Configuration Master backup |

To see what specialized service role(s) a restored node is currently playing, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

If you want to change the node's current role assignment(s), see the instructions for **"Change Node Role Assignments"** (page 305).

### 5.9.2. Restoring a Node That Has Been Down for More Than Four Hours

In **mts.properties.erb** the setting **"hyperstore.proactiverepair.queue.max.time"** (page 470) sets the maximum time for which proactive repair jobs can be queued for a node that is unavailable. The default is 4 hours. This time limit prevents Cassandra from being over-loaded with metadata relating to proactive repair, and ensures that proactive repair is used only for its designed purpose, which is to repair object data from a relatively brief time period.

**If a node is unavailable for longer** than *hyperstore.proactiverepair.queue.max.time*, then the metadata required for implementing proactive repair on the node will stop being written to Cassandra, and an alert will display in the CMC's **Alerts** page (**Alerts -> Alerts**). When the node comes back online you will need to:

1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until the proactive repair completes. You can check the CMC's **Repair Status** page periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the new and changed objects from during the period when proactive repair metadata was still being written to the Cassandra for the node.
2. After proactive repair on the node completes, manually initiate a full repair of the node. For information on manually initiating a repair see **hsstool repair** and **hsstool repairec**. This will repair the new and changed objects from the period after the proactive repair queueing time maximum was reached and before the node came back online.

### 5.9.3. Restoring a Node That Has Been Down for More Than a Few Days

If you have a node that has been down for more than a few days, consult with Cloudian Support for guidance on restoring the node. Special precautions may need to be taken in regard to the Metadata DB (Cassandra) on the node.

## 5.10. Putting a Node or DC into Maintenance

When you need to perform maintenance on a node or an entire data center, HyperStore supports temporarily removing that node or DC from service and disabling alerts from that node or DC.

### 5.10.1. Putting a Node into Maintenance

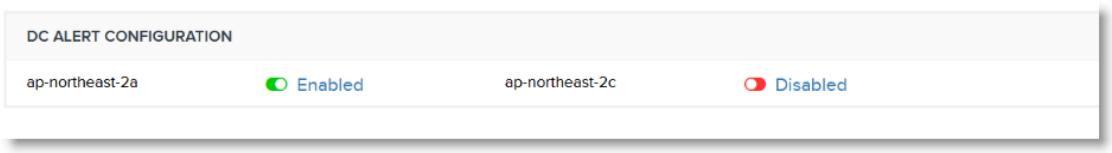
For instructions on putting a node into maintenance, while logged into the **Node Advanced** page (**Cluster -> Nodes -> Advanced**) click **Help**.

### 5.10.2. Putting a Data Center into Maintenance

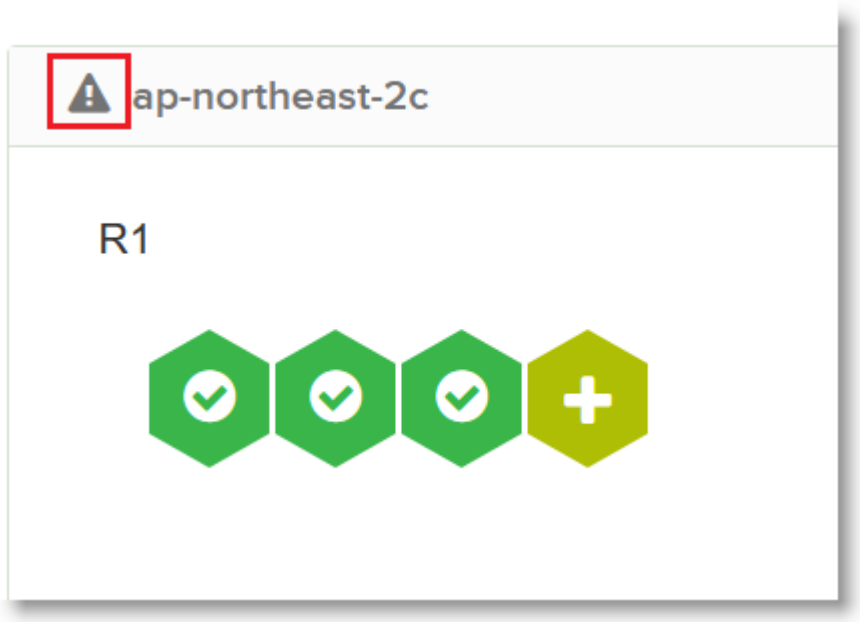
At a high level, the procedure for putting a data center into and out of maintenance is as described below. Do not put more than one DC into maintenance at a time.

1. For all "Replication Across Data Centers" and "Replicated EC" storage policies that you have configured in your system and that include the DC that you want to put into maintenance, use the CMC's **Storage Policies** page (**Cluster -> Storage Policies**) to change the read and write consistency requirement to "Local Quorum" (if that is not already the consistency requirement that you are using).

2. Use the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**) to see whether any of the following specialized service roles are being performed by nodes in the DC that you want to put into maintenance; and if so, move each such role to a node in a different DC. For instructions see **"Change Node Role Assignments"** (page 305).
   - Credentials DB master node
   - QoS DB master node
   - Redis Monitor primary node
   - Cron job primary node
   - Configuration Master primary node
3. Use the CMC's **Alert Rules** page (**Alerts -> Alert Rules**) to disable alerts related to the DC that you want to put into maintenance. Toward the bottom of the page, in the **DC Alert Configuration** section, set the DC to Disabled.




   To confirm that alerts for the DC are disabled, go to the **Data Centers** page (**Cluster -> Data Centers**) and next to the DC name you should see a gray triangle with an exclamation mark.




4. Perform your planned maintenance on the DC.

   **IMPORTANT !** Do not change the IP addresses of the nodes in the DC for which you are performing maintenance.

5. After completing maintenance on the DC, verify network connectivity from your operational DC(s) to the HyperStore nodes in the DC for which you performed maintenance.

6. Use the CMC's **Alert Rules** page to disable alerts related to the DC that you want to put into maintenance. Toward the bottom of the page, in the **DC Alert Configuration** section, set the DC to Enabled. To confirm that alerts for the DC are enabled, go to the **Data Centers** page and next to the DC name you should no longer see a gray triangle with an exclamation mark.
7. For all "Replication Across Data Centers" and "Replicated EC" storage policies that you have configured in your system and that include the DC for which you performed maintenance, use the CMC's **Storage Policies** page to change the read and write consistency requirement back to your desired settings.
8. Use the CMC's **Node Advanced** page to execute *hsstool repair* on each node in the DC for which you performed maintenance, one node at a time, allowing the repair to complete on one node before starting a repair on the next node. These are long-running operations. (This step is applicable only if you are using one or more replication [not erasure coding] storage policies in the DC.)
9. Use the CMC's **Node Advanced** page to execute *hsstool repairec* on any one node in the DC for which you performed maintenance. This repairs all erasure coded data in the DC. This is a long-running operation. (This step is applicable only if you are using one or more erasure coding storage policies in the DC.)

# 5.11. Backing Up and Restoring a Cluster

*Subjects covered in this section:*



This procedure is for backing up and restoring an entire HyperStore cluster (i.e. an entire HyperStore service region). This procedure should not be used for partial backups and restores.

Throughout this procedure, it's assumed that you have used the default installation directories for HyperStore binaries. If not, adjust the command paths stated in the procedure.

**IMPORTANT !** Make sure you have **enough space for the backup**. If you back up on to nodes in your HyperStore cluster, this will double disk usage within the cluster.

**Note** This procedure for backing up and restoring a cluster is **not supported if you have the HyperStore Shell enabled and the root password disabled**.

## 5.11.1. Note About Redis Backups

Daily at 11PM HyperStore implements a Redis cron job (configured in */etc/cron.d/redis-crontab*) that invokes a script (*/opt/redis/redisBackup*) on your Redis Credentials master node and on your Redis QoS master node. The script backs up the Redis Credentials database into a */var/lib/redis/dump-credentials.rdb* file and backs up the Redis QoS database into a */var/lib/redis/dump-qos.rdb* file. Prior to doing so it first moves the previous day's *dump-credentials.rdb* or *dump-qos.rdb* file into a */var/lib/redis/backup* directory, compresses the file, and appends a date-time stamp to its name (for example */var/lib/redis/backup/dump-credentials.rdb.2020111616.gz*). In the */var/lib/redis/backup* directory, files are rotated such that the most recent

seven days worth of Redis backups are retained and older backups are deleted.

So you always have access to Redis backup files that are generated at 11PM daily (more specifically, the backup starts at 11PM -- it may take some time to complete). However, if you want a more current Redis backup you can follow steps 1a and 1b below to re-generate the *dump-credentials.rdb* and *dump-qos.rdb* files.

Logging output from the *redisBackup* script runs is written to */var/log/redis/redisBackup.log*. (A separate log file *cloudian-redisBackup.log* merely records information regarding the launching of the backup script by *cron.d*.)

## 5.11.2. Backing Up an Entire Cluster

1. Back up the Redis Credentials and Redis QoS databases.
    a. On the Redis Credentials master node, use the Redis CLI to perform a **BGSAVE** operation:

    ```
    # /opt/redis/redis-cli -h <hostname> -p 6379 BGSAVE
    ```

    The above command will update the *dump-credentials.rdb* file in the Redis data directory (*/var/lib/redis* by default).

    > **Note** The BGSAVE operation saves the database in the background. To check when the save is done you can use the Redis CLI **LASTSAVE** command, which returns the Unix time of the most recently completed save.

    b. On the Redis QoS master node, use the Redis CLI to perform a **BGSAVE** operation:

    ```
    # /opt/redis/redis-cli -h <hostname> -p 6380 BGSAVE
    ```

    The above command will update the *dump-qos.rdb* file in the Redis data directory.

    c. Place a copy of the *dump-credentials.rdb*, *dump-qos.rdb*, *appendonly_cred.aof*, and *appendonly_qos.aof* files in a safe place.
2. Back up Cassandra and Hyperstore data **on each of your HyperStore nodes**, using the third party backup tool of your choice:
    - For Cassandra data, on each node:
        - First flush Cassandra:

        ```
        # /opt/cassandra/bin/nodetool -h <hostname> flush
        ```

        - Then with your third party tool, back up the Cassandra data directory. (To check which directory is the Cassandra data directory, see the configuration setting *common.csv*: **"cassandra_data_directory"** (page 418)).
    - For HyperStore data, on each node: With your third party tool, back up the HyperStore data directories. (To check which directories are the HyperStore data directories, see the configuration setting *common.csv*: **"hyperstore_data_directory"** (page 394)).
3. Back up HyperStore binaries and configuration files:
    - On the Configuration Master node:
        - */etc/cloudian-<version>-puppet* for HyperStore version 5.2 or later, or */etc/puppet* for versions older than 5.2
        - The current HyperStore installation staging directory (where the *cloudianInstall.sh* tool is)

- On each of your HyperStore nodes:
  - */opt/cloudian*
  - */opt/cassandra*
  - */opt/redis*
  - */opt/tomcat*
  - */opt/cloudianagent*
  - */opt/dnsmasq*

### 5.11.3. Restoring an Entire Cluster

This restore procedure assumes that the restored cluster is the same configuration as what you backed up — i.e. the same IP addresses and mount points.

1. Restore HyperStore binaries and configuration files.
   - On the Configuration Master node:
     - */etc/cloudian-<version>-puppet* for HyperStore version 5.2 or later, or */etc/puppet* for versions older than 5.2
     - The HyperStore installation staging directory (where the *cloudianInstall.sh* tool is)
   - On each of your HyperStore nodes:
     - */opt/cloudian*
     - */opt/cassandra*
     - */opt/redis*
     - */opt/tomcat*
     - */opt/cloudianagent*
     - */opt/dnsmasq*
2. Restore Cassandra and HyperStore data directories **on each of your HyperStore nodes**.
3. Restore the Redis Credentials and Redis QoS databases.

## 5.12. Changing Node Role Assignments

### 5.12.1. Change Node Role Assignments

In a HyperStore cluster there are certain specialized service roles that are assigned to some nodes and not to others. When you install a HyperStore system, the installer assigns these roles automatically and in a way that's appropriate to your cluster topology. The roles are implemented in such a way that no node constitutes a single point of failure.

For a summary of where specialized service roles are currently assigned in your cluster, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

If you wish you can change node role assignments, by using the HyperStore installer on the Configuration Master node. The installer supports the role assignment operations listed below.

**Note** Each of the role assignment change operations listed below entails doing a configuration push and a restart of the affected service (as described in the instructions for each operation). If you need to perform multiple of these role assignment change operations, do them one operation at a time and with a configuration push and affected service restart at the end of each operation. Do **not** perform multiple different role assignment change operations while deferring the configuration push and service restart.

- **"Move the Credentials DB Master Role or QoS DB Master Role"** (page 317)
- **"Move or Add a Credentials DB Slave or QoS DB Slave"** (page 306)
- **"Move the Cassandra Seed Role"** (page 308)
- **"Reduce or Change the List of CMC Hosts"** (page 309)
- **"Move the Redis Monitor Primary or Backup Role"** (page 320)
- **"Move the Cron Job Primary or Backup Role"** (page 310)
- **"Move the Configuration Master Primary or Backup Role"** (page 313)
- **"Change Internal NTP Servers or External NTP Servers"** (page 312)

## 5.12.2. Move or Add a Credentials DB Slave or QoS DB Slave

By default the HyperStore installer deploys two **Credentials DB slaves** and one **QoS DB slave** per data center. The system supports moving a slave from its current host to a different host -- for example, moving a Credentials DB slave to a host that is not currently running the Credentials DB service. It also supports adding a new Credentials DB or QoS DB slave rather than moving one — so that you end up with more slaves than when you started.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

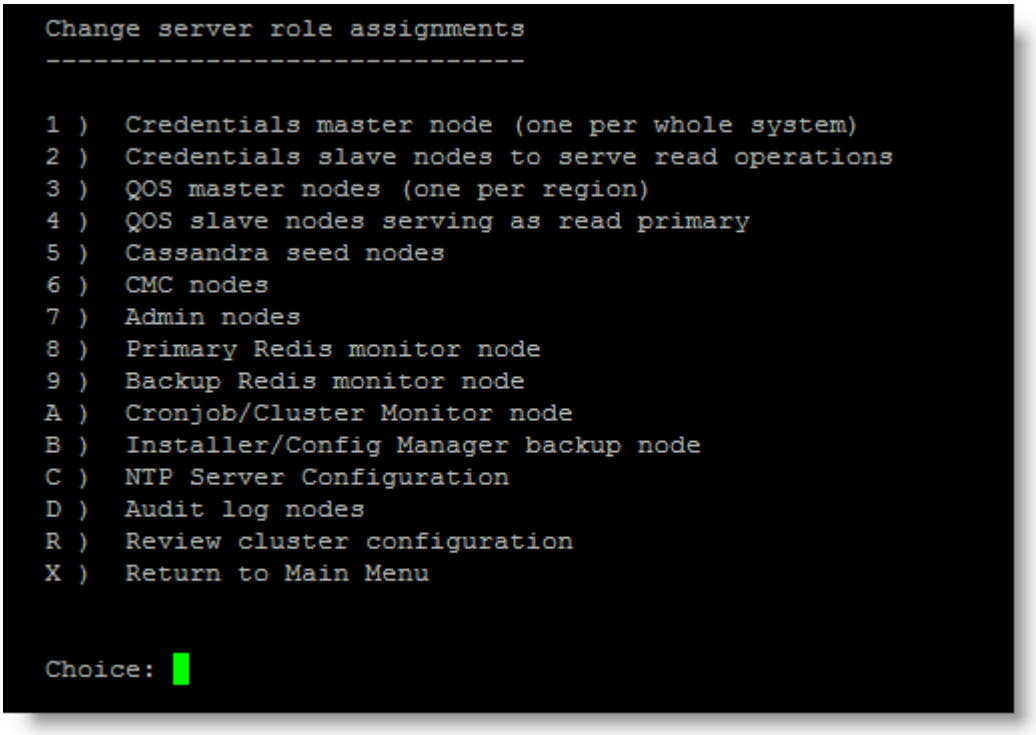
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




3. From the Change Server Role Assignments menu select the option for Credentials slave nodes or the option for QoS slave nodes.
4. The installer prompts you to specify a comma-separated list of all the hosts on which you want slaves to run. The installer's prompt text indicates [in brackets] which hosts are the **current** slaves. You can use your entry at the prompt to either move a slave or add a slave.

   Example of moving a slave:

   ```
   Enter a comma separated list of Credentials slave hosts in region1's
   DC1 data center [deneb,vega]: deneb,altair
   ```

   Example of adding a slave:

   ```
   Enter a comma separated list of Credentials slave hosts in region1's
   DC1 data center [deneb,vega]: deneb,vega,altair
   ```

   > **Note** If your HyperStore system has multiple data centers, the installer will prompt you separately for each data center's slave host list. If for some data centers you want to continue using the same slave(s) you can just press enter at the prompt rather than entering a host list.

5. After completing the interaction for specifying the slave locations, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu, then choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the following services:
   - Redis Credentials or Redis QoS (whichever service you moved or added a slave for)
   - Redis Monitor

- S3 Service (restarting this service also results in a restart of the Admin Service)

After these services have successfully restarted you can exit the installer.

To verify your change, log into the CMC and go to the **Node Status** page (**Cluster -> Nodes -> Node Status**). Review the service status information for the node(s) on which you've located the slave(s). Among the listed services on the node(s) should be "Redis Cred" (for a Redis Credentials slave) or "Redis QoS" (for a Redis QoS slave). The absence of "(Master)" appended to the service name indicates it's a slave instance, not a master.

## 5.12.3. Move the Cassandra Seed Role

HyperStore's Metadata DB is built on Cassandra, which runs on every node in your HyperStore cluster. A subset of the nodes serve as Cassandra "seed" nodes. Cassandra seed nodes serve to bootstrap the Gossip process for new nodes when you add new nodes to your cluster. Gossip is the peer-to-peer communication protocol by which Cassandra nodes regularly exchange state information. The recommended configuration is to have three of your Cassandra nodes act as seed nodes in each data center in which you have deployed HyperStore. The HyperStore installer automatically makes these role assignments when you install your system.

If you wish you can change the list of Cassandra seed nodes for your system, by following the steps below.

**Note** Any of your nodes can play the "seed" role. But bear in mind the recommendation that you have three seed nodes per data center. For performance reasons it's not advisable to have more seed nodes than this.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

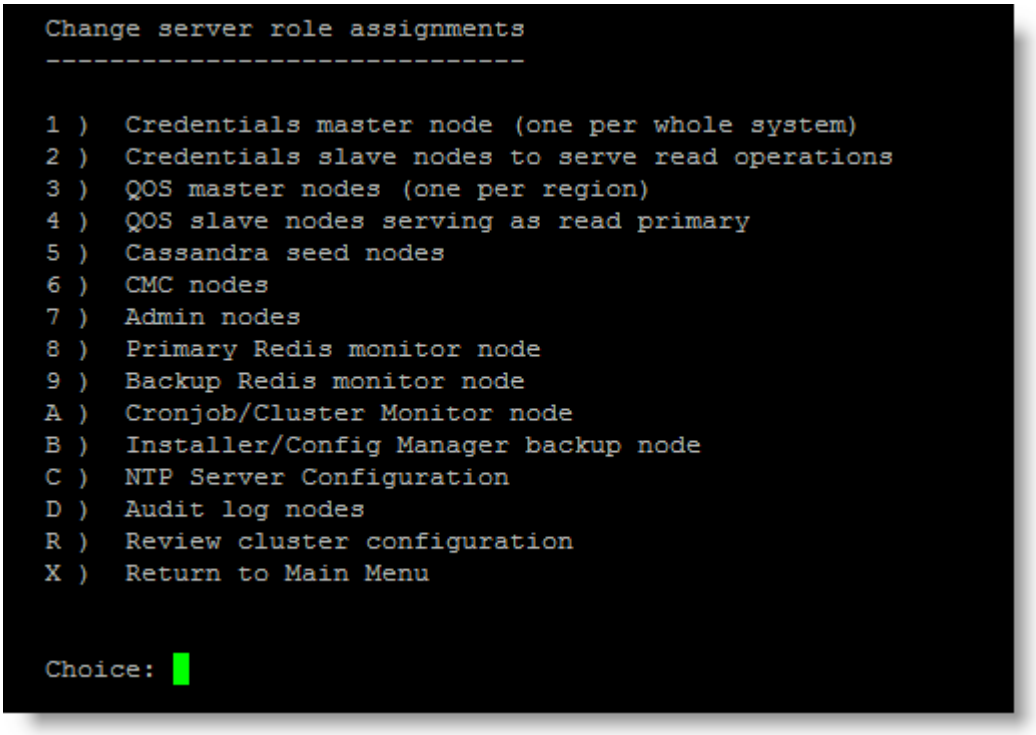
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




3. From the Change Server Role Assignments menu select "Cassandra seed nodes".
4. At the prompt enter a comma-separated list of hosts that you want to serve as Cassandra seed nodes. If you want to keep the same host just press Enter rather than specifying a different host. (If you have a multi-region system, you will be prompted for a list of Cassandra seed nodes for each region.)
5. After completing the interaction for specifying NTP Server Configuration, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the Cassandra service. After the Cassandra service has successfully restarted you can exit the installer.

To verify your change, you can look at the */opt/cassandra/conf/cassandra.yaml* configuration file on any one of your nodes and confirm that the "seeds:" parameter is set to the list of hosts that you specified.

## 5.12.4. Reduce or Change the List of CMC Hosts

By default the Cloudian Management Console (CMC) service is installed and runs on all of your HyperStore hosts. If you wish, after initial deployment of your system you can use the installer to specify a list of HyperStore hosts on which to have the CMC run, rather than having it run on all nodes.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

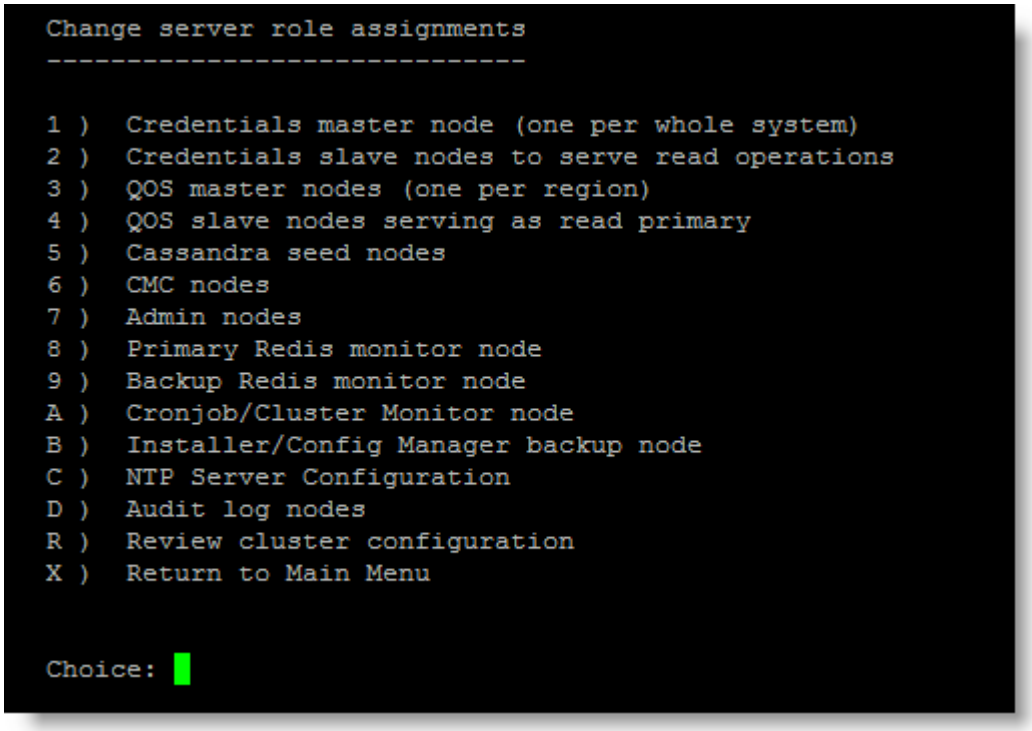
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

```
$ hspkg install
```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




3. From the Change Server Role Assignments menu select "CMC nodes".
4. At the prompt enter a comma-separated list of hosts on which you want the CMC to run.
5. After completing the interaction for specifying your CMC host list, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the CMC service. After the CMC service has successfully restarted you can exit the installer.

## 5.12.5. Move the Cron Job Primary or Backup Role

Certain HyperStore **system maintenance tasks are implemented by cron jobs** that run on a regular schedule, as configured by a crontab. When your HyperStore system is installed the install script designates one node to host the crontab configuration and run the cron jobs, and a second node to serve as a backup. In the event that the primary cron job host goes offline (or if *crond* goes down) the backup host **automatically** takes over as the active cron job host.

The HyperStore **Monitoring Data Collector** resides on the same primary host and backup host as the cron jobs. If the cron jobs role automatically fails over from the primary host to the backup host, the Monitoring Data Collector role also fails over to the backup.

The system supports moving the primary cron job role (and with it, the primary Monitoring Data Collector role) to a different host as described below. The same procedure also supports moving the backup cron job role (and with it, the backup Monitoring Data Collector role) to a different host.

**Note** Do not assign the primary cron job role to the same host as your Configuration Master role. If the cron job primary and the Configuration Master are on the same host and that host goes down, automated fail-over from the cron job primary to the cron job backup will not work.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

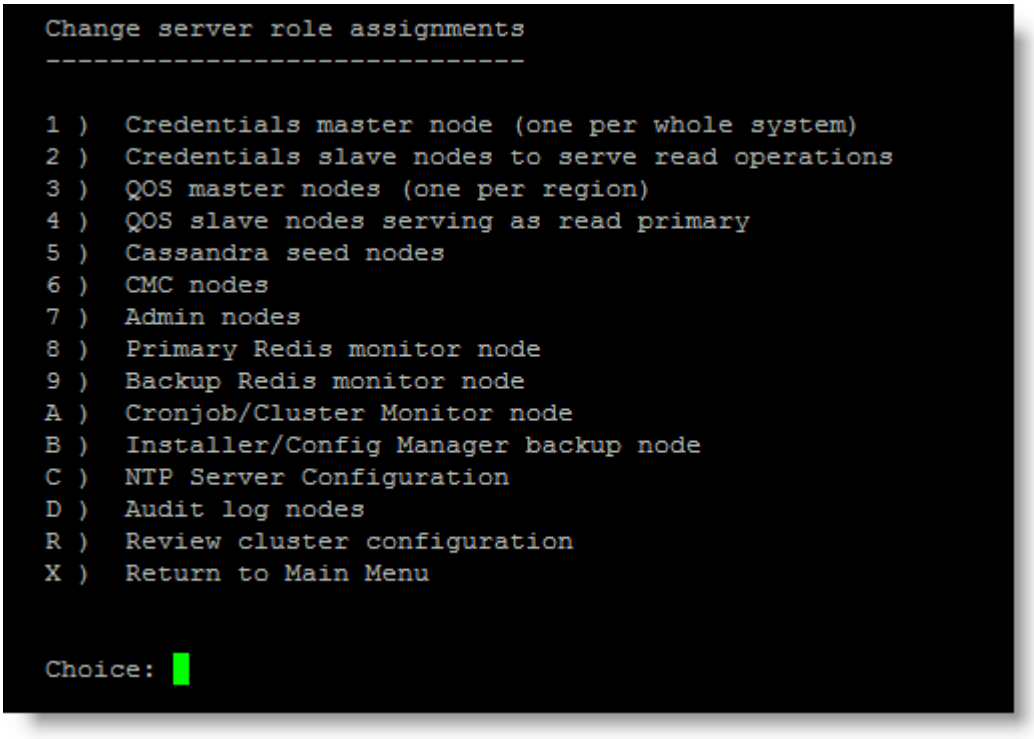
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**,rom any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

3. From the Change Server Role Assignments menu select "Cronjob/Cluster Monitor node".
4. At the prompts specify your desired primary host and backup host for the cron jobs / cluster monitor.

5. After completing the interaction for specifying cron job hosts, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. When Puppet pushes the current configuration settings to the cluster it will also automatically restart *cron.d* on the affected nodes. You do not need to manually restart any services. When the Puppet push completes you can exit the installer.

To verify your change, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that your System Monitoring/Cronjob primary and backup hosts are what you want them to be.

## 5.12.6. Change Internal NTP Servers or External NTP Servers

When you install your HyperStore cluster, for each of your data centers the installation script automatically configures four of your HyperStore nodes to act as internal NTP servers for the cluster. These internal NTP servers synchronize with external NTP servers. By default the set of external servers is:

- 0.centos.pool.ntp.org
- 1.centos.pool.ntp.org
- 2.centos.pool.ntp.org
- 3.centos.pool.ntp.org

> **Note** For more on how HyperStore automatically configures an NTP set-up for the cluster, see **"NTP Automatic Set-Up"** (page 496)

As described below, the system supports changing the list of internal NTP servers (for data centers in which you have more than four HyperStore nodes) and/or changing the list of external NTP servers.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

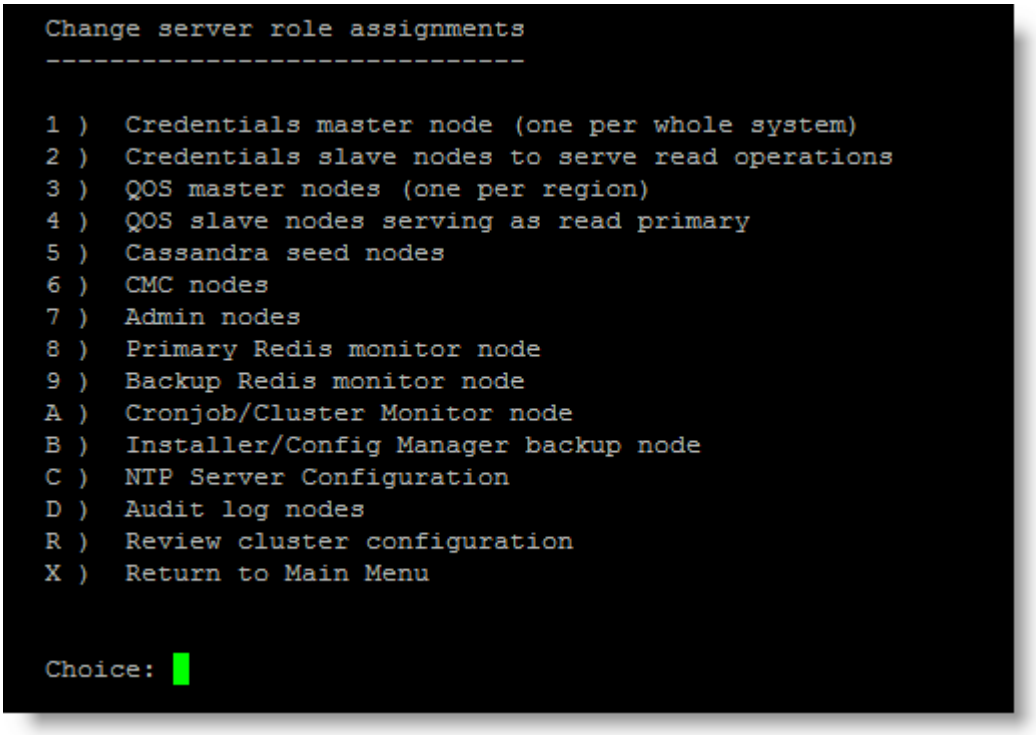
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




3. From the Change Server Role Assignments menu select "NTP Server Configuration".
4. At the first prompt specify the HyperStore hosts that you want to act as the internal NTP servers, as a comma-separated list. You must use host names, not IP addresses. If you want to keep the same hosts that the system is currently using, just press Enter rather than specifying different hosts.

   **Note** If you have multiple HyperStore data centers you will be prompted separately for the internal NTP host list for each data center.

5. At the next prompt specify the external NTP servers with which you want the internal NTP servers to synchronize, as a comma-separated list. For these external servers you can use either FQDNs or IP addresses. If you want to keep the same external servers that the system is currently using, just press Enter rather than specifying a different server list.
6. After completing the interaction for specifying NTP Server Configuration, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
7. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. When Puppet pushes the current configuration settings to the cluster it will also automatically restart *ntpd* on the affected nodes. You do not need to manually restart *ntpd*. When the Puppet push completes you can exit the installer.

To verify your change, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that your internal NTP hosts and external NTP hosts are what you want them to be.

## 5.12.7. Move the Configuration Master Primary or Backup Role

When you install your HyperStore system, you choose the node that you want to serve as the Configuration Master for cluster configuration management, and you run the installer on that node. The installer configures

that node to be the primary Configuration Master, and also configures a second node to be the Configuration Master backup. Any edits that you make to configuration templates on the Configuration Master primary are automatically sync'd to the Configuration Master backup. If the primary goes down, you can **manually** fail over the active Configuration Master role to the backup host.

There are two different operations that you can perform in regard to the Configuration Master role:

## Move the Configuration Master Backup

In this scenario your primary Configuration Master is up and running properly, and you're simply looking to relocate the Puppet Master backup role to a different host than it is currently on.

> **Note** You can find out which host is currently the Configuration Master backup host in the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

1. On the Configuration Master primary host, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Puppet master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

   ```
   Change server role assignments
   ------------------------------

   1 )  Credentials master node (one per whole system)
   2 )  Credentials slave nodes to serve read operations
   3 )  QOS master nodes (one per region)
   4 )  QOS slave nodes serving as read primary
   5 )  Cassandra seed nodes
   6 )  CMC nodes
   7 )  Admin nodes
   8 )  Primary Redis monitor node
   9 )  Backup Redis monitor node
   A )  Cronjob/Cluster Monitor node
   B )  Installer/Config Manager backup node
   C )  NTP Server Configuration
   D )  Audit log nodes
   R )  Review cluster configuration
   X )  Return to Main Menu


   Choice:
   ```

3. From the Change Server Role Assignments menu select "Installer/Config Manager backup node". ("Installer/Config Manager" is what this menu calls the Configuration Master.)
4. At the prompt specify the host to which you want to move the Configuration Master backup role.
5. After completing the interaction for specifying the Configuration Master backup host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
7. Exit the installer, wait at least one minute, then log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that your Configuration Master primary and backup hosts are what you want them to be.

## Manually Fail Over the Configuration Master Role from the Primary to the Backup

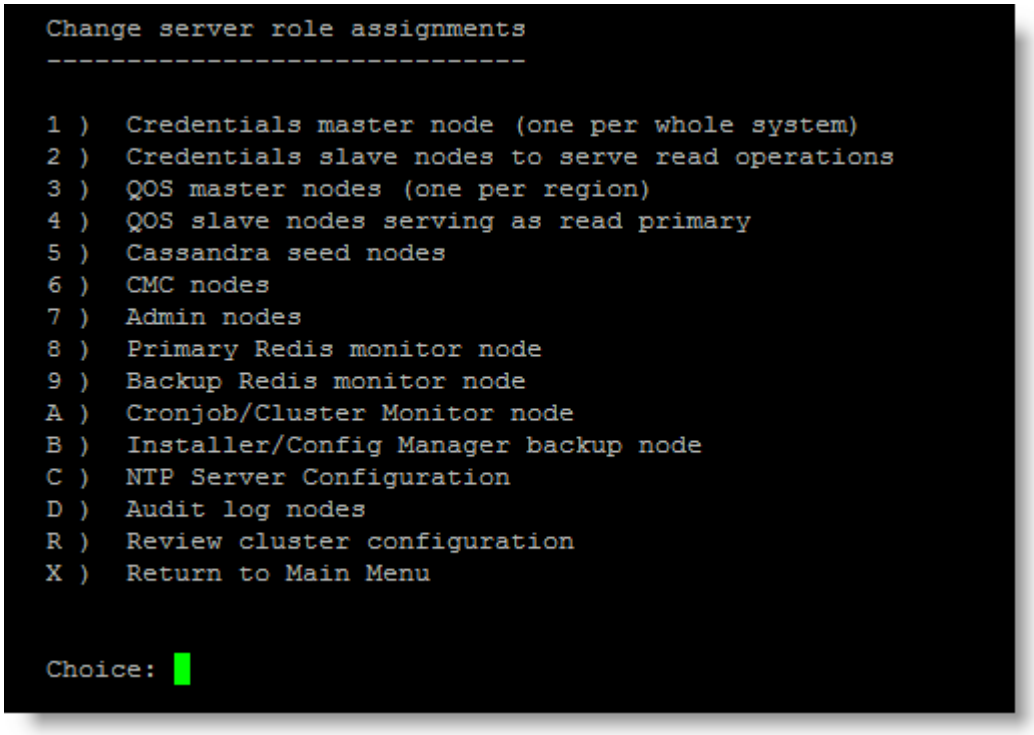
In this scenario there is a problem with your primary Configuration Master, and you want the backup Configuration Master to become active.

**IMPORTANT !** For this procedure you **log into the current Configuration Master backup host** and implement the whole procedure from that host. You can find out which host is currently the Configuration Master backup host in the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

1. **On the Configuration Master backup host**, change into the installation directory and then launch the HyperStore installer.

```
# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Start or stop Puppet daemon".
3. At the prompt specify that you want to stop Puppet. This stops any Puppet daemons that are currently running in your cluster. (Puppet is the underlying technology that HyperStore uses for cluster Configuration Management.)
4. After the installer indicates that Puppet has been stopped, return to the Advanced Configuration Options menu and select "Remove existing Puppet SSL certificates". This will remove existing Puppet SSL certificates, with no further prompts.
5. Return to the Advanced Configuration Options menu and select "Change server role assignments". This displays the Change Server Role Assignments menu.




6. From the Change Server Role Assignments menu select "Installer/Config Manager backup node".
7. At the prompt specify the host to which you want to move the Configuration Master (config manager) backup role. You need a new backup because you are converting the original backup into the primary (by running through this procedure using the installer on the original backup -- this has the effect of turning it into the new primary).
8. After completing the interaction for specifying the Configuration Master backup host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
9. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
10. Go to "Cluster Management" → "Manage Services" and restart the CMC.
11. Optionally, if you want to **leave the Puppet daemons running**, from the installer's main menu select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Start or stop Puppet daemon", and choose to start the daemons. After the daemons have successfully started you can exit the installer.

To verify your change, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that your Configuration Master primary and backup hosts are what you want them to be. The former backup (from which you implemented the above procedure) should now be the primary and the new backup should be as you specified during the procedure.

**Note** If the Configuration Master primary is now on the same host as the cron job primary role you should **move the cron job primary role** to a different host. If the cron job primary and the running Configuration Master are on the same host and that host goes down, automated fail-over from the cron job primary to the cron job backup will not work.

## 5.12.8. Move the Credentials DB Master Role or QoS DB Master Role

The **Credentials DB Master** role is assigned to just one node in your entire HyperStore system (the system has just one Credentials DB Master even if there are multiple service regions). The **QoS DB Master** role is assigned to one node in each of your service regions (each region has its own QoS DB Master). The HyperStore installer automatically makes these role assignments when you install your system.

If you wish you can move the Credentials DB Master role to a current Credentials DB slave node, or move a QoS Master DB role to a current QoS DB slave node, by following the procedure in this section. If you do so, the node that had been Master will automatically become a slave.

> **Note** The system does not support moving the Master role to a node that is not currently a slave.

### 5.12.8.1. Preparing to Move the Master Role

For safety, before moving the Credentials DB Master role or QoS DB Master role make a backup copy of the current database from the Master. You can do so by using the Redis CLI command **BGSAVE** as described below.

To back up a **Credentials DB Master** database:

```
# /opt/redis/redis-cli -h <hostname_of_master_node> -p 6379 BGSAVE
```

The above command will update the *dump-credentials.rdb* file in the Redis data directory (*/var/lib/redis* by default).

To back up a **QoS DB Master** database:

```
# /opt/redis/redis-cli -h <hostname_of_master_node> -p 6380 BGSAVE
```

The above command will update the *dump-qos.rdb* file in the Redis data directory (*/var/lib/redis* by default).

> **Note** The BGSAVE operation saves the database in the background. To check to see whether the save is completed yet you can use the Redis CLI **LASTSAVE** command, which returns the Unix time of the most recently completed save. Do not move the Master role until the saving of the database backup is completed.

### 5.12.8.2. Moving the Credentials DB Master Role or QoS DB Master Role

1. Submit a Redis CLI *info* command to the **slave node** to which you want to move the Master role. Confirm that the slave node returns the following status information:
   - *role* is slave
   - *master_host* is the current Master
   - *master_link_status* is up
   - *master_sync_in_progress* is 0

   When submitting the Redis CLI command use port 6379 if the target node is a Credentials DB slave, or use port 6380 if the target node is a QoS DB slave.

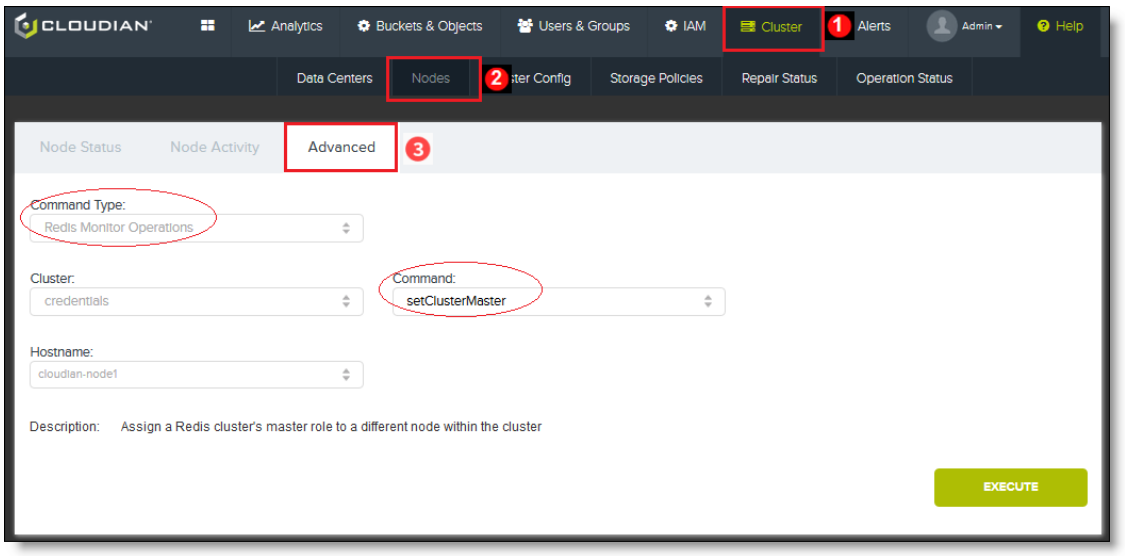
For example, if the Credentials DB Master role is currently on host "cloudian-node2" and you want to move it to host "cloudian-node7" where there is currently a Credentials DB slave:

```
# /opt/redis/redis-cli -h cloudian-node7 -p 6379 info | egrep
"role|master_host|master_link_status|master_sync_in_progress"

role:slave
master_host:cloudian-node2
master_link_status:up
master_sync_in_progress:0
```

2. Dynamically switch the Master role to the slave:

   Log into the CMC and go to **Cluster** → **Nodes** → **Advanced**. Select Command Type "Redis Monitor Operations", then select a Cluster type (Credentials or QoS) and select Command "setClusterMaster".




   For "Hostname" select the host to which you want to move the Master role. The drop-down list will show only nodes that are currently slaves within the cluster type (Credentials or QoS) that you selected.

   After making your selections, click **Execute**. The chosen slave will become the Master, while the Master becomes a slave. The change happens immediately upon command execution.

3. Make the switch of the Master and slave permanent by updating your system configuration:

   a. On the Configuration Master node (not the Credentials DB Master or QoS DB Master but rather the Configuration Master from which cluster configuration is managed) change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

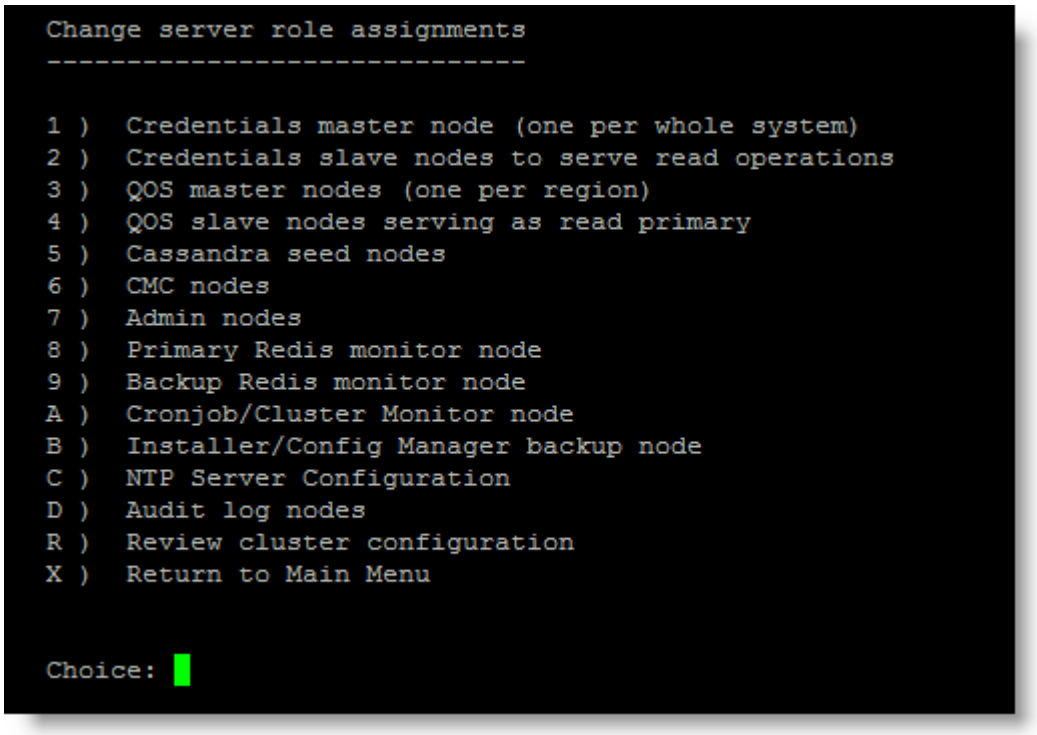
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

b. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




c. From the Change Server Role Assignments menu select the option for the master that you want to move — Credentials DB Master or QoS DB Master.

d. At the prompt indicate the host to which you want to move the Master role. This should be the same host that you checked in Step 1.

e. After completing the interaction for specifying the new Master host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.

f. Return to the installer's main menu, then choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.

g. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the following services, one service at a time:

- The DB service for which you've changed the master role (either Redis Credentials or Redis QoS)
- HyperStore Service
- S3 Service
- Redis Monitor
- IAM Service

After these services have successfully restarted you can exit the installer.

To verify your change, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that the Master that you moved is now where you want it to be.

Now, the former Credentials DB or QoS DB slave has been promoted to Master and the former Master has been demoted to slave.

**IMPORTANT !** **If you want to remove the demoted host** (the former Master that's now a slave) from your cluster, you must first move its slave role to a different host in your cluster. For instructions see **"Move or Add a Credentials DB Slave or QoS DB Slave"** (page 306).

## 5.12.9. Move the Redis Monitor Primary or Backup Role

The **Redis Monitor** runs on two nodes in your cluster, with one being the primary Redis Monitor instance and the other being a backup instance. In the event that the Redis Monitor primary goes offline the Redis Monitor backup **automatically** detects this and takes over as the active Redis Monitor.

**IMPORTANT !** In a multi- data center HyperStore system, the Redis Monitor backup must remain in the same data center as the Redis Monitor primary, and this must be the same data center as where the Credentials DB Master is located.

The system supports moving the primary or backup Redis Monitor to a different host as described below.

1. On the Configuration Master node, change into the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the HyperStore installer.

   ```
   # ./cloudianInstall.sh
   ```

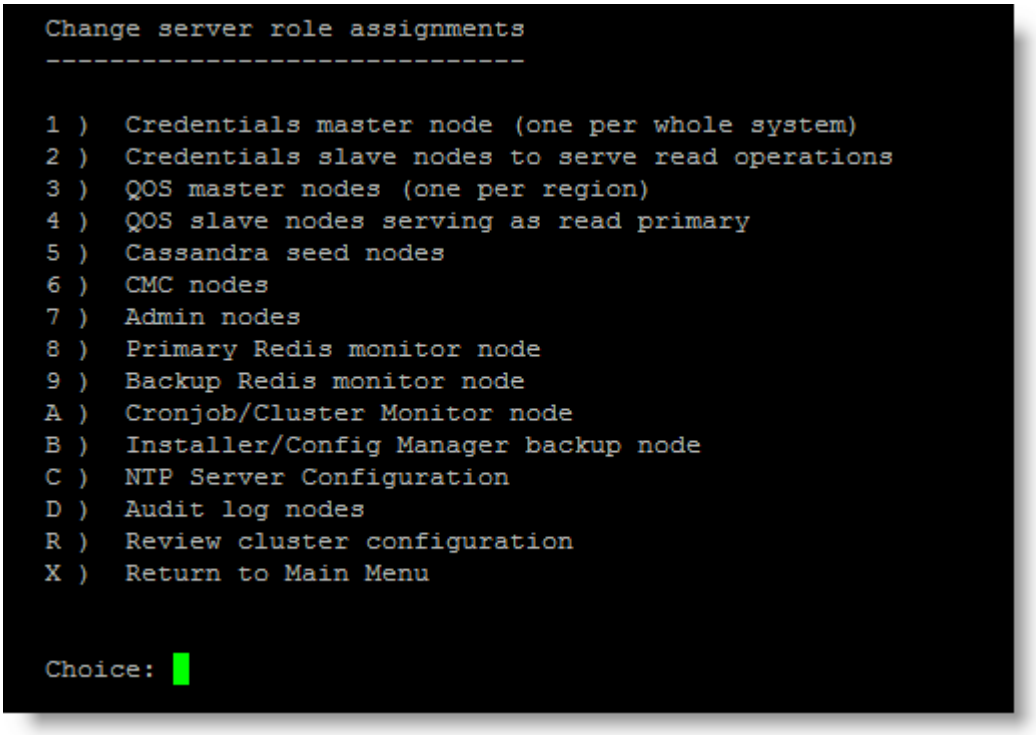
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.




3. From the Change Server Role Assignments menu select the option for the Redis Monitor instance that you want to move (Primary Redis Monitor or Backup Redis Monitor).
4. At the prompt specify the host to which you want to move the Redis Monitor instance.
5. After completing the interaction for specifying the new Redis Monitor location, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Configuration Master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
7. From the "Cluster Management" menu choose "Manage Services" and restart the Redis Monitor service. After the Redis Monitor successfully restarts you can exit the installer.

To verify your change, log into the CMC and go to the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). Review the service information section to confirm that your Redis Monitor primary and backup hosts are what you want them to be.

# 5.13. Cron Jobs and Automated System Maintenance

The HyperStore system automatically executes a variety of maintenance tasks. This section describes the automated maintenance jobs that the system implements. The covered topics are:

- **"System cron Jobs"** (page 321)
- **"Cassandra Data Compaction"** (page 328)

## 5.13.1. System cron Jobs

Most HyperStore automated maintenance routines are implemented as cron jobs. Maintenance cron jobs are run from one HyperStore node in each of your service regions. To see which of your nodes is running the cron

jobs, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). In the Service Information section you will see the identity of the System Monitoring / Cronjob Primary Host, from which the cron jobs are run.

> **Note** You will also see the identity of the System Monitoring / Cronjob Backup Host. If the primary cron job instance goes down (due to the host going down or *crond* going down on the host) and remains down for 10 minutes, the backup cron job instance will automatically take over the primary role and start running the system cron jobs.

The cron jobs themselves are configured in the */etc/cron.d/cloudian-crontab* file on the host node. **If you want to adjust the scheduling of these cron jobs** you should do so via Puppet configuration management, by editing the configuration template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/cloudian-crontab.erb* on the Configuration Master node (and then **push your change to the cluster and restart the S3 Service**). For general information on how to configure cron job scheduling, refer to any reputable resource on the topic.

Note that most of the cron jobs configured in *cloudian-crontab.erb* have "> /dev/null 2>&1" appended to them and therefore they direct all output to */dev/null*.

For information on the Admin API calls referenced in this section, see the ***Cloudian HyperStore Admin API Reference***.

System cron jobs are implemented for the following system tasks:

## System Monitoring Data Collection

Scope: One job per region

Frequency: Every 1 minute

Operation invoked: S3 Service internal process */bin/snapshotData*

This cron job invokes a process that logs "snapshot" system statistics each minute in support of the Cloudian Management Console's system monitoring functionality.

## Diagnostic Logs Upload

### Smart Support Diagnostics Upload

Scope: One job per region

Frequency: Once per day

Operation invoked: S3 Service internal process */bin/phoneHome*

This cron job uploads a daily system diagnostics file to a configurable S3 URI, if the HyperStore "Smart Support" feature (also known as "Phone Home") is enabled. By default Smart Support is enabled and the S3 URI is the S3 URI of Cloudian Support. For more information see **"Smart Support and Diagnostics Feature Overview"** (page 108).

> **Note** The upload will occur within an hour of the time specified in the crontab. A random wait time is built into the upload process so that not all Cloudian customer environments are uploading to Cloudian Support at the same time.

### Node Diagnostics Logs Upload for Support Case Processing

Scope: One job per region

Frequency: Once every 10 minutes

Operation invoked: S3 Service internal process */bin/onDemandLogging*

If you have enabled automatic log collection for support case processing, this cron job will upload to Cloudian Support any node diagnostics logs that Cloudian Support needs to process support cases that you open. By default configuration this feature is disabled. For more information see *common.csv:* **"logcollect_enable"** (page 402).

## Usage Data Processing

Several cron jobs are involved in processing service usage data for users and groups.

> **Note** For an overview of how the HyperStore system tracks service usage by groups and users, see **"Usage Reporting Feature Overview"** (page 218).

### Saving Storage Usage Data for Active Users

Scope: One job per region

Frequency: Every 5 minutes

Operation invoked: Admin API method *POST /usage/storage*

This cron job writes snapshots of per-user and per-group counts for stored bytes and number of stored objects from the Redis QoS database over to the "Raw" column family in the Metadata DB's "Reports" keyspace. The operation is applied only to users and groups that have uploaded or deleted objects in the time since the operation was last executed.

### Saving Storage Usage Data for All Users

Scope: One job per region

Frequency: Once a day

Operation invoked: Admin API method *POST /usage/storageall*

This cron job writes snapshots of per-user and per-group counts for stored bytes and number of stored objects from the Redis QoS database over to the "Raw" column family in the Metadata DB's "Reports" keyspace. The operation is applied to **all** users and groups.

### Repairing Usage Data for Active Users

Scope: One job per region

Frequency: Every 12 hours

Operation invoked: Admin API method *POST /usage/repair/dirtyusers*

This cron job repairs Redis QoS stored bytes and stored object counts for up to 1000 active or "dirty" users. See the Admin API method description for more detail.

### Creating Usage Roll-Up Reports

Scope: One job per region for each roll-up granularity (hour, day, month)

Frequency: Hourly, daily, monthly

Operation invoked: Admin API method *POST /usage/rollup*

- One job with granularity=hour (runs once per hour)
- One job with granularity=day (runs once per day)
- One job with granularity=month (runs once per month)

These three cron jobs create summary (or "roll-up") usage reports data from more granular reports data. The hourly roll-up data is derived from the raw data (the transactional data and stored bytes/stored object count snapshot data). The daily roll-up data and monthly roll-up data are both derived from the hourly roll-up data.

**Note** Hourly rollup jobs that fail -- such as if a relevant service is down or unreachable -- will be retried. The time span for which failed rollup jobs are eligible for retry is configuration by the **"usage.rollup.hour.maxretry"** (page 458) property in **mts.properties.erb**. The default is 6 hours.

## Bucket Log Processing

Scope: One job per region

Frequency: Once every 10 minutes

Operation invoked: S3 Service internal process */.system/dumpbucketlogs*

This cron job moves bucket logging data from the Metadata DB's BucketLog column family into the S3 storage system, in support of the S3 Bucket Logging feature.

## Bucket Lifecyle Processing

Two cron jobs are involved in S3 Bucket Lifecycle implementation.

### Auto-Tiering and Auto-Expiring Objects

Scope: One job per region

Frequency: Once a day

Operation invoked: S3 Service internal process */.system/autoexpire*

This cron job performs two tasks in support of S3 bucket lifecycle policies:

- Transitions (auto-tiers) objects to a tiering destination system, if the objects have reached their scheduled auto-tiering interval or date.
- Deletes objects from HyperStore storage (or from the remote tiered storage system if they've been auto-tiered), if the objects have reached their scheduled expiration interval or date.

The cron job distributes auto-tiering and auto-expiry processing workload across the cluster. Specifically, the auto-tiering and auto-expiry processing workload associated a given storage policy is spread out across all the nodes that participate in that storage policy. For example, suppose you have a HyperStore system with two data centers named DC1 and DC2, and one of your storage policies stores data only in DC1. When the */.system/autoexpire* cron job runs, the auto-tiering and auto-expiry processing work for buckets that use that storage policy will be allocated among the nodes in DC1, so that each node in DC1 is doing the processing for a

subset of those buckets. If you have a storage policy that stores data in both DC1 and DC2, then the auto-tiering and auto-expiry processing workload associated with buckets that use that storage policy will be spread among the nodes in both data centers.

**Note** For more information on the HyperStore auto-tiering feature, see **"Auto-Tiering Feature Overview"** (page 169).

### Restoring Auto-Tiered Objects

Scope: One job per region

Frequency: Every six hours

Operation invoked: S3 Service internal process */.system/restore*

This cron job executes queued object-restore jobs. Restore jobs are placed in queue when S3 clients invoke the S3 API method *RestoreObject*, to restore a local copy of objects that have been auto-tiered to a remote storage system. The cron job executes queued restored jobs every six hours.

## Bucket Inventory Processing

Scope: One job per region

Frequency: Once a day

Operation invoked: S3 Service internal process */.system/processinventory*

This cron job generates bucket content inventories, for any buckets that have bucket inventories configured. For the relevant S3 API see "PutBucketInventoryConfiguration" in the ***Cloudian HyperStore AWS APIs Support Reference***. For CMC support of this feature, while on the CMC's **Bucket Properties** page click **Help**.

## Tombstone Cleanup Processing

Scope: One job per region

Frequency: Every hour

Operation invoked: S3 Service internal process */.system/cleantombstones*

This hourly operation cleans (removes) Cassandra "tombstones", which are markers that indicate that a table cell or row has been deleted.

### Dealing with Excessive Tombstone Build-Up

Under normal circumstances the hourly running of the */.system/cleantombstones* cron job should ensure that there is no excessive build-up of tombstones in Cassandra (the Metadata DB). However, it is possible to encounter TombstoneOverwhelmingException errors in Cassandra logs and an inability to successfully execute an S3 *ListObjects* operation against a specific bucket, in unusual circumstances.

You can trigger tombstone removal by connecting to the S3 Service's JMX port (19080) on any HyperStore node and submitting a *purgeTombstone* command with the bucket name as input. If you are using JConsole, after connecting to port 19080 select the *MBeans* tab, then select *com.cloudian.ss.cassandra.cl* → *BatchJobs* → *Operations* → *purgeTombstone*. On the right side of the console enter the bucket name as the *p1* value and then execute the operation (by clicking *purgeTombstone* on the right side of the console).

Here is the *purgeTombstone* operation triggered by using the *jmxclient* tool (which exists on each HyperStore node). Replace *<version>* with the *jmxclient* version (you can check this under */opt/cloudian/tools*), *<IP address>* with the node's IP address, and *<bucketname>* with the target bucket name.

```
java -jar /opt/cloudian/tools/cmdline-jmxclient-<version>.jar -:- <IP address>:19080 \
com.cloudian.ss.cassandra.cl:type=BatchJobs \
purgeTombstone=<bucketname>
```

## User Deletion Cleanup

Scope: One job per region

Frequency: Once a day

Operation invoked: Admin API method *POST /system/deletedUserCleanup* (for system use only; this Admin API method is not covered in the Admin API documentation)

This cron job attempts to complete the user deletion process for users for whom the deletion process failed to complete on the original attempt. These are users who are stuck in "Deleting" status for one reason or another.

## Storage Policy Deletion or Creation Processing

Scope: One job per region

Frequency: Once a day

Operation invoked: Admin API method **POST /system/processProtectionPolicy**

This cron job processes any pending storage policy delete jobs. System operators can initiate the deletion of an unused storage policy (a storage policy that is not assigned to any buckets) through the CMC's **Storage Policies** page. This operator action marks the policy with a "DELETED" flag and makes it immediately unavailable to service users. However, the full deletion of the storage policy from the system (specifically, the deletion of the Metadata DB keyspace associated with the policy) is not processed until this cron job runs.

This cron job also processes any pending storage policy creation jobs, in the event that multiple storage policy creation requests have been initiated in a short amount of time -- which can result in queueing of storage policy creation jobs. More typically, storage policy creation completes shortly after the creation is initiated through the CMC.

## Retries of Pending Cross-Region Replication Tasks

Scope: One job per region

Frequency: Every four hours

Operation invoked: S3 Service internal process */.system/processcrr*

This cron job processes any pending **cross-region replication** tasks. Typically with cross-region replication, objects are replicated to the destination bucket as soon as they are uploaded to the source bucket. Pending cross-region replication tasks result when an attempt to replicate an object from the source bucket to the destination bucket returns a temporary error such as a connection failure or an HTTP 5xx error. This cron job retries the replication of such objects. For such objects, the retries will recur once every four hours until either the objects are successfully replicated to the destination system or a permanent error is encountered.

**Note** If you set *mts.properties.erb: cloudian.s3.crr.syncbucket* to *true*, then for each source bucket for which there are pending replication retry tasks this cron job will also perform a complete sync-up of the destination bucket contents to the source bucket contents. By default *cloudian.s3.crr.syncbucket* is set to *false*. For more information see **"cloudian.s3.crr.syncbucket"** (page 454).

## Processing of Pending Metadata Search Update Requests

Scope: One job per region

Frequency: Every hour

Operation invoked: S3 Service internal process */.system/processelasticsearch*

If you are using the **HyperStore Search Service**, this hourly cron job triggers the following actions in the HyperStore Search Service:

- Create new search indices for buckets that were created within the past hour and that use a **search-enabled storage policy**.
- Update search indices for all buckets that use a search-enabled storage policy, to reflect changes to bucket content during the past hour -- such as object uploads or object deletions.
- Delete search indices for buckets that were deleted during the past hour and that had used a search-enabled storage policy.

Until the cron job runs, these search index update tasks are queued in the Metadata DB. Any tasks that fail when this cron job runs are retried at the next running of the cron job.

**Note** If your HyperStore Search Service cluster is unavailable for more than a few hours, the search index update task queue may become too large to be completely processed by the hourly cron job. When the HyperStore Search Service cluster comes back online after being down for more than a few hours, use the *search-mgr.sh* tool to trigger an on-demand update of the search indexes in the HyperStore Search Service cluster. For more information about this tool see **"Using the search-mgr.sh Tool For On-Demand Indexing"** (page 205).

## Retries of Pending S3 Bucket Notifications

Scope: One job per region

Frequency: Every hour

Operation invoked: S3 Service internal process */.system/processNotification*

This cron job retries S3 Bucket Notification messages that failed to be successfully sent on previous attempts. For more information about the S3 Notification feature see the SQS section in the ***Cloudian HyperStore AWS APIs Support Reference***.

## Rekeying KMIP-Enabled Buckets

Scope: One job per region

Frequency: Once per month

Operation invoked: S3 Service internal process */.system/rekey*

By default this cron job is disabled (commented out). For information about enabling the cron job, see **"Re-Keying a Bucket"** (page 159).

## 5.13.2. Cassandra Data Compaction

The Metadata DB is built on Cassandra. Cassandra stores data to disk in the form of "Sorted String Tables" (SSTables), a type of append-only commit log. During Cassandra operations, many SSTables may be written to disk for each column family. It's important that SSTables be compacted regularly to minimize the amount of disk space that they use. Compaction merges multiple SSTables into one.

Cassandra regularly implements a "minor" compaction process that occurs **automatically**. This automatic compaction process is sufficient in the context of the HyperStore system. For the HyperStore system, **you do not need perform "major" compactions** using the *nodetool* utility.

You can monitor the compaction process by using JConsole or another JMX client to connect to a Cassandra node's JMX listening port (7199 by default). By accessing the *CompactionManagerMBean* through the JMX console, you can check the progress of any compactions that are currently executing, and also check on the number of completed and pending compactions.

# Chapter 6. Disk Operations

## 6.1. Disabling a HyperStore Data Disk

*Subjects covered in this section:*



HyperStore supports a method for **temporarily disabling a HyperStore data disk drive** so that you can perform planned maintenance such as a firmware upgrade.

Note that you **cannot disable a disk that is the sole available data disk on the node** -- that is, you cannot disable a data disk if all the other data disks on the node are already disabled or are in **stop-write condition**.

> **Note** If you are **replacing** a disk, follow the instructions for **"Replacing a HyperStore Data Disk"** (page 332) rather than the instructions below.

### 6.1.1. The Impact of Disabling a Disk

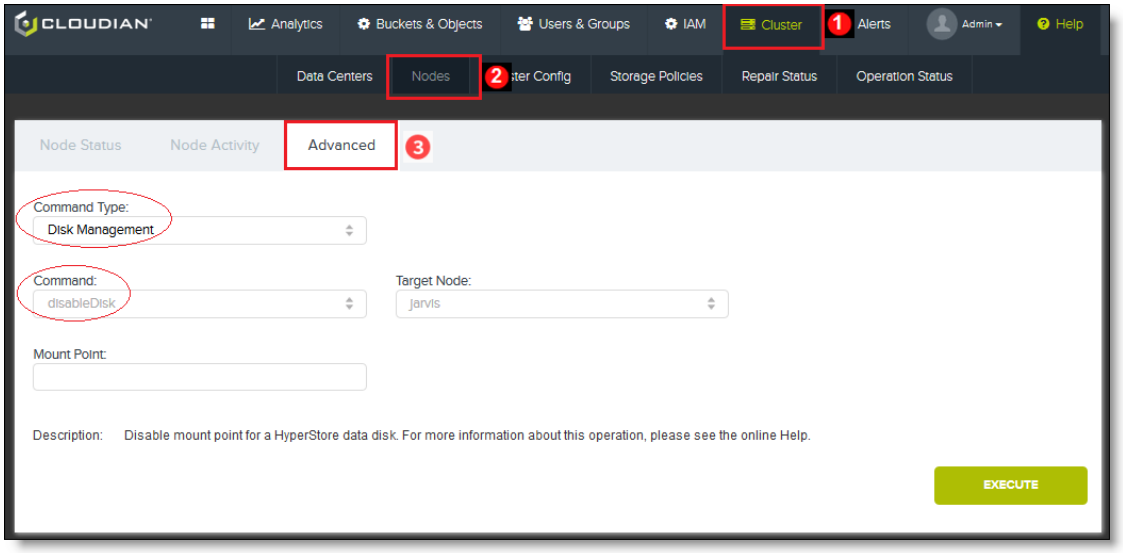
When you execute the HyperStore *disableDisk* function, the system automatically does the following:

- Unmounts the disk's file system, comments out its entry in */etc/fstab*, and marks the disk as unavailable for HyperStore reads and writes.
- Moves the disk's assigned storage tokens to the remaining HyperStore data disks on the same host, in a way that's approximately balanced across those disks. This is a temporary migration of tokens which will be reversed when you later re-enable or replace the disk. While the tokens are on the other disks, writes of new or updated S3 object data that would have gone to the disabled disk will go to the other disks on the host instead.

The existing object data on the disabled disk is **not** recreated on the other disks and therefore that data will be unreadable on the host. Whether the system as a whole can still provide S3 clients with read access to the affected S3 objects depends on the availability of other replicas or erasure coded fragments for those objects, elsewhere within the cluster.

### 6.1.2. Disabling a Disk

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "disableDisk" command.




2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the disk that you want to disable (for example */cloudian6*).
3. Click **Execute**.

After the *disableDisk* operation completes, go to the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**). In the "Disk Detail Info" section, the device that you disabled should now have this red status icon (indicating that it's disabled and its tokens have been migrated to other disks on the same host):

Later, after completing your maintenance work on the disk, follow the instructions for **"Enabling a HyperStore Data Disk"** (page 330). When you re-enable the disk, the tokens that had been moved away from the disk will be moved back to it.

## 6.2. Enabling a HyperStore Data Disk

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"The Impact of Enabling a Disk"** (page 331)*
- ***"Enabling a Disabled Disk"** (page 331)*

HyperStore supports a method for **enabling an existing HyperStore data disk that is currently disabled**. You can tell that a disk is disabled by viewing its status in the "Disk Detail Info" section of the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**). A disk can go into a disabled state either because you disabled it by using the HyperStore *disableDisk* function (as described in **"Disabling a HyperStore Data Disk"** (page 329)) or because the HyperStore system automatically disabled it in response to disk errors (as described in **"Automated Disk Usage Management Feature Overview"** (page 113)).

You can enable a disk if you know that the disk problem was only temporary and that the disk is still healthy enough to use.

**Note** If you are **replacing** a disk, follow the instructions for **"Replacing a HyperStore Data Disk"** (page 332) rather than the instructions below.

## 6.2.1. The Impact of Enabling a Disk

When you execute the HyperStore *enableDisk* function, the system automatically does the following:

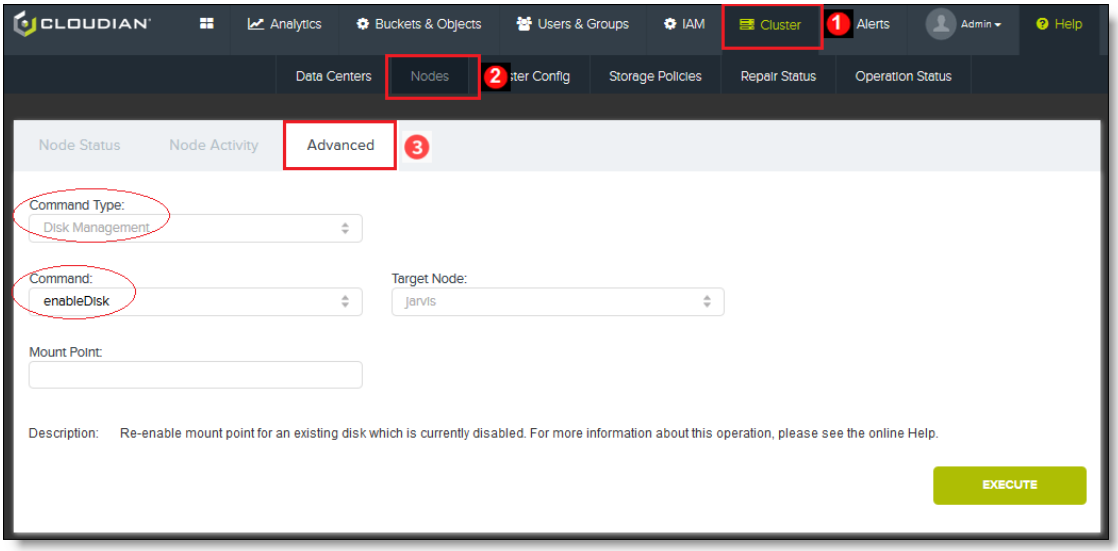
- Remounts the disk (using the same mount point that the disk previously had), uncomments its entry in */etc/fstab*, and marks the disk as available for HyperStore reads and writes.
- Moves back to the disk the same set of storage tokens that were automatically moved away from the disk when it was disabled.

After a disk is re-enabled in this way, writes of new or updated S3 object data associated with the returned tokens will go to the re-enabled disk. And the existing object data that was already on the disk will once again be readable. Meanwhile object data that was written to the affected token ranges while the disk was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks. That data will not be moved to the re-enabled disk.

**Note** For information on how HyperStore tracks token location over time so that objects can be written to and read from the correct disks, see **"Dynamic Object Routing"** (page 116).

## 6.2.2. Enabling a Disabled Disk

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "enableDisk" command.




2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the disk that you want to enable.
3. Click **Execute**.

After the *enableDisk* operation completes, go to the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**). In the "Disk Detail Info" section, the device that you enabled should now have this green status icon (indicating that its status is OK):

If instead the disk icon is displaying in red (indicating an "Error" status), click the **Clear Error History** button. Doing so should return the disk to OK status.

**Note** If the CMC continues to show status information for the disk's old device address (as well as a new device address), and if clicking the **Clear Error History** button fails to clear the old information, *ssh* into the node on which you enabled the disk and run the command *systemctl restart cloudian-agent*. Then wait at least one minute, and check the CMC again. If the old information is still displaying, click the **Clear Error History** button again.

# 6.3. Replacing a HyperStore Data Disk

*Subjects covered in this section:*



HyperStore supports a method for **activating a replacement HyperStore data disk and restoring data to it**. This procedure applies to either of these circumstances:

- You are replacing a disk that is currently disabled. You can tell that a disk is disabled by viewing its status in the "Disk Detail Info" section of the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**). A disk can go into a disabled state either because you disabled it by using the HyperStore *disableDisk* function (as described in **"Disabling a HyperStore Data Disk"** (page 329)) or because the HyperStore system automatically disabled it in response to disk errors (as described in **"Automated Disk Usage Management Feature Overview"** (page 113)).
- You are replacing a disk that is not currently disabled. In this case, it is not necessary for you to use the *disableDisk* function before replacing the disk. When you pull the disk from the host machine HyperStore will automatically disable the associated mount point, and you can proceed to replace the disk.

**After you've pulled the bad disk and physically installed the replacement disk**, HyperStore will take care of all the remaining set-up and restoration tasks when you follow the steps in **"Replacing a HyperStore Data Disk"** (page 333) below.

**You can replace disks on multiple nodes concurrently, if there is only erasure coded data in your system.** Replacing multiple disks concurrently is **not** supported in either of these circumstances:

- You have replication storage policies in your system (storage policies that use object replication rather than erasure coding). Note that in a multi-DC environment, "Replicated EC" is considered an erasure coding storage policy not a replication storage policy.
- You are using the new "hybrid" storage policies introduced in HyperStore 7.5.1 (which use replication for small objects and erasure coding for large objects).

Also, you cannot replace multiple disks on the same node concurrently. You can only replace multiple disks concurrently if the disks are on different nodes.

### 6.3.1. The Impact of Replacing a HyperStore Data Disk

When you physically install a new disk and then execute the HyperStore *replaceDisk* function, the system automatically does the following:

- Creates a primary partition and an *ext4* file system on the new disk.
- Establishes appropriate permissions on the mount.
- Remounts the new disk (using the same mount point that the prior disk had), uncomments its entry in */etc/fstab*, and marks the disk as available for HyperStore reads and writes.
- Moves back to the new disk the same set of storage tokens that were automatically moved away from the prior disk when it was disabled.
- Performs a data repair for the new disk (populating the new disk with its correct inventory of object replicas and/or erasure coded object fragments).

Going forward, writes of new or updated S3 object data associated with the returned tokens will go to the new disk. Meanwhile object data that was written to the affected token ranges while the mount point was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks. That data will not be moved to the new disk.

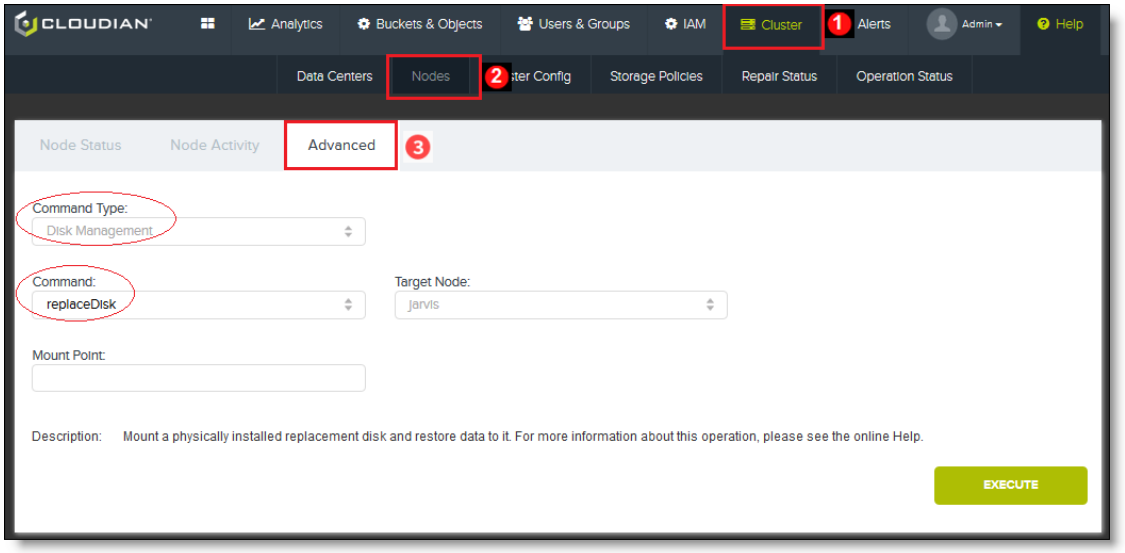
For information on how HyperStore tracks token location over time so that objects can be written to and read from the correct disks, see **"Dynamic Object Routing"** (page 116).

### 6.3.2. Replacing a HyperStore Data Disk

**Note** HyperStore supports replacing a bad disk on an existing node while you are in the midst of expanding your cluster. Wait until the **"Add Node" operation** completes successfully, and then you can follow the steps below either before you trigger a rebalance operation on the new node(s) or while the rebalance operation(s) are in progress. You do not need to wait until the rebalance operation(s) complete. However, see **"Extra Step After Disk Replacements Performed During Cluster Expansion"** (page 334).

After you've physically installed the replacement disk, follow these steps to activate the replacement disk and restore data to it:

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "replaceDisk" command.




2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the replacement disk. This must be the same as the mount point of the disk that you replaced.
3. Click **Execute**.

After the *replaceDisk* operation completes, go to the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**). In the "Disk Detail Info" section, the replacement disk should now have this green status icon (indicating that its status is OK):

The system then automatically runs *repair* and *repairec* on the disk mount point. You can monitor the repair progress in the **Operation Status** page (**Cluster -> Operation Status**).

Note that:

- If in the Node Status page the disk icon is displaying in red (indicating an "Error" status), click the **Clear Error History** button. Doing so should return the disk to OK status.
- If the CMC continues to show status information for the old disk (as well as the new disk), and if clicking the **Clear Error History** button fails to clear the old information, *ssh* into the node on which you replaced the disk and run the command *systemctl restart cloudian-agent*. Then wait at least one minute, and check the CMC again. If the old information is still displaying, click the **Clear Error History** button again.

### 6.3.2.1. Extra Step After Disk Replacements Performed During Cluster Expansion

If you executed the *replaceDisk* during the rebalance operations associated with a cluster expansion (or after the Add Node[s] operation completed and before you kicked off the rebalance operations), then after the *replaceDisk* operation and **all rebalance operations have completed**, run **hsstool repairec** on any one of the new nodes. This will replace any EC data that is supposed to be on the new nodes but that may be missing because of the disk having gone bad on an existing node during the cluster expansion.

If you added new nodes to multiple data centers, run **hsstool repairec** on one of the new nodes in each data center.

## 6.4. Replacing a Cassandra Disk

If you need to replace a disk that stores Cassandra (the Metadata DB) and the OS, please consult with Cloudian Support.

## 6.5. Responding to Data Disks Nearing Capacity

HyperStore implements an **automatic mechanism for helping to ensure balanced disk usage** among the disks on a host. However, if service utilization is heavy for the size of your cluster, there may be times when one or more disks nears their capacity.

> **Note** For guidance about HyperStore capacity management and cluster resizing, see **"Capacity Monitoring and Expansion"** (page 338).

If an individual disk or a node as a whole is running low on available capacity, HyperStore alerts administrators:

- Alerts are triggered if an individual disk drops below 15% available capacity or if a node as a whole drops below 10% available capacity. When such alerts are triggered, they appear in the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**) and **Alerts** page (**Alerts -> Alerts**) as well as being sent to the system administrator email address(es). Optionally, alerts can also be transmitted as SNMP traps. Alert thresholds and options are configurable in the **Alerts Rules** page (**Alerts -> Alert Rules**).
- If a node as a whole reaches 80% utilization of its data disk capacity, a Warning is display on the CMC's **Dashboard** page.

If a **HyperStore data disk** (a disk storing S3 object data) is nearing capacity, the first two things to try are:

- Use **hsstool cleanup** (and **hsstool cleanupec** if you use erasure coding in your system) on the node to clear it of any data that's no longer supposed to be there. Such "garbage data" may be present if, for example, S3 objects have been deleted from the system as a whole but the deletion operations on the node in question failed.
- Delete S3 objects. Note that the associated files will not be deleted from the disk immediately since HyperStore uses batch processing for purging of S3 object data. The batch purging processing occurs hourly on each node.

  > **Note** For additional guidance on removing data to free up disk space, consult with Cloudian Support.

If these measures do not free up sufficient disk space, the solution is to increase system capacity by adding one or more new nodes to your cluster. For the procedure see **"Adding Nodes"** (page 264). When you add a node, a portion of the data on your existing nodes is copied to the new node and then (when you subsequently run a cleanup operation) deleted from the existing nodes — thereby freeing up space on the existing nodes. The degree to which space will be freed up on existing nodes depends on the number of new nodes that you add in proportion to the size of your existing cluster — for example, adding two nodes to a four node cluster would free up a larger percentage of the existing nodes' disk space than would adding two nodes to a twenty node cluster.

For information about managing a **Cassandra data disk** (a disk storing system and object metadata stored in the Metadata DB) that is nearing capacity see **"Responding to Cassandra Disks Nearing Capacity"** (page 336).

## 6.6. Responding to Cassandra Disks Nearing Capacity

If a Cassandra (Metadata DB) data drive is nearing capacity, the first two things to try are:

- Use **hsstool cleanup** on the host node, using the *allkeyspaces* option. This will clear any Cassandra data that is no longer supposed to be on the node.
- Selectively delete Cassandra data. To do this, consult with Cloudian Support.

If these measures do not free up sufficient space, the solution is to increase system capacity by adding one or more new nodes. For the procedure see **"Adding Nodes"** (page 264). When you add a node, a portion of the Cassandra data on your existing nodes is copied to the new node and then (when you subsequently run a cleanup operation) deleted from the existing nodes — thereby freeing up space on the existing nodes.

## 6.7. Adding Disks is Not Supported

**HyperStore does not support adding disks to an existing node within the cluster**. To increase cluster storage capacity, the only supported method is to add one or more nodes as described in **"Adding Nodes"** (page 264).

> **Note** For guidance about HyperStore capacity management and cluster resizing, see **"Capacity Monitoring and Expansion"** (page 338).

# Chapter 7. Monitoring

## 7.1. Cloudian HyperIQ

Cloudian HyperIQ is a solution for dynamic visualization and analysis of HyperStore system monitoring data and S3 service usage data. HyperIQ is a separate product available from Cloudian that deploys as virtual appliance on VMware or VirtualBox and integrates with your existing HyperStore system. For more information about HyperIQ contact your Cloudian representative.


## 7.2. Using the CMC to Monitor HyperStore

The Cloudian Management Console (CMC) provides extensive support for monitoring the health and performance of your HyperStore system. The CMC also supports a configurable mechanism for alerting administrators when system events occur.

The table below shows the system monitoring and alert management tasks that are supported by the CMC.

| Category | Tasks | CMC Page |
|---|---|---|
| System Monitoring | Check high level status information for each service region, including current storage capacity usage and remaining available capacity, project capacity usage for the next 120 days, recent S3 transaction performance, and whether there are any system alerts | **Dashboard** ( ) |

| Category | Tasks | CMC Page |
|---|---|---|
| | Check the remaining available storage capacity in each service region, as well as capacity remaining in each data center and on each node | **Analytics -> Capacity Explorer** |
| | Check how your storage capacity usage has changed over the past 30 days, per service region | **Analytics -> Cluster Usage** |
| | For each data center, see which nodes have any alerts and whether any HyperStore services are down on any node | **Cluster -> Data Centers** |
| | Check a specific node's storage capacity usage and remaining available capacity, CPU and memory usage, recent S3 transaction performance, and HyperStore services status. Also check capacity usage and status information for individual disks ona node. | **Cluster -> Nodes -> Node Status** |
| | Checka specific node's performance trends over the past 30 days, for metrics such as CPU utilization, disk read and write throughput, and S3 transactions per second. | **Cluster -> Nodes -> Node Activity** |
| System Alerts Management | Configure system alert rules, for having the system automatically notify administrators regarding system events. Also view pre-configured alert rules that come with the system. | **Alerts -> Alert Rules** |
| | Review and acknowledge alerts generated by any node in the system | **Alerts -> Alerts** |
| | Review and acknowledge alerts generated by a specific node | **Cluster -> Nodes -> Node Status** |

# 7.3. Capacity Monitoring and Expansion

*Subjects covered in this section:*

- *Introduction (immediately below)*
- ***"Monitoring Cluster Storage Capacity Utilization"** (page 339)*
- ***"Adding Nodes to Your System"** (page 341)*

**HyperStore is horizontally scalable**, allowing you to gain additional storage and request processing capacity by adding more nodes to your cluster. When you add new nodes to your cluster, the storage capacity associated with the new nodes becomes immediately available to the system. However, the automated processes that re-distribute data from existing nodes to newly added ones -- thereby reducing storage capacity usage on the existing nodes -- may take up to several days or more to complete, depending on factors such as data volume and network bandwidth. Therefore it's important to closely monitor your current and projected system capacity usage, plan ahead for needed cluster expansions, and implement such expansions well before you've filled your current capacity.

Use the CMC Dashboard to monitor your system's current and projected storage utilization level (for more information see **"Monitoring Cluster Storage Capacity Utilization"** (page 339), further below). As best practices for cluster expansion timing, Cloudian recommends the following:

- **Start expansion planning and preparation when either of the following occur (whichever occurs first)**:
  - The Dashboard shows your **utilization of system storage capacity has reached 70%**.
  - The Dashboard shows your **utilization of system storage capacity is projected to reach 90% within 120 days**. If your system has a high rate of ingest relative to capacity, this projection may occur even if your current usage has not yet reached 70%.

  When planning your expansion keep in mind that:

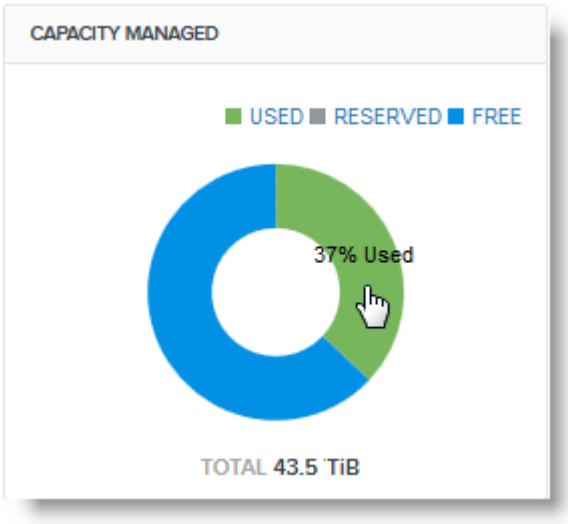
  - The minimum unit of expansion is a node. HyperStore does not support adding disks to existing nodes.
  - You need to allow time to acquire host machines and prepare them for being added to your cluster.
  - Preferably, cluster expansions should be substantial enough that the expanded cluster will allow you to meet your projected storage needs for at least an additional six months after the expansion. In this way you can avoid having to frequently add nodes to your system.
  - Cloudian Support is available to review and provide feedback on your expansion plan.
- **Execute your expansion when either of the following occur (whichever occurs first)**:
  - The Dashboard shows your **utilization of system storage capacity has reached 80%**.
  - The Dashboard displays a Warning that your **utilization of system storage capacity is projected to reach 90% within 90 days**. If your system has a high rate of ingest relative to capacity, this projection may occur even if your current usage has not yet reached 80%.

**IMPORTANT !** Each HyperStore node is designed to **reject new writes if it reaches 90% storage capacity utilization**. Allowing your system to surpass 80% capacity utilization poses the risk of having to rush into an urgent cluster expansion operation.

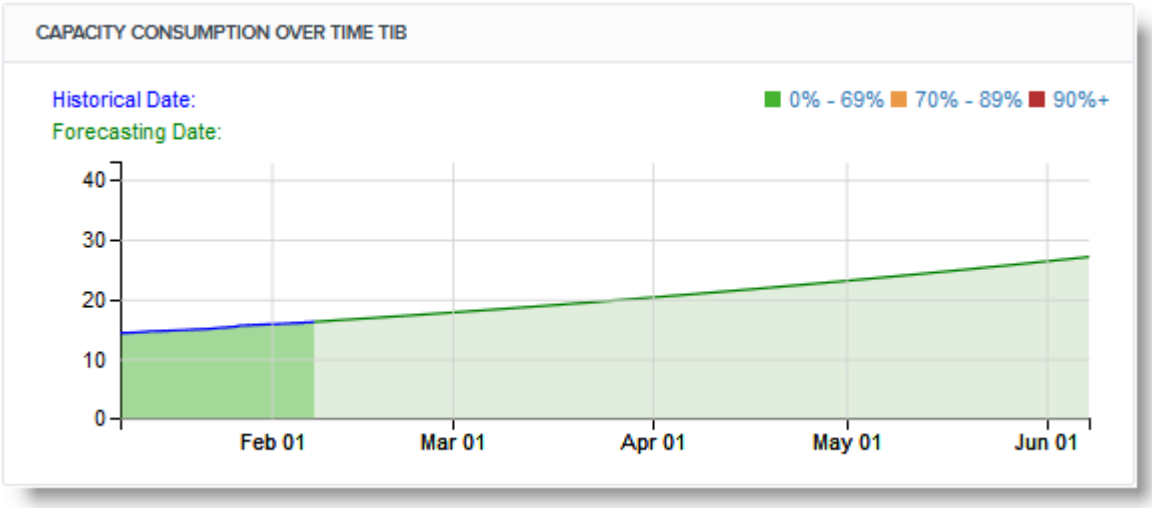
## 7.3.1. Monitoring Cluster Storage Capacity Utilization

HyperStore makes it easy to regularly monitor your system storage capacity utilization. In the CMC Dashboard you can view:

- Current storage capacity utilization.




- Projected storage capacity utilization over the next 120 days.




In both the current capacity utilization graphic and the projected utilization graphic, color-coding is used to highlight utilization levels of 70% or higher and 90% or higher.

The Dashboard also displays:

- A Warning message if the cluster is projected to reach 90% storage utilization in 90 days or less
- A Critical message if the cluster has reached 90% storage utilization

For more detail, while on the CMC's Dashboard ( ) click **Help**.

In the CMC's **Capacity Explorer** page (**Analytics -> Capacity Explorer**) you can view your system's remaining free storage capacity broken down by service region (cluster), data center, and node. If less than 30% storage space remains at any one of these levels -- that is, if more than 70% of capacity is utilized in a given node, data center, or region -- this is highlighted in the interface.

In the CMC's **Node Status** page (**Cluster -> Nodes -> Node Status**) you can view each node's storage capacity utilization as well as the utilization level for each disk on each node.

### 7.3.2. Adding Nodes to Your System

To add nodes to an existing HyperStore data center, follow the documented procedure **"Adding Nodes"** (page 264).

To add a new data center (with new nodes) to an existing HyperStore service region, follow the documented procedure **"Adding a Data Center"** (page 276).

To add a new service region (with new nodes) to an existing HyperStore system, follow the documented procedure **"Adding a Region"** (page 283).

**IMPORTANT !** With the addition of a new DC or service region you will need to create new storage policies that utilize the new DC or region. **Adding a new DC or region does not create additional storage capacity for existing buckets that use existing storage policies**. Only new buckets that utilize the new storage policies will make use of the additional storage capacity created by adding a new DC or region. In the current HyperStore version, you cannot revise an existing storage policy or reassign a new storage policy to an existing bucket.

To create additional storage capacity for your existing storage policies you must add nodes to your existing data center(s).

## 7.4. Using the Admin API to Monitor HyperStore

The HyperStore Admin API supports methods for monitoring HyperStore health and performance. The CMC invokes these Admin API methods in implementing the CMC's system monitoring functions. If you wish you can invoke these Admin API methods directly, through a client application of your own making or through third party command line tools that enable you to construct HTTP requests.

For more information see "monitor" section of the ***Cloudian HyperStore Admin API Reference***.

## 7.5. Using Linux Utilities to Monitor System Resource Usage

The Linux OS on which the HyperStore system runs includes several useful commands for checking on system resource utilization on individual host machines. For example, the ***top*** command returns a summary of host-wide resource utilization as well as a breakdown of resource usage per process. Using ***top -c*** (which returns more information in the "Command" column than *top* alone) makes it easier to distinguish between the several Java-based HyperStore processes that will be running on each host — including Cassandra, the S3 Service (cloudian_s3), the Admin Service (cloudian_admin), the HyperStore Service (storage_s3), and the CMC (tomcat).

Important statistics are the memory usage and CPU usage:

- VIRT: Virtual memory.
- RES: Resident memory. VIRT minus RES is the amount of memory swapped out.
- %CPU: Percentage of CPU used

Other useful Linux utilities for monitoring system resource usage include:

- ***vmstat***
- ***iostat***
- ***dstat***

# 7.6. Using JMX to Monitor HyperStore Services

**IMPORTANT !** Cloudian recommends that you **do not use JMX for monitoring a HyperStore production system**, as it may negatively impact performance (particularly if you run JConsole on one of your production nodes). However, JMX may be useful for monitoring a HyperStore testing or evaluation system.

The S3 Service, Admin Service, HyperStore Service, and Cassandra Service support monitoring via Java Management Extensions (JMX). You can access JMX statistics using the graphical JMX client JConsole, which comes with your Java platform. By default the full path to the JConsole executable is */usr/java/latest/bin/jconsole*.

After launching JConsole, in the JConsole GUI specify the *<host>:<jmx-port>* that you want to connect to. Each of the HyperStore system's Java-based services has a different JMX listening port as indicated in the sections that follow. **The statistics that you view will be only for the particular node to which you are connected via JConsole.**

**Note** By default a JConsole connection does not require a user name and password, so in the JConsole GUI these fields can be left empty. For general information about using JConsole, including password protection and security options, see the JConsole Help.

**Note** This section on JMX statistics presumes that you are using JConsole, but there are other JMX clients available. Your HyperStore system comes with two command-line JMX clients — *cmdline-jmxclient* and *jmxterm* — which are in the */opt/cloudian/tools* directory.

## S3 Service JMX Statistics

Default JMX port: 19080

For the S3 Service, these categories of JMX statistics are supported:

*S3 operation timing and rate stats*

In JConsole's MBeans tab, timing performance statistics for S3 operations are available under the *metrics* MBean. Under *metrics* there is *com.cloudian.s3.stats.<operation>*, where *<operation>* is an S3 API operation such as *putObject*, *getObject*, *deleteObject*, *getBucket*, and so on. For each operation type, under *Attributes* there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the S3 Service. Statistics will only be available for operations that have been performed since the last S3 Service restart — for example, if no *deleteObject* operations have been performed since the last restart, then no *deleteObject* statistics will be available.

**Note** These timing and rates stats are implemented with the Metrics Core library.

*S3 operation success stats*

In JConsole's MBeans tab, success rate statistics for S3 operations are available under *com.gemini.cloudian.s3 → Accounting → Success Rate → Attributes*. For each S3 operation, the success rate is expressed as a double between 0 and 1 indicating the percentage of successful processing for that S3 operation. The success rate statistics are cumulative since the last start of the S3 Service on the node to which you are connected.

For example, the statistic "DeleteBucket" would have a value such as 1.0 or 0.92, indicating a 100% or 92% success rate for S3 "DELETE Bucket" operations since the last start-up of the S3 Service.

*S3 Service's Cassandra client stats*

For its client interface to Cassandra, the S3 Service leverages open source Hector technology. In JConsole's MBeans tab, Hector statistics are available under *me.prettyprint.cassandra.service_Cloudian<regionName> → hector → hector → Attributes*. Descriptions of these statistics are available in the **Health Check Attributes Available for Hector** section of the online Hector user guide.

The list of supported statistics is below.

- ExhaustedPoolNames
- KnownHosts
- NumActive
- NumBlockedThreads
- NumConnectionErrors
- NumExhaustedPools
- NumIdleConnections
- NumPoolExhaustedEventCount
- NumPools
- NumRenewedIdleConnections
- NumRenewedTooLongConnections
- ReadFail
- RecoverableErrorCount
- RecoverableLoadBalancedConnectErrors
- RecoverableTimedOutCount
- RecoverableTransportExceptionCount
- RecoverableUnavailableCount

- SkipHostSuccess
- StatisticsPerPool
- SuspendedCassandraHosts
- WriteFail

*HTTP server thread pool stats*

For serving HTTP requests, the S3 Service leverages open source Jetty technology. In JConsole's MBeans tab, Jetty thread pool statistics are available under *org.eclipse.jetty.util.thread → queuedthreadpool → 0 → Attributes*. The statistics available are:

- threads
- idleThreads
- queueSize

## Admin Service JMX Statistics

Default JMX port: 19081

In JConsole's MBeans tab, timing performance statistics for Admin Service operations are available under the *metrics* MBean. Under *metrics* there is *com.cloudian.admin.stats.<operation>*, where *<operation>* is an S3 API operation such as *getUser*, *createUser*, and so on. For each operation type, under *Attributes* there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the S3 Service (the Admin Service stops and starts together with the S3 Service). Statistics will only be available for operations that have been performed since the last S3 Service restart — for example, if no *createUse*r operations have been performed since the last restart, then no *createUser* statistics will be available.

**Note** These timing and rates stats are implemented with the Metrics Core library.

## HyperStore Service JMX Statistics

Default JMX port: 19082

For the HyperStore Service, these categories of JMX statistics are supported:

*HyperStore operation timing and rate stats*

In JConsole's MBeans tab, timing performance statistics for HyperStore Service operations are available under the *metrics* MBean. Under metrics there is *com.cloudian.hybrid.stats.<operation>*, where *<operation>* is a

HyperStore Service operation such as *put*, *getBlob*, *getDigest*, or *delete*. For each operation type, under *Attributes* there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the HyperStore Service. Statistics will only be available for operations that have been performed since the last HyperStore Service restart — for example, if no *delete* operations have been performed since the last restart, then no *delete* statistics will be available.

**Note** These timing and rates stats are implemented with the Metrics Core library.

*HyperStore operation success stats*

In JConsole's MBeans tab, HyperStore file system operations statistics are available under *com.gemini.cloudian.hybrid.server → HyperstoreOperationsStats → Attributes*. The statistics available are:

- NumberOfSuccessfulDeleteOperations
- NumberOfSuccessfulReadOperations
- NumberOfSuccessfulWriteOperations
- TotalNumberOfDeleteOperations
- TotalNumberOfReadOperations
- TotalNumberOfWriteOperations

*Cassandra client stats*

For its client interface to Cassandra, the HyperStore Service leverages open source Hector technology. In JConsole's MBeans tab, Hector statistics are available under *me.prettyprint.cassandra.service_Cloudian<regionName> → hector → hector → Attributes.*

The list of supported statistics is the same as indicated for the **S3 Service's Hector statistics**. Descriptions of these statistics are available in the **Health Check Attributes Available** for Hector section of the online Hector user guide.

*HTTP server thread pool stats*

For serving HTTP requests, the HyperStore Service leverages open source Jetty technology. In JConsole's MBeans tab, Jetty thread pool statistics are available under *org.eclipse.jetty.util.thread → queuedthreadpool → 0 → Attributes*. The statistics available are:

- threads
- idleThreads
- queueSize

### Cassandra Service (Metadata DB) JMX Statistics

Default JMX port: 7199

In JConsole's MBeans tab, under *org.apache.cassandra.metrics*, Cassandra supports a wide range of statistics.

The *Compaction* MBean exposes statistics including:

- Number of completed compactions.
- Number of pending compaction tasks.
- Progress of currently running compactions.

> **Note** If the number of pending compaction tasks grows over time, this is an indicator of a need to increase cluster capacity.

The *ColumnFamily* MBean exposes statistics for active, pending, and completed tasks for each of Cassandra's thread pools.

> **Note** A significant and sustained increase in the pending task counts for the Cassandra thread pools is an indicator of a need to increase cluster capacity.

The *ColumnFamily* MBean exposes individual column family statistics including:

- Memtable data size
- Number of live SSTables
- Average read latency
- Average write latency

> **Note** A sustained increase in read and write latencies may indicate a need to increase cluster capacity.

## 7.7. Checking HTTP(S) Responsiveness

The HyperStore system supports a health check that lets administrators or applications -- such as a load balancer -- determine whether HTTP(S) interfaces are responsive on a particular HyperStore host. To perform the health submit an HTTP(S) **HEAD** request to:

*<hostname or IP address>:<port>/.healthCheck*

If the service is up and running and listening on its assigned port, you will receive back an HTTP 200 OK status. If not, your request will time out.

This feature is supported for the following HTTP(S) based services and ports:

| Service | Default HTTP Port | Default HTTPS Port |
|---|---|---|
| S3 Service | 80 | 443 |
| IAM Service | 16080 | 16443 |
| SQS Service | 18090 | not applicable |

| Service | Default HTTP Port | Default HTTPS Port |
|---|---|---|
| Admin Service | 18081 | 19443 |
| HyperStore Service | 19090 | not applicable |

**Note**
* To do health checks against an HTTPS port, the client executing the check must support TLS/SSL. Note also that health checks against HTTPS ports are a more expensive operation (in terms of resource utilization) than health checks against HTTP ports.
* HTTP is disabled by default for the Admin Service, so that only HTTPS is supported. See the Introduction section of the ***Cloudian HyperStore Admin API Reference***.
* The HTTP(S) health check of the Admin API service does not require Basic Authentication credentials. Basic Authentication is required for all other HTTP(S) requests to the Admin API, but is not required for health check requests.
* The SQS Service is disabled by default. See the introduction to the SQS section of the ***Cloudian HyperStore AWS APIs Support Reference***.

The example below shows a health check of an S3 Service instance that is responsive.

```
HEAD http://192.168.2.16:80/.healthCheck
Status Code: 200 OK
Content-Length: 0
Date: Wed, 25 Aug 2021 12:51:50 GMT
Server: CloudianS3
```

In the case of health checks of the S3 Service, each health check request results in a special entry in the **S3 Request Log**, such as in this example entry:

```
2021-08-16 15:01:33,757|127.0.0.1||healthCheck|||81|0|0|0|81|11820||200|
544cdd90-822f-1c98-b780-525400e89933|0|0|
```

## 7.7.1. Cloudian Management Console (CMC)

To check the responsiveness of the CMC, submit an HTTPS **OPTIONS** request to the CMC login URL:

*https://<host>:8443/Cloudian/login.htm*)

**Note** Sending a GET or HEAD request to the CMC login URL will result in the CMC sending a *GET /group/list* call to the Admin Service which in turn sends a request to Cassandra. To avoid this, the more lightweight way to check CMC responsiveness is to send an OPTIONS request to the CMC login URL.

This page left intentionally blank

# Chapter 8. Logging

## 8.1. HyperStore Logs

*Subjects covered in this section:*



The major HyperStore services each generate their own application log. The S3 Service, Admin Service, and HyperStore Service, in addition to generating application logs, also generate transaction (request) logs.

The log descriptions below indicate each log's default location, logging level, rotation and retention policy, log entry format, and where to modify the log's configuration.

**Note** With the exception of Cassandra and Redis logs, all HyperStore logs are located in */var/log/cloudian*.

**Note** For information on viewing logs from within the HyperStore Shell, see **"Using the HSH to View Logs"** (page 374).

### 8.1.1. Admin Service Logs

*Admin Service application log (cloudian-admin.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-admin.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE`<br>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry Example | `2021-05-04 15:48:03,158 ERROR[main]HS081003 CassandraProtectionPolicy:Caught: me.prettyprint.hector.api.exceptions.` |

| | |
|---|---|
| | `HectorException: [10.50.10.21(10.50.10.21):9160] All host pools marked down. Retry burden pushed out to client.` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-admin.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-admin.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="ADMINAPP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*Admin Service request log (cloudian-admin-request-info.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-admin-request-info.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|ClientIpAddress\|HttpMethod\|Uri\|QueryParams\|DurationMicrosecs\|HttpStatus`<br>**Note**<br>* Query parameters are not logged for requests that involve user credentials.<br>* The request log records Admin API requests for which authentication fails, as well as requests for which the authentication succeeds. Success or failure is indicated by the HttpStatus. |
| Log Entry Example | `2021-10-27 14:54:01,170\|10.20.2.57\|GET\|/group/list\|limit:100\|188212\|200` |
| Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-admin-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 2GB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-admin.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="ADMINREQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.2. Cassandra (Metadata DB) Logs

*Cassandra application log (system.log)*

| | |
|---|---|
| Location | On every node, *var/log/cassandra/system.log* |
| Log Entry Format | `PriorityLevel [ThreadId] Date(ISO8601) CallerFile:Line# - MESSAGE` |
| Log Entry Example | `INFO [FlushWriter:3] 2016-11-10 21:09:59,487 Memtable.java:237 - Writing Memtable-Migrations @445036697(12771/15963 serialized/live bytes, 1 ops)` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 20MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *system.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 200MB **or** if oldest rotated file age reaches 30 days. |
| Configuration | Configuration: */etc/cloudian-<version>-puppet/modules/cassandra/templates/logback.xml.erb*. For setting descriptions see the online documentation for Logback:<br>• **FixedWindowRollingPolicy**<br>• **SizeBasedTriggeringPolicy** |

*Cassandra request log (cassandra-s3-tx.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cassandra-s3-tx.log*. Note that these request logs are written on the **client side** as the S3 Service sends requests to Cassandra. |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS LogEntryType S3RequestId MESSAGE`<br>The LogEntryType is one of NORMAL, ERROR, or SLOW. |
| Log Entry Example | `2016-11-22 13:06:27,192 c.d.d.c.Q.SLOW [cluster1] [/10.20.2.146:9042] Query too slow, took 7674 ms: alter table "UserData_20d784f480b0374559bdd71054bbb8c1"."CLOUDIAN_METADATA" with gc_grace_seconds=864000 and bloom_filter_fp_chance=0.1 and compaction = {'class':'LeveledCompactionStrategy'};` |
| Default Logging Level | DEBUG<br>**Note** In *log4j-s3.xml.erb* on your Configuration Master node there are three different *AsyncLogger* instances for Cassandra request logging. The ERROR logger logs entries when a Cassandra request results in an error; the SLOW logger logs entries when a Cassandra request takes more than 5 seconds to process; and the NORMAL logger logs all Cassandra requests. The three loggers all write to */var/log/cloudian/cassandra-s3-tx.log*, and the implementation prevents duplicate entries across the three loggers. All three loggers are set to DEBUG level by default; and each logger works only if set to DEBUG or TRACE. To disable a logger, set its level |

| | |
|---|---|
| | to INFO or higher. For example to disable the NORMAL logger so that only error and slow requests are recorded, set the NORMAL logger's level to INFO. Then do a Puppet push and restart the S3 Service. |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cassandra-s3-tx.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="CASSANDRATX"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

**Note** Currently **only a fraction of request types from the S3 Service to Cassandra support this request logging feature**. These are request types that use a new DataStax Java driver (which supports the request logging) rather than the older Hector driver (which does not).

## 8.1.3. CMC Logs

*CMC application log (cloudian-ui.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-ui.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId] ClassName:MESSAGE`<br>In the case of log entries for user logins to the CMC, the MESSAGE value will be formatted as follows:<br>Normal login<br>`Login <groupId>\|<userId> from: <ipAddress> Success`<br>`Login <groupId>\|<userId> from: <ipAddress> Failed [<reason>]`<br>SSO login<br>`SSOLogin <groupId>\|<userId> from: <ipAddress> Success`<br>`SSOLogin <groupId>\|<userId> from: <ipAddress> Failed [<reason>]` |
| Log Entry Example | `2017-05-04 11:56:48,475  INFO [localhost-startStop-1] ServiceMapUtil:Loading service map info from: cloudianservicemap.json` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-ui.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*. |

| | |
|---|---|
| | Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cmc/templates/log4j.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*CMC user audit log (ui-action.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/ui-action.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|SourceIP\|AdminID\|Action\|CanonicalUserId\|UserId\|GroupId\|Result`<br>The *ui-action.log* captures CMC actions relating to login, logout, and the creation, editing, and deletion of user accounts. This log is generated on each node running the CMC, and on each node records only activity processed by that node's CMC instance. Through the CMC you can download a CSV (comma-separated value) file that aggregates all the *ui-action.log* content from across all CMC instances. For more information about downloading the CSV file, and for more information about the content of this log, while on the CMC's Manage Users page click **Help**. |
| Log Entry Example | `2021-01-16 09:22:13,812\|172.16.6.184\|0%7Cadmin\|CreateUser\|\|testU2\|test\|Success` |
| Default Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *ui-action.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 90 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cmc/templates/log4j.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="UIREQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.4. HyperStore Firewall Log

*HyperStore firewall log (firewall.log)*

If the HyperStore firewall is enabled in your system then DROP'd connections are logged in the HyperStore firewall log. For information about the firewall see **"HyperStore Firewall"** (page 491).

| | |
|---|---|
| Location | On every node, */var/log/cloudian/firewall.log* |
| Log Entry | The log records information about dropped packets, including the timestamp, host, firewall |

| | |
|---|---|
| Format | zone (*cloudian-backend* [for the designated internal interface] or *cloudian-frontend* [for all other interfaces]), interface name and MAC address, source and destination address, protocol, TCP flags, and so on. |
| Log Entry Example | `Apr 18 11:03:19 demo4-node1 kernel: IN_cloudian-frontend_DROP: IN=eth0 OUT= MAC=52:54:00:e3:82:d7:52:54:00:11:dd:79:08:00 SRC=10.254.254.103 DST=10.254.254.118 LEN=44 TOS=0x00 PREC=0x00 TTL=57 ID=43247 PROTO=TCP SPT=52377 DPT=74 WINDOW=1024 RES=0x00 SYN URGP=0` |
| Default Rotation and Retention Policy | Rotation occurs hourly if the live file size has reached 10MB; or else daily regardless of file size (except that there is no rotation of an empty live log file).<br>Rotated files are named as *firewall.log-YYYY-MM-DD.HH.gz*. Rotated files are compressed with *gzip*.<br>Rotated files are retained for 180 days and then automatically deleted. |
| Configuration | Rotation of this log is managed by the Linux *logrotate* utility. In the current version of HyperStore, the rotation settings for the HyperStore firewall log are not configurable. |

## 8.1.5. HyperStore Service Logs

*HyperStore Service application log (cloudian-hyperstore.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-hyperstore.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[S3RequestId][ThreadId]MessageCode ClassName:MESSAGE`<br>• The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help.<br>• The S3RequestId value is present only in messages associated with implementing S3 requests. |
| Log Entry Example | `2017-05-04 23:58:34,634 ERROR[][main]HS220008 CloudianAbstractServer:Unable to load configuration file: storage.xml` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-hyperstore.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*HyperStore Service request log (cloudian-hyperstore-request-info.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-hyperstore-request-info.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|IpAddressOfClientS3Server\|S3RequestId\|HttpStatus\|HttpOperation\|OriginalUri\|HyperStoreFilePath\|ContentLength\|DurationMicrosecs\|Etag\|ECSuffix`<br>• The IpAddressOfClientS3Server is the IP address of the S3 Server node that submits the request to the HyperStore Service.<br>• The HttpOperation is the HyperStore Service HTTP API operation that the S3 Service invokes and will be a simple operation like "PUT" or "GET". In the case of a "secure delete", the operation will be "SECURE-DELETE" and this request won't be logged until the third and final data overwriting pass for the object is completed. By contrast a regular delete will be logged as operation "DELETE". The secure delete feature is disabled by default. For more information see **"Enabling Secure Delete"** (page 167).<br>• The OriginalUri field shows the group ID, bucket name, and object name from the originating S3 API request, in URI-encoded form ("CloudianTest1", "buser1", and "514kbtes", respectively, in the example above).<br>• The Etag field will be "0" for operations other than PUT.<br>• The ECSuffix field value is "null" if not an erasure coded data request. |
| Log Entry Example | Erasure coded object:<br>`2020-11-03 06:23:35,090\|10.112.1.79\|3f1b624f-d7b7-1e2c-a6b1-0a690b692ef3\|200\|GET\|/ec/ngrp%2Fm-MDA0OTI1MjcxNjA0Mzg0NTM2MzYx%2Fecbn%2Fmpu..0002.1\|/cloudian1/ec/std8ZdRJDskcPvmOg4/db35e009d75fff7fce122105f6d3b877/196/037/90637448863864709563044774788357063423.1604384536362536381-0A70014F\|5242880\|15277\|0\|3`<br>Replicated object:<br>`2020-11-03 06:42:22,201\|10.112.1.79\|3f1b627a-d7b7-1e2c-a6b1-0a690b692ef3\|200\|PUT\|/file/ngrp%2Fhsfsbn%2Fhsfs\|/cloudian1/hsfs/std8ZdRJDskcPvmOg4/9894597460837bab308e7327144b4bcc/227/232/16918770883092121100470776750103075026 9.1604385742187742235-0A70014F\|1048576\|186769\|379228ccb18fb5f795aaaa17a459f0ed\|null` |
| Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 300MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-hyperstore-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 3GB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="REQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*HyperStore Service cleanup log (cloudian-hyperstore-cleanup.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-hyperstore-cleanup.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|Command#\|ChunkName\|ChunkFilePath`<br><br>This log has entries when an **hsstool cleanup** or **hsstool cleanupec** operation results in files being deleted from the node. A cleanup operation that determines that no files need to be deleted from the node will not cause any entries to this log.<br><br>**Note** For background information about "chunks" and chunk names see **"Creating an Input File"** (page 561). For information about chunk file paths see **"HyperStore Service and the HSFS"** (page 63). |
| Log Entry Example | `2018-02-28 05:57:25,743\|1\|buser1/obj1\|/var/lib/cloudian/hsfs/SalA11OVu6oCThSSRafvH/7cf10597b0360421d7564e7c248b2445/165/206/163985596354481463889148061573011679 71.1476861996241` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *cloudian-hyperstore-cleanup.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="CLEANUP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*HyperStore Service repair log (cloudian-hyperstore-repair.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-hyperstore-repair.log* |
| Log Entry Format | This log has entries when a repair operation results in an attempt to repair data on the node. A repair operation that determines that no repairs are needed on the node will not cause any entries to this log (but will result in entries to the application log *cloudian-hyperstore.log*).<br><br>In *cloudian-hyperstore-repair.log* there are two different types of entries, with different fields. The first type of entry records information about a repair action being taken and has this format:<br><br>`yyyy-mm-dd HH:mm:ss,SSS\|RepairType\|Command#\|Coordinator\|RepairEndpoint\|StreamFromEndpoint\|ChunkName\|Path\|ChunkSize\|Md5Hash\|RepairLatencyMillisecs\|Suffix`<br><br>In most cases the RepairType field value will be one of:<br><br>• RR -- regular repair for replicas |

<table>
<tr><td></td><td>
<ul>
<li>PRR -- proactive repair for replicas</li>
<li>REC -- regular repair for erasure coded data</li>
<li>PREC -- proactive repair for erasure coded data</li>
<li>UECD -- update of EC digest fields</li>
</ul>
The Coordinator is the node to which the hsstool repair or hsstool repairec command was submitted. The StreamFromEndpoint is the node from which a replica or erasure coded fragment was streamed in order to repair missing or bad data at the RepairEndpoint node. For erasure coded data repair, the fragment is streamed from the node that performed the decoding and re-encoding of the repaired object.
The Suffix field indicates the fragment suffix in the case of erasure coding repairs. For replica repairs, suffixes are not applicable and this field's value will be "-1".
Note For background information about "chunks" and chunk names see "Creating an Input File" (page 561). For information about chunk file paths see "HyperStore Service and the HSFS" (page 63).
The other type of entry in cloudian-hyperstore-repair.log records the outcome of successful erasure coding repair tasks. These entries have this format:
<pre>Ok|yyyy-mm-dd HH:mm:ss,SSS|RepairType|ChunkName|Path|TaskType</pre>
The ECRepairTaskType field value indicates the type of erasure coded fragment problem that was successfully repaired -- one of "[MD5_MISMATCH]", "[MISSING]", or "[DUPLICATE]".
Note Failed erasure coding repair tasks are logged in cloudian-hyperstore-repair-failure.log (described further below).
</td></tr>
<tr><td>Log Entry Examples</td><td>
Example #1 (information about an EC repair operation)
<pre>2021-06-10 23:00:12,265|REC|13|10.20.1.81|10.20.1.81|10.20.1.81|
m-MDAzYTMyMTcxNjIzMzkwODA5Mzk2/ecbn/mpmc..0002.2|/cloudian3/ec/
1SZe38K1qp98VDadplVxyq/257cb007cd57283bad02763e83864bb7/075/133/
279365030550901462400171341501819848 43.1623390809396809455-0A140151|
524288|8ad63b8f18e934333bdc2164bd7f8553|8|2</pre>
Example #2 (a successful EC repair task outcome):
<pre>Ok|2021-06-10 23:00:12,272|REC|m-MDAzYTMyMTcxNjIzMzkwODA5Mzk2/ecbn/mpmc..0002.2|
/cloudian3/ec/1SZe38K1qp98VDadplVxyq/257cb007cd57283bad02763e83864bb7/075/133/
279365030550901462400171341501819848 43.1623390809396809455-0A140151|[MISSING]</pre>
</td></tr>
<tr><td>Default Logging Level</td><td>Not applicable</td></tr>
<tr><td>Default Rotation and Retention Policy</td><td>
Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.
Rotated files are named as cloudian-hyperstore-repair.log.YYYY-MM-DD.i.gz, where i is a rotation counter that resets back to 1 after each day. Rotated files are compressed with gzip
Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
</td></tr>
</table>

| | |
|---|---|
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="REPAIR"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*HyperStore Service EC repair failure log (cloudian-hyperstore-repair-failure.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-hyperstore-repair-failure.log* |
| Log Entry Format | When *repairec* has been run, this log records an entry for each chunk for which the repair attempt failed on the node. These entries have this format:<br>`ChunkName\|Path\|OperationId\|yyyy-mm-dd HH:mm:ss,SSS\|FailReason[\|TaskType]`<br>The OperationId is a system-generated unique identifier of the EC repair run (i.e., each time you run *hsstool repairec* the run will be assigned its own unique ID, so that different runs can be distinguished from each other).<br>The FailReason field value indicates the failure reason -- for example "LESS_THAN_K" in the event that fewer than K good fragments were available for the chunk.<br>The TaskType field may or may not be present, depending on the FailReason. When present, the TaskType field indicates the type of EC fragment problem that the failed repair task was attempting to address -- either "[MD5_MISMATCH]" or "[MISSING]" or "[DUPLICATE]".<br>**Note** The *cloudian-hyperstore-repair-failure.log* file -- or excerpts from this log file, that you save into a separate file -- can be used as input to a *repairec -f <input-file>* operation, to try again to repair the chunks for which the prior repair attempt failed. See **"hsstool repairec"** (page 555).<br>**Note** For background information about "chunks" and chunk names see **"Creating an Input File"** (page 561). For information about chunk file paths see **"HyperStore Service and the HSFS"** (page 63). |
| Log Entry Examples | Example #1:<br>`ecbn/hoge\|/cloudian2/ec/std8ZdRJDskcPvmOg4/96f96c866408a3fc9e42237ae94bce5d/170/045/156312679751569259476355507179662176759.1631169767765768030-0A32C89D\|d8cd52cc-0e72-1cd4-b03c-000c29b5a219\|2021-09-08 23:47:42,654\|NODE_DOWN`<br>Example #2:<br>`ecbn/sig\|/cloudian1/ec/1IVfPFBIijTqx0GvLCwoi0/35418348e30905d622f692c451868e73/172/190/258795835870696908070155874865642461l8.1626338080577080582-0A700108\|3bad2bf2-95ce-1070-b320-0acf12cb1e97\|2021-07-15 08:47:55,126\|EC_DECODE_FAILED\|[MISSING]` |
| Default Logging Level | Not applicable |
| Default Rotation and | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size. |

| | |
|---|---|
| Retention Policy | Rotated files are named as *cloudian-hyperstore-repair-failure.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 10000 MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="REPAIR_FAILURE"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*HyperStore Service whereis log (whereis.log)*

This log is generated if and only if you execute the *hsstool whereis -a* command, to output location detail for every S3 object in the system. For information about this log see **hsstool whereis**.

## 8.1.6. HyperStore Shell (HSH) Log

*HyperStore shell log (hsh.log)*

If the HyperStore shell (HSH) is enabled in your system then HSH user logins and commands are logged in the HyperStore shell log. An instance of the HyperStore shell log resides on each HyperStore node and records any HSH logins and commands on that node. For information about the HSH see **"HyperStore Shell (HSH) Feature Overview"** (page 95).

| | |
|---|---|
| Location | On every node, */var/log/hsh/hsh.log* |
| Log Entry Examples | HSH user login:<br>`time="2019-09-04T20:26:21-07:00" level=info msg="New Session" session=15C16CFDA4A46863 user=sa_admin`<br>HSH user running a command:<br>`time="2019-09-04T21:22:30-07:00" level=info msg="Running command." args="<s:setup>" command=hspkg session=15C16CFDA4A46863 type=interactive`<br><br>`time="2019-09-04T21:22:30-07:00" level=info msg="Executing command." cwd=/home/sa_admin mode=User path=/opt/cloudian-staging/7.2/system_setup.sh runuser=root session=15C16CFDA4A46863 tty=true`<br><br>`time="2019-09-04T21:24:38-07:00" level=info msg="Command complete." args="<s:setup>" command=hspkg session=15C16CFDA4A46863 status=0 type=interactive` |
| Default Rotation and Retention Policy | Rotation occurs monthly or if the live file size reaches 500MB.<br><br>Rotated files are named as *hsh.log.<n>*, with *hsh.log.1* being the most recently rotated file. The most recently rotated file is not compressed; all other rotated files are compressed with *gzip* and the file names will have a *.gz* suffix.<br><br>12 rotated files are retained. Older files are automatically deleted. |
| Configuration | Rotation of this log is managed by the Linux *logrotate* utility. In the current version of Hyper- |

| | |
|---|---|
| | Store, the rotation settings for the HyperStore shell log are not configurable. |

## 8.1.7. IAM Service Logs

*IAM application log (cloudian-iam.log)*

| | |
|---|---|
| Location | On every node in the default service region, */var/log/cloudian/cloudian-iam.log*<br><br>**Note** For more information on the IAM Service see the "IAM API" section of the ***Cloudian HyperStore AWS APIs Support Reference***. |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE`<br><br>• The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry Example | `2018-02-16 10:16:36,246  INFO[qtp214187874-41] CloudianHFactory:Creating DynamicKS for: AccountInfo` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *cloudian-iam.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-iam.xml.erb* , this log is configurable in the block that starts with *<RollingRandomAccessFile name="IAMAPP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*IAM request log (cloudian-iam-request-info.log)*

| | |
|---|---|
| Location | On every node in the default service region, */var/log/cloudian/cloudian-iam-request-info.log*<br><br>**Note** This log records Security Token Service (STS) requests as well as IAM requests. |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|ClientIpAddress\|AccountRootUserCanonicalId\|RequestorUserId\|GroupId\|Protocol:Action\|IamUserId\|RoleSessionArn\|RoleSessionId\|TempCredentialsAccessKey\|HttpStatus\|ErrorCode\|ResponseData\|DurationMicrosecs`<br><br>• For the IamUserId field: |

| | |
|---|---|
| | ◦ If request is by IAM user, this is the UserId of the IAM user<br>◦ If request is by a non-SAML role session, this is the UserId of the IAM user who assumed the role<br>◦ If request is by a SAML role session, this is the Subject field value from the SAML Assertion (the user identifier)<br>◦ If request is by an account root user, this field is empty<br>• If request is by a role session, the RoleSessionArn, RoleSessionId, and TempCredentialsAccessKey fields provide information about the role session making the request. Otherwise these fields are empty.<br>• If the action is AssumeRole or AssumeRoleWithSAML, the ResponseData field provides information about the role being assumed (RoleSessionArn&RoleSessionId&TempCredentialsAccessKey). Otherwise this field is empty. |
| Log Entry Examples | `2020-08-10 18:58:59,607\|10.20.2.34\|679d95846fb0f0047f5926ba16546552\|testu159986\|myGroup8732\|iam:PutRolePolicy\|\|\|\|200\|\|\|6`<br><br>`2020-08-10 18:58:59,619\|10.20.2.34\|679d95846fb0f0047f5926ba16546552\|testu159986\|myGroup8732\|sts:AssumeRole\|aidcc54d3e60de2a74e89ad639561df0\|\|\|200\|\|arn:aws:iam::679d95846fb0f0047f5926ba16546552:role/iammypath/rolen134094&rolesn54301&asicbd8ef4bdd6e7e03c\|118` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-iam-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 2GB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-iam.xml.erb* , this log is configurable in the block that starts with *<RollingRandomAccessFile name="IAMREQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.8. Monitoring Agent and Collector Logs

*Monitoring Agent application log (cloudian-agent.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-agent.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId] ClassName:MESSAGE` |
| Log Entry Example | `2017-05-04 11:57:03,698  WARN [pool-2-thread-7] LogFileTailerListener:Log file not found for service:cron` |
| Default Log- | WARN |

| | |
|---|---|
| ging Level | |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-agent.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudianagent/templates/log4j-agent.xml.erb*, this log is configurable in the block that starts with *RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*Monitoring Data Collector application log (cloudian-datacollector.log)*

| | |
|---|---|
| Location | On the Monitoring Data Collector node, */var/log/cloudian/cloudian-datacollector.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE`<br>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry Example | `2017-05-04 00:00:05,898  WARN[main]DC040073 SmtpNotification: Failed to send due to messaging error: Couldn't connect to host, port: smtp.notification.configure.me, 465; timeout 5000` |
| Default Logging Level | WARN |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-datacollector.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-datacollector.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.9. Phone Home (Smart Support) Log

*Phone Home application log (cloudian-phonehome.log)*

| | |
|---|---|
| Location | On the Monitoring Data Collector node, */var/log/cloudian/cloudian-phonehome.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE` |

| | |
|---|---|
| | The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry Example | `2017-05-04 19:13:03,384 ERROR[main]HS200043 S3:All Redis read connection pools are unavailable. Redis HGETALL of key BPP_MAP fails.` |
| Default Logging Level | WARN |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-phonehome.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-phonehome.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.10. Redis (Credentials DB and QoS DB) and Redis Monitor Logs

*Redis Credentials application log (redis-credentials.log)*

| | |
|---|---|
| Location | On Redis Credentials nodes, */var/log/redis/redis-credentials.log* |
| Log Entry Format | `PID.role dd MMM hh:mm:ss.ms loglevel MESSAGE`<br>The role will usually be "M" for master or "S" for slave.<br>The loglevel will be "*" for NOTICE or "#" for ERROR. |
| Log Entry Example | `18290:S 28 Jul 01:23:42.416 # Connection with master lost.` |
| Default Logging Level | NOTICE |
| Default Rotation Policy | Not rotated by default. You can set up rotation by using **logrotate**. |
| Configuration | Redis Credentials application logging is configured in the main Redis configuration file. The file name depends on the Redis node type -- master or slave. These templates are on the Configuration Master node, under */etc/cloudian-<version>-puppet/modules/redis/templates/*:<br>• Redis Credentials master node: *redis-credentials.conf.erb*<br>• Redis Credentials slave node: *redis-credentials-slave.conf.erb*<br>The only configurable logging settings are the log file name and the logging level. See the commenting in the configuration file for more detail. |

*Redis QoS application log (redis-qos.log)*

| | |
|---|---|
| Location | On Redis QoS nodes, */var/log/redis/redis-qos.log* |
| Log Entry Format | `PID.role dd MMM hh:mm:ss.ms loglevel MESSAGE`<br>The role will usually be "M" for master or "S" for slave.<br>The loglevel will be "*" for NOTICE or "#" for ERROR. |
| Log Entry Example | `24401:M 28 Jul 01:41:46.963 * Calling fsync() on the AOF file.` |
| Default Logging Level | NOTICE |
| Default Rotation Policy | Not rotated by default. You can set up rotation by using **logrotate**. |
| Configuration | Redis QoS application logging is configured in the main Redis configuration file. The file name depends on the Redis node type -- master or slave. These templates are on the Configuration Master node, under */etc/cloudian-<version>-puppet/modules/redis/templates/*:<br>• Redis QoS master node: *redis-qos.conf.erb*<br>• Redis QoS slave node: *redis-qos-slave.conf.erb*<br>The only configurable logging settings are the log file name and the logging level. See the commenting in the configuration file for more detail. |

*Redis request logs (redis-{s3,admin,hss}-tx.log)*

| | |
|---|---|
| Location | If enabled, these logs are written to */var/log/cloudian/redis-{s3,admin,hss}-tx.log* on the S3 Service, Admin Service, and/or HyperStore nodes (that is, these request logs are written on the **client side** as the S3, Admin, and HyperStore services instances send requests to Redis). |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS S3RequestId MESSAGE` |
| Log Entry Example | `2016-11-17 23:54:01,695  CLIENT setname ACCOUNT_GROUPS_M` |
| Default Logging Level | INFO<br>**Note** The default logging level of INFO disables these logs. If you want these logs to be written, you must edit the Puppet template files *log4j-s3.xml.erb* (for logging S3 Service access to Redis), *log4j-admin.xml.erb* (for logging Admin Service access to Redis), and/or *log4j-hyperstore.xml.erb* (for logging HyperStore Service access to Redis). Find the *AsyncLogger name="redis.clients.jedis"* block and change the *level* from "INFO" to "TRACE". Then do a Puppet push, and then restart the relevant services (S3-Admin and/or HyperStore). |
| Default Rotation and Retention | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size. |

| | |
|---|---|
| Policy | Rotated files are named as *redis-{s3,admin,hss}-tx.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template files */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-{s3,admin,hss}.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="REDISTX"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*Redis Monitor application log (cloudian-redismon.log)*

| | |
|---|---|
| Location | On Redis Monitor nodes, */var/log/cloudian/cloudian-redismon.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE`<br>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry Example | `2017-05-04 00:01:13,590  INFO[pool-23532-thread-3] RedisCluster: Failed to connect to cloudian jmx service [mothra:19082]` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-redismon.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-redismon.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.11. S3 Service Logs (including Auto-Tiering, CRR, and WORM)

*S3 Service application log (cloudian-s3.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-s3.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[S3RequestId][ThreadId]MessageCode ClassName:MESSAGE`<br>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the "Log Message Codes" section of the CMC's online Help. |
| Log Entry | `2017-05-04 00:33:39,435 ERROR[bc6f3fca-9037-135f-a964-0026b95cedde]` |

| | |
|---|---|
| Example | `[qtp1718906711-94]HS204017 XmlSaxParser:Rule doesn't have AllowedMethods/AllowedOrigins.` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-s3.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="S3APP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*S3 Service request log (cloudian-request-info.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-request-info.log* |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|ClientIpAddress\|BucketOwnerUserId\|Operation\|BucketName\|RequestorUserId\|RequestHeaderSize\|RequestBodySize\|ResponseHeaderSize\|ResponseBodySize\|TotalRequestResponseSize\|DurationMicrosecs\|UrlEncodedObjectName\|HttpStatus\|S3RequestId\|Etag\|ErrorCode\|SourceBucketName/UrlEncodedSourceObjectName\|GroupId\|CanonicalUserId\|IamUserId\|RoleSessionArn\|RoleSessionId\|AccessKey\|UserAgent\|DeleteSize\|BucketCanonicalId` |

- The Operation field indicates the S3 API operation. Note that "getBucket" indicates GET Bucket (List Objects) Version 1 whereas "getBucketV2" indicates GET Bucket (List Objects) Version 2. (In the case of a **health check request**, the Operation field indicates "healthCheck". In the case of requests submitted to the S3 Service by a **system cron job**, the Operation field indicates the name of the cron job action, such as "sytemBatchDelete").
- The Etag field is the Etag value from the response, if applicable to the request type. For information about Etag see for example **Common Response Headers** from the Amazon S3 REST API spec. This field's value will be 0 for request/response types that do not use an Etag value.
- The ErrorCode field is the Error Code in the response body, applicable only for potentially long-running requests like PUT Object. If there is no Error Code in the response body this field's value will be 0. For possible Error Code values see **Error Responses** from the Amazon S3 REST API spec.

> **Note** In the case where the Operation field value is *deleteObjects*, the ErrorCode field will be formatted as *objectname1-errorcode1,objectname2-errorcode2,objectname3-errorcode3...*, and the object names will be URL-encoded. If there are no errors the field is formatted as *objectname1-0,ob-*

| | |
|---|---|
| | *jectname2-0,objectname3-0....*<br>• The SourceBucketName/UrlEncodedSourceObjectName field is populated only for copyObject and uploadPartCopy operations and is empty for other operation types.<br>• For the IamUserId field:<br>◦ If request is by IAM user, this is the UserId of the IAM user<br>◦ If request is by a non-SAML role session , this is the UserId of the IAM user who assumed the role<br>◦ If request is by a SAML role session, this is the Subject field value from the SAML Assertion (the user identifier)<br>◦ If request is by an account root user, this field is empty<br>• If request is by a role session, the RoleSessionArn and RoleSessionId fields provide information about the role session making the request. Otherwise these fields are empty.<br>• The AccessKey is the access key of the account root user, IAM user, or IAM role session that submitted the request. This is from the user's or role session's security credentials, which consist of an access key and a secret key.<br>• The UserAgent value is truncated to 35 characters if longer than that<br>• The DeleteSize is the number of net bytes deleted, if the operation is a deleteObject operation (this is the deleted object size, excluding any overhead from replication or erasure coding). For operations other than deleteObject this field's value will be the string 'null'.<br>• The BucketCanonicalId is a unique system-generated ID assigned to the bucket. (Note that if an administrator runs the Admin API operation *POST /bucketops/purge* against a bucket -- which deletes every object in the bucket -- the system assigns the bucket a new canonical ID.)<br>NOTE:<br>Cloudian HyperIQ is a solution for dynamic visualization and analysis of HyperStore system monitoring data and S3 service usage data. HyperIQ is a separate product available from Cloudian that deploys as virtual appliance on VMware or VirtualBox and integrates with your existing HyperStore system. For more information about HyperIQ contact your Cloudian representative. |
| Log Entry Example | `2022-06-17 01:09:07,138\|10.50.41.207\|user1\|putObject\|bucket2\|user1\|396\|70\|180\|405\|1051\|6619\|obj1\|307\|668d5360-071c-153f-becd-000c29bd5995\|0\|TemporaryRedirect\|\|CloudianTest1\|f1a312a31d912441be2fea02f8677982\|\|\|5cb6967a8f8569facb6d\|Boto/2.42.0 Python/2.7.5 Linux/3.10\|null\|620dbabd2b540109d7f945b15c5b6079` |
| Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*. |

| | |
|---|---|
| | Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 2GB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="S3REQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## Logging the True Originating Client IP Address

If you use a load balancer in front of your S3 Service (as would typically be the case in a production environment), then the *ClientIpAddress* in your S3 request logs will by default be the IP address of a load balancer rather than that of the end client. If you want the S3 request logs to instead show the end client IP address, your options depend on what load balancer you're using.

If your load balancer is HAProxy or a different load balancer that supports the PROXY Protocol, enable S3 support for the PROXY Protocol (see **"s3_proxy_protocol_enabled"** (page 411) in **common.csv**) and configure your load balancer to use the PROXY Protocol for relaying S3 requests to the S3 Service. Consult with your Cloudian Sales Engineering or Support representative for guidance on load balancer configuration.

If your load balancer does not support the PROXY Protocol:

1. Configure your load balancers so that they pass the HTTP *X-Forwarded-For* header to the S3 Service. This is an option only if you run your load balancers run in "HTTP mode" rather than "TCP mode". Consult with your Cloudian Sales Engineering or Support representative for guidance on load balancer configuration.
2. Configure your S3 Service to support the *X-Forwarded-For* header. You can enable S3 Service support for this header by editing the configuration file *s3.xml.erb* on your Configuration Master node. The needed configuration lines are already in that file; you only need to uncomment them.

   Before uncommenting:

   ```
   <!-- Uncomment the block below to enable handling of X-Forwarded- style headers -->
   <!--
   <Call name="addCustomizer">
     <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/></Arg>
   </Call>
   -->
   ```

   After uncommenting:

   ```
   <!-- Uncomment the block below to enable handling of X-Forwarded- style headers -->
   <Call name="addCustomizer">
   <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/></Arg>
   </Call>
   ```

   After making this configuration edit, **do a Puppet push and restart the S3 Service** to apply your change.

*Auto-Tiering request log (cloudian-tiering-request-info.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-tiering-request-info.log* |

| | |
|---|---|
| | **Note** All nodes participate in tiering objects to their intended destinations systems. In the case of regular auto-tiering based on user-defined schedules, the tiering processing workload is randomly spread across the nodes in the service region in which the cron job host resides. In the case of Bridge Mode tiering (also known as proxy tiering), whichever node processes the S3 uploading of an object into its source bucket also processes the immediate tiering of that object to its intended destination system. For more information on auto-tiering see **"Auto-Tiering Feature Overview"** (page 169). |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|Command\|Protocol\|SourceBucket/Object\|SourceObjectVersion\|TargetBucket\|TargetObjectVersion\|ObjectSize\|TotalRequestSize\|Status\|DurationMicrosecs\|Mode\|AttemptCount`<br>• The Mode field will be one of AUTO_TIERING (for regular auto-tiering), BRIDGE_MODE (for Bridge Mode auto-tiering), or SCHEDULER (for retry attempts for Bridge Mode tiering of objects)<br>• The AttemptCount is applicable only to Bridge Mode tiering and associated retries, and indicates how many attempts have been made to tier the object. This field's value will be "-1" if the Mode is AUTO_TIERING. |
| Log Entry Example | `2021-04-23 05:57:35,914\|MOVEOBJECT\|S3\|tps01/abc801\|null\|tps02\|null\|33\|39\|ERR\|268776\|BRIDGE_MODE\|1` |
| Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br>Rotated files are named as *cloudian-tiering-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="TIER"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*Cross-Region Replication request log (cloudian-crr-request-info.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-crr-request-info.log*<br>**Note** Whichever S3 Service node processes a PUT of an object into a source bucket configured for CRR will be the node that initiates the replication of the object to the destination bucket. This node will have an entry for that object in its CRR request log. In the case of retries of replication attempts that failed with a temporary error the first time, the retries will be logged in the CRR request log on the cron job node. For general information on the cross-region replication feature, see **"Cross-Region Rep-** |

| | |
|---|---|
| | **lication Feature Overview"** (page 179). |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS\|SourceBucket/Object\|ObjectVersionId\|DestinationBucket\|CrrOperation\|Status\|DurationMillisecs\|Size`<br><br>The Status will be one of:<br>• COMPLETED -- The object was successfully replicated to the destination bucket.<br>• FAILED -- The object replication attempt resulted in an HTTP 403 or 404 response from the destination system. This is treated as a permanent error and no retry attempt will occur for replicating this object. This type of error could occur if for example the destination bucket has been deleted or if versioning has been disabled for the destination bucket. A FAILED status triggers an S3 service error Alert in the CMC's **Alerts** page.<br>• PENDING -- The object replication attempt encountered an error other than an HTTP 403 or 404 response. All other errors -- such as a connection error or an HTTP 5xx response from the destination system -- are treated as temporary errors. The object replication will be retried again once every four hours until either the object is successfully replicated to the destination bucket or a permanent error is encountered. Each retry attempt results in a new log entry in *cloudian-crr-request-info.log*. |
| Log Entry Example | `2020-03-04 15:04:58,423\|prod-2020-01-16/0299-0166180000362773.pdf\|fe15a1de-de25-f8af-9a28-b4a9fc08ca7e\|prod-backup-2020-01-16\|REPLICATEOBJECT\|COMPLETED\|61\|328471` |
| Logging Level | Not applicable |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *cloudian-crr-request-info.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="CRR"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*WORM audit log (s3-worm.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/s3-worm.log*<br><br>**Note** For information about the WORM feature see **"Object Lock Feature Overview"** (page 187). |
| Log Entry Format | `DateTime\|Hostname\|S3RequestId\|S3Operation\|Headers\|Bucket\|Object\|ObjectVersionId\|CanonicalUserId\|CanonicalIamUserId\|StatusCode\|` |

| | |
|---|---|
| | `StatusMessage`<br><br>The S3Operation will be any of:<br>• An object lock operation (*GET/PUT Bucket object lock configuration*or *GET/PUT Object legal hold* or *GET/PUT Object retention*)<br>• A regular S3 operation that includes object lock related headers (such as a *PUT Bucket* request that includes an *x-amz-object-lock-enabled: true* header or a *PUT Object* request that includes an *x-amz-object-lock-mode* or *x-amz-object-lock-retain-until-date* or *x-amz-object-lock-legal-hold* header). For such operations the relevant header(s) will be shown in the Headers field of the log entry. Multipart uploads are logged only if completed, and the operation is indicated as *PUT Object*.<br>• A *DELETE Object Version* request in regard to a locked object version.<br><br>For a regular S3 operation such as *PUT Bucket* or *PUT Object*, the Headers field will show the object lock related headers. For *GET/PUT Bucket object lock configuration* operations the Headers field will convey information about the bucket's default object lock configuration, coded as follows:<br>• First character: "T" for object lock is enabled on bucket<br>• Second character: "G" for Governance mode or "C" for Compliance mode<br>• Third character: "D" or "Y" for retention period unit of measurement (days or years)<br>• Remaining characters: Integer for number of days or years<br><br>Example 1: *TGD30* for a Governance mode configuration with 30 day retention period<br><br>Example 2: *T* for a bucket with object lock enabled, but no default object lock configuration<br><br>If the submitter of the request is an IAM user, the log entry shows the canonical user ID of the IAM user as well as the canonical user ID of the parent user account. |
| Log Entry Example | `2019-10-06 09:59:30,767\|arcturus\|a9d7e5b1-e85a-11e9-8519-52540014b047\|s3:GetBucketObjectLockConfiguration\|TGD90\|newbucket\|\|\|b584fb57480af5108e32d17f10c5cb7b\|\|200\|OK` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *s3-worm.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb*, this log is configurable in the block that starts with *<RollingRandomAccessFile name="S3WORM"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.1.12. SQS Service Logs

*SQS application log (cloudian-sqs.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-sqs.log*<br><br>**Note** For an SQS overview (including information about how to enable the SQS Service, which is disabled by default) see the SQS API section in the ***Cloudian HyperStore AWS APIs Support Reference***. |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE` |
| Log Entry Example | `2019-11-05 02:32:44,453  INFO[SQSRetentionCheckerThread - pool-5-thread-1] SQSMessageRetentionChecker:Another thread processing. Do nothing...` |
| Default Logging Level | INFO |
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *cloudian-sqs-req.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-sqs.xml.erb* , this log is configurable in the block that starts with *<RollingRandomAccessFile name="SQSAPP"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

*SQS request log (cloudian-sqs-request.log)*

| | |
|---|---|
| Location | On every node, */var/log/cloudian/cloudian-sqs-request.log*<br><br>**Note** For an SQS overview (including information about how to enable the SQS Service, which is disabled by default) see the SQS section in the ***Cloudian HyperStore AWS APIs Support Reference***. |
| Log Entry Format | `yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE`<br><br>The MESSAGE includes the client IP address, the user ID, and the request type (the SQS "action"). |
| Log Entry Example | `2019-07-11 13:50:21,736 INFO[qtp1548962651-122] RequestLogger:2019-07-11 13:50:21,732\|10.20.2.61\|user1\|CreateQueue\|testQueue\|200\|4` |
| Default Logging Level | INFO |

| | |
|---|---|
| Default Rotation and Retention Policy | Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.<br><br>Rotated files are named as *cloudian-sqs-req.log.YYYY-MM-DD.i.gz*, where *i* is a rotation counter that resets back to 1 after each day. Rotated files are compressed with *gzip*.<br><br>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB **or** if oldest rotated file age reaches 180 days. |
| Configuration | In the Puppet template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-sqs.xml.erb* , this log is configurable in the block that starts with *<RollingRandomAccessFile name="SQSREQ"*. For setting descriptions see **"Log Configuration Settings"** (page 373). |

## 8.2. Log Configuration Settings

The S3 Service, HyperStore Service, Redis Monitor, Admin Service, Monitoring Data Collector, Monitoring Agent, and CMC each have their own XML-formatted *log4j-*.xml.erb* configuration template in which you can adjust logging settings. Within a *log4j-*.xml.erb* file, specific logs -- such as the S3 application log and the S3 request log -- are configured by named instances of *RollingRandomAccessFile*. The **"HyperStore Logs"** (page 349) overview section indicates the specific *log4j-*.xml.erb*file and the specific *RollingRandomAccessFile* name by which each log is configured (for example the S3 application log is configured by the *RollingRandomAccessFile* instance named "S3APP" in the *log4j-s3.xml.erb* file).

**Note** After making any configuration file edits, be sure to **trigger a Puppet sync-up and then restart the affected service** (for example, the S3 Service if you've edited the *log4j-s3.xml.erb* file).

Within a particular log's *RollingRandomAccessFile* instance there are these editable settings:

- **PatternLayout pattern="<pattern>"** — The log entry format. This flexible formatting configuration is similar to the *printf* function in C. For detail see **PatternLayout** from the online Apache Log4j2 documentation.
- **TimeBasedTriggeringPolicy interval="<integer>"** — Roll the log after this many days pass. (More precisely, the log rolls after *interval* number of time units pass, where the time unit is the most granular unit of the date pattern specified within the *filePattern* element — which in the case of all HyperStore logs' configuration is a day). Defaults to rolling once a day if *interval* is not specified. All HyperStore logs use the default of one day.
- **SizeBasedTriggeringPolicy size="<size>"** — Roll the log when it reaches this size (for example "10 MB"). Note that this trigger and the *TimeBasedTriggeringPolicy* operate together: the log will be rolled if either the time based trigger **or** the size based trigger occur.
- **IfLastModified age="<interval>"** — When a rolled log file reaches this age the system automatically deletes it (for example "180d").
- **IfAccumulatedFileSize exceeds="<size>"** — When the aggregate size (after compression) of rolled log files for this log reaches this size, the system automatically deletes the oldest rolled log file (for example "100 MB"). Note that this setting works together with the *IfLastModified* setting -- old rolled log files will be deleted if either the age based trigger **or** the aggregate size based trigger occur.

> **Note** Each *RollingRandomAccessFile* instance also includes a *DefaultRolloverStrategy max="<integer>"* parameter which specifies the maximum number of rolled files to retain from a single day's logging. However, by default this parameter is not relevant for HyperStore because HyperStore logs are configured such that the *IfAccumulatedFileSize* trigger will be reached before the *DefaultRolloverStrategy* trigger.

For each log's default value for the settings above, see the **"HyperStore Logs"** (page 349) overview section.

In the *log4j-*.xml.erb* files, in addition to *RollingRandomAccessFile* instances there are also ***Logger*** instances. Each *Logger* instance contains an *AppenderRef* element that indicates which log that *Logger* instance applies to, by referencing the log's *RollingRandomAccessFile* name (for example *AppenderRef ref="S3APP"* means that the *Logger* instance is associated with the S3 application log). Note that multiple *Logger* instances may be associated with the same log — this just means that multiple core components of a service (for example, multiple components within the S3 Service) have separately configurable loggers. If you're uncertain about which *Logger* instance to edit to achieve your objectives, consult with Cloudian Support.

The *Logger* instances are where you can configure a logging level, using the *level* attribute:

- **level="<level>"** — Logging level. The following levels are supported (only events at the configured level and above will be logged):
  - OFF = Turn logging off.
  - ERROR = Typically a fail-safe for programming errors or a server running outside the normal operating conditions. Any error which is fatal to the service or application.
  - WARN = Anything that can potentially cause application oddities, but where the server can continue to operate or recover automatically. Exceptions caught in "catch" blocks are commonly at this level.
  - INFO = Generally useful information to log (service start/stop, configuration assumptions, etc). Info to always have available. Normal error handling, like a user not existing, is an example.
  - DEBUG = Information that is diagnostically helpful.
  - TRACE = Very detailed information to "trace" the execution of a request or process through the code.
  - ALL = Log all levels.

For each log's default log level, see the **"HyperStore Logs"** (page 349) overview section.

For information about how to aggregate HyperStore logs to your own centralized logging server see **"Aggregating Logs to a Central Server"** (page 122).

## 8.3. Using the HSH to View Logs

If you are using the **HyperStore Shell (HSH)** to manage your HyperStore nodes, the HSH supports a command for viewing HyperStore logs -- specifically, log files under the directory */var/log/cloudian*.

To use the HSH to view HyperStore log files, first log into the Configuration Master node (via SSH) as an HSH user. Upon successful login the HSH prompt will appear as follows:

```
<username>@<hostname>$
```

For example:

```
sa_admin@hyperstore1$
```

**To view a HyperStore log:**

```
$ hslog /var/log/cloudian/<logfilename>
```

**Note** You must include the file path */var/log/cloudian/* -- not just the log file name.

For example:

```
$ hslog /var/log/cloudian/cloudian-admin.log
```

The command *hslog /var/log/cloudian/<logfilename>* writes the log to *stdout*.

If you wish you can use command piping -- *hsslog /var/log/cloudian/<logfilename>* | *<command>* -- to pipe the output to *head* or *tail* or *less* or *grep*, for example.

For details about the logs you can view, see **"HyperStore Logs"** (page 349).

This page left intentionally blank

# Chapter 9. Configuration Settings

## 9.1. CMC's Configuration Settings Page

Through the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**) you can dynamically change a variety of HyperStore system configuration settings. For detail, while logged into the **Configuration Settings** page click **Help**.

## 9.2. Installer Advanced Configuration Options

The HyperStore installation tool supports several types of advanced system configurations which can be implemented at any time after initial installation of the system.

> **Note** As a best practice, you should complete basic HyperStore installation first and confirm that things are working properly (by running the installer's Validation Tests, under the "Cluster Management" menu) before you consider using the installer's advanced configuration options.

To access the advanced configuration options, on the Configuration Master node change into your installation staging directory (*/opt/cloudian-staging/7.5.2*) and launch the installer.

```
# ./cloudianInstall.sh
```

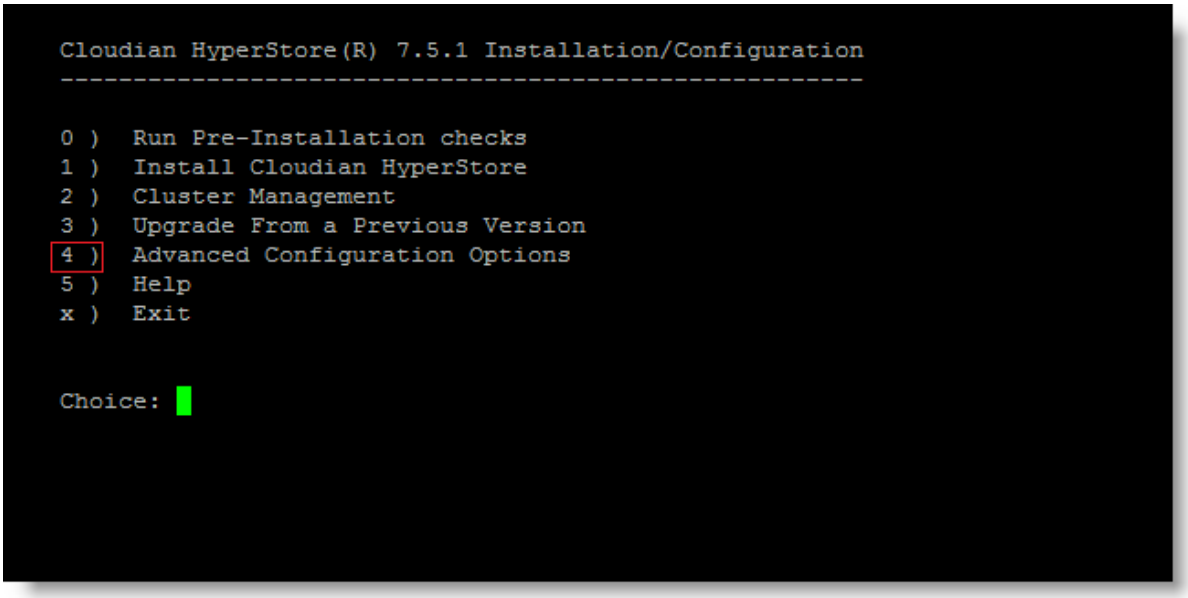
*If you are using the **HyperStore Shell***

If you are using the HyperStore Shell (HSH) as a Trusted user, from any directory on the Configuration Master node you can launch the installer with this command:

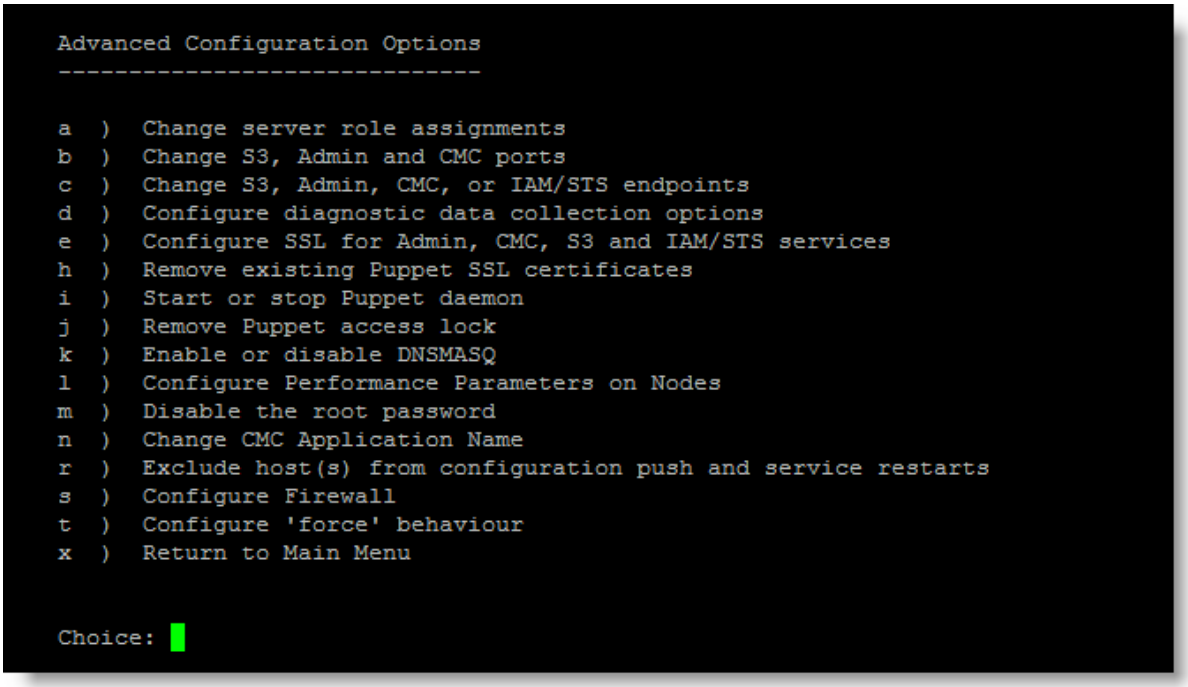
```
$ hspkg install
```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

At the installer main menu's Choice prompt enter **4** for Advanced Configuration Options.




This opens the "Advanced Configuration Options" sub-menu.




From this menu you can choose the type of configuration change that you want to make and then proceed through the interactive prompts to specify your desired settings.

## a) Change server role assignments

This is for shifting role assignments — such as the QoS DB Master role or Credentials DB Master role — from one of the hosts in your cluster to another. You can also change the list of external NTP servers through this menu option. For more information see **"Change Node Role Assignments"** (page 305). .

## b) Change S3, Admin or CMC ports

This lets you interactively change listening ports for the S3, Admin, and CMC services. For more information

see **"Changing S3, Admin, or CMC Listening Ports"** (page 498).

### c) Change S3, Admin, CMC, or IAM/STS endpoints

This lets you interactively change the S3 service endpoint, S3 static website endpoint, Admin service endpoint, CMC endpoint, or IAM service endpoint. For more information see **"Changing S3, Admin, CMC, or IAM Service Endpoints"** (page 498).

### d) Configure diagnostic data collection options

This lets you interactively configure Phone Home (Smart Support) settings.

1. After selecting this option from the Advanced Configuration Options menu, follow the prompts to specify your desired settings. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value. The final prompt will ask whether you want to save your changes -- type *yes* to do so.
2. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
3. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service.

### e) Configure SSL for Admin, CMC, S3 and IAM/STS services

These options are for managing SSL certificates in support of using HTTPS for the Admin, CMC, S3, and IAM/STS services. For more information see **"Security and Privacy Features"** (page 141).

### h) Remove existing Puppet SSL certificates

This is a troubleshooting measure in the event of Puppet connectivity problems, as described in the "Installation Troubleshooting" section of the ***Cloudian HyperStore Installation Guide***.

This operation is specifically in regard to Puppet certificates (used during cluster configuration management processes) and has nothing to do with SSL for the S3 Service.

### i) Start or stop Puppet daemon

By default, after HyperStore installation the Puppet master (Configuration Master) and Puppet agent daemons are left running, and every 10 minutes the agents check the master to see if there are updated HyperStore configuration settings to download to the agent nodes. Alternatively you can use this option from the Advanced menu to stop the Puppet agent daemons.

Note that if you don’t leave the Puppet agent daemons running, you must remember to trigger a one-time Puppet sync-up each time you make a HyperStore configuration change on the Puppet master. By contrast, if you leave the Puppet agent daemons running, a sync-up will happen automatically every 10 minutes even if you don’t specifically trigger a sync-up after making a configuration change.

If you ever want to check on the current status of your Puppet master and agent daemons, you can do so by choosing "Cluster Management" from the installer’s main menu; then choose "Manage Services"; then in the Service Management sub-menu choose "Puppet service (status only)".

### j) Remove Puppet access lock

When Puppet is performing a cluster configuration sync-up, the system temporarily locks Puppet access (so that no additional Puppet instances are launched by other operators or systems). If the sync-up operation is

terminated unexpectedly — such as by a <CTRL>-c command, or a Puppet master node shutdown — the temporary lock may fail to release. This unreleased lock would prevent any subsequent sync-ups from being implemented.

You can clear the lock by running the "Remove Puppet access lock" operation.

Afterward, to confirm that the lock is cleared you can check to make sure that no */tmp/cloudian.installer.lock* directory exists on the Puppet master node.

## k) Enable or disable DNSMASQ

This menu option is for either of these circumstances:

- When you ran the *cloudianInstall.sh* script to install your HyperStore system, you had the script automatically install and configure **dnsmasq** to perform DNS resolution for HyperStore service domains. (That is, you used the *configure-dnsmasq* option when you launched the install script.) However, now you want to use your own DNS solution for resolving HyperStore domains, and you've completed your DNS configuration as described in **"DNS Set-Up"** (page 50). You can use the installer's "Enable or disable DNSMASQ" menu option to disable dnsmasq. This stops *dnsmasq* on all HyperStore nodes and disables *dnsmasq* by configuration.
- When you ran the *cloudianInstall.sh* script to install your HyperStore system, you did **not** have it install *dnsmasq* (the default installer behavior is not to install *dnsmasq*). However, now you want to use *dnsmasq* for HyperStore domain resolution. With the installer's "Enable or disable DNSMASQ" menu option you can enable *dnsmasq*:
    1. After selecting this option from the Advanced Configuration Options menu, follow the prompts to choose to enable *dnsmasq*.
    2. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
    3. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the DNSMASQ service.

## l) Configure Performance Parameters on Nodes

This lets you run a HyperStore performance configuration optimization script. The script runs automatically when you install your cluster and when you add nodes, so under normal circumstances you should not need to use this installer Advanced Configuration option. For more information see **"Tuning HyperStore Performance Parameters"** (page 500).

## m) Disable the root password

For information on this option see **"Enabling the HSH and Managing HSH Users"** (page 96).

## r) Exclude host(s) from configuration push and service restarts

Use this option if you want to temporarily exclude a particular node or nodes from installer-driven configuration pushes and service restarts. One scenario where it would be appropriate to do this is if you have a node that's down and can't be brought back up anytime soon, and in the interim you want to make a configuration change to your system. With a node down, the installer's "Cluster Management" → "Push Configuration Settings to Cluster" function will fail if you try to push to the whole cluster. So before doing a push, use the "Advanced Configuration Options" → "Exclude host(s) from configuration push and service restarts" to specify the down node. Then, when you subsequently do a configuration push to the cluster, the installer will automatically exclude that node and the push to the cluster will succeed.

The excluded host will also be excluded when you use the installer's "Cluster Management" → "Manage Services" menu to stop, start, or restart particular services in the cluster (such as the S3 Service or the Cassandra Service).

> **Note** This "exclude from configuration push and service restarts" status is different than "maintenance mode". The "excluded" status merely excludes a node from the list of nodes that the installer uses when it pushes out configuration changes to the cluster or restarts services across the cluster; it has no effect other than that. (For more information on "maintenance mode" see Start Maintenance Mode).

If you put a node into "excluded" status, and you later exit the installer, then if you subsequently launch the installer again it will display a message indicating that there is a node in excluded status. In the sample below the node "cloudian-node6" is in this status.

```
# ./cloudianInstall.sh
The following nodes have been excluded for configuration updates and service
restarts: cloudian-node6
Press any key to continue ...
```

When the node is back up again and you are ready to have it again be eligible for installer-managed configuration pushes and service restarts, return to the "Advanced Configuration Options" → "Exclude host(s) from configuration push and service restarts" function and enter "none" at the prompt.

If you made a system configuration change when a node was down and excluded, then after the node is back up and you've taken the node out of excluded status, do a configuration push and a service restart (whichever service restart is appropriate to the configuration change you made). This will bring the node's configuration up to date with the rest of the cluster.

> **Note** There are a small number of circumstances where the system will **automatically** place a down node into the "excluded" status. One such circumstance is when the system is executing automatic failover of the system cronjob host role, in the event that the primary cronjob host goes down. The next time that you launch the installer it will display a message identifying the node that's in "excluded" status.

## s) Configure Firewall

This lets you enable and configure the built-in HyperStore firewall, on all the HyperStore nodes in your system. For more information see **"HyperStore Firewall"** (page 491).

## t) Configure 'force' behavior

If you specify the *force* option when running the installer on the command line, the *force* option will "stick" and will be used automatically for any subsequent times the installer is run to install additional nodes (such as when you do an "Add Node" operation via the Cloudian Management Console, which invokes the installer in the background). To turn the *force*option off so that it is no longer automatically used when the installer is run to add more nodes, launch the installer and go to the Advance Configuration Options. Then choose option **t** for **Configure 'force' behavior** and follow the prompts.

# 9.3. Pushing Configuration File Edits to the Cluster and Restarting Services

*Subjects covered in this section:*



## 9.3.1. Puppet Overview

During HyperStore installation, the open source version of the cluster configuration management application Puppet is automatically installed and set up. The Puppet framework consists of:

- A **Puppet master node** on which all the HyperStore configuration templates reside. This is one of your HyperStore nodes -- specifically, the node on which you ran the HyperStore installation script. In HyperStore this is called the **Configuration Master** node.
- A **Puppet agent on every node** in your HyperStore system (including the node on which the Master is running). In HyperStore these are called the **Configuration Agents**.

This cluster configuration management system enables you to edit HyperStore configuration templates in one location — on the Configuration Master node — and have those changes propagate to all the nodes in your HyperStore cluster, even across multiple data centers and multiple service regions. There are two options for implementing Puppet sync-up: you can trigger an immediate sync-up by using the HyperStore installer, or you can wait for an automatic sync-up which by default occurs on a 10 minute interval. With either approach, after the sync-up **you must restart the affected service(s) to apply the configuration changes**.

**IMPORTANT !** Do not directly edit configuration files on individual HyperStore nodes. If you make edits on an individual node, those local changes will be overwritten when the local Configuration Agent does its next sync-up with the Configuration Master.

**Note** To ensure high availability of the Configuration Master role, HyperStore automatically sets up a backup Configuration Master node and supports a method for manually failing over the Configuration Master role in the event of problems with the primary Configuration Master node. See **"Move the Configuration Master Primary or Backup Role"** (page 313)

## 9.3.2. Installation Staging Directory

During installation or upgrade of your HyperStore system, when you unpack the HyperStore product package on your Configuration Master node an installation staging directory is created. For HyperStore version 7.5.2, the installation staging directory is:

```
/opt/cloudian-staging/7.5.2
```

If you forget the location of your staging directory, you can find it displayed in the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**) toward the bottom of the "Service Information" section.

Among the important files in your installation staging directory is the HyperStore installation script *cloudianInstall.sh* -- also known as the HyperStore installer -- which you can use for a variety of purposes including pushing configuration changes out to the cluster and restarting services.

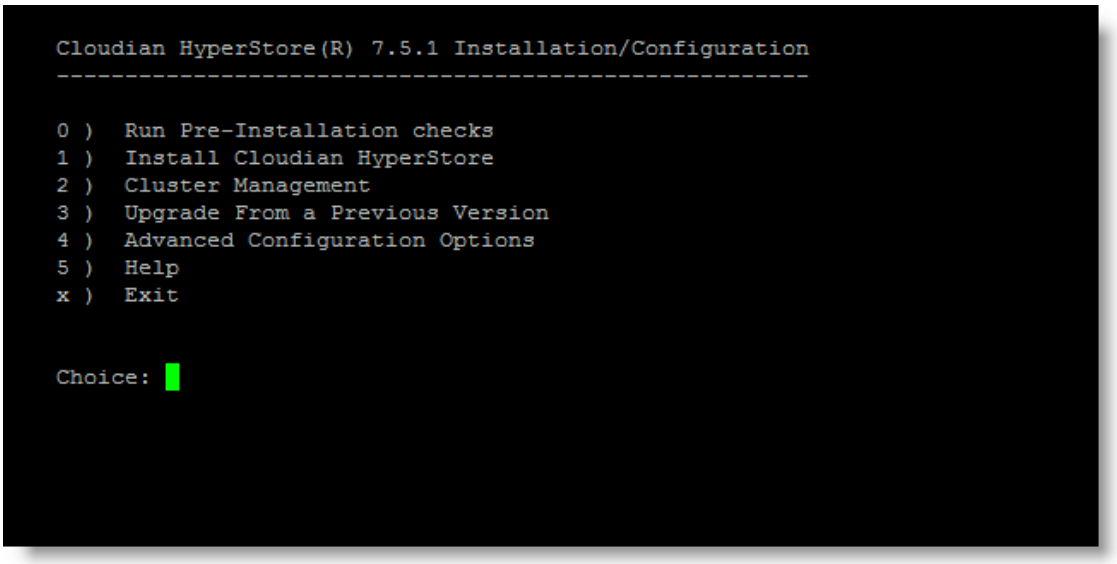
### 9.3.3. Using the Installer to Push Configuration Changes and Restart Services

After you've edited configuration file templates on the Configuration Master you can use the HyperStore installer to trigger a Puppet sync-up and then restart the affected service(s):

1. After logging into the Configuration Master node as *root*, change into your installation staging directory (*/opt/cloudian-staging/7.5.2*). Once in the staging directory, launch the HyperStore installer:

   ```
   # ./cloudianInstall.sh
   ```

   This displays the installer's main menu:

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. Enter "2" for Cluster Management. This displays the Cluster Management menu.

```
Cluster Management
------------------

a )  Review Cluster Configuration
b )  Push Configuration Settings to Cluster
c )  Manage Services
d )  Run Validation Tests
x )  Return to Main Menu


Choice:
```

3. Enter "b" for Push Configuration Settings to Cluster. You will then be prompted to list the hosts on which you want the Configuration Agents to sync up with the Configuration Master. The default is all your HyperStore hosts; for this you can just press enter at the prompt. In a multi-region system you are also given the option to sync-up only the agents in a particular region.
4. After the Puppet run completes for all the agents (and a success message displays on the console), **restart the affected service(s) to apply your configuration change**:
    a. From the Cluster Management menu, enter "c" for Manage Services. This displays the Service Management menu.

```
      Service Management
-------------------------------------
    0 ) All services
    1 ) Redis Credentials
    2 ) Redis QOS
    3 ) Cassandra
    4 ) HyperStore service
    5 ) S3 service
    6 ) Redis Monitor
    7 ) Cloudian Agent
    8 ) DNSMASQ
    9 ) Cloudian Management Console (CMC)
    10) IAM
    11) SQS
    P ) Puppet service (status only)
    X ) Quit

You can execute the following list of commands:
start,stop,status,restart,version,node-start,node-stop


Select a service to manage:
```

    b. From the Service Management menu, enter the number for the service to restart. The service to restart will depend on which configuration setting(s) you edited. For example:

| File in Which You Edited Setting(s) | Service to Restart |
|---|---|
| *hyperstore-server.properties.erb* | HyperStore Service |
| *mts-ui.properties.erb* | CMC |
| *mts.properties.erb* | Typically the S3 Service (but instead Cassandra in a small number of cases; see **mts.properties.erb** for the setting[s] that you changed) |
| *common.csv* | Depends on the specific setting(s) that you edited; see **common.csv** |

c. After entering the service to manage, enter "restart". Watch the console for messages indicating a successful stop and restart of the service. Note that the service restart occurs on each node on which the service is running.

## 9.3.4. Option for Triggering a Puppet Sync-Up from the Command Line

HyperStore supports an alternative way of using the installer to trigger a Puppet sync-up, directly from the command line. To use this method, run the installer like this:

```
# ./cloudianInstall.sh runpuppet="[<region>,<host>,<host>,...]"
```

This triggers a Puppet sync-up, and the sync-up progress displays in your terminal.

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can run the following command to trigger a Puppet sync-up:

```
$ hspkg install runpuppet="[<region>,<host>,<host>,...]"
```

Here are some examples for how to specify the *runpuppet* option:

- *runpuppet=""* — Sync-up all the agents in your whole HyperStore system.
- *runpuppet="region1"* — Sync-up only all the agents in region1.
- *runpuppet="region1,host1"* — Sync-up only host1 in region1.
- *runpuppet="region1,host1,host2"* — Sync-up only host1 and host2 in region1.

**Note** If you use this method, you still need to subsequently launch the installer in the normal way and then **restart the affected service(s)** as described in Step 4 from the **"Using the Installer to Push Configuration Changes and Restart Services"** (page 383) section, in order to apply your changes.

## 9.3.5. Excluding a Down Node from an Installer-Driven Configuration Push

If you use the installer to push a configuration change out to your cluster (by triggering a Puppet sync-up throughout the cluster) and the installer detects that a node is down or unreachable, the whole configuration push operation will fail. Therefore, if you want to make a system configuration change at a time when a node is down or unreachable you need to configure the installer to exclude that node from the cluster configuration push. For instructions on place a node into "excluded" status (and how to take a node out of this status), see **"r) Exclude host(s) from configuration push and service restarts"** (page 380).

**Note** There are a small number of circumstances where the system will **automatically** place a down node into the "excluded" status. One such circumstance is when the system is executing automatic fail-over of the system cronjob host role, in the event that the primary cronjob host goes down. The next time that you launch the installer it will display a message identifying the node that's in "excluded" status.

### 9.3.6. Automatic Puppet Sync-Up on an Interval

If you leave the Puppet master and Puppet agents running as background daemons (as is the default behavior after HyperStore installation), the Puppet agents on all of your up HyperStore nodes will automatically check every 10 minutes to see if changes have been made to the HyperStore configuration templates on the master node. If configuration changes have been made, each the Puppet agent will download those changes to its local host machine.

Note however that this automatic Puppet sync-up **does not apply the configuration changes to the currently running services**. Applying the configuration changes requires a **service restart**.

To apply your changes, first wait long enough to be sure that the automatic Puppet sync-up has occurred (the default is every 10 minutes). Then, restart the affected service(s) by logging into your Configuration Master node, changing into the installation staging directory, launching the installer, then going to the Cluster Management menu and restarting the affected service(s) as described in Step 4 in the **"Using the Installer to Push Configuration Changes and Restart Services"** (page 383) section.

## 9.4. Using the HSH to Manage Configuration Files

If you are using the **HyperStore Shell (HSH)** to manage your HyperStore nodes, the HSH supports commands for viewing and editing HyperStore configuration files. To use the HSH to view or edit HyperStore configuration files, first log into the Configuration Master node (via SSH) as an HSH user. Upon successful login the HSH prompt will appear as follows:

```
<username>@<hostname>$
```

For example:

```
sa_admin@hyperstore1$
```

**Note** To use the HSH to manage configuration files you must be an **HSH Trusted user**.

**To view the list of configuration files that you can view and edit using the HSH**:

To see the complete list of configuration files that you can view and edit in the HSH run this command:

```
$ hspkg config --help
```

The list includes all the files covered in the "HyperStore Configuration Files" section of this documentation as well as a few additional system configuration files.

**Note** Although *cloudian-crontab.erb* is listed, using the HSH you can only view this file -- not edit it.

**To view a configuration file using the HSH**:

```
$ hspkg config <filename>
```

Specify just the configuration file name (such as *common.csv*), not the full path to the file.

In the background this invokes the Linux command *less* to display the configuration file. Therefore you can use the standard keystrokes supported by *less* to navigate the display; for example:

- ***f*** key or Space bar -- Page down
- ***b*** key -- Page up
- Down arrow key or Enter -- Go down one line
- Up arrow key -- Go up one line
- <n>***f*** key or <n>Space bar -- Go down <n> number of lines
- <n>***b*** key -- Go up <n> number of lines
- */string* Enter-- Search down for the specified string
- *?string* Enter -- Search up for the specified string
- ***q*** key -- Quit the file display and return to the HSH prompt

**To edit a configuration file using the HSH**:

```
$ hspkg config -e <filename>    (or $ hspkg config --edit <filename>)
```

Specify just the configuration file name (such as *common.csv*), not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the standard keystrokes supported by *vi* to make and save changes to the file; for example:

- Up or down arrow keys -- Move cursor up or down one line
- Left or right arrow keys -- Move cursor left or right one character
- ***i*** key -- Start insert mode (to actually make edits to file)
- Escape key -- End insert mode and return to command mode (so that in command mode you can save or discard your changes)
- ***:w*** key combination -- In command mode, save changes and keep the file open so you can keep working on it
- ***:wq*** key combination -- In command mode, save changes, close the file, and return to the HSH prompt
- ***:q!*** key combination -- In command mode, discard changes, close the file, and return to the HSH prompt
- ***:q*** key combination -- In command mode, close the file and return to the HSH prompt (appropriate only if you made no changes to the file)

**Note** For more information about using the *vi* text editor, see any reputable online source.

If you made and saved a change to a configuration file, to apply the change you must use the installer to push the change out to the cluster and restart the relevant service(s). From the HSH you can launch the installer as follows:

```
$ hspkg install
```

For more information about using the installer to push your configuration change and restart services, see **"Using the Installer to Push Configuration Changes and Restart Services"** (page 383).

# 9.5. common.csv

The *common.csv* file is the main HyperStore configuration file and under typical circumstances this is the only configuration file that you may want to edit. On the Configuration Master node the path to the file is:

```
/etc/cloudian-7.5.2-puppet/manifests/extdata/common.csv
```

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can edit this configuration file with this command:

```
$ hspkg config -e common.csv
```

Specify just the configuration file name, not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the **standard keystrokes supported by *vi*** to make and save changes to the file.

All *common.csv* settings that require environment-specific customization are automatically pre-configured by the install script, based on information that you provided during the installation process. Making any further customizations to this configuration file is optional.

**IMPORTANT !** If you make any edits to *common.csv*, be sure to push your edits to the cluster and restart the affected services to apply your changes. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

Also, if you make changes to...
* any listening port settings
* the *cloudian_user* setting
* the *iam_service_enabled* setting or the *sqs_enabled* setting
... then before using the installer to push the change to the cluster, at the installer main menu you must enter **2** for Cluster Management then **a** for Review Cluster Configuration, then enter **yes** when asked if you want to update the Puppet server with your changes. After you've done so you can push your change to the cluster and restart the affected service(s).

## 9.5.1. common.csv Details

### 9.5.1.1. Settings in common.csv

#### release_version

Current HyperStore release version.

Default = 7.5.2

Do not edit.

#### cloudian_license_file_name

Name of your Cloudian license file.

Default = Set during installation based on operator input.

Do not edit manually. To apply an updated license file, use the CMC's Update License function (**Cluster → Cluster Config → Cluster Information → Update License**). That function will automatically update this configuration setting as appropriate and dynamically apply the change to your live system. No service restart is necessary.

### default_region

The default **service region** for your S3 service. Must be one of the regions listed for the regions setting. In a multi-region HyperStore system, the default region plays several roles in the context of S3 storage bucket LocationConstraint functionality. For example:

- For PUT Bucket requests that lack a CreateBucketConfiguration element in the request body, the bucket will be created in the default region.
- For PUT Bucket requests that do include a CreateBucketConfiguration element (with LocationConstraint attribute), the PUT requests typically resolve to the default region, and S3 Service nodes in the default region then initiate the process of creating the bucket in the requested region.

If your HyperStore system has only one service region, make that region the default region.

Default = Set during installation based on operator input.

Do not change this setting after your system is installed and running. If for some reason you need to change which region is your default region, please consult with Cloudian Support.

### regions

Comma-separated list of **service regions** within your HyperStore system. Region names must be lower case with no dots, dashes, underscores, or spaces. Even if you have only one region, you must give it a name and specify it here.

Default = Set during installation based on operator input.

### javahome

Home directory of Java on your HyperStore nodes.

Default = Path to OpenJDK (set automatically during installation)

### java_minimum_stack_size

Memory stack size to use for the HyperStore system's Java-based services (Cassandra, S3 Service, HyperStore Service, Admin Service)

Default = 256k

> **Note** Optionally you can override this stack size value on a per-server-type basis by adding any of the following settings to the configuration file and assigning them a value:
>
> *cassanda_stack_size*
> *cloudian_s3_stack_size*
> *cloudian_hss_stack_size*
> *cloudian_admin_stack_size*

### installation_root_directory

Directory in which HyperStore service packages will be physically installed.

Default = /opt/cloudian-packages

### run_root_directory

Root directory in which HyperStore services will be run. Links will be created from this directory to the physical installation directory.

Default = /opt

### pid_root_directory

Root directory in which HyperStore service PID (process ID) files will be stored. A */cloudian* sub-directory will be created under this root directory, and the PID files will be stored in that sub-directory.

Default = /var/run

### cloudian_user

User information for the user as which to run Cloudian HyperStore services, in format *<user_name>,<group_name>,<optional_numeric_UID>,<login_shell>*. If you want this user to be something other than the default, edit this setting and also the *cloudian_runuser* setting.

Default ="cloudian,cloudian,,/bin/bash"

### cloudian_runuser

User name of the user as which to run Cloudian HyperStore services. This must match the first field from the *cloudian_user* setting. If you edit the *cloudian_runuser* setting you must also edit the first field from the *cloudian_user* setting.

Default = cloudian

### user_bin_directory

Directory in which certain user-invocable scripts are stored.

Default = /usr/local/bin

### user_home_directory

Directory under which to create the home directory of the HyperStore services runtime user. The system will append the *cloudian_runuser* value to the *user_home_directory* value to get the full home directory path. For example, with "cloudian" as the *cloudian_runuser* and "/export/home" as the *user_home_directory*, the Cloudian user's home directory is "/export/home/cloudian".

Default = /export/home

### cloudian_userid_length

Maximum number of characters allowed in a HyperStore user ID. The highest value you can set this to is 256.

> **Note** This maximum applies also to the user full name. For example if this is set to 64, then when you are creating a user through the CMC or the Admin API the user ID can be a maximum of 64 characters long, and also the user full name can be a maximum of 64 characters long.

Default = 64

To apply change, after Puppet propagation restart S3 Service and CMC

### service_starts_on_boot

Whether to have HyperStore services start on host reboot, true or false.

Default = true

### service_restart_on_failure

Whether to automatically restart a service if the service crashes (stops with a non-zero exit code -- a failure code). When enabled, this feature is implemented for each node individually. For example, if *service_restart_on_failure* is set to *true*, and if the S3 Service crashes on particular node, then the S3 Service on will be automatically restarted on that node.

The first automatic attempt to restart a crashed service will occur 30 seconds after the crash. If four attempts to automatically restart a service on a node fail within a 10 minute interval, no more automatic restart attempts of that service on that node will occur. (When you later manually start the service again, this resets the automatic restart mechanism, so that subsequently it is ready to work again if necessary.)

This setting applies only to the S3 Service, the Admin Service, the HyperStore Service, and the Cassandra Service (Metadata DB). Automatic restart is not supported for other services.

Default = false

> **Note** If you change this setting to *true*, **push your change to the cluster and then restart the S3 Service, the HyperStore Service, and the Cassandra Service**. Restarting the S3 Service also restarts the Admin Service.

### cloudian_log_directory

Directory into which to write application logs for the S3 Service, Admin Service, HyperStore Service, Redis Monitor, Cloudian performance monitoring Agent, and Cloudian performance monitoring Data Collector.

Default = /var/log/cloudian

### cleanup_directories_byage_withmatch

The *cleanup_directories_byage_** settings configure Puppet to automatically delete certain files from your HyperStore nodes after the files reach a certain age. The *cleanup_directories_byage_withmatch* setting is a comma-separated list of directories in which to look for such files.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = "/var/log/cloudian,/tmp,/var/log/puppetserver,/opt/tomcat/logs"

### cleanup_directories_byage_withmatch_timelimit

The *cleanup_directories_byage_** settings configure Puppet to automatically delete certain files from your HyperStore nodes after the files reach a certain age. The *cleanup_directories_byage_withmatch_timelimit* setting specifies the age at which such files will be deleted (based on time elapsed since file creation). The age can be specified as *<x>m* or *<x>h* or *<x>d* where *<x>* is a number of minutes, hours, or days.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = 15d

### cleanup_directories_byage_matches

The *cleanup_directories_byage_** settings configure Puppet to automatically delete certain files from your HyperStore nodes after the files reach a certain age. The *cleanup_directories_byage_matches* setting specifies the file types to delete.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = "diagnostics*.gz,diagnostics*.tgz,hiq_metrics*.gz,jna*.tmp,liblz4-java*.so,snappy-*.so,*.cloudian-bak,cloudian_system_info*.tar.gz,puppetserver*.log.zip,localhost*.log,localhost_access_log*.txt,catalina*.log"

### cleanup_sysinfo_logs_timelimit

Retention period for Node Diagnostics packages. After a package reaches this age, Puppet will automatically delete the package.

A Node Diagnostics package is created under a */var/log/cloudian/cloudian_sysinfo* directory on a node if you use the Collect Diagnostics feature for that node. For general information on Node Diagnostics see **"Smart Support and Diagnostics Feature Overview"** (page 108).

This cleanup feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior).

Specify this value as a number of days, hours, or minutes, with the value formatted as *<n>d* or *<n>h* or *<n>m* respectively (such as 15d or 12h or 30m).

Default = 15d

> **Note** When you use the CMC to collect diagnostics on a node (**Cluster -> Nodes -> Advanced**), the UI gives you the option to have the diagnostics package automatically uploaded to Cloudian Support, and also the option to have the system delete the package immediately after it's been successfully uploaded to Cloudian Support. The retention period set by *cleanup_sysinfo_logs_timelimit* comes into play only if you do not use the "upload and then immediately delete" options.

### path_style_access

Whether the CMC (and also the installer's basic validation test script) should use "path style" request formatting when submitting S3 requests to the HyperStore S3 Service. In path style S3 requests, the bucket name is part of the Request-URI rather than being part of the Host header value.

Options are:

- *true* — The CMC will use path style HTTP request formatting when submitting S3 requests to the HyperStore S3 Service. The bucket name associated with the request will be in the Request-URI. For example:

```
PUT /bucket1/objectname HTTP/1.1
```

```
Host: s3-region1.mycompany.com
```

- *false* — The CMC will not use path style HTTP request formatting when submitting S3 requests to the HyperStore S3 Service. Instead it will use "virtual host" style access. The bucket name associated with the request will be in the HTTP Host header. For example:

```
PUT /objectname  HTTP/1.1
Host: bucket1.s3-region1.mycompany.com
```

Note that this setting affects only the behavior of the CMC and the behavior of the installer's basic validation test script, in their role as S3 clients. Meanwhile the HyperStore S3 Service always supports both path style access and virtual host style access.

**IMPORTANT !** If the CMC or any other S3 client applications use virtual host style access to the HyperStore S3 Service, then your DNS environment must be configured to resolve this type of Host value. See **"DNS Set-Up"** (page 50).

Default = true

To apply change, after Puppet propagation restart CMC

### fips_enabled

For information about this setting see **"FIPS Support"** (page 168).

### sshdconfig_disable_override

**Note** This setting is relevant only to software-only deployments of HyperStore that were first installed as HyperStore version 7.2.4 or later. It does not apply to HyperStore appliances, or older HyperStore software deployments for which the upgrade path included versions 7.2.0, 7.2.1, 7.2.2, or 7.2.4. For such systems HyperStore has already managed the *sshd_config* file and the *sshdconfig_disable_override* setting should be left at its default of *false*.

If this setting is left at its default value of *false* when you install HyperStore then on each HyperStore host your existing */etc/ssh/sshd_config* file will be appended with a HyperStore-managed section of SSH settings. This HyperStore-managed section of the file allows HyperStore to implement the *fips_enabled* setting, if you set *fips_enabled* to true; and it also allows HyperStore Shell users to log in to nodes, if you **enable the HyperStore Shell (HSH)**.

If you set *sshdconfig_disable_override* to *true* before executing the HyperStore installation then HyperStore will not append the existing */etc/ssh/sshd_config* file on HyperStore host machines. **In this case you will not be able to use the *fips_enabled* setting or use the HyperStore Shell.**

For detail about when to make this edit within the context of the installation procedure, see "Installing a New HyperStore System" in the ***Cloudian HyperStore Installation Guide***.

Default = false

To apply a change to this setting on an already installed system (a change that should not be needed in typical circumstances), on the Configuration Master node run this command:

```
hsctl config apply ssh
```

**Note** *hsctl* is a new node management tool that remains mostly behind the scenes in HyperStore 7.5.2 but will be more prominent in future HyperStore releases. You can run the *hsctl* command above from any directory.

### hyperstore_data_directory

A quote-enclosed, comma-separated list of mount points to use for S3 object storage. In a production environment, use dedicated disks for S3 object storage. Do not use the same disk(s) that are storing the OS and Cassandra.

The system will automatically assign virtual nodes (vNodes) to each of your S3 data mount points, in a manner that allocates an approximately equal **total token range** to each mount point.

Do not change the *hyperstore_data_directory* setting once your cluster is operational.

Do not use symbolic links when specifying your mount points for the *hyperstore_data_directory* setting. The HyperStore system does not support symbolic links for these directories.

Example of a multiple mount point configuration = "/cloudian1,/cloudian2,/cloudian3,/cloudian4"

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cloudian

### hyperstore_listen_ip

The IP interface on which each HyperStore Service node listens for data operations requests from clients. This setting must match the *cassandra_listen_address* setting.

**Specify this as an IP address alias**. Puppet will use the alias to determine the actual IP address for each node.

Options are %{::cloudian_ipaddress}, %{::ipaddress_eth#}, %{::ipaddress_lo}, or %{::ipaddress_bind#}

Default = %{::cloudian_ipaddress}

To apply change, after Puppet propagation restart HyperStore Service

### hyperstore_timeout

For the S3 Service's connections to the HyperStore Service, the transaction completion timeout (session timeout) in milliseconds.

Default = 10000

To apply change, after Puppet propagation restart S3 Service

For a diagram showing the place of this timeout within the S3 request processing flow, see the description of *mts.properties.erb*: **"cassandra.cluster.CassandraThriftSocketTimeout"** (page 442).

### hyperstore_connection_timeout

For the S3 Service's connections to the HyperStore Service, the connection establishment timeout in milliseconds.

Default = 10000

To apply change, after Puppet propagation restart S3 Service

For a diagram showing the place of this timeout within the S3 request processing flow, see the description of *mts.properties.erb*: **"cassandra.cluster.CassandraThriftSocketTimeout"** (page 442).

### hyperstore.maxthreads.repair

Maximum number of simultaneous client threads for one S3 Service node to use on HyperStore File System data repairs automatically performed during read operations. For more information on the "repair on read" mechanism see Automated Data Repair Feature Overview.

Default = 50

### hyperstore_jetty_minThreads

Each HyperStore Service node maintains a thread pool to process incoming HTTP requests from clients (S3 Service nodes). Idle threads are terminated if not used within a timeout period — unless the number of threads in the pool is down to the required minimum pool size, in which case the idle threads are kept.

The *hyperstore_jetty_minThreads* parameter sets the minimum number of threads to keep in the thread pool.

Default = 100

### auto_repair_computedigest_run_number

This property configures the **scheduled auto-repair feature** such that every *Nth* run of **hsstool repair** (for replicated object data) and **hsstool repairec** (for erasure coded object data) will use the "-computedigest" option in order to detect and repair any data corruption on disk ("bit rot"). For example, if this property is set to 3, then on each node every 3rd run of *repair* will use the "-computedigest" option and for each data center every 3rd run of *repairec* will use the "-computedigest" option.

By default the auto-repair interval for *repair* is 30 days, and each individual node has its own every-30-days repair schedule. So if for example you set *auto_repair_computedigest_run_number* to 3, then on a given node the automatically triggered *repair* runs would be implemented like this:

- Day 0: *repair* without "-computedigest"
- Day 30: *repair* without "-computedigest"
- Day 60: *repair* with "-computedigest"
- Day 90: *repair* without "-computedigest"
- Day 120: *repair* without "-computedigest"
- Day 150: *repair* with "-computedigest"
- etc

By default the auto-repair interval for *repairec* is 29 days. With erasure coded data repair, running *hsstool repairec* on any one node repairs all the erasure coded data in the local data center. Consequently the auto-repair feature runs the command on just one randomly selected node in each data center every 29 days.

So if for example you set *auto_repair_computedigest_run_number* to 3, then for a given data center the automatically triggered *repairec* runs would be implemented like this:

- Day 0: *repairec* without "-computedigest"
- Day 29: *repairec* without "-computedigest"
- Day 58: *repairec* with "-computedigest"
- Day 87: *repairec* without "-computedigest"
- Day 116: *repairec* without "-computedigest"

- Day 145: *repairec* with "-computedigest"
- etc

Setting *auto_repair_computedigest_run_number* to 1 would result in all auto-repair runs using "-computedigest".

By default *auto_repair_computedigest_run_number* is set to 0, which disables using "-computedigest" for auto-repair runs. **So by default no auto-repair runs will use the "-computedigest" option.**

Default = 0

> **Note** Because it entails recalculating a fresh MD5 hash of each replica or erasure coded fragment on the target node, using "-computedigest" on repair runs is an expensive operation in terms of resource utilization.

> **Note** The *auto_repair_computedigest_run_number* setting has no impact on *hsstool repair* or *hsstool repairec* runs that you manually execute on a node. The setting only impacts the auto-repair feature.

### hyperstore.maxthreads.write

Maximum number of simultaneous client threads for one S3 Service node to use on writes to the HyperStore File System.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore.maxthreads.read

Maximum number of simultaneous client threads for one S3 Service node to use on reads of the HyperStore File System.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_maxperrouteconnections

The maximum allowed number of concurrently open connections between each S3 Service node and each HyperStore Service node. This allows for limiting the traffic load between each front-end S3 Service node (as it processes incoming requests from S3 clients) and any single HyperStore Service node.

Note that each of your S3 Service nodes has its own pool of connections to the HyperStore storage layer, so the total possible connections from the S3 Service as a whole to a single HyperStore Service node would be the number of S3 Service nodes multiplied by the value of "Max Connections from One S3 Service Node to One HyperStore Service Node".

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_maxtotalconnections

The maximum allowed number of concurrently open connections between each S3 Service node and all HyperStore Service nodes, combined. This allows for limiting the traffic load between each front-end S3 Ser-

vice node (as it processes incoming requests from S3 clients) and the whole back-end HyperStore storage layer.

Note that each of your S3 Service nodes has its own pool of connections to the HyperStore storage layer, so the total possible connections from the front-end S3 Service as a whole to the back-end HyperStore storage layer as a whole would be the number of S3 Service nodes multiplied by the value of "Max Connections from One S3 Service Node to All HyperStore Service Nodes".

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_jetty_maxThreads

Each HyperStore Service node maintains a thread pool to process incoming HTTP requests from clients (S3 Service nodes). When there is a request to be serviced, a free thread from the pool is used and then returned to the pool afterward. If a thread is needed for a job but no thread is free, a new thread is created and added to the pool — unless the maximum allowed number of threads in the pool has been reached, in which case queued jobs must wait for a thread to become free.

The *hyperstore_jetty_maxThreads* parameter sets the maximum number of threads to allow in the thread pool.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_messaging_service_threadpool

Maximum size of the thread pool used by the HyperStore inter-node messaging service. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_repair_session_threadpool

Maximum size of the thread pool used for **hsstool repair** or **hsstool repairec** operations. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_repair_digest_index_threadpool

Maximum size of the thread pool used for reading file digests on a node in order to build the index required by Merkle Tree based repair (the default **hsstool repair** type). When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_rangerepair_threadpool

Maximum size of the thread pool used for running multiple range repair tasks in parallel during **hsstool repair**. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_stream_outbound_threadpool

Maximum size of the thread pool used for streaming files from one HyperStore node to another during Merkle Tree based **hsstool repair**. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_downloadrange_session_threadpool

Maximum size of the thread pool used for range download sessions conducted by a HyperStore Service node. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_uploadrange_session_threadpool

Maximum size of the thread pool used for range upload sessions conducted by a HyperStore Service node. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_decommission_threadpool

Maximum size of the thread pool used for uploading files away from a node that is being decommissioned. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_cleanup_session_threadpool

This thread pool size limit determines the maximum number of blobs (object replicas or erasure coded fragments) to process in parallel within each cleanup "job" taking place on a node. Processing a blob entails checking the blob's corresponding object metadata to determine whether the blob is supposed to be where it is or rather should be deleted.

The maximum number of "jobs" running in parallel is set by **"cleanupjobs.threadpool.corepoolsize"** (page 436) in *hyperstore-server.properties.erb*.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### hyperstore_auto_repair_threadpool

Maximum size of the thread pool used by the HyperStore auto-repair feature. When there is an auto-repair to kick off and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

> **Note** For more information about the auto-repair feature see Automated Data Repair Feature Overview.

### hyperstore_repairec_sessionscan_threadpool

During an **hsstool repairec** operation, the threads in this thread pool are tasked with identifying erasure coded objects in the system and batching them for evaluation. This parameter sets the maximum size of the thread pool per node.

Default = 50

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECSessionScanThreadPoolCorePoolSize)

### hyperstore_repairec_digestrequest_threadpool

During an **hsstool repairec** operation, the threads in this thread pool are tasked with reading the digests associated with batches of erasure coded objects, to determine whether any of those objects need repair. This parameter sets the maximum size of the thread pool per node.

Default = 30

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECDigestRequestThreadPoolFixedPoolSize)

### hyperstore_repairec_task_threadpool

During an **hsstool repairec** operation, the threads in this thread pool are tasked with repairing erasure coded objects that have been determined to need repair. This parameter sets the maximum size of the thread pool

per node.

Default = 60

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECTaskThreadPoolCorePoolSize)

### hyperstore_repairec_rocksdbscan_threadpool

During an **hsstool repairec** operation, the threads in this thread pool enable concurrent reads of digest data in each **RocksDB** instance, as digest read requests come in from multiple digest request threads on multiple nodes. This parameter sets the maximum size of the thread pool per RocksDB instance.

Default = 30

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECRocksDBScanThreadPoolFixedPoolSize)

### hyperstore_disk_check_interval

The interval (in minutes) at which each HyperStore Service node will run a check to see if there is significant imbalance in disk usage on the node. If an imbalance is found in excess of that configured by *disk.balance.delta*, then one or more tokens are automatically moved from the over-used disk(s) to one or more less-used disk(s) on the same host. For more information see **"Automated Disk Usage Management Feature Overview"** (page 113).

Default = 4320 (72 hours)

To apply change, after Puppet propagation restart the HyperStore Service

> **Note** This feature applies only to HyperStore data disks (on which are stored S3 object data). It does not apply to disks that are storing only the OS and Cassandra.

### phonehome_proxy_host

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the hostname or IP address of the proxy.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

> **Note** For this feature you should configure your forward proxy to support access to **.s3-support.cloudian.com* (that is, to any sub-domain of *s3-support.cloudian.com*).

> **Note** By default any proxy settings that you configure for the daily Smart Support uploads will also apply to the sending of on-demand Node Diagnostics packages (triggered by your using the CMC's Collect Diagnostics function [**Cluster -> Nodes -> Advanced**]). If you want to use a different proxy for Node Diagnostics sending than you do for the daily Smart Support upload, use the **sysinfo.proxy.*** settings in *mts.properties.erb* to separately configure proxy information for Node Diagnostics sending.

For more background information on these features see **"Smart Support and Diagnostics Feature Overview"** (page 108).

### phonehome_proxy_port

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the proxy's port number.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

### phonehome_proxy_username

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the username that HyperStore should use when connecting to the proxy (if a username and password are required by the proxy).

Default = empty

### phonehome_proxy_password

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the password that HyperStore should use when connecting to the proxy (if are username and password are required by the proxy).

Default = empty

### phonehome_uri

S3 URI to which to upload system-wide diagnostics data each day. By default this is the S3 URI for Cloudian Support.

If you set this to a different S3 destination, include the HTTP or HTTPS protocol part of the URI (*http://* or *https://*).

Default = https://s3-support.cloudian.com:443

To apply change, after Puppet propagation restart the S3 Service

**Note** If you set *phonehome_uri* to a URI for your own HyperStore S3 Service (rather than the Cloudian Support URI), and if your S3 Service is using HTTPS, then your S3 Service's SSL certificate must be a CA-verified certificate — not a self-signed certificate. By default the phone home function cannot upload to an HTTPS URI that's using a self-signed certificate. If you require that the upload go to an HTTPS URI that's using a self-signed certificate, contact Cloudian Support for guidance on modifying the phone home launch script. For information on HTTPS set-up for the S3 Service, see **"Security and Privacy Features"** (page 141).

### phonehome_bucket

- If you leave *phonehome_uri* at its default value -- which is Cloudian Support S3 URI -- you can leave the *phonehome_bucket*, *phonehome_access_key*, and *phonehome_secret_key* properties empty. The Smart Support feature will automatically extract the Cloudian Support S3 bucket name and security

credentials from your encrypted HyperStore license file.

- If you set *phonehome_uri* to an S3 URI other than the Cloudian Support URI, set the *phonehome_bucket*, *phonehome_access_key*, and *phonehome_secret_key* properties to the destination bucket name and the applicable S3 access key and secret key.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

### phonehome_access_key

See **"phonehome_bucket"** (page 401).

Default = empty

To apply change, after Puppet propagation restart the S3 Service

### phonehome_secret_key

See **"phonehome_bucket"** (page 401).

Default = empty

To apply change, after Puppet propagation restart the S3 Service

### phonehome_gdpr

For description of how to use this setting, see **"Encrypting Sensitive Fields in Uploaded Log File Copies to Protect Data Privacy"** (page 111).

Default = false

### phonehome_gdpr_bucket

For description of how to use this setting, see **"Encrypting Sensitive Fields in Uploaded Log File Copies to Protect Data Privacy"** (page 111).

Default = false

### logcollect_enable

This setting is applicable only if Smart Support is enabled for your HyperStore system, as it is by default (*mts.properties: phonehome.enabled=true*).

If Smart Support is enabled in your system and *logcollect_enable* is set to *true*, then after you open a support case with Cloudian Support, the system will automatically upload to Cloudian Support any logs that Cloudian Support needs to process your case.

> **Note** Your **GDPR settings** will be applied to these automatic uploads.

If *logcollect_enable* is set to *false*, then when you open a support case with Cloudian Support, Cloudian Support will ask you to collect and upload from your HyperStore system any logs that Cloudian Support needs to process your case (which you can do as described in Collect Diagnostics).

Default = false

To apply change, after Puppet propagation restart the S3 Service.

### admin_auth_user

If the Admin Service is configured to require HTTP(S) Basic Authentication from clients (if **"admin_auth_enabled"** (page 403) is set to *true*), this is the username for clients to use when submitting HTTP(S) requests to the Admin Service.

Default = sysadmin

To apply change, after Puppet propagation restart S3 Service and CMC

### admin_auth_pass

If the Admin Service is configured to require HTTP(S) Basic Authentication from clients (if **"admin_auth_enabled"** (page 403) is set to *true*), this is the password for clients to use when submitting HTTP(S) requests to the Admin Service. In this setting the password is configured as a comma-separated pair of "<Jetty_obfuscated_password>,<cleartext_password>".

Do not use special characters in the cleartext password.

For information about creating a Jetty-obfuscated password (based on a cleartext password that you provide), see the Introduction section in the ***Cloudian HyperStore Admin API Reference***.

Default if original HyperStore install was version 7.2.2 or newer = "<obfuscated>,<cleartext>" of a random password generated by the system upon installation.

Default if original HyperStore install was older than version 7.2.2 = "1uvg1x1n1tv91tvt1x0z1uuq,public"

To apply change, after Puppet propagation restart S3 Service and CMC

### admin_auth_realm

If the Admin Service is configured to require HTTP(S) Basic Authentication from clients (if **"admin_auth_enabled"** (page 403) is set to *true*), this is the realm name used by the Admin Service for Basic Authentication purposes.

Default = CloudianAdmin

To apply change, after Puppet propagation restart S3 Service and CMC

### admin_auth_enabled

Whether to have the Admin Service require HTTP(S) Basic Authentication from connecting clients. Set to *true* to have the Admin Service require Basic Authentication, or *false* to not require it.

Default if original HyperStore install was version 6.0.2 or newer = true

Default if original HyperStore install was older than version 6.0.2 = false

To apply change, after Puppet propagation restart S3 Service

### admin_secure

If set to "true", the Admin Service will accept **only** HTTPS connections from clients (through port 19443 by default). If set to "false", the Admin Service will allow regular HTTP connections from clients (through port 18081) as well as HTTPS connections (through port 19443).

This setting also controls CMC client-side behavior when the CMC calls the Admin Service for tasks such as creating users, creating storage policies, and retrieving system monitoring data. If *admin_secure* is "true", the

CMC will exclusively use HTTPS when making requests to the Admin Service. If *admin_secure* is "false", the CMC will exclusively use regular HTTP when making requests to the Admin Service.

Note however that even if you set *admin_secure* to "false" -- so that the Admin Service accepts HTTP requests as well as HTTPS requests; and so that the CMC sends only HTTP requests to the Admin Service -- the Admin Service's HTTPS port will still be accessed by other HyperStore system components. In particular, some of the **"System cron Jobs"** (page 321) use HTTPS to make calls to the Admin Service.

Default = true

> **Note** If your original HyperStore install was **older than version 6.0.2** and you have upgraded to the current version, the *admin_secure* setting does not appear in the *common.csv* file and an internal default value of "false" is used. In such systems, if you want the Admin Service to accept only HTTPS connections from clients, add the line *admin_secure,true* to the *common.csv* file.

To apply change, after Puppet propagation restart S3 Service and CMC

### cmc_admin_secure_port

If the CMC is using HTTPS to connect to the Admin Service (as it will if *admin_secure* is set to "true"), this is the Admin Service listening port number to which it will connect. Note that this setting controls CMC client-side configuration, not Admin Service configuration.

Default = 19443

To apply change, after Puppet propagation restart CMC.

### user_password_min_length

For users' CMC passwords, the minimum required length in number of characters. The system will reject a user's attempt to set a new password that does not meet this requirement.

Default = 9

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_dup_char_ratio_limit

When a CMC user creates a new password, no more than this percentage of characters in the new password can be characters that are in the user's current password. The system will reject a user's attempt to set a new password that does not meet this requirement.

If this is set to 0, then none of the characters in a user's new password can be characters that are in the user's current password.

If this is set to -1, then this password requirement is disabled.

Default = -1

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_unique_generations

When a CMC user creates a new password, the new password cannot be the same as one of the user's past passwords, going back this many passwords into the user's password history. The system will reject a user's attempt to set a new password that does not meet this requirement.

For example if you set this to 10, then a user's new password cannot match against any of the user's past 10 passwords.

If this is set to 0, then this password requirement is disabled.

Default = 0

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_rotation_graceperiod

When a CMC user creates a new password, they must wait at least this many days before replacing that password with another new password.

If this is set to 0, then this password requirement is disabled.

Default = 0

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_rotation_expiration

After a CMC user has had the same password for this many days, the password expires and the CMC will require the user to create a new password before they can log in again. This will be enforced by the CMC's login function.

If this is set to 0, then this password requirement is disabled.

Note that if you change this setting, the lifespan of each user's **existing** password is considered to have started **when the password was created**. For example, if you change this setting's value from 0 to 90, then all users whose existing passwords are already older than 90 days old will be required to change their passwords the next time that they log into the CMC.

Default = 0

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_lock_enabled

This setting controls whether or not the CMC password lock feature is enabled:

- If *true*, the CMC password lock feature is enabled and its behavior is configured by the **"user_password_lock_durationsec"** (page 406) and **"user_password_lock_maxfailedattempts"** (page 406) settings.

  > **Note** The CMC password lock feature applies only to users for which local authentication (based on a user ID and password defined within HyperStore) is performed, not users whose CMC login attempts are authenticated by reference to an LDAP system. For users authenticated by an LDAP system you can configure a password lockout policy within the LDAP system if desired.

- If *false*, the CMC password lock feature is disabled.

Default = *true* for fresh installations of HyperStore 7.3 or later; *false* for systems upgraded to HyperStore 7.3 from an earlier HyperStore version.

To apply change, after Puppet propagation restart the S3 Service and CMC.

### user_password_lock_durationsec

Applicable only if *user_password_lock_enabled* is set to *true*.

If within an interval of *user_password_lock_durationsec* seconds, a user tries *user_password_lock_maxfailedattempts* number of times to log in to the CMC with an incorrect password, then the system locks out that user for an interval of *user_password_lock_durationsec* seconds. During this lock-out interval the system will not allow the user to log into the CMC even if the user supplies the correct password. If during the lock-out interval the user again attempts to log in with an incorrect password, the lock-out interval starts over. After the lock-out interval expires the user will be allowed to log into the CMC if the user supplies the correct password.

For example, if *user_password_lock_durationsec* is set to 1800 (the default) and *user_password_lock_maxfailedattempts* is set to 6 (the default), then the system will lock out a user who tries 6 times within a 30 minute period to log into the CMC with an incorrect password. The lock-out will last 30 minutes, starting from the 6th invalid login attempt. During the 30 minute lock-out period:

- If the user tries again to log in with an incorrect password, this restarts the 30 minute lock-out period.
- If the user does not try again to log in with an incorrect password, the lock-out period automatically expires at the end of the 30 minutes, and the user is then allowed to log in if they supply the correct password.
- Optionally, the system administrator can intervene to release the lock on a user -- before the lock reaches its automatic expiration -- by using the CMC (**Users & Groups -> Manage Users**; while on that page click **Help** for details) or the Admin API (see the "user" section of the ***Cloudian HyperStore Admin API Reference***).

Default = 1800

To apply change, after Puppet propagation restart the S3 Service and CMC.

> **Note** If a system administrator becomes locked out of the CMC -- due to too many login attempts within an incorrect password -- the lock on that administrator can be released by a different system administrator using the CMC, or by the Admin API method mentioned above. Alternatively, as with any type of locked out user the lock-out will release automatically after the configurable time interval.

### user_password_lock_maxfailedattempts

Applicable only if *user_password_lock_enabled* is set to *true*.

For a description of the *user_password_lock_maxfailedattempts* setting see **"user_password_lock_durationsec"** (page 406). These two settings work in combination.

Default = 6

To apply change, after Puppet propagation restart the S3 Service and CMC.

### iam_service_enabled

If this is set to "true" then HyperStore's IAM Service is enabled and IAM functionality will display in the CMC. For more information about the IAM API see the IAM section of the ***Cloudian HyperStore AWS APIs Support Reference***. For more information about IAM in the CMC, while on any of the pages in the CMC's **IAM** section click Help.

Default = true

To apply change, after Puppet propagation restart the IAM Service.

### iam_port

Port on which the HyperStore IAM Service listens for regular HTTP connections.

Default = 16080

To apply change, after Puppet propagation restart the IAM Service.

### iam_secure

If set to "true", the IAM Service will accept **only** HTTPS connections from clients (through port 16443 by default). If set to "false", the IAM Service will allow regular HTTP connections from clients (through port 16080) as well as HTTPS connections (through port 16443).

This setting also controls CMC client-side behavior when the CMC calls the IAM Service for tasks such as creating IAM or creating IAM policies. If *iam_secure* is "true", the CMC will exclusively use HTTPS when making requests to the IAM Service. If *iam_secure* is "false", the CMC will exclusively use regular HTTP when making requests to the IAM Service.

Default = false

To apply change, after Puppet propagation restart the IAM Service and the CMC.

### iam_secure_port

Port on which the HyperStore IAM Service listens for HTTPS connections.

Default = 16443

To apply change, after Puppet propagation restart the IAM Service.

### iam_service_endpoint

IAM Service endpoint. This setting is controlled by the installer. Do not edit this setting directly. For instructions on changing the IAM Service endpoint, see **"Changing S3, Admin, CMC, or IAM Service Endpoints"** (page 498).

Default = Set during installation based on operator input

### iam_max_groups

The maximum number of IAM groups allowed per Cloudian account root.

Default = 300

To apply change, after Puppet propagation restart the IAM Service.

> **Note** The maximum number of **IAM users** per Cloudian account root is 5000. This limit is not configurable.

### iam_max_groups_per_user

The maximum number of IAM groups that an IAM user can belong to at a time. If an IAM user belongs to this many groups, the IAM Service will not allow her to be joined to any more groups.

**IMPORTANT !** The more IAM groups an IAM user belongs to, the more complex and time-consuming will be the operation to assess the various IAM policies that apply to the user when the user submits an S3 request (to determine whether the user has permission for that request). Therefore, exercise caution in raising the value of this setting.

Default = 10

To apply change, after Puppet propagation restart the IAM Service.

### mfa_enforced

This setting controls whether or not Multi-Factor Authentication is mandatory for CMC users:

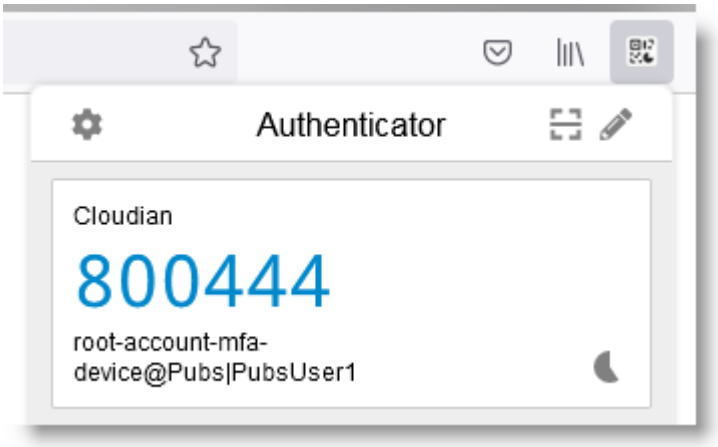
- If *mfa_enforced* is set to *true*, then when a user tries to log into the CMC, the CMC will prompt them to set up MFA and will present them an interface for doing so. Before using the CMC interface to set up MFA the user must install a virtual MFA device -- a software application that generates MFA codes -- for the browser from which they will be accessing the CMC. This can be any MFA application that supports the open TOTP (time-based one-time password) standard, also known as RFC 6238. Examples of free MFA applications include Google Authenticator and Microsoft Authenticator. Once the user has installed a virtual MFA device and set up MFA for the CMC, the CMC will log them out and then to log back in they will need to provide an MFA authentication code. Subsequently, whenever they log into the CMC they will need to provide MFA authentication codes. For more information on MFA set-up, in the CMC Help use the browse panel to navigate to **Cloudian Management Console (CMC) -> My Account -> Security Credentials**.
    - For already existing CMC users, setting up MFA will be required the next time they try to log into the CMC, after you've set *mfa_enforced* to *true* and applied the change.
    - For new users created after you've applied the change, setting up MFA will be required the first time they try to log into the CMC.
    - If *mfa_enforced* is set to *true*, the requirement to use MFA for CMC logins applies to system admin users (including the default system admin user named 'admin') as well as the group admins and regular users.
- If *mfa_enforced* is set to *false*, then MFA is optional for CMC logins. CMC users can optionally set up MFA for their CMC accounts, as described in  in the CMC Help (**Cloudian Management Console (CMC) -> My Account -> Security Credentials**). But users will not be required to set up and use MFA.

Default = false

To apply change, after Puppet propagation restart the CMC Service.

### mfa_totp_issuer

Users can enable multi-factor authentication (MFA) on their CMC login accounts (see **"CMC Multi-Factor Authentication"** (page 166)). The *mfa_totp_issuer* setting specifies the MFA TOTP (time-based one-time password) Issuer name that will display in a user's virtual MFA device if the user enables MFA on their CMC account. The example below is from the virtual MFA device Google Authenticator, with the Issuer at its default value "Cloudian".




Default = Cloudian

To apply change, after Puppet propagation restart the IAM Service.

### cloudian_s3admin_min_threads

Minimum number of threads to keep in each Admin Service node's HTTP request processing thread pool.

The Admin Service uses a thread pool to process incoming HTTP requests from clients. An initial pool of threads is created at server initialization time, and additional threads may be added if needed to process queued jobs. Idle threads are terminated if not used within a thread timeout period — unless the number of threads in the pool is down to *cloudian_s3admin_min_threads*. If only this many threads are in the pool, then threads are kept open even if they've been idle for longer than the thread timeout.

Default = 10

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3admin_max_threads

Maximum number of threads to allow in the Admin Service's HTTP request processing thread pool. If there are fewer than this many threads in the pool, new threads may be created as needed in order to handle queued HTTP request processing jobs. If the maximum thread pool size is reached, no more threads will be created — instead, queued HTTP request processing jobs must wait for an existing thread to become free.

Default = 50

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3admin_max_idletime

When the Admin Service processes HTTP requests from clients, the maximum allowed connection idle time in milliseconds. If this much idle time passes before a new request is received on an open connection with a client, or if this much idle time passes during the reading of headers and content for a request, or if this much idle time passes during the writing of headers and content of a response, the connection is closed.

Default = 60000

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3admin_lowres_maxidletime

Special, "low resource" maximum idle time to apply to Admin Service HTTP connections when the number of simultaneous connections to an Admin Service node exceeds *cloudian_s3admin_lowres_maxconnections*. Configured in milliseconds. With this setting, you can have the Admin Service be less tolerant of connection idle time during times of high concurrent usage. (For general idle timer behavior, see the description of *cloudian_s3admin_max_idletime* above.)

Default = 5000

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3admin_lowres_maxconnections

If the number of simultaneous HTTP connections to an Admin Service node exceeds this value, the special idle timer configured by *cloudian_s3admin_lowres_maxidletime* is applied to that node rather than the usual *cloudian_s3admin_max_idletime*.

Default = 1000

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_max_threads

Maximum number of threads to allow in the S3 Service's HTTP request processing thread pool. If there are fewer than this many threads in the pool, new threads may be created as needed in order to handle queued HTTP request processing jobs. If the maximum thread pool size is reached, no more threads will be created — instead, queued HTTP request processing jobs must wait for an existing thread to become free.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### cloudian_s3_max_idletime

When the S3 Service processes HTTP requests from clients, the maximum allowed connection idle time in milliseconds. If this much idle time passes before a new request is received on an open connection with a client, or if this much idle time passes during the reading of headers and content for a request, or if this much idle time passes during the writing of headers and content of a response, the connection is closed.

Default = 60000

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_lowres_maxidletime

Special, *low resource* maximum idle timer to apply to S3 Service HTTP connections when the number of simultaneous connections to an S3 Service node exceeds *cloudian_s3_lowres_maxconnections*. Configured in milliseconds.

Default = 5000

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_lowres_maxconnections

If the number of simultaneous HTTP connections to an S3 Service node exceeds this value, the special idle timer configured by *cloudian_s3_lowres_maxidletime* is applied to that node.

Default = 2000

To apply change, after Puppet propagation restart S3 Service

### cloudian_tiering_useragent

User agent string used by HyperStore when it acts as an S3 client for auto-tiering to an external S3 system.

Default = "APN/1.0 Cloudian/1.0 HyperStore/"

To apply change, after Puppet propagation restart the S3 Service

### s3_proxy_protocol_enabled

If you are using **HAProxy** as the load balancer in front of your S3 Service -- or a different load balancer that supports the **PROXY Protocol** -- you can set *s3_proxy_protocol_enabled* to *true* if you want the S3 Service to support the PROXY Protocol. If you do so, the following will happen:

- The S3 Server will create dedicated PROXY Protocol connectors listening on port 81 (for regular PROXY Protocol) and port 4431 (for PROXY Protocol with SSL/TLS). These connectors will be enabled and configured in *s3.xml.erb*. By default these connectors are disabled that template.
- If you configure your load balancer to use the PROXY Protocol for communicating with the S3 Service, the load balancer when relaying each S3 request to the S3 Service will pass along the originating client's IP address.

  > **Note** For guidance on load balancer configuration consult with your Cloudian Sales Engineering or Professional Services representative.

- In the **S3 request log**, the S3 request entries will then show the true client IP address as the source address, rather than showing the loader balancer's IP address as the source. Also, the true client IP address for each request will be available to support using S3 bucket policies that filter based on source IP address; or billing rating plans that "allowlist" certain source IP addresses.

Setting *s3_proxy_protocol_enabled* to *true* is appropriate if you're using HAProxy for load balancing -- or a different load balancer that supports the PROXY Protocol -- and you want S3 Service request logging to show the true origin address associated with S3 requests; and/or you want to implement bucket policies or rating plans that are responsive to the origin address. If you're using a load balancer that supports PROXY Protocol, using this protocol is the preferred method for providing the originating client IP address to the S3 layer, rather than using the *X-Forwarded-for* header.

> **Note** If you intend to use the PROXY Protocol with TLS/SSL (on S3 Service listening port 4431) you must set up TLS/SSL for the S3 Service, if you have not already done so. For instructions see **"Security and Privacy Features"** (page 141).
>
> If *s3_proxy_protocol_enabled* is set to *true* then when you configure TLS/SSL for the S3 Service your configuration information will be applied to PROXY Protocol port 4431 as well as to the regular S3 HTTPS port 443.

Default = false

To apply change, after Puppet propagation restart the S3 Service

### cloudian_s3_heap_limit

Maximum heap size limit for the S3 Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_s3_max_heap_percent*.

Default = 30g

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_max_heap_percent

Maximum heap size for the S3 Service application, as a percentage of host system RAM. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_s3_heap_limit*.

Default = 15

To apply change, after Puppet propagation restart S3 Service

### cloudian_hss_heap_limit

Maximum heap size limit for the HyperStore Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_hss_max_heap_percent*.

Default = 30g

To apply change, after Puppet propagation restart HyperStore Service

### cloudian_hss_max_heap_percent

Maximum heap size for the HyperStore Service application, as a percentage of host system RAM. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_hss_heap_limit*.

Default = 15

To apply change, after Puppet propagation restart HyperStore Service

### cloudian_admin_heap_limit

Maximum heap size limit for the Admin Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_admin_max_heap_percent*.

Default = 16g

To apply change, after Puppet propagation restart S3 Service

### cloudian_admin_max_heap_percent

Maximum heap size for the Admin Service application, as a percentage of host system RAM. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_admin_heap_limit*.

Default = 5

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the S3 Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart S3 Service

### cloudian_hss_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the HyperStore Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart HyperStore Service

### cloudian_admin_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the Admin Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart S3 Service

### cloudian_s3_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the S3 Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart S3 Service

### cloudian_hss_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the HyperStore Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart HyperStore Service

### cloudian_admin_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the Admin Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart S3 Service

### cloudian_heapdump_on_out_of_memory

Whether to enable Java heap dump on an S3 Service or HyperStore Service or Monitoring Data Collector out-of-memory error. Dump paths are */var/log/cloudian/s3.hprof* and */var/log/cloudian/hss.hprof* and */var/-log/cloudian/datacollector.hprof*, respectively. The dump file is in HPROF binary format.

Default =true

To apply change, after Puppet propagation restart S3 Service and HyperStore Service

### s3_bucket_logging_internalnetworks

This setting applies to the S3 bucket logging feature, whereby a bucket owner can choose to have access logs for that bucket regularly saved into that bucket or a different bucket owned by that user. For more information on this feature see Enable/Disable Bucket Logging.

The *s3_bucket_logging_internalnetworks* setting gives you as the system administrator the ability to mask certain fields in bucket log entries for S3 requests that originate from within your internal network(s). This would include for example S3 requests that originate from the CMC, or from other hosts within your internal network (s). This prevents users of the bucket logging feature from seeing information that you might consider sensitive in their bucket logs. In particular the bucket log entry fields that can be masked are:

- Remote IP
- User-Agent

For each internal network (specified in CIDR format) that you list in the *s3_bucket_logging_internalnetworks* setting, when S3 requests originate from within those networks the Remote IP and User-Agent field values in bucket log entries will be replaced by a dash ("-").

Example of a correctly formatted setting value:

```
s3.bucket.logging.internalnetworks = 150.5.2.0/24, 150.5.3.0/24
```

> **Note** To mask the Remote IP and User-Agent fields for S3 requests that originate from the **CMC**, include 127.0.0.0/8 in the list of internal networks (since the CMC usually uses *localhost* when submitting requests to the S3 Service).

Default = empty

To apply change, after Puppet propagation restart S3 Service

### cassandras_per_s3node

Maximum number of Cassandra nodes to which an individual S3 Service node may keep simultaneous live connections.

Default = 9

To apply change, after Puppet propagation restart S3 Service

### cassandra_max_active

The maximum allowed number of simultaneously active connections in a Cassandra connection pool. If this limit has been reached and a thread requires a new connection to Cassandra, the thread will wait for a period configured by *cassandra.cluster.MaxWaitTimeWhenExhausted* in *mts.properties.erb* (default 9 seconds) before returning an error to the client.

If this is set to a negative value (e.g. -1) this disables the limit on active connections.

This setting is controlled by a HyperStore performance optimization script that takes the local hardware environment into account. **Do not manually edit this setting** -- your edits will not be applied by the system.

### cloudian_s3_aes256encryption_enabled

This setting is for enabling AES-256 in HyperStore, for use with server-side encryption. If AES-256 is not enabled, AES-128 is used instead.

Default = false

To apply change, after Puppet propagation restart the S3 Service.

### cloudian_s3_autoinvalidateiamcache

When set to *true*, each region's cache of IAM user information (including IAM group membership and IAM policy-based permissions) is automatically cleared out every six hours, as well as cache data being updated whenever there is a change to IAM user information.

When set to *false*, each regions cache of IAM user information is not automatically cleared out every six hours, and instead the cache data is only updated whenever there is a change to IAM user information. This is the preferable setting if you have a multi-region HyperStore system with high latency between regions. The *false* setting reduces the need for the HyperStore S3 Service in non-default regions to make calls to the default region to retrieve IAM information when IAM users submit S3 requests to the non-default regions.

Default = true

To apply change, after Puppet propagation restart S3 Service and IAM Service

### s3_perbucket_qos_enabled

This setting enables/disables the keeping of per-bucket stored-bytes counts (the number of net bytes -- excluding replication or erasure coding overhead -- stored in each bucket) and stored-objects counts (the number of objects in each bucket) in the system.

If you set *s3_perbucket_qos_enabled* to *true*, then:

- The system will keep track of stored-bytes and stored-objects counts for each bucket. These counts will be kept in the QoS DB.

  > **Note** In some atypical circumstances the keeping of these per-bucket stored-bytes and stored-object counts, which will be updated after every S3 transaction, may cause a minor-to-moderate decline in S3 write performance. Example circumstances would be if there are an exceptionally large number of buckets in your HyperStore system, or if you have a multi-data center HyperStore deployment with more than usual latency between the data centers. If you want to enable per-bucket stored-bytes and stored-objects counters in your system but are concerned about potential performance impacts, consult with Cloudian Support.

- You will be able to query the current stored-bytes and stored-objects counts for individual buckets, by using the Admin API calls *GET /system/bytecount* and *GET /system/objectcount*. You will also be able to retrieve the byte and object counts for all buckets owned by all users in a specified group, by using the Admin API call *GET /system/bucketusage*. For more information on this Admin API calls see the "system" section of ***Cloudian HyperStore Admin API Reference***.
- If you use Cloudian's monitoring and visualization product **HyperIQ**, HyperIQ will be able to report stored-bytes and stored-objects counts for individual buckets.
- In the CMC, the **object listing page for a bucket** will display the bucket's current stored-bytes and stored-objects counts.

**IMPORTANT !** After setting *s3_perbucket_qos_enabled* to *true*, pushing the change out to the cluster, and restarting the S3 Service, execute the Admin API call *POST /usage/repair?groupId=ALL* to bring the counters up to date for buckets that already have objects in them. For more information about this Admin API call see the "usage" section of the ***Cloudian HyperStore Admin API Reference***.

Subsequently, the counters will automatically be updated for each bucket each time there is an S3 transaction that impacts the byte count or object count for the bucket.

If you leave *s3_perbucket_qos_enabled* at its default value *false*, then per-bucket stored-bytes and stored-objects counts will not be maintained in the QoS DB; you will not be able to query per-bucket counts through the Admin API calls referenced above; HyperIQ will not be able to report on per-bucket stored-bytes and stored-objects counts; and the CMC's object listing page for a bucket will not show stored-bytes and stored-objects counts.

Default = false

To apply change, after Puppet propagation restart the S3 Service.

### redis_credentials_master_port

Port on which the Redis Credentials master node listens for data storage requests from clients. The Redis Credentials slave nodes will listen on this port as well.

Default = 6379

To apply change, after Puppet propagation restart Redis Credentials, S3 Service, and HyperStore Service

### redis_monitor_subscription_check

This setting controls certain aspects of HyperStore services start-up behavior, including during a HyperStore version upgrade operation. Leave this setting at its default value.

Default = false

### redis_qos_master_port

Port on which the Redis QoS master node listens for data storage requests from clients. The Redis QoS slave nodes will listen on this port as well.

Default = 6380

To apply change, after Puppet propagation restart Redis QoS, S3 Service, and HyperStore Service

### redis_monitor_listener_port

Port on which the Redis Monitor can be queried for information about the Redis cluster state, via the Redis Monitor CLI.

Default = 9078

To apply change, after Puppet propagation restart Redis Monitor Service

### redis_lib_directory

Directory in which to store Redis data files.

Default = /var/lib/redis

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

### redis_log_directory

Directory in which to store Redis log files.

Default = /var/log/redis

To apply change, after Puppet propagation restart Redis Credentials and Redis QoS

### cassandra_max_heap_size

Max Heap size setting (memory allocation) for the Cassandra application. If this setting is assigned a value, this value will be used for *MAX_HEAP_SIZE* in *cassandra-env.sh*. By default the *cassandra_max_heap_size* setting is commented out and not used by the system. Instead, the default behavior for Cloudian HyperStore is for *MAX_HEAP_SIZE* in *cassandra-env.sh* to be automatically set based on the host's RAM size.

Default = commented out and unused

To apply change, after Puppet propagation restart Cassandra Service

### cassandra_enable_gc_logging

Whether to enable Java garbage collection (GC) logging for Cassandra. The log is written to */var/-log/cassandra/gc.log*.

Default = true

To apply change, after Puppet propagation restart Cassandra Service

> **Note** GC logging is also enabled for the S3 Service, Admin Service, and HyperStore Service. These GC logs are under */var/log/cloudian* and are named *s3-gc.log*, *admin-gc.log*, and *hss-gc.log*, respectively.

### cassandra_heapdump_on_out_of_memory

Whether to enable Java heap dump on a Cassandra out-of-memory error. Dump path is */var/-log/cassandra/cassandra.hprof*. The dump file is in HPROF binary format.

Default = true

To apply change, after Puppet propagation restart Cassandra Service

### cassandra_lib_directory

Directory in which to store Cassandra application state data.

Default = /var/lib/cassandra

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

### cassandra_log_directory

Directory in which to store Cassandra application log files.

Default = /var/log/cassandra

To apply change, after Puppet propagation restart Cassandra Service

### cassandra_saved_cache_directory

Directory in which to store the Cassandra saved_caches file.

Default = /var/lib/cassandra

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

### cassandra_commit_log_directory

Directory in which to store the Cassandra commit log file.

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cassandra_commit

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

### cassandra_data_directory

Directory in which to store Cassandra data files. By default, Cassandra is used only for storing S3 object metadata (metadata associated with individual objects) and service metadata such as account information, usage data, and system monitoring data.

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cassandra/data

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

**IMPORTANT !** The Cassandra data directory should **not** be mounted on a shared file system such as a NAS device.

### cassandra_port

Port on which Cassandra listens for data operations requests from clients (such as the S3 Service, the Admin Service, and the CMC all of which make calls to the Cassandra Service)

Default = 9160

To apply change, after Puppet propagation restart Cassandra Service

### concurrent_compactors

Number of simultaneous Cassandra compactions to allow, not including validation "compactions" for anti-entropy repair. Simultaneous compactions can help preserve read performance in a mixed read/write work-load, by mitigating the tendency of small sstables to accumulate during a single long running compaction.

Default = 2

### cassandra_tombstone_warn_threshold

If while processing a Cassandra query it is found that a single row within a column family has more than this many tombstones (deleted data markers), a tombstone warning is logged in the **Cassandra application log**.

An example of query that can potentially encounter a high number of tombstones is a metadata query triggered by an S3 Get Bucket (List Objects) operation.

Default = 50000

To apply change, after Puppet propagation restart Cassandra

### cassandra_tombstone_failure_threshold

If while processing a Cassandra query it is found that a single row within a column family has more than this many tombstones (deleted data markers), the query fails and a tombstone error is logged in the Cassandra application log.

An example of query that can potentially encounter a high number of tombstones is a metadata query triggered by an S3 Get Bucket (List Objects) operation.

Default = 100000

To apply change, after Puppet propagation restart Cassandra

### cassandra_tombstone_cleanup_threshold

If while processing a Cassandra query it is found that a single row within a CLOUDIAN_METADATA or MPSession column family has more than this many tombstones (deleted data markers), a tombstone purge process is automatically triggered for that column family.

An example of query that can potentially encounter a high number of tombstones is a metadata query triggered by an S3 Get Bucket (List Objects) operation.

Default = 75000

To apply change, after Puppet propagation restart the S3 Service

> **Note** You can also manually trigger a tombstone purge for a specific bucket, as described in **"Tombstone Cleanup Processing"** (page 325).

### cassandra_tombstone_gcgrace

While doing a purge of tombstones (deleted data markers) in a CLOUDIAN_METADATA or MPSession column family, the system will not purge tombstones that are fewer than this many seconds old.

If this is set to 0, then no tombstones are exempted from the purge.

Default = 0

To apply change, after Puppet propagation restart the S3 Service

### cassandra_listen_address

For each Cassandra node, the IP interface on which the node will listen for cluster management communications from other Cassandra nodes in the cluster. **Specify this as an IP address alias**. Puppet will use the alias to determine the actual IP address for each node.

Options are %{::cloudian_ipaddress}, %{::ipaddress_eth#}, %{::ipaddress_lo}, or %{::ipaddress_bind#}

Default = %{::cloudian_ipaddress}

To apply change, after Puppet propagation restart Cassandra Service

### cassandra_rpc_address

For each Cassandra node, the IP interface on which the node will listen for data operations requests from clients, via Thrift RPC. **Specify this as an IP address alias**. Puppet will use the alias to determine the actual IP address for each node.

Options are %{::cloudian_ipaddress}, %{::ipaddress_eth#}, %{::ipaddress_lo}, or %{::ipaddress_bind#}

If desired, this can be the same IP address alias as used for *cassandra_listen_address*.

Default = %{::cloudian_ipaddress}

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

### cassandra_default_node_datacenter

This instance of this setting is not used. Instead, the instance of *cassandra_default_node_datacenter* in *region.csv* is used. Typically you should have no need to edit that setting.

Default = commented out

### admin_whitelist_enabled

Whether to enable the billing "allowlist" feature that allows favorable billing terms for a specified list of source IP addresses or subnets. If this feature is enabled, allowlist management functionality displays in the CMC. This functionality is available only to HyperStore system administrators, not to group admins or regular users.

Default = true

To apply change, after Puppet propagation restart S3 Service and CMC

### allow_delete_users_with_buckets

If this is set to *true*, then a user who owns buckets can be deleted via the CMC or the Admin API, and the system will not only delete the user but will also **automatically delete the user's buckets and all the data in those buckets**. This includes buckets and data belonging to any IAM users who have been created under the user account root. The deleted data will not be recoverable.

If this is set to *false*, then the system will not allow a user who owns buckets to be deleted via the CMC or the Admin API. Instead, the user or an administrator must first delete all buckets owned by the user -- via the CMC or a different S3 client application -- including any buckets belonging to IAM users under the user account root. Only after all such buckets are deleted can the user then be deleted via the CMC or the Admin API.

Default = *true* for systems originally installed as 7.2.x or earlier; *false* for systems originally installed as 7.3 or later.

To apply change, after Puppet propagation restart S3 Service

> **Note** Regarding an administrator's ability to delete a user's buckets through the CMC, see the setting *common.csv:* **"cmc_view_user_data"** (page 424).

### cmc_log_directory

Directory in which to store CMC application logs.

Default = /var/log/cloudian

To apply change, after Puppet propagation restart CMC

### cmc_admin_host_ip

Fully qualified domain name for the Admin Service. The CMC connects to this service.

If you have multiple service regions for your HyperStore system, this FQDN must be the one for the Admin Service in your default service region.

Default = Set during installation based on operator input.

To apply change, after Puppet propagation restart CMC.

### cmc_cloudian_admin_user

For the CMC, the login user name of the default system admin user. The system admin will use this user name to log into the CMC.

Default = admin

**This cannot be changed.**

### cmc_domain

Service endpoint (fully qualified domain name) for the CMC. This endpoint must be resolvable for CMC clients.

Default = Set during installation based on operator input.

To change this endpoint, use the **"Installer Advanced Configuration Options"** (page 377).

### cmc_web_secure

Whether the CMC should require HTTPS for all incoming client connections, true or false.

Set this property to "true" to require HTTPS. In this mode of operation, requests incoming to the CMC's regular HTTP port will be redirected to the CMC's HTTPS port.

Set this property to "false" to allow clients to connect through regular HTTP. In this mode of operation, requests incoming to the CMC's HTTPS port will be redirected to the CMC's regular HTTP port.

Default = true

To apply change, after Puppet propagation restart CMC.

### cmc_http_port

Port on which the CMC listens for regular HTTP requests.

Default = 8888

To apply change, after Puppet propagation restart CMC.

### cmc_https_port

Port on which the CMC listens for HTTPS requests.

Default = 8443

To apply change, after Puppet propagation restart CMC.

### cmc_admin_secure_ssl

If the CMC is using HTTPS to connect to the Admin Service (as it will if **"admin_secure"** (page 403) is set to "true"), this setting controls the CMC's requirements regarding the Admin Service's SSL certificate:

- If set to *true*, then when the CMC makes HTTPS connections to the Admin Service the CMC's HTTPS client will require that the Admin Service's SSL certificate be **CA validated** (or else will drop the connection).
- If set to *false*, then when the CMC makes HTTPS connections to the Admin Service the CMC's HTTPS client will allow the Admin Service's SSL certificate to be self-signed.

> **Note** Note that the SSL certificate that is used with the Admin Service by default is self-signed.

Default = false

To apply change, after Puppet propagation restart CMC.

### cmc_application_name

Name of the CMC web application, to be displayed in the URL paths for the various CMC UI pages. The CMC's URL paths are in the form *https://<host>:<port>/<application_name>/<page>.htm*.

Use only alphanumeric characters. Do not use spaces, dashes, underscores, or other special characters..

Default = Cloudian (and so URL paths are in form *https://<host>:<port>/Cloudian/<page>.htm*. For example *https://enterprise2:8443/Cloudian/dashboard.htm*).

To apply change, after Puppet propagation restart CMC.

### cmc_storageuri_ssl_enabled

If this is set to "true", the CMC uses HTTPS to connect to the HyperStore S3 Service (in implementing the CMC **Buckets & Objects** functionality). If "false", the CMC uses regular HTTP to connect to the HyperStore S3 Service.

Do not set this to "true" unless you have set up HTTPS for your HyperStore S3 Service (for more information see **"Security and Privacy Features"** (page 141) ).

Default = false

### cmc_grouplist_enabled

This setting controls whether the CMC will show a drop-down list of group names when selection of a group is necessary in interior parts of the CMC UI (parts other than the login page). This is relevant only for system administrators, since only system administrators have the opportunity to choose among groups for certain features (such as user management, group management, or usage reporting). Set this to "false" to have the UI instead present a text box in which the administrator can type the group name.

Default = true

To apply change, after Puppet propagation restart CMC.

> **Note** If the number of groups in your system exceeds the value of the *cmc_grouplist_size_max* setting (100 by default), then group drop-down lists are not supported and the UI will display a text input box for group name regardless of how you've set *cmc_grouplist_enabled*.

### cmc_login_languageselection_enabled

This setting controls whether to display at the top of the CMC's **Sign In** screen a selection of languages from which the user can choose, for rendering the CMC's text (such as screen names, button labels, and so on). The supported languages are English, Japanese, Spanish, German, and Portuguese. With this set to "true", the CMC language will initially be based on the user's browser language setting, but in the **Sign In** screen the user will be able to select a different supported language if they wish.

If you set this to "false", then the language selection will not display at the top of the CMC's **Sign In** screen, and instead the CMC text language will be exclusively based on the user's browser language setting. If the user's browser language setting matches one of the supported CMC languages, then that language will be used for the CMC text. If the user's browser language setting does not match any of the CMC's supported languages, the CMC text will display in English.

Default = true

To apply change, after Puppet propagation restart CMC.

### cmc_login_grouplist_enabled

This setting controls whether to enable the **Group** drop-down list on the CMC's **Sign In** screen. The **Group** drop-down list lists all groups registered in the HyperStore system, and when users log in they can choose their group from the list. If disabled, the drop-down list will not display and instead users will need to enter their group name in a Group Name text input box when logging into the CMC.

Set this to "false" if you don't want users to see the names of other groups.

Default = true

To apply change, after Puppet propagation restart CMC.

**Note** If the number of groups in your system exceeds the value of the *cmc_grouplist_size_max* setting (100 by default), then group drop-down lists are not supported and the UI will display a text input box for group name regardless of how you've set *cmc_login_grouplist_enabled*.

### cmc_login_grouplist_admincheckbox_enabled

This setting is relevant only if the group name drop-down list is not being displayed in the CMC login page (because *cmc_login_grouplist_enabled* is set to *false* or *cmc_grouplist_size_max* is exceeded). With no group name drop-down list in the login page, then:

- If *cmc_login_grouplist_admincheckbox_enabled* is set to *true* (the default), then underneath the Group Name text input field an "Admin Group" checkbox displays and system admin users can check this box rather than typing anything in the Group Name field.
- If *cmc_login_grouplist_admincheckbox_enabled* is set to *false*, then "Admin Group" checkbox does not display and instead system admin users must enter the system admin group ID in the Group Name text input field when they log into the CMC. **The system admin group ID is 0**. With this configuration, system admin users must enter **0** in the Group Name field to log in.

Default = true

To apply change, after Puppet propagation restart CMC.

### cmc_grouplist_size_max

Maximize number of groups that can be displayed in a CMC drop-down list.

If you have more than this many groups in your HyperStore system, then in parts of the CMC interface that require the user to select a group the interface will display a text input box rather than a drop-down list of groups to select from. The CMC will do this regardless of your setting for *cmc_login_grouplist_enabled* and *cmc_grouplist_enabled*.

For example, if *cmc_login_grouplist_enabled* and *cmc_grouplist_enabled* are set to "true" and *cmc_grouplist_size_max* is set to 100 (the default values), then the CMC will display drop-down lists for group selection if you have up to 100 groups in your system, or text input boxes for group name entry if you have more than 100 groups in your system.

Default = 100

To apply change, after Puppet propagation restart CMC.

### cmc_session_timeout

Session timeout for a user logged in to the CMC, as a number of minutes. After a logged-in user has been inactive for this many minutes, the CMC will terminate the user's session.

Default = 30

To apply change, after Puppet propagation restart CMC.

### cmc_view_user_data

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability of admin users to access regular users' storage buckets. When allowed this access, admin users can view regular users' data, add data to users' buckets, and delete users' data. Also when allowed this access, admin users can change the properties of regular users' buckets and objects.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any admin users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = false

To apply change, after Puppet propagation restart CMC.

> **Note** The *cmc_view_user_data* setting does not apply to file shares owned by users for whom you have enabled **File Services**. Regardless of how the *cmc_view_user_data* setting is configured, you as an administrator cannot add, view, edit, or delete file shares.

### cmc_crr_external_enabled

This controls whether settings for replicating from a local source bucket to an external S3 system -- a system other than the HyperStore system in which the source bucket resides -- will appear in the **Cross Region Replication** tab of the CMC's **Bucket Properties** screen. The default is *false*, so that such settings do not appear in the dialog, and the dialog can only be used to configure replication from a source bucket to a destination bucket in the same HyperStore system.

If you are considering setting *cmc_crr_external_enabled* to *true* you should first read **"Cross-System Replication"** (page 182) including the limitations and caveats noted in that section.

Default = false

To apply change, after Puppet propagation restart CMC.

### cmc_login_banner_*

For information about using the *cmc_login_banner_size*, *cmc_login_banner_title*, *cmc_login_banner_message*, and *cmc_login_banner_button_confirm* settings see **"Configuring a Login Page Acknowledgment Gate"** (page 234).

### cmc_csrf_origin_check_enabled

Set this to *true* if you want the CMC to implement request origin checks as a safeguard against Cross-Site Request Forgery (CSRF). CSRF exploits users who, while logged in to a secure web application such as the CMC, are concurrently doing things like checking their email or surfing web sites in other browser tabs. CSRF tricks such users into clicking on seemingly harmless links or images that are designed to submit malicious requests to the targeted web application without the user's knowledge. The malicious requests, sent from the user's browser, can reach the secure web application because the user has already successfully logged in.

If you set *cmc_csrf_origin_check_enabled* to *true* then the CMC will check standard HTTP request headers to confirm that, apart from login requests, all other requests to the CMC domain are originating from the CMC domain (as would be expected for legitimate requests, such as if the user is clicking links within the CMC or taking actions within a CMC page). If an HTTP request's origin domain doesn't match the target domain (the CMC domain), the CMC will reject the request.

If you set *cmc_csrf_origin_check_enabled* to *true* and you have the CMC behind a proxy or a load balancer, then you must also configure the *cmc_csrf_origin_allowlist* setting, described below.

Default = false

To apply change, after Puppet propagation restart CMC.

### cmc_csrf_origin_allowlist

If you set *cmc_csrf_origin_check_enabled* to *true* and you have the CMC behind a proxy or a load balancer, use the *cmc_csrf_origin_allowlist* setting to specify the CMC domain(s) associated with the proxy or load balancer (the CMC FQDN[s] mapped to the proxy or load balancer in your DNS configuration). This is necessary because in this environment, when the CMC application applies the origin check to guard against CSRF, the origin domain in incoming HTTP requests should match the CMC domain associated with the proxy or load balancer -- rather than matching the internal CMC application domain to which the proxy or load balancer forwards the requests.

When configuring *cmc_csrf_origin_allowlist*, include the transfer protocol along with the FQDN -- for example *https://cmc.domain.com*. If there are multiple CMC domains associated with your proxies or load balancers, configure *cmc_csrf_origin_allowlist* as a vertical bar separated list -- for example *https://cmc.domain1.com|https://cmc.domain2.com*.

Default = empty

To apply change, after Puppet propagation restart CMC.

### cmc_purgebucket_enabled

This setting controls whether or not CMC users will be given the option to purge all the content in a bucket in a single operation, as a precursor to deleting the bucket.

- If this setting is *false*, and in the CMC's **Bucket** page a user attempts to delete a bucket that has objects in it, the CMC will inform the user that they must delete the objects in the bucket before they can delete the bucket. The user can then go and delete each object, either through the CMC's **Objects** page, or through a third party S3 application.
- If this setting is *true*, and in the CMC's **Bucket** page a user attempts to delete a bucket that has objects in it, the CMC will inform the user that there are objects in the bucket and will present the user an option to purge the entire contents of the bucket by typing 'PURGE' into a confirmation text box. If the user confirms that they want to purge the bucket contents, the system will initiate a purge of the bucket contents as a background operation and inform the user that they can now delete the bucket.

  > **Note** The CMC will not allow users to purge contents from a bucket that has Object Lock enabled, even if you have set *cmc_purgebucket_enabled* to *true*.

Default = false

To apply change, after Puppet propagation restart CMC.

### cmc_sso_enabled

Whether to enable CMC support for single sign-on functionality, true or false. When this is set to false, if a user attempts to access the CMC via SSO, the access will be denied and an error will be written to *cloudian-ui.log*.

Default = false

> **IMPORTANT !** If you enable CMC SSO functionality, then for security reasons you should **set custom values for *cmc_sso_shared_key*** and ***cmc_sso_cookie_cipher_key*.** (the next two settings below *cmc_sso_enabled*). Do not leave these settings at their default values.

> **Note** For more information on CMC SSO, see **"Implementing Single Sign-On for the CMC"** (page 239).

### cmc_sso_shared_key

Shared security key used for hash creation, when using the "auto-login with one way hash" method of single sign-on access to the CMC.

Default = ss0sh5r3dk3y

### cmc_sso_cookie_cipher_key

Triple DES key used for cookie encryption.

Default = 123456789012345678901234

**Note** If you change this value after CMC SSO has already been in service, end users who had used SSO previously will have on their browser a Cloudian SSO cookie that is no longer valid. If such users access the CMC after your *cmc_sso_cookie_cipher_key* change, the CMC detects the invalid cookie, deletes it, and drops a new, valid one.

### cmc_bucket_tiering_default_destination_list

The list of auto-tiering destinations to display in the CMC interface that bucket owners use to configure auto-tiering for a bucket (for more information on this interface, while on the CMC's **Bucket Properties** page (**Buckets & Objects -> Buckets -> Properties**) click **Help**. Specify this as a quote-enclosed list, with comma-separation between destination attributes and vertical bar separation between destinations, like this:

```
"<name>,<endpoint>,<protocol>|<name>,<endpoint>,<protocol>|..."
```

This can be multiple destinations (as it is by default), or you can edit the setting to have just one destination in the "list" if you want your users to only use that one destination.

For multiple destinations you can have as many as you want, within reason (bear in mind that in the interface the dialog box will expand to accommodate the additional destinations).

The *<name>* will display in the CMC interface that bucket owners use to configure auto-tiering, as the auto-tiering destination name. The *<protocol>* must be one of the following:

- s3
- glacier
- azure
- spectra

If you wish you can include multiple destinations of the same type, if those destinations have different endpoints. For example, "Spectra 1,<endpoint1>,spectra|Spectra 2,<endpoint2>,spectra". Each such destination will then appear in the CMC interface for users configuring their buckets for auto-tiering.

Default = "AWS S3,https://s3.amazonaws.com,s3|AWS GLACIER,https://s3.amazonaws.com,glacier|Google,https://storage.googleapis.com,s3|Azure,https://blob.core.windows.net,azure"

To apply change, after Puppet propagation restart CMC.

**Note** If your original HyperStore version was older than 7.1.4, then after upgrade to 7.1.4 or later your default value here will also include a Spectra destination.

**Note** For more information about setting up auto-tiering in the system, see **"Preparing the Auto-Tiering Feature"** (page 175). That section includes information about how to enable the auto-tiering feature (which is disabled by default in the CMC interface) and about the option of having all end users use the same system-configured security credentials for accessing the tiering destination (rather than supplying their own security credentials for tiering, which is the default behavior). Note that if you choose to have all users use the same tiering security credentials you will need to specify a single system default tiering destination -- using a setting in the CMC's **Configuration Settings** page, as described in **"Preparing the Auto-Tiering Feature"** (page 175) -- and that configuration setting will **override** the *cmc_bucket_tiering_default_destination_list* setting.

### awsmmsproxy_host

This setting is obsolete and will be removed from a future HyperStore release. Do not use.

### bucketstats_enabled

Whether to enable usage statistics reporting on a per-bucket basis, true or false.If you set this to true, you can subsequently retrieve usage data for a specified bucket or buckets by using the Admin API's *GET /usage* and *POST /usage/bucket* methods. Per-bucket usage statistics will be available only dating back to the point in time that you enabled this feature. Per-bucket usage statistics are not tracked by default and will not be available for time periods prior to when you set this parameter to true.

Note that:

- While the *s3_perbucket_qos_enabled* setting (elsewhere in *common.csv*) is for enabling per-bucket stored-bytes and stored-object counters in the system -- so that you can at any time check the current counts for a given bucket -- the *bucketstats_enabled* setting is for enabling per-bucket usage tracking so that you can check to see the usage activity that occurred for a given bucket **during a specified period of time**.
- Setting *bucketstats_enabled* to *true* will result in additional metadata being stored in the Metadata DB, and will create additional work for the cron jobs that roll up usage data into hourly, daily, and monthly aggregates.

Default = false

To apply change, after Puppet propagation restart S3 Service.

### blink_disk_intervalsec

If you have HyperStore Appliance machines, and if in the Appliance section of the CMC's **Node Status** page you click the **Blink** function for one of the disk drives on one of your appliances, the light on the drive will blink for this many seconds.

Default = 180 (three minutes)

### blink_appliance_intervalsec

If you have HyperStore Appliance machines, and if in the Appliance section of the CMC's **Node Status** page you click the **Blink Chassis** function for one of your appliances, the light on the appliance chassis will blink for this many seconds.

Default = 180 (three minutes)

### file_port

This setting is related to the optional **File Services** component and is controlled by the HyperStore install script. Do not manually edit.

### file_secure_port

This setting is related to the optional **File Services** component and is controlled by the HyperStore install script. Do not manually edit.

### file_secure_enabled

This setting is related to the optional **File Services** component and is controlled by the HyperStore install script. Do not manually edit.

### file_api_path

This setting is related to the optional **File Services** component and is controlled by the HyperStore install script. Do not manually edit.

### search_enabled

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_auth_username

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_auth_pass

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_truststore_pass

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_fqdn

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_intranet_ip

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_port

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_secure_enabled

This setting is related to the optional **Metadata Search** component and is controlled by the HyperStore install script. Do not manually edit.

### search_indexer_bulkPut_maxObjects

This setting has to do with how object metadata is transmitted to the HyperStore Search Service cluster, if you are using the HyperStore Search Service. Do not edit this setting unless instructed to do so by Cloudian Support.

Default = 10000

### sqs_*

For information about these settings, see the Introduction to the "SQS API" section of the ***Cloudian HyperStore AWS APIs Support Reference***..

### kmip_*

These settings configure how HyperStore will make connections to a KMIP-compliant Key Management System (KMS) of your choosing. This is applicable only if you want to use the SSE-KMIP type of **server-side encryption**.

- **kmip_host** -- For the KMS that you are using, the IP address or the DNS-resolved FQDN or hostname. Do not include the transfer protocol prefix (i.e. do not include *https://*). (Mandatory; default = empty)
- **kmip_port** -- KMIP listening port used by the KMS. (Mandatory; default = 5696)
- **kmip_username** -- User name for accessing your KMS account. (Mandatory if required by the particular KMIP KMS that you are using, otherwise optional; default = empty)
- **kmip_password** -- Password for accessing your KMS account. (Mandatory if required by the particular KMIP KMS that you are using, otherwise optional; default = empty)
- **kmip_keystore_password** -- Password for accessing the local TLS keystore used for implementing TLS connections to the KMS. (Mandatory; default = empty)
- **kmip_ssl_protocol** -- The allowed TLS protocols, for the TLS connection with the KMIP KMS. If multiple values are specified, they must be comma-separated in double quotes -- for example "TLSv1.2,TLSv1.3" (Mandatory if required by the particular KMIP KMS that you are using, otherwise optional; default = empty. Note that this setting is required for connecting to HashiCorp Vault or PyKMIP.)
- **kmip_ssl_cipher_suites** -- The allowed TLS cipher suites, for the TLS connection with the KMIP KMS. If multiple values are specified, they must be comma-separated in double quotes. (Mandatory if required by the particular KMIP KMS that you are using, otherwise optional; default = empty. Note that this setting is required for connecting to HashiCorp Vault or PyKMIP.)

For more information about how to prepare HyperStore for using a KMIP KMS -- including the requirement to set up the local TLS keystore -- see **"Using SSE-KMIP"** (page 155).

## 9.6. hyperstore-server.properties.erb

The *hyperstore-server.properties* file configures the HyperStore Service. On each of your HyperStore nodes, the file is located at the following path by default:

```
/opt/cloudian/conf/hyperstore-server.properties
```

**Do not directly edit the *hyperstore-server.properties* file on individual HyperStore nodes**. Instead, if you want to make changes to the settings in this file, edit the configuration template file *hyperstore-server.properties.erb* on the Configuration Master node:

```
/etc/cloudian-7.5.2-puppet/modules/cloudians3/templates/hyperstore-server.properties.erb
```

Some *hyperstore-server.properties.erb* properties get their values from settings in **common.csv** or from settings that you can control through the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**). In the *hyperstore-server.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like <%= … %>. In the property documentation below, the description of such a property indicates "Gets its value from *<location>: <setting>*." Rather than editing such a property in *hyperstore-*

*server.properties.erb*, edit the property's corresponding *<location>: <setting>* instead (most often this will be a setting in *common.csv*).

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can edit this configuration file with this command:

```
$ hspkg config -e hyperstore-server.properties.erb
```

Specify just the configuration file name, not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the **standard keystrokes supported by *vi*** to make and save changes to the file.

**IMPORTANT !** If you do make edits to *hyperstore-server.properties.erb*, be sure to push your edits to the cluster and restart the HyperStore Service to apply your changes. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

## 9.6.1. hyperstore-server.properties.erb Details

### 9.6.1.1. Settings in hyperstore-server.properties.erb

#### cloudian.storage.datadir

Gets its value from *common.csv*:**hyperstore_data_directory**.

#### secure.delete

Set this to true if you want HyperStore to use "**secure delete**" methodology whenever implementing the deletion of an object from a bucket.

For information about how secure delete works, see **"Enabling Secure Delete"** (page 167)

**IMPORTANT !** Using secure delete has substantial impact on **system performance for delete operations**. Consult with your Cloudian representative if you are considering using secure delete.

Default = false

#### messaging.service.listen.address

Gets its value from *common.csv:* **hyperstore_listen_ip**.

#### messaging.service.listen.port

The port on which a HyperStore Service node listens for messages from other HyperStore Service nodes. This internal cluster messaging service is used in support of cluster management operations such as node repair.

Default = 19050

### messaging.service.read.buffer.size

When a HyperStore Service node reads data it has received over the network from other HyperStore Service nodes -- such as during repair operations -- this is the read buffer size.

Default = 65536

### messaging.service.write.buffer.size

When a HyperStore Service node writes data to the network for transferring to other HyperStore Service nodes -- such as during repair operations -- this is the write buffer size.

Default = 1048576

### messaging.service.threadpool.corepoolsize

Gets its value from *common.csv:***hyperstore_messaging_service_threadpool**.

### messaging.service.maxconnections

Maximum simultaneous number of connections that the HyperStore messaging service will accept.

Default = 2000

### messaging.service.repairfile.timeout

Maximum time in seconds to allow for repair of a single file on a HyperStore node. File repair entails checking other HyperStore nodes to find the most recent copy of the file and then downloading that copy.

Default = 120

### messaging.service.connection.timeout

Maximum time in seconds that a HyperStore node will allow for establishing a connection to another Hyper-Store node, for conducting inter-node operations.

Default = 300

### repair.session.threadpool.corepoolsize

Gets its value from *common.csv:* **hyperstore_repair_session_threadpool**.

### repair.session.rangeslice.maxrows

During **hsstool repair** or **hsstool repairec** operations, the maximum number of row keys to retrieve per get_range_slice query performed on Cassandra <GROUPID>_METADATA column families.

Default = 2

> **Note** Cloudian, Inc recommends that you leave this setting at its default value. Do **not** set it to a value lower than 2.

### repair.session.columnslice.maxcolumns

During **hsstool repair** or **hsstool repairec** operations, the maximum number of columns to retrieve per get_slice or get_range_slice query performed on Cassandra <GROUPID>_METADATA column families.

Default = 1000

### repair.session.slicequery.maxretries

During **hsstool repair** or **hsstool repairec** operations, the maximum number of times to retry get_slice or get_range_slice queries after encountering a timeout. The timeout interval is configured by *cassandra.cluster.CassandraThriftSocketTimeout* in *mts.properties.erb*, and retries are attempted as soon as a timeout occurs.

Default = 3

### repair.session.updateobjs.queue.maxlength

During **hsstool repair** operations, the target maximum number of object update jobs to queue for processing. Object update jobs are placed in queue by a differencer mechanism that detects discrepancies between object metadata on remote replicas versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for *repair.session.updateobjs.queue.maxwaittime* (below).

Default = 1000

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionJobQueueMaxLength)

### repair.session.updateobjs.queue.waittime

If during **hsstool repair** operations the differencer detects that the number of queued object update jobs is at or above the target maximum (as configured by *repair.session.updateobjs.queue.maxlength*), the number of seconds to wait before checking the queue size again. During this interval the differencer adds no more object update jobs to the queue.

Default = 2

### repair.session.updateobjs.queue.maxwaittime

During **hsstool repair** operations, the maximum total number of seconds for the differencer to wait for the object update job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of update requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object update requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

### repair.session.object.download.maxretries

During **hsstool repair** or **hsstool repairec** operations, the maximum number of times to retry object download requests after encountering a timeout. The timeout interval is configured by *mts.properties: cassandra.cluster.CassandraThriftSocketTimeout*, and retries are attempted as soon as a timeout occurs.

Default = 3

### repair.session.inmemory.fileindex

When performing Merkle Tree based **hsstool repair** (the default repair type) for files in the HyperStore File System, whether to hold the file indexes in memory rather than writing them to disk. Options are:

- true — For each vNode being repaired, a file index directory is created and is held in memory unless its size exceeds a threshold in which case it is written to disk (under the HyperStore data mount point that

the vNode is associated with). For most vNodes it will not be necessary to write the file indexes to disk. If file indexes are written to disk, they are automatically deleted after the repair operation completes.

- false — For each vNode being repaired, a file index directory is created and written to disk (under the HyperStore data mount point that the vNode is associated with), regardless of size. The file indexes are automatically deleted after the repair operation completes.

Default = true

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionInMemoryFileIndex)

### repair.digest.index.threadpool.corepoolsize

Gets its value from *common.csv:* **hyperstore_repair_digest_index_threadpool**.

### repair.merkletree.response.waittime

During *hsstool repair* operations, the repair coordinator node retrieves Merkle Trees from HyperStore endpoint nodes. When contacted by the coordinator node, the endpoint nodes first must construct the Merkle Trees before returning them to the coordinator node. Constructing the trees can take some time if a very large number of objects is involved.

The *repair.merkletree.response.waittime* property sets the maximum amount of time in minutes that the coordinator node will wait for Merkle Trees to be returned by all endpoint nodes. If not all Merkle Trees have been returned in this time, the repair operation will fail.

Default = 360

### rangerepair.threadpool.corepoolsize

Gets its value from *common.csv:* **hyperstore_rangerepair_threadpool**.

### stream.outbound.threadpool.corepoolsize

Gets its value from *common.csv:* **hyperstore_stream_outbound_threadpool**.

### repairec.sessionscan.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_repairec_sessionscan_threadpool**.

### repairec.digestrequest.threadpool.fixedpoolsize

Gets its value from *common.csv*:**hyperstore_repairec_digestrequest_threadpool**.

### repairec.task.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_repairec_task_threadpool**.

### repairec.rocksdbscan.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_repairec_rocksdbscan_threadpool**.

### repairec.session.queue.maxlength

During **"hsstool repairec"** (page 555) operations, the target maximum number of object update jobs to queue for processing. Object update jobs are placed in queue by a differencer mechanism that detects discrepancies between object metadata on remote replicas versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for *repairec.session.queue.maxwaittime*(below).

Default = 2000

### repairec.session.queue.waittime

If during **hsstool repairec** operations the differencer detects that the number of queued object update jobs is at or above the target maximum (as configured by *repairec.session.updateobjs.queue.maxlength*), the number of seconds to wait before checking the queue size again. During this interval the differencer adds no more object update jobs to the queue.

Default = 2

### repairec.session.queue.maxwaittime

During **hsstool repairec** operations, the maximum total number of seconds for the differencer to wait for the object update job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of update requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object update requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

### downloadrange.session.threadpool.corepoolsize

Gets its value from *common.csv:* **hyperstore_downloadrange_session_threadpool**.

### uploadrange.session.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_uploadrange_session_threadpool**.

### decommission.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_decommission_threadpool**.

### cleanup.session.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_cleanup_session_threadpool**.

### cleanup.session.deleteobjs.queue.maxlength

During **hsstool cleanup** or **hsstool cleanupec** operations, the target maximum number of object delete jobs to queue for processing. Object delete jobs are placed in queue by a cleanup job that detects discrepancies between object metadata in Cassandra versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for *cleanup.session.deleteobjs.queue.maxwaittime*.

Default = 2000

### cleanup.session.deleteobjs.queue.waittime

If during **hsstool cleanup** or **hsstool cleanupec** operations a cleanup job detects that the number of queued object delete jobs is at or above the target maximum (as configured by *cleanup.session.deleteobjs.queue.maxlength*), the number of seconds to wait before checking the queue size again. During this interval the cleanup job adds no more object delete jobs to the queue.

Default = 2

### cleanup.session.deleteobjs.queue.maxwaittime

During **hsstool cleanup** or **hsstool cleanupec** operations, the maximum total number of seconds for the differencer to wait for the object delete job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of delete requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object delete requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

### cleanup.session.delete.graceperiod

During **hsstool cleanup** or **hsstool cleanupec** operations, only consider an object for deletion if at least this many seconds have passed since the object's Last Modified timestamp.

Default = 86400

### cleanupjobs.threadpool.corepoolsize

During **hsstool cleanup** or **hsstool cleanupec** operations, this setting controls how many cleanup "jobs" can run in parallel on a single HyperStore node. For each HyperStore data mount point on a node, the object data directory structure is as follows:

```
<mountpoint>/<hsfs|ec>/<base62-encoded-vNode-token>/<policyid>/<000-255>/<000-255>/<filename>
```

Under the *hsfs* (for replica data) or *ec* (for erasure coded data) directory level, there are sub-directories for each of the mount point's vNodes (identified by token), and under those, sub-directories for each storage policy configured in your system (identified by system-generated policy ID). For more information on this directory structure, see **"HyperStore Service and the HSFS"** (page 63).

When a physical HyperStore node is cleaned, there is a separate cleanup "job" for each *<policyId>* sub-directory on the physical node. The *cleanupjobs.threadpool.corepoolsize* setting controls how many such jobs can run in parallel on a given physical node. Each concurrent job will run on a different HyperStore data disk on the node.

Within each job, the separate setting *common.csv:***"hyperstore_cleanup_session_threadpool"** (page 399) controls how many blobs (object replicas or erasure coded fragments) can be processed in parallel. Processing a blob entails checking the blob's corresponding object metadata to determine whether the blob is supposed to be where it is or rather should be deleted.

So for example with *cleanupjobs.threadpool.corepoolsize = 10* and *hyperstore_cleanup_session_threadpool = 10*, then on a given physical node being cleaned a maximum of 10 cleanup jobs would run in parallel (with each job working on a different disk), with a maximum of 10 blobs being processed in parallel within each of the 10 jobs.

Default = 10

### cleanup.task.batch.threadpool.size

This setting and the *cleanup.task.batch.integritycheck.threadpool.size* and *cleanup.session.threadpool.batch* settings provide additional performance tuning controls over various aspects of the *hsstool cleanupec* operation. The defaults are appropriate for most environments, and typically you should have no need to edit these settings unless instructed to do so by Cloudian Support.

Default = 20

### cleanup.task.batch.integritycheck.threadpool.size

See the description of **cleanup.task.batch.threadpool.size**.

Default = 10

### cleanup.session.threadpool.batch

See the description of **cleanup.task.batch.threadpool.size**.

Default = 15

### max.cleanup.operations.perdc

Maximum number of **hsstool cleanup** or **hsstool cleanupec** operations to allow at one time, per data center.

The limit is applied separately to *cleanup* and *cleanupec* operations. For example, if this property is set to 1 (as it is by default), then within a DC you can run one *cleanup* operation at a time and one *cleanupec* operation at a time. The limit does not prevent you from running one *cleanup* operation and one *cleanupec* operation simultaneously.

If you have multiple DCs in your HyperStore system, the limit is applied separately to each DC. For example if you have two data centers named DC1 and DC2, and if this property is set to 1, you can run one *cleanup* operation and one *cleanupec* operation in DC1 at the same time as you are running one *cleanup* operation and one *cleanupec* operation in DC2.

If you are considering raising this limit from its default of 1, first consult with Cloudian Support.

Default = 1

### hyperstore.proactiverepair.poll_time

At this recurring interval each HyperStore node checks to see whether any proactive repair jobs are queued for itself, and executes those jobs if there are any. Configured as a number of minutes.

This check is also automatically performed when a HyperStore Service node starts up. Subsequently the recurring check occurs at this configured interval.

For more information about the proactive repair feature, see Automated Data Repair Feature Overview .

Default = 60

> **Note** If for some reason you want to trigger proactive repair on a particular node immediately, you can do so by running the**"hsstool proactiverepairq"** (page 538) command with the "-start" option.

> **Note** For information about temporarily disabling the proactive repair feature, see Disabling or Stopping Data Repairs.

### stream.throughput.outbound

During **hsstool repair** or **hsstool repairec** operations, HyperStore nodes may stream large amounts of data to other HyperStore nodes. On each node this setting places an upper limit on outbound streaming throughput during repair operations, in megabits per second.

Default = 800

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileStreamingService → Attributes → MaxStreamThroughputOutbound)

### auto.repair.threadpool.corepoolsize

Gets its value from *common.csv*:**hyperstore_auto_repair_threadpool**.

### auto.repair.scheduler.polltime

Interval (in minutes) at which each HyperStore node's auto-repair scheduler will check the auto-repair queues for HSFS repair, Cassandra repair, and erasure coded data repair to see whether it's time to initiate a repair on that node.

Default = 10

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → AutoRepairSchedulerPollTime)

### auto.repair.schedule.interval

Gets its value from the "Replicas Repair Interval (Minutes)" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### auto.repairec.schedule.interval

Gets its value from the "EC Repair Interval (Minutes)" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### auto.repaircassandra.schedule.interval

Gets its value from the "Cassandra Repair Interval (Minutes)" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.storage.jmx.port

The port on which the HyperStore Service listens for JMX requests.

Default = 19082 (set elsewhere in the manifest structure; do not edit this property)

### disk.fail.action

Gets its value from the "HyperStore Disk Failure Action" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### disk.repair.rebuild

When HyperStore implements a **replaceDisk** operation it automatically executes *hsstool repair* and *hsstool repairec* for the replacement disk. With *disk.repair.rebuild=true* (the default setting), the automatic executions of *hsstool repair* and *hsstool repairec* will use the *-rebuild* option that those operations support. Using the *-rebuild* option is the most efficient way to rebuild data on to a replacement disk. Typically the only occasion you would have for setting *disk.repair.rebuild=false* -- so that the *-rebuild* option is not used -- is if you are instructed to do so by Cloudian Support in the context of troubleshooting a failed attempt to replace a disk.

Default = true

### disk.fail.error.count.threshold

This setting in combination with the *disk.fail.error.time.threshold* setting provides you the option to specify a read/write error frequency threshold that must be met before the system takes the automated action that is specified by the "HyperStore Disk Failure Action" setting.

The threshold, if configured, is in the form of "If *disk.fail.error.count.threshold* number of HSDISKERROR messages are logged in *cloudian-hyperstore.log* in regard to the same disk within an interval of *disk.fail.error.time.threshold* seconds, then take the automated action specified by the *disk.fail.action* setting."

For example, if *disk.fail.error.count.threshold* = 3, and *disk.fail.error.time.threshold* = 60, and *disk.fail.action* = "disable", then the system will disable any HyperStore data disk for which 3 or more HSDISKERROR messages appear in *cloudian-hyperstore.log* within a 60 second time span.

If you set the two threshold settings to "0", then no threshold behavior is implemented, and instead the automated action specified by the *disk.fail.action* setting is triggered by any single occurrence of an HSDISKERROR message in *cloudian-hyperstore.log*.

Default = 100

### disk.fail.error.time.threshold

Disk error threshold time span in seconds. For more information see **disk.fail.error.count.threshold**.

Default = 1800

### disk.check.interval

Gets its value from *common.csv*:**hyperstore_disk_check_interval**.

### disk.balance.delta

When the disk balance check is run (at the *disk.check.interval*), token migration is triggered if a disk's utilization percentage differs from the average disk utilization percentage on the node by more than the configured *disk.balance.delta*. If *disk.balance.delta* = 10 (the default), then, for example:

- If the average disk space utilization on a node is 35%, and the disk space utilization for Disk4 is 55%, then one or more tokens will be migrated away from Disk4 to other disks on the node (since the actual delta of 20% exceeds the maximum allowed delta of 10%).
- If the average disk utilization on a node is 40%, and the disk utilization for Disk7 is 25%, then one or more tokens will be migrated to Disk7 from the other disks on the node (since the actual delta of 15% exceeds the maximum allowed delta of 10%).

For more information on this feature see **"Automated Disk Usage Management Feature Overview"** (page 113).

Default = 10

### disk.audit.interval

At this configurable interval (in number of minutes), the system tries to write one byte of data to each HyperStore data disk. If any of these writes fail, */var/log/messages* is scanned for messages indicating that the file system associated with the disk drive in question is in a read-only condition (message containing the string "Remounting filesystem read-only"). If any such message is found, the disk is automatically disabled in accordance with your configured "HyperStore Disk Failure Action" (in the CMC, **Cluster -> Cluster Config -> Configuration Settings**).

The scan of */var/log/messages* will be limited to the time period since the the last time the disk audit was run.

This recurring audit of disk drive health is designed to proactively detect disk problems even during periods when there is no HyperStore Service read/write activity on a disk.

Default = 60

### disk.error.check.fs

When scanning */var/log/messages* as part of the disk audit, the messages will be first filtered by this file system name string.

Default = "EXT4-fs"

### max.diskusage.percentage

The **hsstool repair** operation will fail if any HyperStore data disk that stores token ranges impacted by the operation is more than this percent full.

Default = 95

### auto.repair.computedigest.run.number

Gets its value from *common.csv*: **auto_repair_computedigest_run_number**.

### hss.errorlogger.appender

Do not edit this setting.

### hss.heallogger.appender

Do not edit this setting.

### enable.cassandra.rangerepair

If this is set to true, HyperStore uses a "range repair" approach when executing Cassandra auto-repairs. Each impacted token range is repaired one range at a time, sequentially. This approach improves the performance for Cassandra auto-repairs.

For background information on the scheduled auto-repair feature see Automated Data Repair Feature Overview.

Default = true

### retry.rebalance.ranges

If this is set to true, HyperStore will automatically retry any failed sub-tasks from an *hsstool rebalance* operation that has completed on a newly added node. Like other types of **proactive repair**, this retry of any failed rebalance sub-tasks occurs once per hour.

Default = true

### rocksdb.*

The *rocksdb.** settings configure the RocksDB database that runs on each HyperStore node and stores file digests. Do not edit these settings.

# 9.7. mts.properties.erb

The *mts.properties* file configures the S3 Service and the Admin Service. On each of your HyperStore nodes, the file is located at the following path by default:

```
/opt/cloudian/conf/mts.properties
```

**Do not directly edit the *mts.properties* file on individual HyperStore nodes**. Instead, if you want to make changes to the settings in this file, edit the configuration template file *mts.properties.erb* on the Configuration Master node:

```
/etc/cloudian-7.5.2-puppet/modules/cloudians3/templates/mts.properties.erb
```

Some *mts.properties.erb* properties get their values from settings in **common.csv** or from settings that you can control through the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**). In the *mts.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like *<%= … %>*. In the property documentation below, the description of such a property indicates "Gets its value from *<location>: <setting>*." Rather than editing such a property in *mts.properties.erb*, edit the property's corresponding *<location>: <setting>* instead (most often this will be a setting in *common.csv*).

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can edit this configuration file with this command:

```
$ hspkg config -e mts.properties.erb
```

Specify just the configuration file name, not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the **standard keystrokes supported by *vi*** to make and save changes to the file.

**IMPORTANT !** If you do make edits to *mts.properties.erb*, be sure to push your edits to the cluster and restart the S3 Service to apply your changes. Note that restarting the S3 Service automatically restarts the Admin Service as well. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

**Note** Settings for enabling and configuring the Veeam Smart Object Storage API (SOSAPI) are not included in *mts.properties.erb* by default. If you want to use this functionality, contact Cloudian Support for guidance on adding the applicable properties to this file.

## 9.7.1. mts.properties.erb Details

### 9.7.1.1. Settings in mts.properties.erb

#### cassandra.cluster.name

Do not edit this setting.

#### cassandra.cluster.Hosts

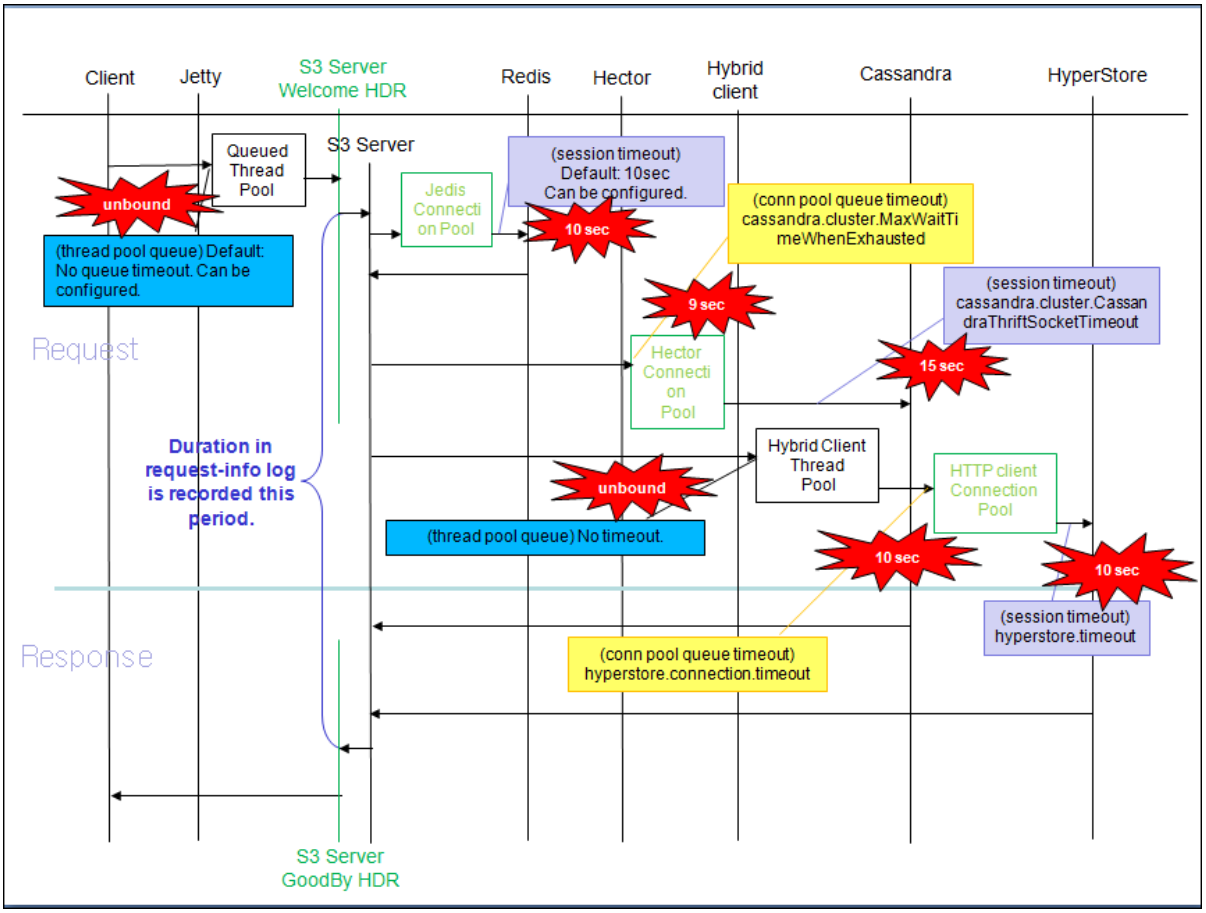
Do not edit this setting.

### cassandra.cluster.CassandraThriftSocketTimeout

After submitting a request to a Cassandra instance via its Thrift socket, the amount of time in milliseconds to wait for a response from Cassandra before failing the operation.

Default = 15000

The diagram below shows the place of *cassandra.cluster.CassandraThriftSocketTimeout* and other timeouts within the S3 Service's request processing flow.




### cassandra.cluster.MaxActive

Gets its value from *common.csv*: **"cassandra_max_active"** (page 414).

> **Note** The S3 Service, Admin Service, and HyperStore Service each separately maintain their own pool of connections to the Cassandra data store. The configuration settings in this section are applied separately to each of the three connection pools. For example, if *MaxActive=10*, then the S3 Service to Cassandra, Admin Service to Cassandra, and HyperStore Service to Cassandra connection pools are **each** allowed a maximum of 10 simultaneously active connections.

### cassandra.cluster.MaxIdle

The maximum allowed number of idle connections in a Cassandra connection pool. Any idle connections in excess of this limit are subject to being closed. Set to a negative value to disable this limit. Note this control is applicable only if *TimeBetweenEvictionRunsMillis* is set to a positive value.

Default = 20

### cassandra.cluster.MaxWaitTimeWhenExhausted

If *cassandra.cluster.MaxActive* has been reached for a target Cassandra host and a thread requires a new connection to the host, the thread will wait this many milliseconds before returning an error to the client.

Default = 9000

For a diagram showing the place of this timeout within the S3 request processing flow, see *cassandra.cluster.CassandraThriftSocketTimeout* (above).

### cassandra.cluster.RetryDownedHosts

Whether or not to periodically retry Cassandra hosts that have been detected as being down, using a background thread. If set to "true", the retry is attempted at configurable interval *cassandra.cluster.RetryDownedHostsDelayInSeconds*.

Default = true

### cassandra.cluster.RetryDownedHostsQueueSize

Maximum number of downed Cassandra hosts to maintain in the downed host retry queue at the same time. If multiple Cassandra nodes are down, and if *cassandra.cluster.RetryDownedHosts=true*, then a queue is maintained for retrying downed nodes. The *cassandra.cluster.RetryDownedHostsQueueSize* sets a limit on the number of nodes that can be in the retry queue simultaneously.

Default = -1 (unlimited)

### cassandra.cluster.RetryDownedHostsDelayInSeconds

The number of seconds to wait between retry attempts for downed hosts. Applicable only if *cassandra.cluster.RetryDownedHosts=true*.

Default = 10

### cassandra.cluster.Lifo

If true, use a “last in, first out” policy for retrieving idle connections from a pool (use the most recently used idle connection). If false, use "first in, first out" retrieval of idle connections (use the oldest idle connection).

Default = true

### cassandra.cluster.MinEvictableIdleTimeMillis

The minimum time in milliseconds that a connection must be idle before it becomes eligible for closing due to maximum idle connection limits.

Default = 100000

### cassandra.cluster.TimeBetweenEvictionRunsMillis

The interval in milliseconds at which to check for idle connections in a pool, for enforcement of maximum idle connection limits.

Default = 100000

### cassandra.cluster.UseThriftFramedTransport

Whether to use framed transport for Thrift. Strongly recommended to leave as true. If set to false, then

*thrift_framed_transport_size_in_mb* in *cassandra.yaml* must be set to 0.

Default = true

### cassandra.cluster.AutoDiscoverHosts

Whether or not to periodically check for the presence of new Cassandra hosts within the cluster and to use these same settings to interface with those new hosts. If set to true, a check for new hosts is performed at configurable interval *cassandra.cluster.AutoDiscoveryDelayInSeconds*.

Default = true

### cassandra.cluster.AutoDiscoveryDelayInSeconds

Number of seconds to wait between checks for new hosts. Applicable only *if cassandra.cluster.AutoDiscoverHosts=true*

Default = 60

### cassandra.cluster.RunAutoDiscoveryAtStartup

Whether or not to perform auto-discovery at cluster start-up. See *cassandra.cluster.AutoDiscoverHosts*.

Default = false

### cassandra.cluster.HostTimeoutCounter

If a Cassandra node returns more than this many host timeout exceptions within an interval of

*cassandra.cluster.HostTimeoutWindow*, mark the node as temporarily suspended. This setting and the other *HostTimeout** settings below are applicable only if *cassandra.cluster.UseHostTimeoutTracker=true*.

Default = 3

### cassandra.cluster.HostTimeoutWindow

If within an interval of this many milliseconds a Cassandra node returns more than *cassandra.cluster.HostTimeoutCounter* host timeout exceptions, mark the node as temporarily suspended.

Default = 1000

### cassandra.cluster.HostTimeoutUnsuspendCheckDelay

How often to check suspended nodes list to see which nodes should be unsuspended, in seconds.

Default = 10

### cassandra.cluster.HostTimeoutSuspensionDurationInSeconds

When the periodic check of the suspended nodes list is performed, if a node has been suspended for more than this many seconds, unsuspend the node and place it back in the available pool.

Default = 30

### cassandra.cluster.UseHostTimeoutTracker

Whether to keep track of how often each Cassandra node replies with a host timeout exception, and to temporarily mark nodes as suspended if their timeout exceptions are too frequent. See the *HostTimeout** setting descriptions above for details.

Default = true

### cassandra.cluster.UseSocketKeepalive

Whether to periodically send keep-alive probes to test pooled connections to Cassandra hosts.

Default = true

### cassandra.data.directories

Gets its value from *common.csv*: **cassandra_data_directory**.

### cassandra.fs.keyspace

Base name of the Cassandra keyspaces in which object metadata is stored. A storage policy ID is appended to this base name to create a keyspace for a particular storage policy (for example, *UserData_b06c5f9213ae396de1a80ee264092b56*). There will be one such keyspace for each storage policy that you have configured in your system.

Default = UserData

### cassandra.jmx.port

Cassandra's JMX listening port.

Default = 7199

### cassandra.tombstone_cleanup_threshold

Gets its value from *common.csv*: **cassandra_tombstone_cleanup_threshold**.

### cassandra.tombstone_gcgrace

Gets its value from *common.csv*: **cassandra_tombstone_gcgrace**.

### cloudian.repair.eventqueue.create.interval

When **proactive repair** is run on a node, it reads from a queue of node-specific object write failure events that is stored in Cassandra. The object write failure events are timestamped and are bundled together based on the time interval in which they occurred. The *cloudian.repair.eventqueue.create.interval* setting controls the time span (in minutes) that's used for the bundling. For example with the default of 60 a new bundle starts each 60 minutes, if a node is unavailable for longer than this and write failure events for the node are accumulating.

When the node comes back online, the first automatic run of proactive repair will repair the objects from all the interval-based bundles except for the most current one (the in-progress interval, such as the current hour if the default of 60 is being used). That bundle will be processed at the next automatic run of proactive repair. By default proactive repair runs (if needed) every 60 minutes — this frequency is configurable by **"hyperstore.proactiverepair.poll_time"** (page 437).

Default = 60

### cloudian.cassandra.default.ConsistencyLevel.Read

For the Reports keyspace (service usage data), AccountInfo keyspace (user account information), and Monitoring keyspace (system monitoring data), the consistency level to require for read operations. The reads are requested by other HyperStore components such as the S3 Service and the Admin Service.

For general information about system metadata storage, see **"System Metadata Replication"** (page 134).

For consistency level definitions, see **"Consistency Levels"** (page 131).

You can optionally use a comma-separated list to implement **Dynamic Consistency Levels**.

Default = LOCAL_QUORUM,ONE

### cloudian.cassandra.default.ConsistencyLevel.Write

For the Reports keyspace (service usage data), AccountInfo keyspace (user account information), and Monitoring keyspace (system monitoring data), the consistency level to require for write operations. The writes are requested by other HyperStore components such as the S3 Service and the Admin Service.

For consistency level definitions, see **"Consistency Levels"** (page 131).

You can optionally use a comma-separated list to implement **Dynamic Consistency Levels**.

Default = QUORUM,LOCAL_QUORUM

**IMPORTANT !** Since this setting applies to writes, using "ONE" is not recommended.

### cloudian.cassandra.UserData.ConsistencyLevel.HyperStore

The consistency level to require when HyperStore is reading or writing to the *UserData_<policyid>* keyspaces in order to implement a repair or cleanup operation. If the configured consistency requirement can't be met, the operation fails and subsequently retries. The operation will abort if two retry attempts fail to achieve the required consistency level.

For consistency level definitions, see **"Consistency Levels"** (page 131).

You can optionally use a comma-separated list to implement **Dynamic Consistency Levels**.

Default = QUORUM

**IMPORTANT !** Since this setting applies to writes as well as reads, using "ONE" is not recommended.

### cloudian.cassandra.localdatacenter

Do not edit this setting.

### cloudian.s3.authorizationV4.singleregioncheck

In Cloudian deployments that consist of only one service region, this setting impacts the system's handling of incoming S3 requests that are using AWS Signature Version 4 Authentication:

- If this setting is set to "true", the system will validate that the region name that the client used when creating the request signature (as indicated in the scope information that the client specifies in the request) is in fact the region name for your single region. If not the request will be rejected.

- If set to "false" the system will not validate the region name. Instead, the system calculates and validates the signature using whatever region name is specified in the request.

Default = false

**Note** Do not set this property to "true" if the S3 service endpoint (URI) for your one region does not include a region name string. For example, you can set this property to "true" -- thereby enabling region name validation --if your S3 endpoint is *s3-tokyo.enterprise.com* (where *tokyo* is the region name), but not if your S3 endpoint is *s3.enterprise.com* (which lacks a region name string).

### cloudian.s3.authorizationV4.multiregioncheck

In Cloudian deployments that consist of multiple service regions, this setting impacts the system's handling of incoming S3 requests that are using AWS Signature Version 4 Authentication:

- If this setting is set to "true", the system will validate that the region name that the client used when creating the request signature (as indicated in the scope information that the client specifies in the request) is in fact the region name for the region to which the request has been submitted. If not the request will be rejected.
- If set to "false" the system will not validate the region name. Instead, the system calculates and validates the signature using whatever region name is specified in the request.

Default = false

**IMPORTANT !** Do not set this property to "true" if the S3 service endpoints (URIs) for your regions do not include region name strings. For example, you can set this property to "true" -- thereby enabling region name validation --if your S3 endpoints are *s3-tokyo.enterprise.com* and *s3-osaka.enterprise.com* (where *tokyo* and *osaka* are region names), but not if your S3 endpoints lack region name strings.

### cloudian.s3.serverside.encryption.keylength

Gets its value from *common.csv*: **cloudian_s3_aes256encryption_enabled**.

### cloudian.s3.qos.enabled

Gets its value from "Enforce Configured QoS Limits for Storage Utilization" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.s3.qos.rate.enabled

Gets its value from "Enforce Configured QoS Limits for Request Rates and Data Transfer Rates" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.s3.qos.cache.enabled

When QoS enforcement is enabled, for each S3 request the S3 Service checks current user and group QoS usage against configured QoS limits. The usage counters and configured limits are stored in the Redis QoS DB. This setting if set to "true" enables the S3 Service to cache QoS counter and limits data and to check that cached data first when performing its QoS enforcement role. If there is no cache hit then Redis is checked.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → QosCacheEnabled)

### cloudian.s3.qos.cache.expiryms

For the S3 Service's QoS data cache (if enabled), the cache entry expiry time in milliseconds.

Default = 10000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → QosCacheExpiryMs)

### cloudian.s3.qos.storagewrite.batch.enabled

If set to "true" then the AccountHandler's updates of storage object and storage bytes counters in Redis are batched together.

Default = false

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → QosStorageWriteEnabled)

### cloudian.s3.qos.storagewrite.intervalms

If batching of storage counter updates is enabled, the batch interval in milliseconds.

Default = 60000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → QosStorageWriteIntervalMs)

### cloudian.s3.qos.maxdelay

If a PUT Object or PUT Object Copy operation that is overwriting an existing object takes more than this many milliseconds to complete, then before the system updates the user's Storage Bytes (SB) count it re-checks the existing object's metadata to ensure that the calculated size differential between the existing object and the new object is based on fresh metadata. This configurable behavior is intended to reduce the likelihood of erroneous SB counts resulting from race conditions wherein multiple client requests are overwriting the same object at the same time.

The context is that in the case of an existing object being overwritten by a new instance of the object, the system updates the user's SB count by calculating the delta between the original object's size (as indicated by its metadata) and the new object's size and then applying that difference to the user's SB count. It's important therefore that the calculated delta be based on up-to-date metadata for the object that's being overwritten, even in the case where the writing of the new object takes a long time to complete.

Default = 60000

### cloudian.s3.qos.bucketcounter.enabled

Gets its value from *common.csv*: **s3_perbucket_qos_enabled**.

### cloudian.s3.metadata.cache.enabled

If set to *true*, bucket metadata is cached by the S3 Service. When S3 request processing requires bucket metadata, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → MDCacheEnabled)

### cloudian.s3.group.cache.enabled

If set to *true*, then user group metadata is cached by the S3 Service. When S3 request processing requires group status, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GroupCacheEnabled)

### cloudian.s3.credentials.cache.enabled

If set to *true*, users active security credentials are cached by the S3 Service. When S3 request processing requires a user's credentials, the cache is checked first. If there is no cache hit then the needed credentials retrieved from Redis, and are cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source credentials in Redis change and then automatically invalidates the cached credentials.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GroupCacheEnabled)

### cloudian.s3.user.cache.enabled

If set to *true*, user metadata is cached by the S3 Service. When S3 request processing requires a user's status and/or display name, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → UserCacheEnabled)

### cloudian.s3.autoinvalidateiamcache

Gets its value from *common.csv*: **cloudian_s3_autoinvalidateiamcache**.

### iam.service.enabled

Gets its value from *common.csv*: **iam_service_enabled**.

### iam.service.port

Gets its value from *common.csv*: **iam_port**.

### iam.service.secure

Gets its value from *common.csv*: **iam_secure**.

### iam.service.secureport

Gets its value from *common.csv*: **iam_secure_port**.

### iam.service.endpoint

Gets its value from *common.csv*: **iam_service_endpoint**.

### cloudian.s3.regions

Gets its value from *common.csv*: **regions**.

### cloudian.s3.home_region

Do not edit this setting.

### cloudian.s3.default_region

Gets its value from *common.csv*: **default_region**.

### cassandra.cluster.name.<region>

Do not edit this setting.

### cassandra.cluster.hosts.<region>

Do not edit this setting.

### cloudian.s3.domain.<region>

Do not edit this setting.

### cloudian.s3.ssl_domain.<region>

Do not edit this setting.

### cloudian.s3.website_endpoint

To change this, use the HyperStore **"Installer Advanced Configuration Options"** (page 377).

### cloudian.util.dns.resolver

Method used to resolve foreign buckets to the correct region, in support of the S3 LocationConstraint feature. Currently only one option is supported:

- com.gemini.cloudian.util.dns.RedirectResolver — Never resolve foreign bucket domain names, always reply with a 307 redirect. The Location value to supply in the redirect response is obtained from the global Redis database.

Default = com.gemini.cloudian.util.dns.RedirectResolver

### cloudian.publicurl.port

To change S3 listening ports, use the HyperStore **"Installer Advanced Configuration Options"** (page 377).

### cloudian.publicurl.sslport

To change S3 listening ports, use the HyperStore **"Installer Advanced Configuration Options"** (page 377).

### cloudian.s3.server

HTTP Server header value to return in responses to S3 requests. If you want no Server header value returned in responses to S3 requests, uncomment this setting and set it to empty.

Default = Commented out; uses internal default "CloudianS3"

### cloudian.s3.tmpdir

The directory in which to temporarily store large files in order to reduce memory usage.

Default = Commented out; uses internal default *java.io.tmpdir*.

### cloudian.fs.read_buffer_size

When interacting with the file storage system in Cassandra, the size of the S3 Service's and Admin Service's read buffer, in bytes. A larger read buffer can enhance performance but will also consume more memory.

Default = 65536

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → ReadBufferSize)

### cloudian.s3.putobject.max_size

Gets its value from the "Put Object Maximum Size (Bytes)" setting on the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.s3.getbucket.max_num_of_keys

When performing an S3 GET Bucket operation (which returns meta-data about the objects in the bucket specified in the request), the maximum number of objects to list in the response. If the client request sets a "max-keys" parameter, then the lower of the client-specified value and the *cloudian.s3.getbucket.max_num_of_keys* value is used.

Default = 1000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GetBucketMaxKeys)

### cloudian.s3.max_user_buckets

Gets its value from the "Maximum Buckets Per User" setting on the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.s3.delimiter_regex

Regular expression indicating allowed delimiters in getBucketList objects.

Default = .+

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → DelimiterRegex)

### cloudian.s3.multipart.maxparts

Gets its value from the "Multipart Upload Maximum Parts" setting on the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### cloudian.s3.multipart.minpartsize

In an S3 Multipart Upload submitted to HyperStore, the required minimum size per part, excluding the last part. Expressed in number of bytes. The operation will fail if a part other than the last part is smaller than this many bytes. For example, if you set this to 5242880 (5MB, which is the Amazon S3 default for minimum part size) then in a Multipart Upload each part uploaded must be at least 5MB in size, except for the last part which is allowed to be as small as necessary to complete the object.

The HyperStore default of 1 byte essentially places no restriction on minimum part size.

Default = 1

### cloudian.s3.unsupported

Comma-separated list of Amazon S3 query arguments that the HyperStore system does not support. This list applies across HTTP methods and across S3 resource types. In response to requests that include an unsupported query argument, the HyperStore system will return 501, Not Implemented.

Default = accelerate,requestPayment,analytics,metrics,select,notification,attributes,intelligent-tiering

### cloudian.util.ntp.Path

Path to *ntpstat*. Setting this value is required only if ntpstat is not in the environment PATH.

Default = commented out; internal default = ' '

### cloudian.util.ntp.MaximumSynchronizationDistance

Maximum system clock skew to allow when servicing S3 requests that require the S3 Service to generate an object versionId, in milliseconds.

- If this setting is set to a positive value, a S3 Service node that is processing S3 requests that require generating a versionid will perform an ntpstat check. If the node's system clock is skewed by more than *cloudian.util.ntp.MaximumSynchronizationDistance* milliseconds, the S3 Service rejects the S3 request with a "503 Service Unavailable" error response. The frequency of the ntpstat check can be limited by using the *cloudian.util.ntp.CheckIntervalMillis* setting.
- If this setting is set to 0, the *npstat* check is not performed when processing S3 requests that require generating a versionId.

Default = 0

### cloudian.util.ntp.CheckIntervalMillis

Minimum time between *ntpstat* checks, in milliseconds.

- If this setting is set to a positive value, then when the S3 Service is processing S3 requests that require generating a versionId, an *ntpstat* check will be performed only if *cloudian.util.ntp.CheckIntervalMillis* milliseconds or more have passed since the last *ntpstat* check was performed.

- If this setting is set to 0, an *ntpstat* check is performed with every S3 request that requires generating a versionId.

Default = 60000

### cloudian.s3.batch.delete.delay

When executing the batch processing to purge deleted objects from disk (see *cloudian.delete.queue.poll.interval* below for background information), the number of milliseconds to pause in between purges of individual objects.

If *cloudian.s3.batch.delete.delay* is set to 0, then when the batch processing job is running there is no delay between individual object purges.

The *cloudian.s3.batch.delete.delay* property serves to "throttle" the execution of the deleted object batch processing job (to slow it down so as to reduce its resource demands).

Default = 30

### cloudian.delete.queue.poll.interval

When S3 requests for object deletion are received and successfully processed by HyperStore, the system immediately marks the objects as having been deleted, and stores this object deletion flag in a queue in the Metadata DB. But the actual purging of object data from disk does not occur until a batch processing job runs, to physically purge objects that have been marked as deleted.

The *cloudian.delete.queue.poll.interval* property sets the interval at which to run the batch processing of queued object deletes, in number of minutes. With the default of 60, the object deletion batch processing job is run once per hour on each node. For performance reasons the kick-off times are staggered across the nodes -- so, all nodes run the job once per hour (by default configuration) but the jobs do not all kick off at the same time. However because a batch processing job takes some time to complete, there may be overlap such that batch processing jobs are running on multiple nodes concurrently. You can limit the degree of concurrency with the *cloudian.dc.delete.instances* property.

Also for performance reasons, when a batch processing job is being run on a node, that node manages the batch but the work of implementing the purge actions contained in that batch is spread across the cluster.

> **Note** This property can be set to "0" to disable batch processing of objects marked for deletion, but if you do this it should only be temporarily, and in consultation with Cloudian Support.

Default = 60

### cloudian.dc.delete.instances

This setting specifies the maximum number of nodes that can concurrently run batch delete processing jobs within one data center. This provides a way of throttling how much batch delete processing work can happen concurrently in a data center.

For example, with *cloudian.dc.delete.instances* set to 2, if batch delete processing jobs are running on two nodes in a data center and the time arrives for a batch delete processing job to kick off on another node in that same data center (based on *cloudian.delete.queue.poll.interval* ), the kick-off of that job will wait until one of the currently running batch delete processing jobs completes -- so that no more than two jobs are running concurrently per data center.

If you set *cloudian.dc.delete.instances* to 0, this means there is no limit -- so batch processing jobs can concurrently run on all nodes in a data center.

Default = 10

### cloudian.delete.purge.bucket.threads

Maximum number of threads to devote to purging a bucket (while executing the purging associated with the Admin API call *POST /bucketops/purge*). For details on this API call see the "bucketops" section of the ***Cloudian HyperStore Admin API Reference***.

Default = 6

### cloudian.s3.deleteins3

This setting applies to HyperStore's **auto-tiering** behavior when the tiering destination has native S3 API support, such as AWS, Google Cloud Platform (GCP), or a remote HyperStore system.

If *cloudian.s3.deleteins3 = true*, then:

- For an object that has been transitioned to the tiering destination system, if a delete is performed on the object in the HyperStore source bucket then the object's metadata will be deleted from the local HyperStore system and **the tiered object itself will be deleted from the tiering destination bucket**.
- If a source bucket that has been configured for auto-tiering is deleted from the local HyperStore system, the **tiering destination bucket will be deleted from the tiering destination system**.

**Note** When HyperStore is deleting a tiered object or a tiering destination bucket from the tiering destination system, this operation must successfully complete before HyperStore deletes the corresponding local object metadata or local bucket.

If *cloudian.s3.deleteins3 = false*, then:

- For an object that has been transitioned to the tiering destination system, if a delete is performed on the object in the HyperStore source bucket then the object's metadata will be deleted from the local HyperStore system -- and so the object will no longer be accessible through the local HyperStore system -- but the tiered object itself will **not** be deleted from the tiering destination bucket.
- If a source bucket that has been configured for auto-tiering is deleted from the local HyperStore system, the tiering destination bucket will **not** be deleted from the tiering destination system.

Default = true

**Note** For tiering destinations that do not have native S3 API support -- such as Azure or Spectra Logic -- if a tiered object or a bucket configured for tiering are deleted from the local HyperStore system, the corresponding object or bucket are deleted from the tiering destination system. For these tiering destinations this behavior cannot be modified.

### cloudian.s3.crr.syncbucket

Once every four hours a HyperStore **cron job** processes any pending **cross-region replication** tasks. Typically with cross-region replication, objects are replicated to the destination bucket as soon as they are uploaded to the source bucket. Pending cross-region replication tasks result when an attempt to replicate an object from the

source bucket to the destination bucket returns a temporary error such as a connection failure or an HTTP 5xx error. The cron job retries the replication of such objects.

- If *cloudian.s3.crr.syncbucket = true*, then each time the cron job runs it also performs a complete sync-up of each bucket for which there are replication tasks in the retry queue. With this complete sync-up, any objects that are in these source buckets but not in their corresponding destination buckets will be replicated to the destination buckets. This includes objects that were in the source buckets before cross-region replication was enabled on the buckets, if any. (The exceptions are objects that were in the source buckets before versioning was enabled on the bucket, if any; and objects that are **encrypted by the SSE-C or AWS-KMS methods**, if any. Such objects will not be replicated.).
- If *cloudian.s3.crr.syncbucket = false*, the cron job will only process pending cross-region replication jobs (retries of object replication attempts that met with temporary errors). This makes the cron job less resource-intensive than it would be if *cloudian.s3.crr.syncbucket = true*.

Default = false

**Note** Setting *cloudian.s3.crr.syncbucket = true* will not affect replication processing for source buckets for which there are not any pending replication tasks in the retry queue. It applies only to buckets for which there are pending replication tasks in the retry queue. If you want to trigger a replication sync-up for a particular source bucket you can do so as described in **"Triggering the Replication of a Bucket's Pre-Existing Objects"** (page 187).

### s3.bucket.logging.internalnetworks

Gets its value from *common.csv*: **s3_bucket_logging_internalnetworks**.

### cloudian.s3.client.ConnectionTimeout

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the connection establishment timeout in milliseconds.

Default = 2000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientConnectionTimeout)

### cloudian.s3.client.SocketTimeout

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the request processing timeout in milliseconds. When a request is sent over an open connection, if a complete response is not received within this interval, the request times out and the connection is closed.

Default = 50000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientSocketTimeout )

### cloudian.s3.client.MaxErrorRetry

When HyperStore's native S3 client is interfacing with a remote S3 Service (such as during auto-tiering), the maximum number of times to retry sending a request that encounters a temporary error response from the remote service (such as a 503 error).

For auto-tiering, after this many retries the request will be put back in the auto-tiering queue and tried again at the next running of the auto-tiering cron job.

Default = 3

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientMaxErrorRetry)

### cloudian.s3.client.UserAgent

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the value of the first part of the HTTP User-Agent header, where the whole header is "<ConfigValue>, <AWS SDK Version> <OS Version> <JDK Version>". For example, if you set this setting to "Agent99", then the resulting User-Agent header will be "Agent99, <AWS SDK Version> <OS Version> <JDK Version>", with the latter three values being populated automatically by the system.

Default = Commented out; uses internal default "Cloudian/<version> <default-region>"

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientUserAgent)

### cloudian.auditlog.enabled

Do not edit this setting.

### reports.raw.ttl

Time-to-live for "raw" S3 usage data in the Reports keyspace in Cassandra, in seconds. Raw service usage data will be automatically deleted this many seconds after its creation. This is set by the system -- **do not manually edit** this setting.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default =

- 604800 (seven days), if the "Track/Report Usage for Request Rates and Data Transfer Rates" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**) is set to "false", as it is by default
- 86400 (one day) if the "Track/Report Usage for Request Rates and Data Transfer Rates" setting is set to "true"

For an overview of how the HyperStore system tracks service usage by groups, users, and buckets, see **"Usage Reporting Feature Overview"** (page 218).

### reports.rolluphour.ttl

Time-to-live for hourly roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Hourly roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 5616000 (65 days)

**IMPORTANT !** This hourly rollup data is the basis for generating billing reports. After hourly rollup data is deleted it is no longer available for generating billing reports.

### reports.rollupday.ttl

Time-to-live for daily roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Daily roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 5616000 (65 days)

### reports.rollupmonth.ttl

Time-to-live for monthly roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Monthly roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 15552000 (180 days)

### reports.auditdata.ttl

Time-to-live for audit data in the Reports keyspace in Cassandra, in seconds. Audit data will be automatically deleted this many seconds after its creation.

Default = 5616000 (65 days)

### events.acknowledged.ttl

Time-to-live for acknowledged system events in the Monitoring keyspace in Cassandra, in seconds. Acknowledged events will be automatically deleted this many seconds after an administrator acknowledges them.

Unacknowledged events are not subject to this timer. The time-to-live countdown on an event record does not begin until an administrator acknowledges the event through the CMC's **Alerts** page or **Node Status** page.

If you want alerts to be deleted immediately after they have been acknowledged, set this property to 1.

**Note** As soon an alert occurs a record of it is written to the Smart Support log (which by default is uploaded to Cloudian Support once a day). So the Smart Support record of alerts persists even after the alerts have been deleted from your system. For more information on Smart Support see **"Smart Support and Diagnostics Feature Overview"** (page 108).

Default = 86400 (one day)

### monitoring.ophistory.ttl

Time-to-live for per-node data repair and cleanup operation status summaries in Cassandra's Monitoring keyspace, in seconds. Each repair and cleanup status summary will be automatically deleted after it has been stored for *monitoring.ophistory.ttl* seconds.

For as long as repair and cleanup status summaries are stored in the Monitoring keyspace, they can be retrieved on a per-node basis by using the **hsstool opstatus** command, with the "-q history" option. This returns the operation history of a specified node. The *monitoring.ophistory.ttl* property controls the maximum length of that retrievable operation history -- for example with *monitoring.ophistory.ttl* at its default value, for each HyperStore node you can retrieve the history of its repair and cleanup operations from the past 90 days. Repair and cleanup operation status information older than that will be deleted.

Default = 7776000 (90 days)

### usage.repair.row.size

When querying Cassandra for the object metadata associated with individual service users, the maximum number of users to retrieve per query.

The *mts.properties* file's "Usage" section has settings for tuning the performance of **usage data repair** operations.

Default = 1000

### usage.repair.column.size

When querying Cassandra for the object metadata associated with individual service users, for each user the maximum number of objects for which to retrieve metadata per query.

Default = 1000

### usage.repair.maxdirtyusers

Maximum number of "dirty" users for whom to verify (and if necessary repair) usage data during a single run of the Admin API *POST /usage/repair/dirtyusers* operation (for details of this API call see the "usage" section in the ***Cloudian HyperStore Admin API Reference***.

This operation is triggered every 12 hours by cron job.

Default = 1000

### usage.rollup.userchunk.size

When performing rollups of usage data, the maximum number of users to take into memory at a time.

Default = 1000

### usage.rollup.usagechunk.size

When performing rollups of usage data, the maximum number of usage records to take into memory at a time.

Default = 1000

### usage.rollup.hour.maxretry

Each time the system performs an hourly rollup of usage data for the hour that just ended, it will check whether the hourly rollup data from the **preceding** hours exists (as it should, unless relevant system services have been down or unreachable). If the hourly rollup data from preceding hours is missing, then the system retries processing the hourly rollups for those hours that are missing the hourly rollup data. The *usage.rollup.hour.maxretry* property sets a maximum on the number of preceding hours to check on and (if needed) perform a retry for.

For example, suppose *usage.rollup.hour.maxretry=6*. With this setting, if the system is for example about to perform the hourly rollup from the 10th hour of the day, it will first check that hourly rollup data exists for the 9th, 8th, 7th, 6th, 5th, and 4th hours of the day -- and if any of those hourly rollups are missing, the system will try again to execute those hourly rollups. After doing so the system will then perform the hourly rollup of the usage data from the 10th hour of the day.

Default = 24

### cloudian.s3.usagerates.enabled

Gets its value from the "Track/Report Usage for Request Rates and Data Transfer Rates" setting on the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### bucketstats.enabled

Gets its value from *common.csv*: **bucketstats_enabled**.

### cloudian.s3.redis.retry

The interval in seconds at which the S3 Service will retry a Redis node that has been unresponsive. If the S3 Service finds a Redis node to be unresponsive the S3 Service will temporarily remove that node from the list of Redis nodes that are available to service requests. At an interval of *cloudian.s3.redis.retry* seconds the S3 Service will retry the Redis node. If it's found to be responsive, the node is added back to the S3 Service's list of available Redis nodes.

This setting is applicable to Redis Credentials nodes and Redis QoS nodes.

Default = 30

### redis.monitor.subscription.check

Gets its value from *common.csv*: **redis_monitor_subscription_check**.

### redis.monitor.primary.pollInterval

The interval at which the backup Redis Monitor instance should check on the health of the primary instance, in seconds. If the primary Redis Monitor instance is unresponsive, the backup instance takes over the monitoring duties.

Default = 5

> **Note** The HyperStore **Redis Monitor Service** monitors Redis cluster health and implements automatic failover of the Redis master node role. For Redis Monitor redundancy, it runs on two of your S3 Service / Admin Service nodes, with the Monitor configured as primary on one node and as backup on the other node.

### redis.credentials.cluster.pollInterval

Interval at which the Redis Monitor application should check the health of the Redis Credentials servers, in seconds. At this interval, the Redis Monitor also checks the S3 Service / Admin Service nodes via JMX to ensure that they are configured to point to the current Redis Credentials master, and updates their configuration if necessary.

Default = 5

### redis.credentials.cluster.client.request.waittime

Maximum time for the Redis Monitor to wait for a JMX connection attempt to a S3 Service / Admin Service node to complete, in seconds. If the connection attempt doesn't complete (with a success or failure result) within this interval, the Redis Monitor marks the S3 Service / Admin Service node as DOWN and writes an INFO level message to *cloudian-redismon.log*. Meanwhile, the connection attempt will continue until completion, and subsequently polling of the S3 Service / Admin Service node will resume at the regular polling interval.

The *redis.credentials.cluster.client.request.waittime* value must be smaller than the *redis-.credentials.cluster.pollInterval* value.

Default = 3

### redis.monitor.alert.limit

This setting throttles the logging output (in *cloudian-redismon.log*) associated with the Redis Monitor's monitoring for conditions of DC partition or "split brain". There is no need to edit this setting unless you are instructed to do so by Cloudian Support.

Default = 100

### redis.monitor.skip.dc.monitoring

For information about this setting please see **"disable dc partition monitoring"** (page 592) and **"enable dc partition monitoring"** (page 593).

Default = true (monitoring is disabled ["skipped"])

### redis.monitor.skip.brain.monitoring

For information about this setting please see **"disable split brain monitoring"** (page 594) and **"enable split brain monitoring"** (page 595).

Default = false (monitoring is enabled [not "skipped"])

### credentials.user.max

Maximum allowed number of S3 credentials per HyperStore user. Each credential is a key pair consisting of a public key (access key) and a private key (secret key). These credentials enable a HyperStore user to access the HyperStore S3 storage system through either the CMC or a third party S3 client.

Inactive credentials count toward this maximum as well as active credentials. Credentials can be created, made active or inactive, and deleted, through either the CMC or the Admin API.

> **Note** If a HyperStore user creates IAM users under their HyperStore account and creates S3 credentials for those IAM users, the IAM users' credentials do **not** count toward the HyperStore user's maximum allowed number of S3 credentials. IAM user credentials are limited separately, by the *credentials.iamuser.max* property.

Default = 5

### credentials.iamuser.max

Maximum allowed number of S3 credentials per IAM user. Each credential is a key pair consisting of a public key (access key) and a private key (secret key). These credentials enable an IAM user to access the Hyper-

Store S3 storage system through a third party S3 client. IAM users cannot access the HyperStore S3 storage system through the CMC.

Inactive credentials count toward this maximum as well as active credentials. Credentials can be created, made active or inactive, and deleted, through either the CMC's **IAM User** section (which is accessible only to HyperStore group administrators or regular users -- not system administrators) or the HyperStore implementation of the Amazon IAM API.

Default = 2

### keystore.pass

Password for the Java keystore file */opt/cloudian/conf/.keystore*. This keystore file stores the Admin Service's pre-generated, self-signed, RSA-based public and private keys for SSL.

Default = adminpass

### secure.transact.alias

Alias identifying the Admin Service's certificate entry within the keystore.

Default = secure

### secure.transact.pass

Password to access the certificate entry that's identified by secure.transact.alias.

Default = private

### admin.auth.realm

Gets its value from *common.csv*: **admin_auth_realm**.

### admin.auth.enabled

Gets its value from *common.csv:* **admin_auth_enabled**.

### admin.secure

Gets its value from *common.csv*: **admin_secure**.

### admin.user.password.length

Maximum allowed character length for users' Cloudian Management Console login passwords.

Default = 64

### user.password.min.length

Gets its value from *common.csv*: **user_password_min_length**.

### user.password.dup.char.ratio.limit

Gets its value from *common.csv*: **user_password_dup_char_ratio_limit**.

### user.password.unique.generations

Gets its value from *common.csv*: **user_password_unique_generations**.

### user.password.rotation.graceperiod

Gets its value from *common.csv*: **user_password_rotation_graceperiod**.

### user.password.rotation.expiration

Gets its value from *common.csv*: **user_password_rotation_expiration**.

### user.password.lock.enabled

Gets its value from *common.csv*: **user_password_lock_enabled**.

### user.password.lock.durationsec

Gets its value from *common.csv*: **user_password_lock_durationsec**.

### user.password.lock.maxfailedattempts

Gets its value from *common.csv*: **user_password_lock_maxfailedattempts**.

### awsmms.proxy.*

These settings are obsolete and will be removed from a future HyperStore release. Do not use.

### admin.whitelist.enabled

Gets its value from *common.csv*: **admin_whitelist_enabled**.

### admin.allow_delete_users_with_buckets

Gets its value from *common.csv*: **allow_delete_users_with_buckets**.

### hyperstore.endport

The **HyperStore Service** listening port to which the S3 Service will submit data operation requests.

Do not edit this setting.

Default = Takes its value from elsewhere within the Puppet manifest structure; default is 19090

### hyperstore.maxthreads.read

Gets its value from *common.csv*: **hyperstore.maxthreads.read**.

### hyperstore.maxthreads.write

Gets its value from *common.csv*: **hyperstore.maxthreads.write**.

### hyperstore.maxthreads.repair

Gets its value from *common.csv*: **hyperstore.maxthreads.repair**.

### hyperstore.maxthreads.delete

Maximum number of simultaneous client threads for one S3 Service node to use on deletes from the HyperStore File System.

Default = 30

### hyperstore.snd.buffer

Socket send buffer size from S3 nodes to HyperStore nodes.

Default = 0

### hyperstore.rcv.buffer

Socket receive buffer size from S3 nodes to HyperStore nodes.

Default = 0

### hyperstore.timeout

Gets its value from *common.csv*: **hyperstore_timeout**.

### hyperstore.connection.timeout

Gets its value from *common.csv*: **hyperstore_connection_timeout**.

### hyperstore.maxtotalconnections

Gets its value from *common.csv:* **hyperstore_maxtotalconnections**, which is controlled by a performance optimization script that runs automatically when you install your cluster or resize your cluster.

### hyperstore.maxperrouteconnections

Gets its value from *common.csv:* **hyperstore_maxperrouteconnections**, which is controlled by a performance optimization script that runs automatically when you install your cluster or resize your cluster

### cassandra.range_repair.max.waiting.time.in_sec

During a Cassandra repair operation each of a node's token ranges are repaired one at a time, sequentially. The system will wait for a maximum of this many seconds for repair of a range to complete. If repair of a range times out by not being completed within this many seconds, the system moves on to repair the next range in sequence. Subsequently, after other ranges are repaired, repair of the range that timed out will be retried a maximum of 3 times. If it still cannot be repaired, the Cassandra repair as a whole will return a Failed status.

Default = 7200

### logcollect.enabled

Gets its value from *common.csv*: **logcollect_enable**.

### phonehome.enabled

The HyperStore Data Collector collects and stores system-wide diagnostic data for your HyperStore system on an ongoing basis. By default this diagnostics data is automatically uploaded to Cloudian Support via the S3 protocol once a day, as part of the Smart Support feature. For more information about this feature -- including what data gets sent to Cloudian Support and how they use it for your benefit -- see **"Smart Support and Diagnostics Feature Overview"** (page 108).

- **If you want diagnostics data automatically uploaded to Cloudian Support via S3 each day**, there's nothing you need to do -- just leave *phonehome.enabled* set to "true" and leave the setting *common.csv*: *phonehome_uri* set to the Cloudian Support S3 URI. This is the default behavior and is the recommended behavior.

- **If you want diagnostics data automatically uploaded each day to an S3 destination other than Cloudian Support**, leave *phonehome.enabled* set to "true" and set the *common.csv*: *phonehome_uri* setting to the desired S3 URI (and also set *common.csv*: *phonehome_{bucket, access_key, secret_key}*).
- **If you do not want diagnostics data automatically uploaded to an S3 destination each day**, set *phonehome.enabled* to "false". This is not recommended.

Even if you choose not to automatically upload the daily diagnostic data to an S3 destination -- that is, even if you set *phonehome.enabled* to "false" -- a diagnostics data file is still generated locally and stored under */var/log/cloudian* on the node on which the HyperStore Monitoring Data Collector runs. (To see which of your nodes is the Data Collector node, go to the CMC's **Cluster Information** page [**Cluster -> Cluster Config -> Cluster Information**] and check for the identity of the "System Monitoring/Cronjob Primary Host".) The "live" diagnostics log -- which is recording the current day's performance statistics -- is named *diagnostics.csv*. The rolled up daily diagnostic packages from previous days -- which include prior days' *diagnostics.csv* files and also various application and transaction logs -- are named *diagnostics_<date/time>_<version>_<region>.tgz*.

> **Note** The deletion of old diagnostics packages is managed by Puppet, as configured by **"cleanup_directories_byage_withmatch_timelimit"** (page 391) in *common.csv*. By default Puppet deletes the diagnostics packages after they are 15 days old. This presumes that you have left the Puppet daemons running in your HyperStore cluster, which is the default behavior. If you do not leave the Puppet daemons running the diagnostics logs will not be automatically deleted. In that case you should delete the old packages manually, since otherwise they will eventually consume a good deal of storage space.

Default = true

### phonehome.uri

Gets its value from *common.csv*: **phonehome_uri**.

### phonehome.{bucket, accessKey, secretKey}

Take their values from *common.csv*: **phonehome_bucket** and the subsequent settings in *common.csv*.

### phonehome.proxy.host

Gets its value from *common.csv*: **phonehome_proxy_host**.

### phonehome.proxy.port

Gets its value from *common.csv*: **phonehome_proxy_port**.

### phonehome.proxy.username

Gets its value from *common.csv*: **phonehome_proxy_username**.

### phonehome.proxy.password

Gets its value from *common.csv*: **phonehome_proxy_password**.

### phonehome.gdpr

Gets its value from *common.csv*: **phonehome_gdpr**.

### phonehome.gdpr.bucket

Gets its value from *common.csv*: **phonehome_gdpr_bucket**.

### sysinfo.uri

S3 URI to which to upload on-demand Node Diagnostics packages, when you use the CMC's Collect Diagnostics function (**Cluster -> Nodes -> Advanced**). By default this is the S3 URI for Cloudian Support, but if you prefer you can set this to a different S3 URI. For an overview of this feature see **"Smart Support and Diagnostics Feature Overview"** (page 108).

Include the HTTP or HTTPS protocol part of the URI (*http://* or *https://*).

Default = https://s3-support.cloudian.com:443

> **Note** If you set *sysinfo.uri* to a URI for your own HyperStore S3 storage system (rather than Cloudian Support), and if your S3 Service is using HTTPS, then your S3 Service's SSL certificate must be a CA-verified, trusted certificate — not a self-signed certificate. By default the Node Diagnostics upload function cannot upload to an HTTPS URI that's using a self-signed certificate. If you require that the upload go to an HTTPS URI that's using a self-signed certificate, contact Cloudian Support.

### sysinfo.{bucket, accessKey, secretKey}

- If you are using the Node Diagnostics upload feature to upload node diagnostic data to **Cloudian Support** via S3, you can leave the *sysinfo.bucket*, *sysinfo.accessKey*, and *sysinfo.secretKey* properties empty. The Node Diagnostics feature will automatically extract the Cloudian Support S3 bucket name and security credentials from your encrypted HyperStore license file. You would only modify these configuration properties if instructed to do so by Cloudian Support.
- If you set *sysinfo.uri* to **your own S3 URI** (rather than the Cloudian Support URI), set the *sysinfo.bucket*, *sysinfo.accessKey*, and *sysinfo.secretKey* properties to the destination bucket name and the applicable S3 access key and secret key.

Default = empty

### sysinfo.proxy.host

Gets its value from *common.csv*: **"phonehome_proxy_host"** (page 400).

> **Note** This property, together with the other *sysinfo.proxy.** properties below, is for using a local forward proxy when sending Node Diagnostics packages to Cloudian Support (or another external S3 destination). By default these properties will inherit the same *common.csv* values that you set for proxying of the daily Smart Support upload (also known as "phone home"). If you want to use a different proxy for sending Node Diagnostics packages -- not the same proxy settings that you use for the phone home feature -- edit the *sysinfo.proxy.** settings directly in *mts.properties.erb*. For example you could change *sysinfo.proxy.host=<%= @phonehome_proxy_host %>* to *sysinfo.proxy.host=proxy2.enterprise.com*.

### sysinfo.proxy.port

Gets its value from *common.csv*: **phonehome_proxy_port**.

### sysinfo.proxy.username

Gets its value from *common.csv*: **phonehome_proxy_username**.

### sysinfo.proxy.password

Gets its value from *common.csv*: **phonehome_proxy_password**.

### s3.client.timeout

When uploading a Node Diagnostics package to an S3 destination such as Cloudian Support, the socket timeout in milliseconds.

Default = 1800000

### s3.upload.part.minsize

When HyperStore uses Multipart Upload to transmit a Node Diagnostics package to an S3 destination such as Cloudian Support, each of the parts will be this many bytes or larger — with the exception of the final part, which may be smaller. For example, if Multipart Upload is used for an 18MiB object, and the configured minimum part size is 5MiB, the object will be transmitted in four parts of size 5MiB, 5MiB, 5MiB, and 3MiB.

Default = 5242880 (5MiB)

### s3.upload.part.threshold

When HyperStore transmits a Node Diagnostics package to an S3 destination such as Cloudian Support, it uses Multipart Upload if the package is larger than this many bytes.

Default = 16777216 (16MiB)

### cloudian.protection.policy.max

Maximum number of bucket protection policies (storage policies) that the system will support. Policies with status "Active", "Pending", or "Disabled" count toward this system limit.

If the policy maximum has been reached, you will not be able to create new policies until you either delete existing policies or increase the value of *cloudian.protection.policy.max*.

Default = 25

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → MaxProtectionPolicies)

For more information about storage policies, see **"Storage Policies Feature Overview"** (page 127).

### cloudian.tiering.useragent

Gets its value from *common.csv*: **cloudian_tiering_useragent**.

### cloudian.s3.tiering.client.maxconnections

Gets its value from *common.csv*: **cloudian_s3_max_threads**.

### cloudian.s3.ssec.usessl

This setting controls whether the S3 servers will require that incoming S3 requests use HTTPS (rather than regular HTTP) connections when the request is using Server Side Encryption with Customer-provided encryption keys (SSE-C). Leaving this setting at its default of "true" -- so that the S3 servers require HTTPS connections for such requests -- is the recommended configuration. The only circumstance in which you might set this to "false" is if:

- You are using a load balancer in front of your S3 servers -- and the load balancer, when receiving an incoming HTTPS request from clients, terminates the SSL and uses regular HTTP to connect to an S3 server over your internal network.
- You trust your internal network to safely transport users' encryption keys from the load balancer to the S3 servers over regular HTTP.

For background information about HyperStore support for server-side encryption (SSE and SSE-C), see **"Server-Side Encryption"** (page 150).

Default = true

### util.awskmsutil.region

Amazon Web Services (AWS) service region to use if you are configuring your HyperStore system to support AWS-KMS as a method of server-side encryption. For complete instructions see **"Using AWS KMS"** (page 160).

Default = us-east-1

### cloudian.s3.torrent.tracker

If you want your service users to be able to use BitTorrent for object retrieval, use this property to specify the URL of a BitTorrent "tracker" (a server that keeps track of the clients that have retrieved a particular object and makes this information available to other clients retrieving the object). This can be a tracker that you implement yourself or one of the many public BitTorrent trackers. HyperStore itself does not provide a tracker.

This must be a single URL, not a list of URLs.

The tracker URL that you specify here will be included in the torrent file that the HyperStore S3 Server returns to clients when they submit a *GetObjectTorrent* request.

### filesvc.secure

Gets its value from *common.csv*: **file_secure_enabled**.

### filesvc.api.root.path

Gets its value from *common.csv*: **file_api_path**.

### cloudian.hypersearch.enabled

Gets its value from *common.csv*: **search_enabled**.

### cloudian.hypersearch.hosts

Gets its value from *common.csv*: **search_fqdn**.

### cloudian.hypersearch.auth.username

Gets its value from *common.csv*: **search_auth_username**.

### cloudian.hypersearch.auth.password

Gets its value from *common.csv*: **search_auth_pass**.

### cloudian.hypersearch.ssl.enabled

Gets its value from *common.csv*: **search_secure_enabled**.

### cloudian.hypersearch.ssl.truststore.path

xxx

### cloudian.hypersearch.ssl.truststore.pass

Gets its value from *common.csv*: **search_truststore_pass**.

### cloudian.hypersearch.legacyhttp

This setting is applicable only if you had been using the legacy Elasticsearch integration feature from a HyperStore version older than 7.5.1, and you had configured your system to transmit object metadata to a custom HTTP destination rather than to an Elasticsearch cluster. On upgrade to HyperStore 7.5.1 this legacy functionality will no longer work as is.

If this is your circumstance, and if you want to keep using the legacy functionality to transmit object metadata to a custom HTTP destination, consult with Cloudian Support. Cloudian Support will guide you on how to appropriately reconfigure your HyperStore 7.5.1 system so that you can continue to use this functionality for a while longer.

**Note** In HyperStore 8.0 this functionality will no longer be supported.

### cloudian.hypersearch.legacyURI

This setting is applicable only if you had been using the legacy Elasticsearch integration feature from a HyperStore version older than 7.5.1, and you had configured your system to transmit object metadata to a custom HTTP destination rather than to an Elasticsearch cluster. On upgrade to HyperStore 7.5.1 this legacy functionality will no longer work as is.

If this is your circumstance, and if you want to keep using the legacy functionality to transmit object metadata to a custom HTTP destination, consult with Cloudian Support. Cloudian Support will guide you on how to appropriately reconfigure your HyperStore 7.5.1 system so that you can continue to use this functionality for a while longer.

**Note** In HyperStore 8.0 this functionality will no longer be supported.

### cloudian.hypersearch.port

Gets its value from *common.csv*: **search_port**.

### cloudian.hypersearch.fqdn

Gets its value from *common.csv*: **search_fqdn**.

### cloudian.hypersearch.prefix

Gets its value from *common.csv*: **search_prefix**.

### search.indexer.bulkPut.maxObjects

Gets its value from *common.csv*: **search_indexer_bulkPut_maxObjects**.

### hss.bring.back.wait, hss.timeout.*, and hss.fail.*

In your storage cluster the **S3 Service** runs on each node and so too does the **HyperStore Service**. S3 requests incoming to your cluster are distributed among the S3 Service instances, and in processing an S3 request an S3 Service instance sends write or read requests to the HyperStore Service instances on multiple nodes (such as when storing a new object in accordance with a 3X replication storage policy, for example).

The settings in the "Node Status Configuration" section of *mts.properties.erb* configure a feature whereby the S3 Service on any node will mark the HyperStore Service on any node as being "Down" if recent requests to that HyperStore Service node have failed at a rate in excess of defined thresholds. When an S3 Service node marks a HyperStore Service node as Down, that S3 Service node will temporarily stop sending requests to that HyperStore Service node. This prevents a proliferation of log error messages that would likely have resulted if requests continued to be sent to that HyperStore Service node, and also allows for the implementation of fall-back consistency levels in the case of storage policies configured with **"Dynamic Consistency Levels"** (page 77).

The configurable thresholds in this section are applied by each individual S3 Service node -- so that each individual S3 Service node makes its own determination of when a problematic HyperStore Service node should be marked as Down.

An S3 Service node will mark a HyperStore Service node as Down in either of these conditions:

- The number of timeout error responses from a HyperStore Service node has exceeded ***hss.timeout.count.threshold*** (default = 10) over a period of ***hss.timeout.time.threshold*** number of seconds (default = 300) and also the percentage of error responses of any type from that HyperStore Service node has exceeded ***hss.fail.percentage.threshold*** (default = 50) over the past ***hss.fail.percentage.period*** number of seconds (default = 300).
- The number of other types of error responses from a HyperStore Service node has exceeded ***hss.fail.count.threshold*** (default = 10) over a period of ***hss.fail.time.threshold*** number of seconds (default = 300) and also the percentage of error responses of any type from that HyperStore Service node has exceeded ***hss.fail.percentage.threshold*** (default = 50) over the past ***hss.fail.percentage.period*** number of seconds (default = 300).

So by default an S3 Service node will mark a HyperStore Service node as Down if during a five minute period the HyperStore Service node has returned either more than 10 timeout responses or more than 10 error responses of other types, while during that same five minute period more than half the requests that the S3 Service node has sent to that HyperStore Service node have failed.

Note that these triggering conditions are based on a combination of number of error responses and percentage of error responses from the problematic HyperStore Service node. This approach avoids marking a HyperStore Service node as down in circumstances when a high percentage of a very small number of requests fail, or when the number of failed requests is sizable but constitutes only a small percentage of the total requests.

Once an S3 Service node has marked a HyperStore Service node as Down, that Down status will persist for ***hss.bring.back.wait*** number of seconds (default = 300) before the Down status is cleared and that S3 Service node resumes sending requests to that HyperStore Service node.

> **Note** If the S3 Service node is restarted during this interval, then the Down status for that HyperStore Service node will be lost and the S3 Service node upon restarting will resume sending requests to that HyperStore Service node.

**Note** There is **special handling in the event that a HyperStore Service node returns a "Connection Refused" error** to an S3 Service node (such as would happen if the HyperStore Service was stopped on the target node). In this case the S3 Service node immediately marks that HyperStore Service node as being down, and will then resume sending requests to that HyperStore Service node after a wait period of 15 seconds. This behavior is not configurable.

### hyperstore.proactiverepair.queue.max.time

When eventual consistency for writes is used in the system -- that is, if you have storage policies for which you have configured the write consistency level to be something less strict than ALL -- S3 writes may succeed in the system even in circumstances when one or more write endpoints is unavailable. When this happens the system's proactive repair feature queues information about the failed endpoint writes, and automatically executes those writes later -- on an hourly interval (by default), without operation intervention. For more information about proactive repair see Proactive Repair.

The proactive repair feature's queueing mechanism entails writing metadata to the Metadata DB, and that metadata is subsequently removed when the endpoint writes are executed by proactive repair. To avoid overburdening the Metadata DB with proactive queueing metadata it's best if a cap be placed on how long the queueing can go on for in a given instance of a write endpoint being unavailable. The *hyperstore.proactiverepair.queue.max.time* property sets this cap, in minutes.

If a node has been unavailable for more than *hyperstore.proactiverepair.queue.max.time* minutes, the system stops writing to the proactive repair queue for that node. The next time the data center receives an incoming S3 PUT request for which the downed node is a write endpoint, an error is logged in the S3 application log, and an alert is generated in the CMC. As indicated by the alert, in this circumstance -- where the node has been down for more than *hyperstore.proactiverepair.queue.max.time* minutes -- then after the node comes back online you need to wait for proactive to complete on the node (you can monitor this in the CMC's **Repair Status** page [**Cluster -> Repair Status**]) and then you **must manually initiate a full repair on the node** (see **hsstool repair** and **hsstool repairec**).

Note that once the node is back up, the timer is reset to 0 in terms of counting against the *hyperstore.proactiverepair.queue.max.time* limit. So if that node subsequently goes down again, proactive repair queueing would again occur for that node for up to *hyperstore.proactiverepair.queue.max.time* minutes.

Default = 240

**Note** To disable this limit -- so that there is no limit on the time for which proactive repair queueing metadata can build up for a node that's unavailable -- set *hyperstore.proactiverepair.queue.max.time* to 0.

### cloudian.s3.enablesharedbucket

To enable the HyperStore S3 API extension that allows an S3 user to list all the buckets that have been shared with him or her, set this property to *true*.

Default = false

**Note** For information about the relevant S3 API call and how to use the extension, see *ListBuckets* in the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***.

### cloudian.userid.length

Gets its value from *common.csv*: **cloudian_userid_length**.

### cloudian.iam.max.groups

Gets its value from *common.csv*: **iam_max_groups**.

### cloudian.iam.max.groups.per.user

Gets its value from *common.csv*: **iam_max_groups_per_user**.

### mfa.totp.issuer

Gets its value from *common.csv*: **mfa_totp_issuer**.

### cloudian.fips.enabled

Gets its value from *common.csv: fips_enabled*. For information about the *fips_enabled* setting see **"FIPS Support"** (page 168).

## 9.8. mts-ui.properties.erb

The *mts-ui.properties* file configures the Cloudian Management Console server (CMC). On each of your HyperStore nodes, the file is located at the following path by default:

```
/opt/tomcat/webapps/Cloudian/WEB-INF/classes/mts-ui.properties
```

**Do not directly edit the *mts-ui.properties* file on individual HyperStore nodes**. Instead, if you want to make changes to the settings in this file, edit the configuration template file *mts-ui.properties.erb* on the Configuration Master node:

```
/etc/cloudian-7.5.2-puppet/modules/cmc/templates/mts-ui.properties.erb
```

Some *mts-ui.properties.erb* properties get their values from settings in **common.csv** or from settings that you can control through the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**). In the *mts-ui.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like *<%= … %>*. In the property documentation below, the description of such a property indicates "Gets its value from *<location>: <setting>*." Rather than editing such a property in *mts-ui.properties.erb*, edit the property's corresponding *<location>: <setting>* instead (most often this will be a setting in *common.csv*).

*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can edit this configuration file with this command:

```
$ hspkg config -e mts-ui.properties.erb
```

Specify just the configuration file name, not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the **standard keystrokes supported by *vi*** to make and save changes to the file.

**IMPORTANT !** If you do make edits to *mts-ui.properties.erb*, be sure to push your edits to the cluster and restart the CMC to apply your changes. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 382).

## 9.8.1. mts-ui.properties.erb Details

### 9.8.1.1. Settings in mts-ui.properties.erb

#### admin.host

Gets its value from *common.csv*: **cmc_admin_host_ip**

#### admin.port

Gets its value from *common.csv*: *ld_cloudian_s3_admin_port*, which is controlled by the installer.

#### admin.secure

Gets its value from *common.csv*: **admin_secure**.

#### admin.secure.port

Gets its value from *common.csv*: **cmc_admin_secure_port**.

#### admin.secure.ssl

Gets its value from *common.csv*: **cmc_admin_secure_ssl**.

#### admin.conn.timeout

The connection timeout for the CMC to use as a client to the Admin Service, in milliseconds. If the CMC cannot connect to the Admin Service within this many milliseconds, the connection attempt times out and the CMC interface displays an error message.

**Note** To provide any of its functions for any type of user, the CMC must successfully connect to the Admin Service.

Default = 10000 (10 seconds)

#### iam.enabled

Gets its value from *common.csv*: **iam_service_enabled**.

#### iam.host

Gets its value from *common.csv*: **iam_service_endpoint**.

#### iam.port

Gets its value from *common.csv*: **iam_port**.

### iam.secure.port

Gets its value from *common.csv*: **iam_secure_port**.

### iam.secure

Gets its value from *common.csv*: **iam_secure**.

### iam.socket.timeout

If during an HTTP/S connection with the IAM Service this many milliseconds pass without any data being passed back by the IAM Service, the CMC will drop the connection.

Default = 30000

### iam.max.retry

If when trying to initiate an IAM request the CMC fails in an attempt to connect to the IAM Service, the CMC will retry this many times before giving up.

Default = 3

### web.secure

Gets its value from *common.csv*: **cmc_web_secure**.

### web.secure.port

Gets its value from *common.csv*: **cmc_https_port**.

### web.nonsecure.port

Gets its value from *common.csv*: **cmc_http_port**.

### storageuri.ssl.enabled

Gets its value from *common.csv*: **cmc_storageuri_ssl_enabled**.

### crr.external.enabled

Gets its value from *common.csv*: **cmc_crr_external_enabled**.

### path.style.access

Gets its value from *common.csv*: **path_style_access**.

### application.name

Gets its value from *common.csv*: **cmc_application_name**.

### s3.client.timeout

Socket timeout on requests from the CMC to the S3 Service, in milliseconds.

Default = 1800000

### s3.upload.part.minsize

When the CMC uses Multipart Upload to transmit an object to the S3 Service, each of the parts will be this many bytes or larger — with the exception of the final part, which may be smaller. For example, if Multipart Upload is used for an 18MiB object, and the configured minimum part size is 5MiB, the object will be transmitted in four parts of size 5MiB, 5MiB, 5MiB, and 3MiB.

Default = 5242880 (5MiB)

### s3.upload.part.threshold

When a CMC user uploads an object larger than this many bytes, the CMC uses Multipart Upload to transmit the object to the S3 Service, rather than PUT Object.

Default = 16777216 (16MiB)

### s3.qos.bucketcounter.enabled

Gets its value from *common.csv*: **s3_perbucket_qos_enabled**.

### query.maxrows

For the CMC's **Usage By Users & Groups** page, the maximum number of data rows to retrieve when processing a usage report request.

Default = 100000

### page.size.default

For the CMC's **Usage By Users & Groups** page, for usage report pagination, the default number of table rows to display on each page of a tabular report.

Default = 10

### page.size.max

For the CMC's **Usage By Users & Groups** page, for usage report pagination, the maximum number of table rows that users can select to display on each page of a tabular report.

Default = 100

### list.multipart.upload.max

In the CMC's **Objects** page, when a user is uploading multiple objects each of which is large enough to trigger the use of the S3 multipart upload method, the maximum number of multipart upload objects for which to simultaneously display upload progress.

For example, with the default value of 1000, if a user is concurrently uploading 1005 objects that require the use of the multipart upload method, the CMC's **Objects** page will display uploading progress for 1000 of those objects.

Default = 1000

### graph.datapoints.max

For the CMC's **Usage By Users & Groups** page, the maximum number of datapoints to include within a graphical report.

Default = 1000

### csv.rows.max

For the CMC's **Usage By Users & Groups** page, the maximum number of rows to include within a comma-separated value report.

Default = 1000

### fileupload.abort.max.hours

When a very large file is being uploaded through the CMC, the maximum number of hours for the CMC to wait for the S3 file upload operation to complete. If this maximum is reached, the upload operation is aborted.

Default = 3

### license.request.email

Email address to which to send Cloudian license requests. This address is used in a request license information link in the CMC interface.

Default = cloudian-license@cloudian.com

### admin.auth.user

Gets its value from *common.csv*: **admin_auth_user**.

### admin.auth.pass

Gets its value from *common.csv*: **admin_auth_pass**.

### admin.auth.realm

Gets its value from *common.csv*: **admin_auth_realm**.

### user.password.min.length

Gets its value from *common.csv*: **user_password_min_length**.

### user.password.dup.char.ratio.limit

Gets its value from *common.csv*: **user_password_dup_char_ratio_limit**.

### user.password.unique.generations

Gets its value from *common.csv*: **user_password_unique_generations**.

### user.password.rotation.graceperiod

Gets its value from *common.csv*: **user_password_rotation_graceperiod**.

### user.password.rotation.expiration

Gets its value from *common.csv*: **user_password_rotation_expiration**.

### mfa.enforced

Gets its value from *common.csv*: **mfa_enforced**.

### acl.grantee.public

In the CMC UI dialogs that let CMC users specify permissions on S3 buckets, folders, or files, the label to use for the ACL grantee "public". If the *acl.grantee.public* property is not set in *mts-ui.properties*, then the system instead uses the *acl.grantee.public* value from your *resources_xx_XX.properties* files (for example, for the U.S. English version of the UI, the value is in *resources_en_US.properties*).

Default = commented out (value is taken from resource file[s])

### acl.grantee.cloudianUser

In the CMC UI dialogs that let CMC users specify permissions on S3 buckets, folders, or files, the label to use for the ACL grantee "all Cloudian HyperStore service users". If the *acl.grantee.cloudianUser* property is not set in *mts-ui.properties*, then the system instead uses the *acl.grantee.cloudianUser* value from your *resources_xx_XX.properties* files (for example, for the U.S. English version of the UI, the value is in *resources_en_US.-properties*).

Default = commented out (value is taken from resource file[s])

### session.timedout.url

URL of page to display if a CMC user's login session times out.

If this value is not set in *mts-ui.properties*, the behavior defaults to displaying the CMC Login screen if the user's session times out.

Default = commented out (CMC Login screen displays)

### admin.manage_users.enabled

This setting controls whether the **Manage Users** function (**Users & Groups -> Manage Users**) will be enabled in the CMC GUI.

Options are:

- true — This function will display for users logged in as a system administrator or group administrator. For group admins this function is restricted to their own group.
- false — This function will not display for any users. If you set this to "false" then the **Manage Users** functionality as a whole is disabled and the more granular *admin.manage_users.*.enabled* properties below are ignored.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

> **Note** If you want to enable some aspects of the **Manage Users** function and not others, you can have *admin.manage_users.enabled* set so that the function is enabled for your desired user types, and then use the granular *admin.manage_users.*.enabled* properties below to enable/disable specific capabilities.

### admin.manage_groups.create.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to create new groups.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_groups.edit.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to edit an existing group's profile and service attributes.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.

  **Note** Even when this capability is enabled for group admins, they will not be able to perform certain group-related actions that are reserved for system admins, such as setting QoS controls for the group as a whole or assigning a default rating plan for the group. Group admins' privileges will be limited to changing their group description and changing the default user QoS settings for the group. The latter capability is controlled by a more granular configuration property *admin.manage_groups.user_qos_groups_default.enabled* (below), which defaults to "true".

- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_groups.delete.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to delete a group.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_users.viewuserdata.enabled

Gets its value from *common.csv*: **cmc_view_user_data**.

### admin.manage_users.edit.user_credentials.enabled

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to change users' CMC login passwords and to view and manage user's S3 access credentials.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

> **Note** Regardless of how you configure the *admin.manage_users.edit.user_credentials.enabled* setting, regular users can manage their own CMC login password and S3 access credentials, in the **Security Credentials** page of the CMC.

### admin.manage_users.edit.user_qos.enabled

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to set Quality of Service (QoS) controls for specific users.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

> **Note** This setting is relevant only if the *admin.manage_users.enabled* and *admin.manage_users.edit.enabled* settings are enabled.

### admin.manage_groups.enabled

This setting controls whether the **Manage Groups** function (**Users & Groups -> Manage Groups**) will be enabled in the CMC GUI.

Options are:

- true — This function will display for users logged in as a system administrator or group administrator. For group admins this function is restricted to their own group.
- false — This function will not display for any users. If you set this to "false" then the **Manage Groups** functionality as a whole is disabled and the more granular *admin.manage_groups.*.enabled* properties below are ignored.

- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

> **Note** If you want to enable some aspects of the **Manage Groups** function and not others, you can have *admin.manage_groups.enabled* set so that the function is enabled for your desired user types, and then use the granular *admin.manage_groups.*.enabled* properties below to enable/disable specific capabilities.

### admin.manage_groups.create.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to create new groups.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_groups.edit.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to edit an existing group's profile and service attributes.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.

  > **Note** Even when this capability is enabled for group admins, they will not be able to perform certain group-related actions that are reserved for system admins, such as setting QoS controls for the group as a whole or assigning a default rating plan for the group. Group admins' privileges will be limited to changing their group description and changing the default user QoS settings for the group. The latter capability is controlled by a more granular configuration property *admin.manage_groups.user_qos_groups_default.enabled* (below), which defaults to "true".

- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_groups.delete.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to delete a group.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### admin.manage_groups.user_qos_groups_default.enabled

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to set default Quality of Service (QoS) controls for users within a specific group.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

> **Note** This setting is relevant only if the *admin.manage_groups.enabled* and *admin.manage_groups.edit.enabled* settings are enabled.

### account.profile.writeable.enabled

This setting controls whether a user can edit his or her own account profile information in the **Account Profile** section of the CMC GUI (accessible through a drop-down list under the user's login name in the upper right of the GUI). For user types for which this editing capability is not enabled, account profile information will be read-only.

Options are:

- true — This capability will be enabled for all user types (system administrator, group administrator, and regular user).
- false — This capability will be disabled for all user types.
- SystemAdmin — This capability will be enabled only for users logged in as a system administrator.
- GroupAdmin — This capability will be enabled only for users logged in as a group administrator.
- User — This capability will be enabled only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### account.credentials.enabled

This setting controls whether the **Security Credentials** function will be enabled in the CMC GUI (accessible through a drop-down list under the user's login name in the upper right of the GUI).

Options are:

- true — This function will display for all user types (system administrator, group administrator, and regular user).
- false — This function will not display for any user types. If you set this to "false" then the Security Credentials functionality as a whole is disabled and the more granular account.credentials.*.enabled properties below are ignored.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.
- User — This function will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this function — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

**Note** If you want to enable some aspects of the **Security Credentials** function and not others, you can have *account.credentials.enabled* set so that the function is enabled for your desired user types, and then use the granular *account.credentials.*.enabled* properties below to enable/disable specific capabilities.

### account.credentials.access.enabled

Within the **Security Credentials** function in the CMC GUI, this setting enables or disables the capability of CMC users to view and change their own S3 storage access keys.

Options are:

- true — This capability will display for all user types (system administrator, group administrator, and regular user).
- false — This capability will not display for any user types.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.
- User — This capability will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this capability — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### account.credentials.signin.enabled

Within the **Security Credentials** function in the CMC GUI, this setting enables or disables the capability CMC users to change their own CMC login password.

Options are:

- true — This capability will display for all user types (system administrator, group administrator, and regular user).
- false — This capability will not display for any user types.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

- User — This capability will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this capability — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### account.activity.enabled

This setting controls whether the **Account Activity** function (**Users & Groups -> Account Activity**) will be enabled in the CMC GUI.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### usage.enabled

This setting controls whether the **Usage By Users and Groups** function (**Analytics -> Usage By User & Group**) will be enabled in the CMC GUI.

Options are:

- true — This function will display for all user types (system administrator, group administrator, and regular user).
- false — This function will not display for any user types.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.
- User — This function will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this function — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### security.serviceinfo.enabled

This setting controls whether HyperStore's S3 service endpoints will display for group admins and regular users in the CMC's **Security Credentials** page (accessible through a drop-down list under the user's login name in the upper right of the GUI).

Options are:

- true — S3 service endpoints will display for group admins and regular users in the CMC's **Security Credentials** page.
- false — S3 service endpoints will not display for group admins or regular users in the CMC's **Security Credentials** page.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

**Note** For HyperStore system admins, the S3 service endpoint information displays in the **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**). The *security.serviceinfo.enabled* property has no effect on this display.

### login.languageselection.enabled

Gets its value from *common.csv*: **cmc_login_languageselection_enabled**.

### login.grouplist.enabled

Gets its value from *common.csv*: **cmc_login_grouplist_enabled**.

### login.grouplist.admincheckbox.enabled

Gets its value from *common.csv*: **cmc_login_grouplist_admincheckbox_enabled**.

### login.banner.*

-- These settings take their values from the *cmc_login_banner_** settings in *common.csv*; use those settings instead. For information about using those settings see **"Configuring a Login Page Acknowledgment Gate"** (page 234).

### csrf.origin.check.enabled

Gets its value from *common.csv*: **cmc_csrf_origin_check_enabled**.

### csrf.target.origin.allowlist

Gets its value from *common.csv*: **cmc_csrf_origin_allowlist**.

### grouplist.enabled

Gets its value from *common.csv*: **cmc_grouplist_enabled**.

### grouplist.size.max

Gets its value from *common.csv*: **cmc_grouplist_size_max**.

### error.stacktrace.enabled

Throughout the CMC UI, when an exception occurs and an error page is displayed to the user, include on the error page a link to a stack trace.

Options are:

- true — Stack trace links will display for all user types (system administrator, group administrator, and regular user).
- false — Stack trace links will not display for any user types.
- SystemAdmin — Stack trace links will display only for users logged in as a system administrator.
- GroupAdmin — Stack trace links will display only for users logged in as a group administrator.
- User — Stack trace links will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this feature — for example, "SystemAdmin,GroupAdmin".

Default = false

### admin.whitelist.enabled

Gets its value from *common.csv*: **admin_whitelist_enabled**.

### sso.enabled

Gets its value from *common.csv*: **cmc_sso_enabled**.

### sso.shared.key

Gets its value from *common.csv*: **cmc_sso_shared_key**.

### sso.tolerance.millis

Maximum allowed variance between the CMC server time and the timestamp submitted in a client request invoking the "auto-login with one way hash" method of single sign-on access to the CMC, in milliseconds. If the variance is greater than this, the request is rejected. This effectively serves as a request expiry mechanism.

Default = 3600000

### sso.cookie.cipher.key

Gets its value from *common.csv*: **cmc_sso_cookie_cipher_key**.

### bucket.storagepolicy.showdetail.enabled

Within the **Bucket Properties** function in the CMC GUI, this setting enables or disables the display of a "Storage Policy" tab that provides information about the storage policy being used by the bucket. This information includes the storage policies replication factor or erasure coding k+m configuration..

Options are:

- true — This tab will display for all user types (system administrator, group administrator, and regular user).
- false — This tab will not display for any user types.
- SystemAdmin — This tab will display only for users logged in as a system administrator.
- GroupAdmin — This tab will display only for users logged in as a group administrator.
- User — This tab will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to display this tab — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

### purgebucket.enabled

Gets its value from *common.csv*: **cmc_purgebucket_enabled**.

### bucket.tiering.enabled

Gets its value from "Enable Auto Tiering" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### tiering.perbucketcredentials.enabled

Gets its value from "Enable Per Bucket Credentials" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### tiering.customendpoint.enabled

Gets its value from "Enable Custom Endpoint" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### bucket.tiering.default.destination.list

Gets its value from *common.csv*: **cmc_bucket_tiering_default_destination_list**.

### bucket.tiering.custom.url

Gets its value from "Default Tiering URL" in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

### puppet.master.licensefile

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

### puppet.master.fileupdate.location

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

### local.ssh.privateKey

Private key to use when the CMC connects via SSH to the Configuration Master node or to other HyperStore nodes when implementing node management functions. The Cloudian installation script automatically populates this setting.

### local.ssh.passphrase

Pass phrase to use when the CMC connects via SSH to the Configuration Master node or to other HyperStore nodes when implementing node management functions. The Cloudian installation script automatically populates this setting.

### local.ssh.applianceKey

Private key to use when the CMC connects via SSH to a new HyperStore Appliance node when the appliance node is being added to an existing system. The Cloudian installation script automatically populates this setting.

### local.temp.dir

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

### remote.ssh.user

The user as which to connect via SSH to the Configuration Master node or other HyperStore nodes.

Default = root

### remote.ssh.port

The port to which to connect via SSH to the Configuration Master node or other HyperStore nodes.

Default = 22

### offload.services.node.options

Used internally by the CMC when invoking the installer script for certain operations. Do not edit.

Default = -m

### uninstall.node.options

Used internally by the CMC when invoking the installer script for certain operations. Do not edit.

Default = -u -r

### blink.disk.intervalsec

Gets its value from *common.csv*: **blink_disk_intervalsec**.

### blink.appliance.intervalsec

Gets its value from *common.csv*: **blink_appliance_intervalsec**.

### filesvc.secure

Gets its value from *common.csv*: **file_secure_enabled**.

### filesvc.api.root.path

Gets its value from *common.csv*: **file_api_path**.

### hypersearch.enabled

Gets its value from *common.csv*: **search_enabled**.

### hypersearch.hosts

Gets its value from *common.csv*: **search_fqdn**.

### hypersearch.auth.username

Gets its value from *common.csv*: **search_auth_username**.

### hypersearch.auth.password

Gets its value from *common.csv*: **search_auth_pass**.

### hypersearch.ssl.enabled

Gets its value from *common.csv*: **search_secure_enabled**.

### hypersearch.fqdn

Gets its value from *common.csv*: **search_fqdn**.

### hypersearch.prefix

Gets its value from *common.csv*: **search_prefix**.

### proactive.repair.queue.warning.enabled

The CMC Dashboard has a feature whereby it displays a "Long proactive repair queue" warning if the **proactive repair queue** for a particular node has more than 10,000 objects in it. This feature requires the Dashboard to retrieve proactive repair queue length data from Cassandra, each time the Dashboard page is loaded in your browser. This can sometimes result in slower loading times for the Dashboard.

The *proactive.repair.queue.warning.enabled* property enables and disables this Dashboard feature. If this property is set to "false", then the Dashboard does not retrieve proactive repair queue length data from Cassandra and does not display any warnings in regard to proactive repair queue length.

Default = false

### cloudian.userid.length

Gets its value from *common.csv*: **cloudian_userid_length**.

## 9.9. survey.csv (Cluster Survey File)

This file resides in your installation staging directory for the life of your HyperStore system. The survey file is automatically updated by the system if you subsequently use the CMC to add more nodes to your cluster; and it is automatically copied to your new installation staging directory when you execute a HyperStore version upgrade.

> **Note** The survey file must be kept in the installation staging directory, not in a different directory. Do not delete or move the survey file.

The survey file contains one line for each HyperStore host in your cluster (including the Configuration Master host), with each line using the format below.

```
<regionname>,<hostname>,<ip4-address>,<datacenter-name>,<rack-name>[,<internal-interface>]
```

- *<regionname>* — HyperStore service region in which the host is located. The HyperStore system supports having multiple service regions with each region having its own independent storage cluster and S3 object inventory, and with S3 application users able to choose a storage region when they create storage buckets. Even if you will have only one region you must give it a name. The maximum allowed length is 52 characters. The only allowed character types are lower case ASCII alphanumerical characters and dashes (a-z0-9 and dashes). Do not include the string "s3" in the region name. Make sure the region name matches the region string that you use in your S3 endpoints in your **"DNS Set-Up"** (page 50).
- *<hostname>* — Short hostname of the host (as would be returned if you ran the *hostname -s* command on the host). This must be the node's short hostname, not an FQDN.

  > **Note** Do not use the same short hostname for more than one node in your entire HyperStore system. Each node must have a unique short hostname within your entire HyperStore system, even in the case of nodes in different data centers or service regions that have different domains. For

example, in your HyperStore system do not have two nodes with the same short hostname *vega* for which the FQDN of one is *vega.east.com* and the FQDN of the other is *vega.west.com*.

- *<ip4-address>* — IP address (v4) that the hostname resolves to. Do not use IPv6. This should be the IP address associated with the host's default, external interface -- not an internal interface.
- *<datacenter-name>* — Name of the data center in which the host machine is located. The maximum allowed length is 256 characters. The only allowed character types are ASCII alphanumerical characters and dashes (A-Za-z0-9 and dashes).
- *<rack-name>* — Name of the server rack in which the host machine is located. The maximum allowed length is 256 characters. The only allowed character types are ASCII alphanumerical characters and dashes (A-Za-z0-9 and dashes).

  **Note** Within a data center, use the same "rack name" for all of the nodes, even if some nodes are on different physical racks than others. For example, if you have just one data center, all the nodes must use the same rack name. And if you have two data centers named DC1 and DC2, all the nodes in DC1 must use the same rack name as the other nodes in DC1; and all the nodes in DC2 must use the same rack name as the other nodes in DC2.

- *[<internal-interface>]* — Use this field only for hosts that will use a different network interface for internal cluster traffic than the rest of the hosts in the cluster do. For example, if most of your hosts will use "eth1" for internal cluster traffic, but two of your hosts will use "eth2" instead, use this field to specify "eth2" for each of those two hosts, and leave this field empty for the rest of the hosts in your survey file. (Later in the installation procedure you will have the opportunity to specify the default internal interface for the hosts in your cluster -- the internal interface used by all hosts for which you do not specify the *internal-interface* field in your survey file.) If all of your hosts use the same internal network interface — for example if all hosts use "eth1" for internal network traffic — then leave this field empty for all hosts in the survey file.

  **Note** Cassandra, Redis, and the HyperStore Service are among the services that will utilize the internal interface for intra-cluster communications.

The example survey file below is for a single-node HyperStore installation:

```
region1,arcturus,65.10.2.1,DC1,RAC1
```

This second example survey file is for a three-node HyperStore cluster with just one service region, one data center, and one rack:

```
tokyo,cloudian-vm7,65.10.1.33,DC1,RAC1
tokyo,cloudian-vm8,65.10.1.34,DC1,RAC1
tokyo,cloudian-vm9,65.10.1.35,DC1,RAC1
```

This third example survey file below is for a HyperStore installation that spans two regions, with the first region comprising two data centers and the second region comprising just one data center. Two of the hosts use a different network interface for internal network traffic than all the other hosts do.

```
boston,hyperstore1,65.1.0.1,DC1,RAC1
boston,hyperstore2,65.1.0.2,DC1,RAC1
boston,hyperstore3,65.1.0.3,DC1,RAC1
boston,hyperstore4,66.2.0.1,DC2,RAC1
boston,hyperstore5,66.2.0.2,DC2,RAC1
```

```
chicago,hyperstore6,68.3.0.1,DC3,RAC1
chicago,hyperstore7,68.3.0.2,DC3,RAC1
chicago,hyperstore8,68.3.2.1,DC3,RAC1,eth2
chicago,hyperstore9,68.3.2.2,DC3,RAC1,eth2
```

# 9.10. Other Configuration Files

The tables below briefly describe additional HyperStore configuration files that reside in the configuration directories on the Configuration Master node. **Under typical circumstances you should not need to manually edit these files.**

All of these files are in sub-directories under the */etc/cloudian-<version>-puppet* directory. For brevity, in the section headings that follow */etc/cloudian-<version>-puppet* is replaced with "..." and only the sub-directory is specified.



*If you are using the **HyperStore Shell***

If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can edit any of these configuration files with this command:

```
$ hspkg config -e <filename>
```

Specify just the configuration file name, not the full path to the file.

In the background this invokes the Linux text editor *vi* to display and modify the configuration file. Therefore you can use the **standard keystrokes supported by *vi*** to make and save changes to the file.

## 9.10.1. Files under .../manifests/extdata/

| File | Purpose |
|---|---|
| *adminsslconfigs.csv* | Configures TLS/SSL implementation for the Admin Service's HTTPS listener. |
| *dynsettings.csv* | Used by the CMC's **Configuration Settings** page. Do not edit this file. |
| *fileupdates.csv* | There may be rare circumstances in which Cloudian Support asks you to use this file, in which case you will be provided instructions for using it. |
| *iamsslconfigs.csv* | Configures TLS/SSL implementation for the IAM Service. This file is updated automatically when you use the installer or the command line tool *hsctl* to manage HTTPS for the IAM Service. For more information see **"Security and Privacy Features"** (page 141) |
| *<node>.csv* | These files are for settings that are tailored to individual nodes, as identified by the *<node>* segment of the file names (for example, *host1.csv*, *host2.csv*, and so on). There will be one |

| File | Purpose |
|---|---|
| | such file for each node in your system. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation. Settings in a *<node>.csv* file override default settings from *common.csv*, for the specified node. |
| *<regionName>_region.csv* | Configures settings that are particular to a service region. Settings that would have different values in different regions are in this file. If you have multiple HyperStore service regions, you will have multiple instances of this file. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation. |
| *s3sslconfigs.csv* | Configures TLS/SSL implementation for the S3 Service. This file is updated automatically when you use the installer or the command line tool *hsctl* to manage HTTPS for the IAM Service. For more information see **"Security and Privacy Features"** (page 141) |
| *<regionName>_topology.csv* | Specifies a topology of nodes, racks, and data centers in a service region. If you have multiple HyperStore service regions, you will have multiple instances of this file. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation. |

## 9.10.2. Files under .../modules/cloudians3/templates/

| File | Purpose |
|---|---|
| *admin.xml.erb* | This file configures the Admin Service's underlying Jetty server functionality, for processing incoming HTTP requests. |
| *admin_realm.properties.erb* | This file configures HTTP Basic Authentication for the Admin Service. Its settings are controlled by corresponding settings in **common.csv**. |
| *cloudian-cron.tab.erb* | This file configures HyperStore system maintenance cron jobs. |
| *kmip.properties.erb* | This file configures HyperStore's connection (if desired) to an external KMIP-compliant Key Management System, which is one option for implementing **server-side encryption**. Its settings are controlled by corresponding settings in **common.csv**. |
| *log4j-*.xml.erb* | These files configure logging behavior for various services such as the S3 Service, Admin Service, HyperStore Service, and so on. For information about settings in these files, see **"Log Configuration Settings"** (page 373). |
| *ntf.properties.erb* | This file configures certain details about the behavior of the S3 Notification service, such as message retention and maximum message size. For more information about this service see the SQS section of the ***Cloudian HyperStore AWS APIs Support Reference***. |
| *s3.xml.erb* | This file configures the S3 Service's underlying Jetty server func- |

| File | Purpose |
|---|---|
| | tionality, for processing incoming HTTP requests. |
| *storage.xml.erb* | This file configures the HyperStore Service's underlying Jetty server functionality, for processing incoming HTTP requests. |
| *tiering-map.txt.erb* | This file is not used currently. |
| *tiering-regions.xml.erb* | In support of the HyperStore auto-tiering feature, this file lists S3 endpoints that objects can be auto-tiered to. By default this file is pre-configured with all the Amazon S3 regional service end-points. If you have a multi-region system, the file is also pre-configured with the S3 endpoints for each of your service regions, and the file is used in support of the **cross-region replication** feature. |

### 9.10.3. Files under .../modules/cmc/templates/

| File | Purpose |
|---|---|
| *server.xml.erb* | This file configures the CMC's underlying Tomcat server functionality, for processing incoming HTTP requests. |

### 9.10.4. Files under .../modules/salt

HyperStore uses the open source version of **Salt** for certain configuration management functions (such as configuration management of the HyperStore firewall -- which from a user perspective is controlled through the installer's Advanced Configuration Options menu). **Do not edit any of the configuration files under */etc/cloudian-<version>-puppet/modules/salt*.**

> **Note** Salt configuration management activity is recorded in the log file */var/log/cloudian/salt.log*.

## 9.11. Configuration Special Topics

### 9.11.1. HyperStore Firewall

*Subjects covered in this section:*



Each HyperStore node includes a built-in HyperStore Firewall that is pre-configured with settings appropriate for a typical HyperStore deployment. The HyperStore Firewall is either enabled or disabled by default depending on whether your original HyperStore installation was older than version 7.2:

- In systems originally installed as version 7.2 or newer, the HyperStore Firewall is enabled by default.
- In systems originally installed as a version older than 7.2 and then later upgraded to 7.2 or newer, the HyperStore Firewall is available but is disabled by default.

You can enable or disable the HyperStore Firewall by using the installer's Advanced Configuration Options, as described below in **"Enabling or Disabling the HyperStore Firewall"** (page 492). When the Firewall is enabled, you can optionally customize certain aspects of the Firewall's behavior, as described further below in **"Customizing the HyperStore Firewall"** (page 494).

**Cloudian, Inc. strongly recommends using a firewall to protect sensitive internal services such as Cassandra, Redis, and so on,** while allowing access to public services -- particularly in environments where no dedicated internal interface(s) have been specified during HyperStore installation; or the internal, back-end network is not a closed network only available between HyperStore nodes. The pre-configured HyperStore Firewall serves this purpose, if enabled. Alternatively, if you have upgraded to HyperStore 7.2 from an older version and you already have a custom firewall in place that you have been successfully using with HyperStore, you may prefer to keep using that firewall -- since in HyperStore 7.2 the HyperStore Firewall is limited as to how much it can be customized.

If you have upgraded to HyperStore 7.2 from an older version and you do wish to enable the HyperStore Firewall rather than continuing to use your own custom firewall, then before enabling the HyperStore Firewall do the following:

- If you have created custom *firewalld* Zone and Service configuration files, make a backup copy of those files for your own retention needs. When you enable the HyperStore Firewall **your existing Zone and Service configuration files will be deleted from the */etc/firewalld* directory**.
- Disable your existing firewall service on each node. For example, to disable *firewalld* do the following on each node:

  ```
  # systemctl stop firewalld
  # systemctl disable firewalld
  ```

  The HyperStore installer will not allow you to enable the HyperStore Firewall on your nodes if existing firewall rules are in effect on any of the HyperStore nodes.

**Note** The HyperStore Firewall service is implemented as a custom version of the *firewalld* service, named *cloudian-firewalld*.

### 9.11.1.1. Enabling or Disabling the HyperStore Firewall

To enable or disable the HyperStore Firewall on all your HyperStore nodes:

1. On the Configuration Master node, change into your current HyperStore version installation staging directory (*/opt/cloudian-staging/7.5.2*). Then launch the installer.

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. At the installer main menu enter **4** for "Advanced Configuration Options".
3. At the Advanced Configuration Options menu enter **s** for "Configure Firewall".
4. At the Cloudian HyperStore Firewall Configuration menu enter **a** for "Enable/Disable Cloudian HyperStore Firewall".
5. In the Enable/Disable Cloudian HyperStore Firewall interface, the Firewall's current status is displayed. At the prompt enter **enable** to enable the Firewall or **disable** to disable the Firewall. After the interface indicates that the Firewall is set as you specified, press any key to return to the Firewall Configuration menu.
6. At the Firewall Configuration menu enter **x** for "Apply configuration changes and return to previous menu". Then at the next prompt that displays enter **yes** to confirm that you want to apply your configuration changes. This will apply your change to all nodes in your HyperStore system (all nodes in all of your data centers and service regions).

   The installer interface should then display a status message "result: **OK**" for each node, to indicate the successful applying of your configuration change to each node.

   > **Note** If the installer displays a Warning about one or more nodes not responding you can try the Firewall Configuration menu's **x** option again, and this retry may work for the node(s) if the problem the first time was a temporary network issue. If one of your nodes is **down**, the node will automatically be updated with your configuration change when the node comes back online.

You are now done with enabling or disabling the HyperStore Firewall. **You do not need to do a Puppet push or restart any services.**

### 9.11.1.2. Default Behavior of the HyperStore Firewall When Enabled

When the HyperStore Firewall is enabled, the default behavior on each HyperStore node is as follows:

- On all IP interfaces, **all TCP ports will allow inbound traffic originating from other HyperStore nodes**. In a multi-DC or multi-region system, this includes inbound traffic originating from HyperStore nodes in other DCs or regions.
- On all IP interfaces, **only the following TCP ports will allow inbound traffic that originates from sources other than HyperStore nodes**:
  - Admin HTTP service port (18081 by default)
  - Admin HTTPS service port (19443 by default)
  - CMC HTTP service port (8888 by default)
  - CMC HTTPS service port (8443 by default)
  - IAM HTTP service port (16080 by default)
  - IAM HTTPS service port (16443 by default)
  - S3 HTTP service port (80 by default)
  - S3 HTTPS service port (443 by default)
  - S3 PROXY HTTP service port (81)
  - S3 PROXY HTTPS service port (4431)

- SSH service port (22)
- SQS HTTP service port (18090 by default)
- SQS HTTPS service port (18443 by default)

Traffic originating from sources other than HyperStore nodes will be blocked (DROP'd) on all TCP ports other than those listed above.

> **Note** The Firewall also allows incoming ICMP traffic originating from sources other than HyperStore nodes.

- On all IP interfaces, **outbound traffic is allowed on all ports**.

When the Firewall is enabled, the Firewall configuration will automatically adjust to system changes in the following ways:

- If you resize your cluster by adding or removing nodes, or by adding or removing a data center or service region, the Firewall accommodates this change automatically. In the case of expanding your cluster, the Firewall will be automatically enabled on new nodes, and the Firewall on the existing nodes will allow inbound traffic from the new nodes, on any port. In the case of removing nodes, the Firewall on the existing nodes will be updated such that the removed nodes are no longer part of the cluster and can only access the cluster's public services.

  > **Note** If the Firewall is disabled when you add new nodes to your cluster, the Firewall will also be disabled on the new nodes.

- If you change the port number that a particular HyperStore public service uses, the Firewall's configuration is automatically adjusted accordingly. After you complete a port change you only need to apply the updated Firewall configuration to the cluster. For instructions see **"Changing S3, Admin, or CMC Listening Ports"** (page 498).

### 9.11.1.3. Customizing the HyperStore Firewall

The default behavior of the HyperStore Firewall (when enabled) is as described above. If you wish you can customize the behavior by denying access to one of the services that the Firewall allows access to by default. For example, you could deny access to the S3 HTTP service so that the S3 HTTPS service is used exclusively. Subsequently, if there is a change in your preferences or circumstances, you could customize the Firewall to once again allow access to that service.

To customize the HyperStore Firewall on all your HyperStore nodes:

1. On the Configuration Master node, change into your current HyperStore version installation staging directory. Then launch the installer.

   ```
   # ./cloudianInstall.sh
   ```

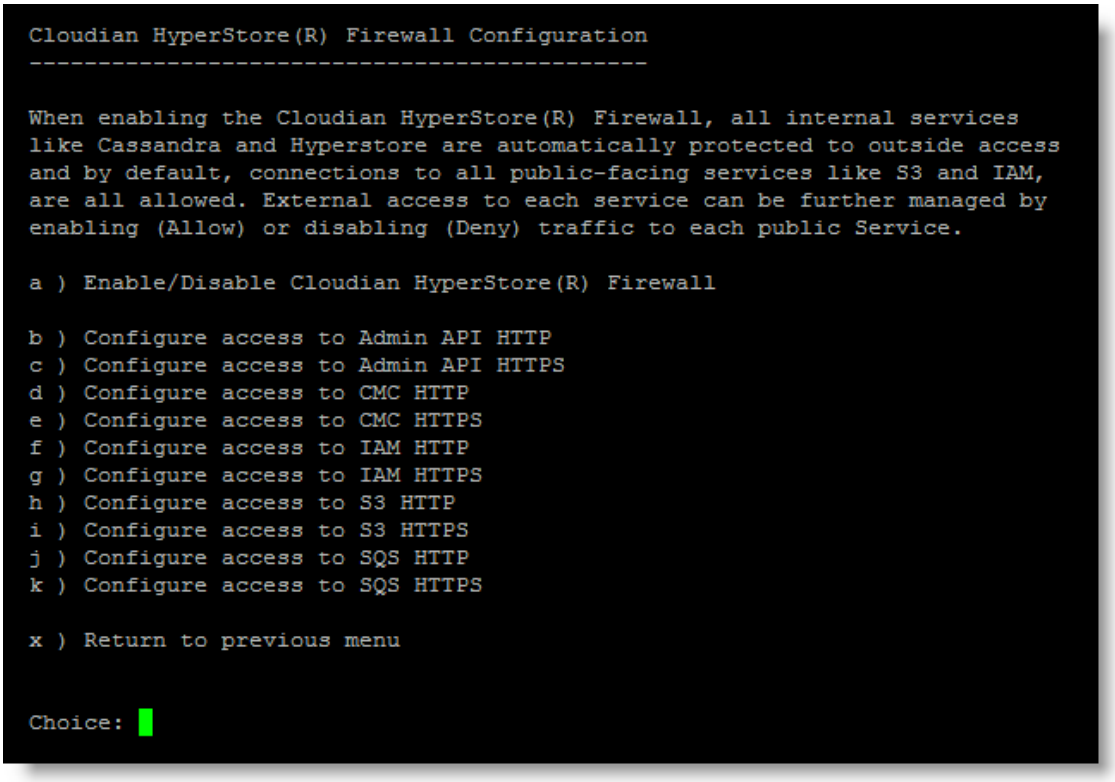
   *If you are using the HyperStore Shell*

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. At the installer main menu enter **4** for "Advanced Configuration Options".
3. At the Advanced Configuration Options menu enter **s** for "Configure Firewall".
4. At the Cloudian HyperStore Firewall Configuration menu enter the menu letter corresponding to the service for which you want to deny or allow access (for example **h** for S3 HTTP).




5. In the Allow/Deny Access to Service <Service Type> interface, the service's current setting is displayed ("Allow" or "Deny"). At the prompt enter **deny** to deny access to the service or **allow** to allow access to the service. After the interface indicates that the service is set as you specified, press any key to return to the Firewall Configuration menu.
6. At the Firewall Configuration menu enter **x** to apply your configuration changes. Then at the next prompt that displays enter **yes** to confirm that you want to apply your configuration changes. This will apply your change to all nodes in your HyperStore system (all nodes in all of your data centers and service regions).

   The installer interface should then display a status message "result: **OK**" for each node, to indicate the successful applying of your configuration change.

   > **Note** If the installer displays a Warning about one or more nodes not responding you can try the Firewall Configuration menu's **x** option again, and this retry may work for the node(s) if the problem the first time was a temporary network issue. If one of your nodes is **down**, the node will automatically be updated with your configuration change when the node comes back online.

You are now done with customizing the HyperStore Firewall. **You do not need to do a Puppet push or restart any services.**

#### 9.11.1.4. HyperStore Firewall Logging

On each node, requests blocked by the HyperStore Firewall are logged to */var/log/cloudian/firewall.log*. For more information about this log including its rotation and retention behavior, see **"HyperStore Firewall Log"** (page 353).

### 9.11.2. Anti-Virus Software

If you are considering using anti-virus software on your HyperStore nodes, be aware that the HyperStore system is provisioned and uses server resources according to the server specifications and use case. Any use of third party software must ensure that it does not interfere with HyperStore's use of these resources (such as disk space, disk I/O, RAM, CPU, network, and ports). Unavailability or sharing of these resources can cause the HyperStore system to not function and/or have reduced performance.

If you want to use anti-virus software to monitor OS files **other than** HyperStore-related files, configure the anti-virus software to exclude these directories from monitoring:

- All HyperStore data directories (as specified by the configuration setting *hyperstore_data_directory* in *common.csv*)
- */var/lib/{cassandra,cassandra_commit,redis}*
- */var/log/{cloudian,cassandra,redis}*
- */opt/{cassandra,cloudian,cloudian-packages,cloudianagent,dnsmasq,redis,tomcat}*
- */etc/cloudian-<version>-puppet**

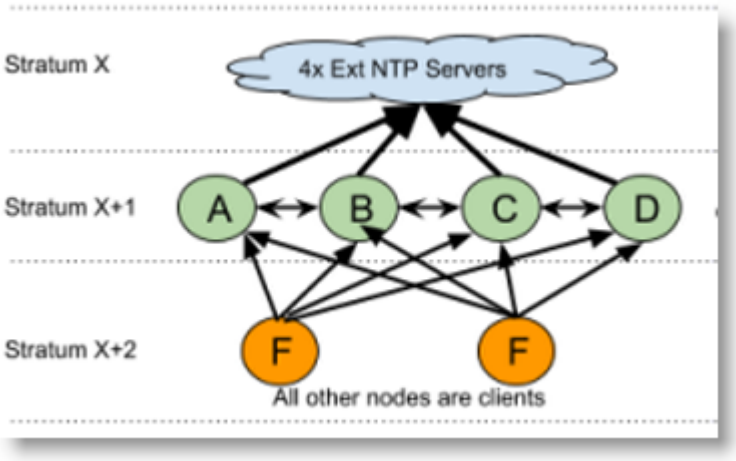
### 9.11.3. NTP Automatic Set-Up

**Note** This topic describes the NTP configuration that HyperStore automatically implements. If instead you are using your own custom NTP set-up, Cloudian recommends using **at least 3 root clock sources**.

Accurate, synchronized time across the cluster is vital to HyperStore service. For example, object versioning relies on it, and so does S3 authorization. It's important to have a robust NTP set-up.

When you install your HyperStore cluster, the installation script **automatically** configures a robust NTP set-up using *ntpd*, as follows:

- In each of your HyperStore data centers, four HyperStore nodes are configured as internal NTP servers. These internal NTP servers will synchronize with external NTP servers -- from the *pool.ntp.org* project by default -- and are also configured as peers of each other. (If a HyperStore data center has only four or fewer nodes, then all the nodes in the data center are configured as internal NTP servers.)
- All other nodes in the data center are configured as clients of the four internal NTP servers.




- In the event that all four internal NTP servers in a DC are unable to reach any of the external NTP servers, the four internal NTP servers will use "orphan mode" -- which entails the nodes choosing one of themselves to be the "leader" to which the others will sync -- until such time as one or more of the external NTP servers are reachable again.

Each HyperStore data center is independently configured, using this same approach.

To see which of your HyperStore hosts are configured as internal NTP servers, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

On the CMC's **Cluster Information** page you can also view the list of external NTP servers to which the internal NTP servers will synchronize. By default the external NTP servers used are public servers from the *pool.ntp.org* project:

- 0.centos.pool.ntp.org
- 1.centos.pool.ntp.org
- 2.centos.pool.ntp.org
- 3.centos.pool.ntp.org

**IMPORTANT !** In order to connect to the default external NTP servers the internal NTP servers must be allowed outbound internet access.

**Note** *ntpd* is configured to automatically start on host boot-up. However, it's recommended that after booting a HyperStore host, you verify that *ntpd* is running (which you can do with the *ntpq -p* command — if *ntpd* is running this command will return a list of connected time servers).

### 9.11.3.1. Changing the List of External NTP Servers or Internal NTP Servers

You can use the HyperStore installer's "Advanced Configuration Options" to change either the list of external NTP servers or the list of internal NTP servers. For more information see **"Change Internal NTP Servers or External NTP Servers"** (page 312).

## 9.11.4. Changing S3, Admin, or CMC Listening Ports

You can use the HyperStore installer's "Advanced Configuration Options" to change listening ports for the S3, Admin, and CMC services.

1. On the Configuration Master node, change to the installation staging directory (*/opt/cloudian-staging/7.5.2*). Then launch the installer.

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. At the installer's main menu enter **4** for "Advanced Configuration Options". Then at the Advanced Configuration Options menu enter **b** for "Change S3, Admin or CMC ports".
3. Follow the prompts to specify your desired port numbers. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value. The final prompt will ask whether you want to save your changes -- type *yes* to do so.
4. Go back to the installer's main menu again and enter **2** for "Cluster Management". Then enter **b** for "Push Configuration Settings to Cluster", and follow the prompts..
5. After returning to the Cluster Management menu again, enter **c** for "Manage Services", and restart the affected services:

   - For changed S3 or Admin ports, restart the S3 Service and the CMC. Note that the Admin service is restarted automatically when you restart the S3 Service.
   - For changed CMC port, restart the CMC

   Do not exit the installer until you complete Step 6 below, if applicable.

6. **If you have the HyperStore firewall enabled** (as described in **"HyperStore Firewall"** (page 491)), the firewall's configuration is automatically adjusted to accommodate the port number change that you made. But you must push the updated firewall configuration out to the cluster by taking these steps with the installer:

   a. At the installer's main menu, enter **4** for "Advanced Configuration Options". Then at the Advanced Configuration Options menu enter **s** for "Configure Firewall".

   b. At the Firewall Configuration menu, enter **x** for "Apply configuration changes and return to previous menu". When prompted, enter **yes** to confirm that you want to apply your configuration changes.

After your changes are successfully applied to the cluster you can exit the installer.

## 9.11.5. Changing S3, Admin, CMC, or IAM Service Endpoints

You can use the HyperStore installer's "Advanced Configuration Options" to change the S3 service endpoint, S3 static website endpoint, Admin service endpoint, CMC endpoint, or IAM endpoint. (For more information on

these HyperStore service endpoints, their default values, and how the endpoints are used, see **"DNS Set-Up"** (page 50).)

**Note** In the current HyperStore release:
* The CMC uses the IAM service endpoint to make STS calls as well as IAM calls. Therefore the installer's function for changing service endpoints will show one shared service endpoint -- the IAM endpoint -- for IAM and STS.
* The installer does not support changing the SQS service endpoint. You can change that endpoint by editing *sqs_endpoint* in **common.csv** and then restarting the SQS service.

1. On the Configuration Master node, change to the installation staging directory (*/opt/cloudian-staging/7.5.2*). Then launch the installer.

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's menu select "Advanced Configuration Options" and then select "Change S3, Admin, CMC, or IAM/STS endpoints".

3. Follow the prompts to specify your desired endpoints. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value.

   For S3 service endpoint, the typical configuration is one endpoint per service region but you also have the option of specifying multiple endpoints per region (if for example you want to have different S3 service endpoints for different data centers within the same region). To do so simply enter a comma-separated list of endpoints at the prompt for the region's S3 service domain URL. Do not enclose the comma-separated list in quotes. If you want to have different S3 endpoints for different data centers within the same service region, the recommended S3 endpoint syntax is *s3-<regionname>.<dcname>.<domain>*. For example if you have data centers named *chicago* and *cleveland* both within the *midwest* service region, and your domain is *enterprise.com*, the S3 endpoints would be *s3-midwest.chicago.enterprise.com* and *s3-midwest.cleveland.enterprise.com*. (Make sure that your DNS set-up resolves the service endpoints in the way that you want -- for example, with one S3 service endpoint resolving to the virtual IP address of a load balancer in your Chicago data center and one S3 service endpoint resolving to the virtual IP address of a load balancer in your Cleveland data center).

   If you have multiple S3 service endpoints for your system, the first S3 service endpoint in your comma-separated list will be the "default" S3 service endpoint (and so will be used in public URLs if CMC users generate public URLs for some of their objects).

   **Note**
   * In an S3 service endpoint the only instance of the string "s3" should be the leading prefix. Do not also include "s3" in the *<regionname>* value, the *<dcname>* value, or the *<domain>* value . This is because having two instances of "s3" in the service endpoint may cause S3 service requests to fail for some S3 client applications. For example, do not have a service endpoint

such as "*s3-tokyo.s3.enterprise.com*".
* HyperStore supports the *s3-<regionname>.<domain>* format and also the *s3.<regionname>.<domain>* format (with a dot after the "s3" rather than a dash).
* Do not use an IP address as your S3 service endpoint.

For S3 static website endpoint you can only have one endpoint per service region. For the Admin service you can only have one endpoint for your whole HyperStore system. For the CMC service you can only have one endpoint for your whole HyperStore system. For the IAM service you can only have one endpoint for your whole HyperStore system.

The final prompt will ask whether you want to save your changes -- type *yes* to do so.

4. Go to the main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
5. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service and the CMC. If you are using DNSMASQ for HyperStore service endpoint resolution, then also restart DNSMASQ.

**Note** If you are using your DNS environment for HyperStore service endpoint resolution, **update your DNS entries** to match your custom endpoints if you have not already done so. For guidance see **"DNS Set-Up"** (page 50). .

**Note** If you are using per-group filtering of S3 endpoint displays in the CMC and you change an S3 endpoint (using the procedure above), then you must go to the CMC's **Manage Groups** page and edit the group configuration for any groups that are using S3 endpoint display filtering. If you do not update the S3 endpoint display filtering for such groups, then neither the original S3 endpoint nor the replacement S3 endpoint will display for those groups. Note that per-group filtering of S3 endpoint displays is not the default behavior (by default all users can see all of your system's current S3 endpoints, listed in the CMC's **Security Credentials** page). If you did not explicitly configure any groups to use S3 endpoint display filtering, then after changing S3 endpoints in the system you do not need to take any action in regard to the CMC's display of S3 endpoints -- all CMC users will automatically be able to see the new S3 endpoints.

## 9.11.6. Tuning HyperStore Performance Parameters

The HyperStore system includes a performance configuration optimization script that is automatically run on each node when you install HyperStore; and that also automatically runs on any new nodes that you subsequently add to your cluster. The script adjusts OS configuration settings on each node, and certain HyperStore system configuration settings, for optimal performance based on your particular environment (taking into account factors such as RAM and CPU specs).

Since the script runs automatically during installation and during cluster expansion, under normal circumstances you should not need to run the script yourself. However, you can run the performance configuration optimization script indirectly through the installer's Advanced Configuration Options menu, if for example you have made configuration changes on your own and your system is now under-performing as a result. Running the script in this way will return the configuration settings to the optimized values as determined by the script.

To run the script from the Advanced Configuration Options menu:

1. On the Configuration Master node, change to the installation staging directory (*/opt/cloudian-staging/7.5.2*) and then launch the installer:

   ```
   # ./cloudianInstall.sh
   ```

   *If you are using the **HyperStore Shell***

   If you are using the **HyperStore Shell (HSH)** as a **Trusted user**, from any directory on the Configuration Master node you can launch the installer with this command:

   ```
   $ hspkg install
   ```

   Once launched, the installer's menu options (such as referenced in the steps below) are the same regardless of whether it was launched from the HSH command line or the OS command line.

2. From the installer's menu select "Advanced Configuration Options" and then select "Configure Performance Parameters on Nodes".
3. At the prompt, specify a node for which to run the performance configuration optimization; or specify a comma-separated list of nodes; or leave the prompt blank and press enter if you want to run the optimization for all nodes in your system. When the script run is done the installer interface will prompt you to continue to the next steps.
4. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
5. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service, the HyperStore Service, and the Cassandra Service.

## 9.11.7. Using JMX to Dynamically Change Configuration Settings

Certain HyperStore configuration settings (a minority of them) can be dynamically reloaded via JMX. You can use a JMX client such as **JConsole** or **Jmxterm** to connect to the JMX listening port of the particular service that you're configuring (S3 Service JMX port = 19080; Admin Service JMX port = 19081; HyperStore Service JMX port = 19082; Redis Monitor JMX port = 19083). *JConsole* is a graphical user interface that's part of the HyperStore system's Java installation and can be found under *JAVA_HOME/bin*. *Jmxterm* is a command line tool that comes bundled with the HyperStore system and can be found under *opt/cloudian/tools*.

When you use JMX to make a configuration setting change, the change is applied to the service dynamically — you do not need to restart the service for the change to take effect. However, the **setting change persists only for the current running session of the affected service**. If you restart the service it will use whatever setting is in the configuration file. Consequently, if you want to change a setting dynamically and also have your change persist through a service restart, you should change the setting in the configuration file as well as changing it via JMX.

In the documentation of HyperStore configuration files, if a setting supports being dynamically changed via JMX, the setting description indicates "Reloadable via JMX". It also indicates the name of the dynamically changeable MBean attribute that corresponds to the configuration file setting. For example, the documentation of the HyperStore Service configuration property *repair.session.threadpool.corepoolsize* includes a note that says:

"Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionThreadPoolCorePoolSize)"

**Note** Some settings in HyperStore configuration files can be set through the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**). The CMC uses JMX to apply setting

changes dynamically, and the CMC also automatically makes the corresponding configuration file change and triggers a Puppet push so that your changes will persist across service restarts. For these settings it's therefore best to use the CMC to make any desired edits rather than directly using JMX. For such configuration file settings, the descriptions in the HyperStore configuration file documentation indicate that you should use the CMC to edit the setting. Consequently these settings are not flagged in the documentation as JMX reloadable.

# Chapter 10. Command Line Tools

## 10.1. hsstool

The HyperStore system includes its own cluster management utility called *hsstool*. This tool has functionality that in many respects parallels the Cassandra utility *nodetool*, with the important distinction that *hsstool* applies its operations to the HyperStore File System (HSFS) as well as to the Cassandra storage layer.

The *hsstool* utility is in the */opt/cloudian/bin* directory of each HyperStore node. Because the HyperStore installation adds */opt/cloudian/bin* to each host's $PATH environment variable, you can run the *hsstool* utility from any directory location on any HyperStore host.

The tool has the following basic syntax:

```
# hsstool -h <host> [-p <port>] <command> [<command-options>]
```

- The *<host>* is the hostname or IP address of the HyperStore node on which to perform the operation. Specify the actual hostname or IP address -- **do not use "localhost"**. For most commands that only retrieve information, the *-h <host>* argument is optional and (if not supplied) defaults to the hostname of the host on which you are executing *hsstool*. For commands that impact system data or processes -- such as a repair or cleanup operation -- the *-h <host>* argument is mandatory.
- The *<port>* is the HyperStore Service's JMX listening port. If you do not supply the port number when using *hsstool*, it defaults to 19082. There is no need to supply the port when using *hsstool* unless you've configured your system to use a non-default port for the HyperStore Service's JMX listener. The syntax summaries and examples in the documentation of individual commands omit the *-p <port>* option.
- The hsstool options *-h <host>* and (if you use it) *-p <port>* must precede the *<command>* and the *<command-options>* on the command line. For example, do not have *-h <host>* come after the *<command>*.
- For best results when running *hsstool* on the command line, run the commands as *root*. While some commands may work when run as a non-root user, others will return error responses.

  *If you are using the **HyperStore Shell***

  If you are using the HyperStore Shell (HSH) you can run *hsstool* from the HSH command line, with the same capabilities and same syntax as if you were running it as *root*:

  ```
  $ hsstool -h <host> [-p <port>] <command> [<command-options>]
  ```

All *hsstool* operations activity is logged in the cloudian-hyperstore.log file on the node to which you sent the *hsstool* command.

For usage information type ***hsstool help*** or ***hsstool help <command>***. The usage information that this returns is not nearly as detailed as what's provided in this documentation, but it does provide basic information about syntax and command options.

**The table below lists the commands that *hsstool* supports**. For command options and usage information, click on a command name.

> **Note** All *hsstool* commands can be executed either on the command line or through the CMC's **Node Advanced** page (**Cluster -> Nodes -> Advanced**). The documentation for these commands shows the CMC interface for each command as well as the command line syntax.

| Command Type | Command | Purpose |
|---|---|---|
| Information Commands | info | View vNode and data load info for a physical node |
| | ring | View vNode info for whole cluster |
| | status | View summary status for whole cluster |
| | opstatus | View status of operations such as cleanup, repair, and rebalance |
| | proactiverepairq | View status of proactive repair queues |
| | repairqueue | View auto-repair schedule or enable/disable auto-repair |
| | ls | View vNode and data load info per mount point |
| | whereis | View storage location information for an S3 object |
| | metadata | View metadata for an S3 object |
| | trmap | View token range snapshot IDs or snapshot content |
| | madd | This is for internal use by the install script, or for use as directed by Cloudian Support. |
| Maintenance Commands | cleanup | Clean a node of replicated data that doesn't belong to it |
| | cleanupec | Clean a node of erasure coded data that doesn't belong to it |
| | autorepair | In the CMC interface this invokes the repairqueue command |
| | repair | Repair the replicated data on a node |
| | repairec | Repair the erasure coded data in a data center |
| | repaircassandra | Repair only the Cassandra metadata on a node, not the object data |
| | rebalance | Shift data load from your existing nodes to a newly added node |
| | opctl | List or stop all repair and cleanup operations currently running in the region |

## 10.1.1. hsstool cleanup

*Subjects covered in this section:*



### 10.1.1.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> cleanup [allkeyspaces|nokeyspaces] [-n] [-l <true|false>]
[-b] [-x] [-no] [-a] [-c <true|false>] [-d <mountpoint>] [-vnode <token>]
[-policy] [-stop]
```

## Options

*allkeyspaces | nokeyspaces  Set scope of metadata cleanup*

(Optional) You have three alternatives for choosing which Cassandra metadata keyspaces to clean up, while also cleaning replicated S3 object data in the HyperStore File System (HSFS):

- Use *allkeyspaces* to clean up replicated S3 objects in the HSFS and also clean up all the Cassandra keyspaces. Cassandra cleanup will be completed first, then HSFS replica cleanup. The Cassandra keyspaces that will be cleaned are: UserData_<storage-policy-ID> keyspaces; AccountInfo; Reports; Monitoring; and ECKeyspace. (For more information see the overview of **Cassandra keyspaces for HyperStore**)
- Use *nokeyspaces* to clean up only replicated objects in the HSFS, and not any Cassandra keyspaces
- If you specify neither *allkeyspaces* nor *nokeyspaces* then the default behavior is to clean up replicated objects in the HSFS and also to clean the *Cassandra UserData_<storage-policy-ID>* keyspaces (which store object metadata). Cassandra cleanup will be completed first, then HSFS replica cleanup.

*-n  Perform dry run without deleting data*

(Optional) Don't actually delete anything. Instead, just do a dry run that identifies the replica data that doesn't belong to the node. If you use the *-n* option you must also:

- Use the *nokeyspaces* option. The ability to do a dry run without actually deleting data is supported only for HSFS replica data. Therefore you must select the *nokeyspaces* option or else the cleanup run will actually clean Cassandra keyspace data.
- Have cleanup operation logging turned on (as it is by default). See the description of the *-l* option below.

**Note** If you use the *-n* option when you run the *cleanup* command, one of the response items will be "Number of files to be deleted count". The specific objects identified for deletion will be listed in the cleanup log file as in the *-l* option description below.

*-l <true|false>  Log list of objects to be cleaned*

(Optional, defaults to true) Write to a log file a list of all the objects that were identified as not belonging to the node. Defaults to true, so you only need to specify the *-l* option if you do **not** want cleanup object logging (in which case you'd specify *-l false*).

If you use logging without using the *-n* option, then the list in the log file is a list of all objects that were deleted by the *cleanup* operation.

If you use logging in combination with the *-n* option, then the list in the log file is a list of HSFS replica objects that will be deleted if you run *cleanup* again without the *-n* option.

The log is named *cloudian-hyperstore-cleanup.log* and is written into the Cloudian HyperStore log directory of the target host. Activity associated with a particular instance of a *cleanup* command run is marked with a unique command number.

*-b  Delete bucketless objects*

(Optional) Delete any objects that are not associated with a valid S3 bucket. If you use this option the cleanup operation will check each object to verify that it is part of a bucket, and delete any objects that are not part of a bucket. In typical cleanup scenarios it's not necessary to use this option.

*-x  Clean only objects from unowned ranges*

(Optional) Remove only data that belongs to token ranges that the target node is not responsible for. This is the recommended option to use when you are cleaning the other nodes after adding a new node to a cluster. In this circumstance, using the *-x* option makes the cleanup more efficient and faster.

*-no Clean only objects that lack metadata*

(Optional) Ignore (do not clean) data that belongs to token ranges that the target node is not responsible for. This setting has the cleanup operation focus on deleting object data for which there is no corresponding object metadata. This is an efficient way to clean up garbage data after you have deleted a very large number of objects (in which case the standard hourly batch delete process may be taking too long to free up disk space). Especially in the case where you have recently added nodes and the cluster is still rebalancing, ignoring out-of-range data during cleanup enables the cleanup operation to more efficiently remove garbage blob data after you have deleted a large number of objects through the S3 interface or the Admin API (*POST bucketops/purge* operation).

Note that the *-no* option is the opposite of the *-x* option, and therefore you cannot use those two options in combination.

> **Note** The CMC interface does not support the *-no* option. This option is supported only on the command line.

*-a Also clean erasure coded data*

(Optional) Clean up garbage replica data and then also clean up garbage erasure coded data. When you use the *-a* option, as soon as the *hsstool cleanup* operation completes on a node an *hsstool cleanupec* operation is automatically run on the node as well. This is a convenient option if you are cleaning a node that is storing both replica data and erasure coded data.

Do **not** use this option if you are using either the *-d <mount-point>* option or the *-vnode <token>* option.

*-c <true|false> Confirm that other replicas exist before cleaning an object*

(Optional, defaults to true) If true, then before removing an object replica because it does not belong to the token ranges that the target node is responsible for, the system will first check to make sure that at least one replica of the object exists on the correct endpoint nodes within in the cluster. If no other replicas exist, then the out-of-range replica will be left in place on the node that's being cleaned and an ERROR level message will be written to the HyperStore Service application log (which will also result in the triggering of an Alert in the CMC, if you are using the default alert rules).

This safety feature guards against the possibility of a cleanup operation deleting an incorrectly placed replica when no other replicas of the object exist in any of the correct locations within the cluster. The trade-off is that the cleanup operation will take longer if this approach is used.

This safety feature defaults to true, so there's no need to use the *-c* option unless you want to specify *-c false* in order to skip this safety check.

*-d <mount-point> Clean only this mountpoint*

(Optional) Clean only the specified HyperStore data mount point (for example */cloudian1*). This option may be useful if you want to delete garbage data from a particular disk, in an effort to free up space on the disk.

*-vnode <token> Clean only this token range*

(Optional) Clean only those objects mapped to the specified vNode (identified by its token such as 183151198631857301055573406308303115 35). This option may be useful if during a full node cleanup (or a

disk-specific cleanup), the operation failed for a particular vNode. In that case you can then use the *-vnode <token>* option to retry cleaning just that one vNode.

*-policy  Clean data from deleted storage policies*

(Optional) By default *cleanup* will evaluate and clean only data associated with storage policies that currently exist in your system. It will not evaluate or delete data associated with storage policies that you've deleted from your system. If you want *cleanup* to also delete from the target node **all data associated with storage policies that you've deleted from the system**, use the *-policy* option.

**Note** Before deleting a storage policy you are required to delete any buckets and objects that are stored under that policy. However, the way that object deletion works in HyperStore is that the system deletes the object metadata immediately but does not delete the actual object data until the next run of an automatic purging process that occurs hourly on each node. Meanwhile, for storage policy deletion, the full deletion of the metadata associated with the policy is implemented by a **daily cron job**. Depending on when you delete buckets and objects associated with a storage policy and when you delete the policy itself, the timing may be such that the daily storage policy deletion cron job executes before some of the object data associated with the policy gets deleted by the hourly object data purging process. It's this residual "garbage data" that will be detected and removed if you use the *-policy* option when running *cleanup* on a node.

**Note** The CMC interface does not support the *-policy* option. This option is supported only on the command line.

*-stop  Stop an in-progress cleanup*

(Optional) Use *hsstool -h <host> cleanup -stop* to stop an in-progress cleanup operation on the specified node.

You can subsequently use the **"hsstool opstatus"** (page 522) command to confirm that the cleanup has been stopped (status = TERMINATED) and to see how much cleanup progress had been made before the stop.

If you terminate an in-progress cleanup operation you will not subsequently be able to resume that operation from the point at which it stopped (the *hsstool cleanup* operation does not support a resume option). Instead, when you want to clean the node you can run a regular full cleanup operation.

### 10.1.1.2. Usage Notes

Use this **hsstool** command on a node when you want to identify and delete replica data that does not belong on the node. Broadly, *hsstool cleanup* removes two classes of "garbage" data from a target node:

- Data that belongs to a token range that the target node is no longer responsible for, as a result of a modified token range allocation within the cluster (as occurs when you add a new node).
- Data that should not be on the node even though the data falls within the token ranges that the node is responsible for. This can occur, for example, if data from objects that have been deleted through the S3 interface (or the Admin API's *POST bucketops/purge* operation) has not yet been removed from disk by the hourly batch delete job; or if an object delete request through the S3 interface succeeds for some but not all of the object's replicas.

By default *hsstool cleanup* performs both types of cleanups, but the command supports an *-x* option to perform only the first type and a *-no* option to perform only the second type.

By default you can only run *hsstool cleanup* on one node at a time per data center. This limit is configurable by the **"max.cleanup.operations.perdc"** (page 437) setting in **hyperstore-server.properties.erb**. If you want to raise this limit, consult with Cloudian Support first.

The system will not allow you to run *hsstool cleanup* on a node on which **hsstool repair** is currently running.

> **Note** The *hsstool cleanup* operation will only clean objects for which the Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *hsstool cleanup*.

## When to Use *hsstool cleanup*

The operational procedures during which you would use *hsstool cleanup* are:

- **"Restoring a Node That Has Been Offline"** (page 299)
- Delete a Storage Policy

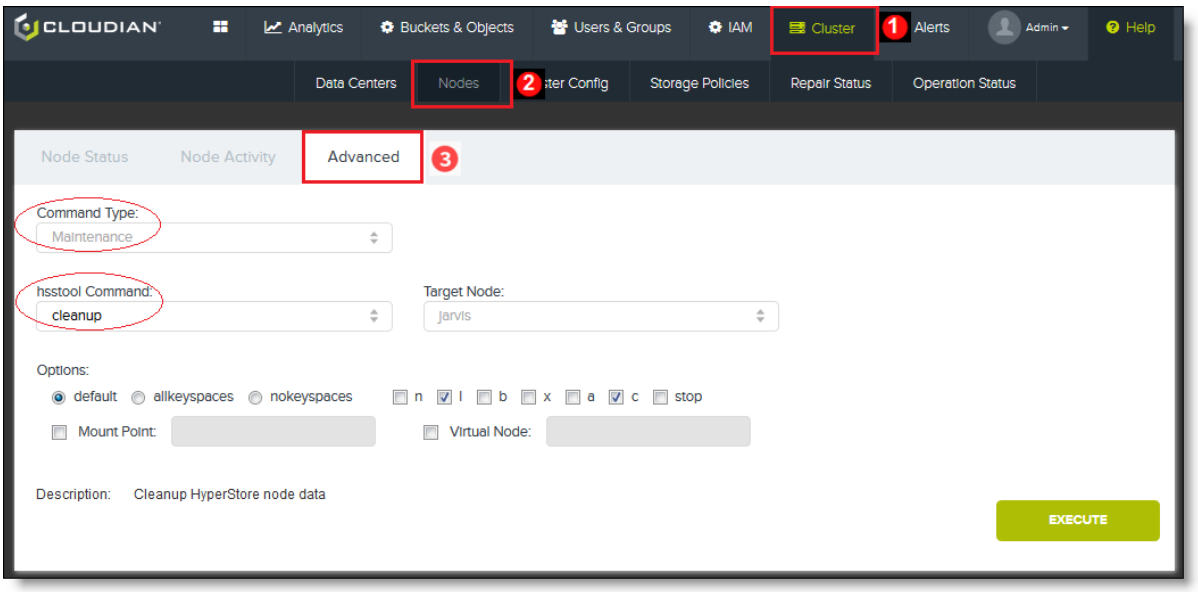
Please refer to those procedures for step-by-step instructions, including the proper use of *hsstool cleanup* within the context of the procedure.

You might also use *hsstool cleanup* at the end of the procedure for **"Adding Nodes"** (page 264). However, using *hsstool cleanup* at the end of the procedure for Adding Nodes is necessary **only if you do not use one of the options that integrates cleanup tasks into the *rebalance* operation** (the *rebalance -cleanupfile* option or the *rebalance -cleanup* option). For more information on these *rebalance* options, see **"hsstool rebalance"** (page 543).

If you do run *hsstool cleanup* at the end of the Adding Nodes procedure, use the *cleanup* options *allkeyspaces*, *-l*, *-x*, *-a*, and *-c*. Note that the *-a* option applies the cleanup operation to erasure coded data as well as replicated data. The system by default only allows you to run cleanup on one node at a time per data center. After cleanup completes on one node, initiate cleanup on a next node, and continue in this way until all of your previously nodes have been cleaned.

## CMC Support For This Command

As an alternative to running *hsstool cleanup* on the command line, you can run command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's **Operation Status** page (**Cluster -> Operation Status**). This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the **hsstool opstatus** command.

#### 10.1.1.3. Checking Operation Status

The *cleanup* operation is a long-running operation. Its duration will depend on multiple factors including the amount of data in your system and the number of nodes in your system. For information about checking the status of an in-progress or recently completed *cleanup* operation see **"hsstool opstatus"** (page 522).

### 10.1.2. hsstool cleanupec

*Subjects covered in this section:*



#### 10.1.2.1. Command Syntax

**Synopsis**

```
# hsstool -h <host> cleanupec [-n] [-l <true|false>] [-b] [-x] [-no]
[-c <true|false>] [-d <mountpoint>] [-vnode <token>] [-policy] [-stop]
```

**Options**

*-n  Perform dry run without deleting data*

(Optional) Don't actually delete anything. Instead, just do a dry run that identifies the erasure coded data that doesn't belong to the node. If you use the *-n* option you must also have cleanup operation logging turned on (as it is by default). See the description of the *-l* option below.

**Note** If you use the *-n* option when you run the *cleanupec* command, one of the response items will be "Number of files to be deleted count". The specific objects identified for deletion will be listed in the cleanup log file as in the *-l* option description below.

*-l <true|false>  Log list of objects to be cleaned*

(Optional, defaults to true) Write to a log file a list of all the objects that were identified as not belonging to the node. Defaults to true, so you only need to specify the *-l* option if you do **not** want cleanup object logging (in which case you'd specify *-l false*).

If you use logging without using the *-n* option, then the list in the log file is a list of all objects that were deleted by the *cleanupec* operation.

If you use logging in combination with the *-n* option, then the list in the log file is a list of objects that will be deleted if you run *cleanupec* again without the *-n* option.

The log is named *cloudian-hyperstore-cleanup.log* and is written into the Cloudian HyperStore log directory of the target host. Activity associated with a particular instance of a *cleanupec* command run is marked with a unique command number.

*-b Delete bucketless objects*

(Optional) Delete any objects that are not associated with a valid S3 bucket. If you use this option the cleanup operation will check each object to verify that it is part of a bucket, and delete any objects that are not part of a bucket. In typical cleanup scenarios it's not necessary to use this option.

*-x Clean only objects from unowned ranges*

(Optional) Remove only data that belongs to token ranges that the target node is not responsible for. This is the recommended option to use when you are cleaning the other nodes after adding a new node to a cluster. In this circumstance, using the *-x* option makes the cleanup more efficient and faster.

*-no Clean only objects that lack metadata*

(Optional) Ignore (do not clean) data that belongs to token ranges that the target node is not responsible for. This setting has the cleanup operation focus on deleting object data for which there is no corresponding object metadata. This is an efficient way to clean up garbage data after you have deleted a very large number of objects (in which case the standard hourly batch delete process may be taking too long to free up disk space). Especially in the case where you have recently added nodes and the cluster is still rebalancing, ignoring out-of-range data during cleanup enables the cleanup operation to more efficiently remove garbage blob data after you have deleted a large number of objects through the S3 interface or the Admin API (*POST bucketops/purge* operation).

Note that the *-no* option is the opposite of the *-x* option, and therefore you cannot use those two options in combination.

**Note** The CMC interface does not support the *-no* option. This option is supported only on the command line.

*-c <true|false> Confirm that other fragments exist before cleaning an object*

(Optional, defaults to true) If true, then before removing an object fragment because it does not belong to the token ranges that the target node is responsible for, the system will first check to make sure that all *k+m* fragments of the object exist on the correct endpoint nodes within in the cluster. If fewer than *k+m* fragments exist, then the out-of-range fragment will be left in place on the node that's being cleaned and an ERROR level message will be written to the HyperStore Service application log (which will also result in the triggering of an Alert in the CMC, if you are using the default alert rules).

This safety feature guards against the possibility of a cleanup operation deleting an incorrectly placed fragment when fewer than *k+m* fragments of the object exist within the cluster. The trade-off is that the cleanup operation will take longer if this approach is used.

This safety feature defaults to true, so there's no need to use the *-c* option unless you want to specify *-c false* in order to skip this safety check.

*-d <mount-point> Clean only this mountpoint*

(Optional) Clean only the specified HyperStore data mount point (for example */cloudian1*). This option may be useful if you want to delete garbage data from a particular disk, in an effort to free up space on the disk.

*-vnode <token> Clean only this token range*

(Optional) Clean only those objects mapped to the specified vNode (identified by its token such as 183151198631857301055573406308303115 35). This option may be useful if during a full node cleanup (or a disk-specific cleanup), the operation failed for a particular vNode. In that case you can then use the *-vnode <token>* option to retry cleaning just that one vNode.

*-policy  Clean data from deleted storage policies*

(Optional) By default *cleanupec* will evaluate and clean only data associated with storage policies that currently exist in your system. It will not evaluate or delete data associated with storage policies that you've deleted from your system. If you want *cleanupec* to also delete from the target node **all data associated with storage policies that you've deleted from the system**, use the *-policy* option.

**Note** Before deleting a storage policy you are required to delete any buckets and objects that are stored under that policy. However, the way that object deletion works in HyperStore is that the system deletes the object metadata immediately but does not delete the actual object data until the next hourly run of the automatic purging process that occurs hourly on each node. Meanwhile, for storage policy deletion, the full deletion of the metadata associated with the policy is implemented by a **daily cron job**. Depending on when you delete buckets and objects associated with a storage policy and when you delete the policy itself, the timing may be such that the daily storage policy deletion cron job executes before some of the object data associated with the policy gets deleted by the hourly object data purging process. It's this residual "garbage data" that will be detected and removed if you use the *-policy* option when running *cleanup* on a node.

**Note** The CMC interface does not support the *-policy* option. This option is supported only on the command line.

*-stop  Stop an in-progress cleanup*

(Optional) Use *hsstool -h <host> cleanupec -stop* to stop an in-progress *cleanupec* operation on the specified node.

You can subsequently use the **"hsstool opstatus"** (page 522) command to confirm that the *cleanupec* operation has been stopped (status = TERMINATED) and to see how much progress had been made before the stop.

If you terminate an in-progress *cleanupec* operation you will not subsequently be able to resume that operation from the point at which it stopped (the *hsstool cleanupec* operation does not support a resume option). Instead, when you want to clean the node you can run a regular full *cleanupec* operation.

### 10.1.2.2. Usage Notes

**Note** If you want to clean **replica data and also erasure coded data** on a node, use **hsstool cleanup** with the *-a* option. If you want to clean **only erasure coded data** on a node, use *hsstool cleanupec* as described below.

Use this **hsstool** command on a node when you want to identify and delete erasure coded data that does not belong on the node. Broadly, *hsstool cleanupec* removes two classes of "garbage" data from a target node:

- Data that belongs to a token range that the target node is no longer responsible for, as a result of a modified token range allocation within the cluster (as occurs when you add a new node).
- Data that should not be on the node even though the data falls within the token ranges that the node is responsible for. This can occur, for example, if data from objects that have been deleted through the S3 interface (or the Admin API's *POST bucketops/purge* operation) has not yet been removed from disk by the hourly batch delete job; or if an object delete request through the S3 interface succeeds for some but not all of the object's replicas.

By default *hsstool cleanupec* performs both types of cleanups, but the command supports an *-x* option to perform only the first type and a *-no* option to perform only the second type.

By default you can only run *hsstool cleanupec* on one node at a time per data center. This limit is configurable by the **"max.cleanup.operations.perdc"** (page 437) setting in **hyperstore-server.properties.erb**. If you want to raise this limit, consult with Cloudian Support first.

The system will not allow you to run *hsstool cleanupec* on a node on which **hsstool repairec** is currently running.

Note that:

- The *hsstool cleanupec* operation will only clean objects for which the Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *hsstool cleanupec*.
- HyperStore 7.5.1 and later supports Hybrid Storage Policies whereby objects larger than a **configurable size threshold** are erasure coded and objects at or smaller than that threshold are replicated. The *hsstool cleanupec* operation will evaluate and clean the erasure coded objects associated with such storage policies, but not the replicated objects. To evaluate and clean the replicated objects use **"hsstool cleanup"** (page 504).
- If you want to clean **replica data and erasure coded data** on a node with one command, use **hsstool cleanup** with the *-a* option.

### When to Use *hsstool cleanupec*

If you have erasure coded data in your HyperStore system, the operational procedure during which you would use *hsstool cleanup* are:

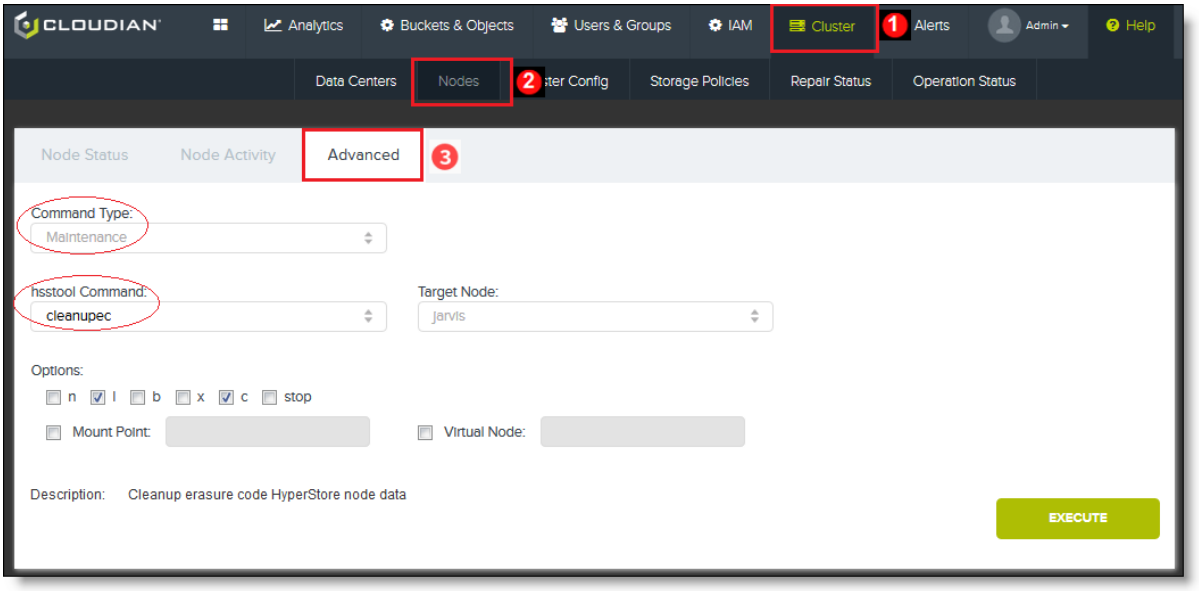
- **"Restoring a Node That Has Been Offline"** (page 299)
- Delete a Storage Policy

Please refer to those procedures for step-by-step instructions, including the proper use of *hsstool cleanupec* within the context of the procedure.

**Note** The *hsstool cleanupec* operation will only clean objects whose Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *hsstool cleanupec*.

### CMC Support For This Command

As an alternative to running *hsstool cleanupec* on the command line, you can run command through the CMC UI:




**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's **Operation Status** page (**Cluster -> Operation Status**). This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the **hsstool opstatus** command.

#### 10.1.2.3. Checking Operation Status

The *cleanupec* operation is a long-running operation. Its duration will depend on multiple factors including the amount of data in your system and the number of nodes in your system. For information about checking the status of an in-progress or recently completed *cleanupec* operation see **"hsstool opstatus"** (page 522).

### 10.1.3. hsstool info

*Subjects covered in this section:*



#### 10.1.3.1. Command Syntax

**Synopsis**

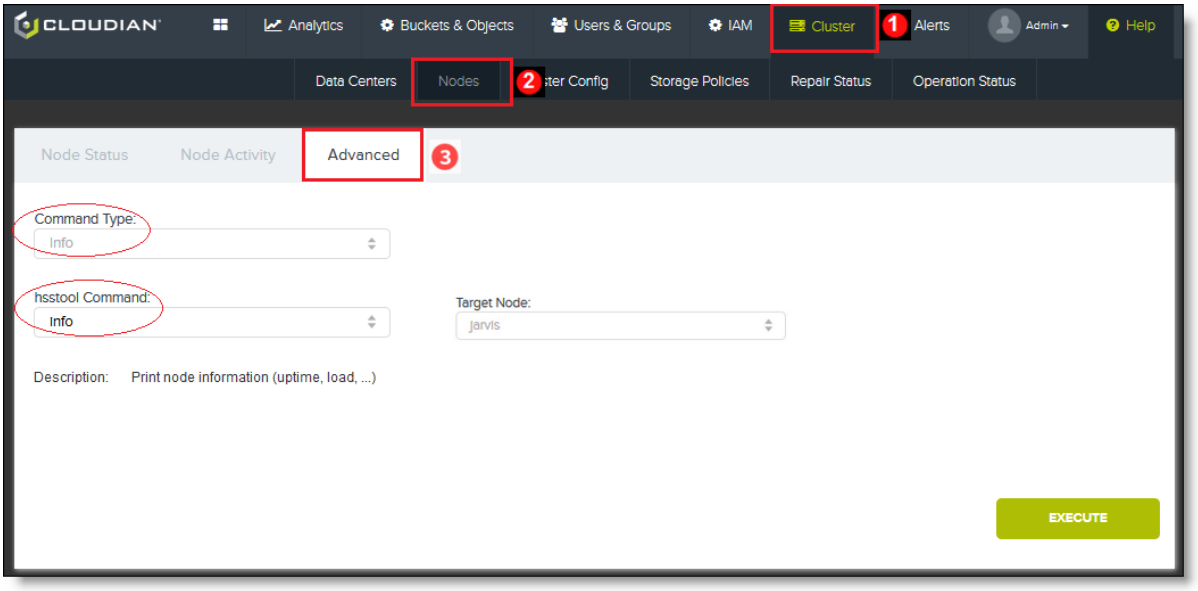
```
# hsstool [-h <host>] info
```

#### 10.1.3.2. Usage Notes

This **hsstool** command returns **virtual node** (token range) information and data load information for a specified physical node within a storage cluster. The return includes a list of virtual nodes (tokens) assigned to the physical node.

### CMC Support for This Command

As an alternative to running *hsstool info* on the command line, you can run it through the CMC UI:




## 10.1.3.3. Command/Response Example

The example below shows an excerpt from a response to the *info* command. The command returns information about a specific node, "cloudian-node1". The node's token (vNode) list is sorted in ascending order.

```
# hsstool -h cloudian-node1 info
Cloudian            : 7.5
Cloudian Load       : 1.12 TB / 42.3 TB (2.6%)
Uptime (seconds)    : 230681
Token               : 18315119863185730105557340630830311535
Token               : 21637484670727122681279388562251225465
Token               : 23572420191725176386844252744138398599
Token               : 25984083804572739863688602357781003456
Token               : 32049925251885239737462386844023262134
Token               : 34776961444994655981702644433932691872
Token               : 39011904510130716900391258282509705889
...
...
Token               : 141833557733030600282377220263378688424
Cassandra           : 2.0.11
Cassandra Load      : 1.9 MB
Data Center         : DC1
Rack                : RAC1
```

**Note** To see which tokens are on which disks on a node, use **hsstool ls**.

### Response Items

*Cloudian*

Cloudian HyperStore software version installed on the node.

*Cloudian Load*

The total volume of S3 object data (replicas and/or erasure coded fragments) stored in the HyperStore File System on the node, across all HyperStore data disks combined.

This field also shows the total volume of disk space allocated for S3 object storage on the node (the total capacity of HyperStore data disks combined); and the percentage of used volume over total capacity.

*Uptime*

The number of seconds that the HyperStore Service has been running since its last start.

*Token*

A storage token assigned to the node. Tokens are randomly generated from an integer token space ranging 0 to $2^{127}$ -1, and distributed around the cluster. Each token is the top of a token range that constitutes a virtual node (vNode). Each vNode's token range spans from the next-lower token (exclusive) in the cluster up to its own token (inclusive). A physical node's set of tokens/vNodes determines which S3 object data will be stored on the physical node.

For more background information see **"How vNodes Work"** (page 80).

*Cassandra*

Cassandra software version installed on the node.

*Cassandra Load*

Cassandra storage load (quantity of data stored in Cassandra) on the node. There will be some Cassandra load even if all S3 object data is stored in the HyperStore File System. For example, Cassandra is used for storage of object metadata and service usage data, among other things.

*Data Center*

Data center in which the node resides.

*Rack*

Rack in which the node resides.

## 10.1.4. hsstool ls

*Subjects covered in this section:*



### 10.1.4.1. Command Syntax

#### Synopsis

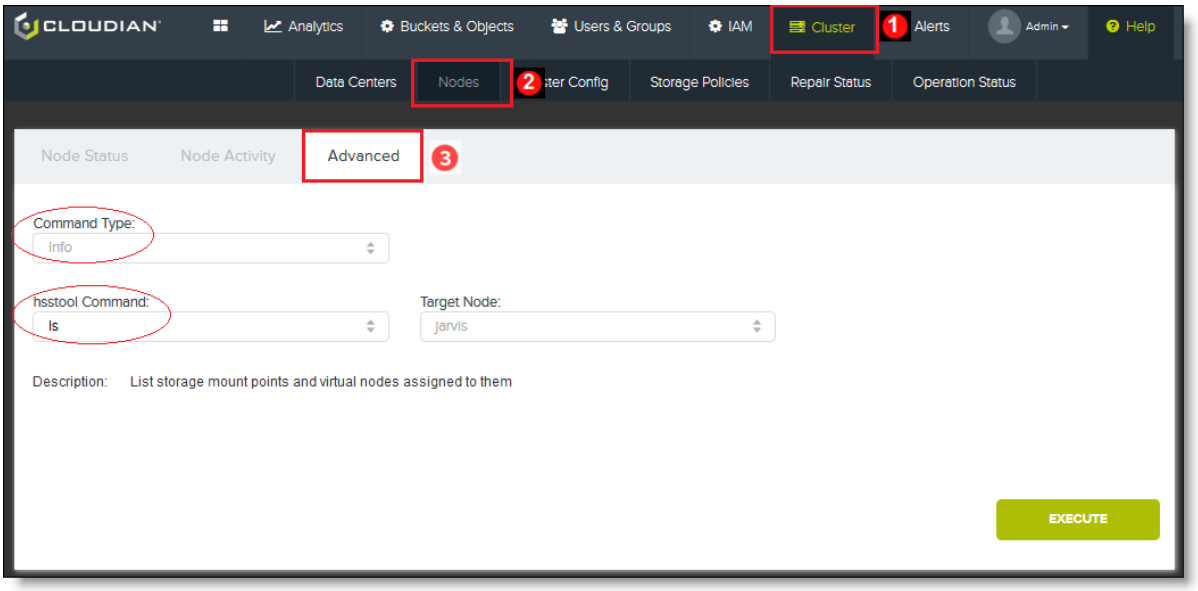
```
# hsstool [-h <host>] ls
```

## 10.1.4.2. Usage Notes

This **hsstool** command returns a node's list of HyperStore data mount points, the list of storage tokens currently assigned to each mount point on the node, the current disk usage per mount point, and the recent errors per mount point if any.

### CMC Support for This Command

As an alternative to running *hsstool ls* on the command line, you can run it through the CMC UI:




## 10.1.4.3. Command/Response Example

The example below shows a response to an *hsstool ls* command. The command response snippet below is truncated; the actual response would list all the tokens assigned to each HyperStore data directory mount point on the node.

```
Filesystem  Size   Used   Avail  Use%   Mounted on   Status  Estimated Digest Keys  Disk Health ..


/dev/md126  3.5T   101G   3.2T    4%     /             OK           0                NORMAL    ..
/dev/sda1   8.9T   9.9G   8.9T    1%    /cloudian1     OK           1877             NORMAL    ..
/dev/sdab1  8.9T   9.9G   8.9T    1%    /cloudian3     OK           1879             NORMAL    ..
...
...
/cloudian1/hsfs:
       124495052739982245195822144173669089 28     HfkGaHQIEduYOZDCRLzW4
       373481913034410151320198600960801505 28     r1D1xZzjR37DQCOlSVBC4
...
...
```

### Response Items

*Filesystem*

Device name of the disk drive

*Size*

Total capacity of the disk

*Used*

Used capacity of the disk

*All*

Remaining available capacity of the disk

*Use%*

Percentage of disk capacity used

*Mounted on*

Mount point of the device

*Status*

Disk status: either OK or ERROR or DISABLED. For more information on the disk's status see the CMC's **Node Status** page, specifically the **Disk Detail Info** section.

*Estimated Digest Keys*

This is a rough estimate of the number of digest keys in the Digest DB for the disk. This corresponds to the number of HyperStore data files on the disk.

*Disk Health*

xxx

*Disk errors*

Count of recent disk errors, if any. These counts are categorized into filesystem errors, I/O errors, and file not found errors. The counts are reset whenever any of these events occur:

- A successful access operation on the disk (such as a read or write)
- A **"disk.fail.error.time.threshold"** (page 439) interval (default 1800 seconds) passes without **"disk.fail.error.count.threshold"** (page 439) total errors (default 100) occurring for the disk. Specifically, the counters reset upon the next disk access operation after the interval passes. In broad terms, by default the counters reset approximately every 30 minutes if not enough disk errors occurred during that interval to trigger the system's configurable automated disk failure response.
- A restart of the node or a restart of the HyperStore Service on the node

*Token list*

For each HyperStore data mount point, the lower section of the *ls* command response lists all the storage tokens currently assigned to that mount point. Displayed alongside the decimal version of each storage token is the base62 encoding of the token. In the HyperStore File System, base62 encoded tokens will be part of the directory structure for stored S3 object data. For example, under directory *<mount-point>/hsfs/<base62-encoded-tokenX>/...* would be the S3 object replica data associated with the token range for which *tokenX* is the upper bound. For more information on S3 storage directory structure see **"HyperStore Service and the HSFS"** (page 63).

## 10.1.5. hsstool metadata

*Subjects covered in this section:*



### 10.1.5.1. Command Syntax

#### Synopsis

```
# hsstool [-h <host>] metadata <bucket>/<object> [-v <version>]
```

#### Options

*<bucket>/<object>  Bucket and object name*

(Mandatory) Bucket name, followed by a forward slash, followed by the full object name (including "folder path", if any). For example, *mybucket/file1.txt* or *mybucket/Videos/Vacation/Italy_2021-06-27.mpg*.

If the object name has spaces in it, enclose the *bucket/object* name pair in quotes. For example, *"mybucket/big document.doc"*.

The *bucket/object* name is case-sensitive.

> **Note** In the CMC UI implementation of this command, you enter the bucket name and the full object name (including folder path) in separate fields. For example, bucket name *mybucket* and full object name *Videos/Vacation/Italy_2021-06-27.mpg*.

*-v <version>  Object version*

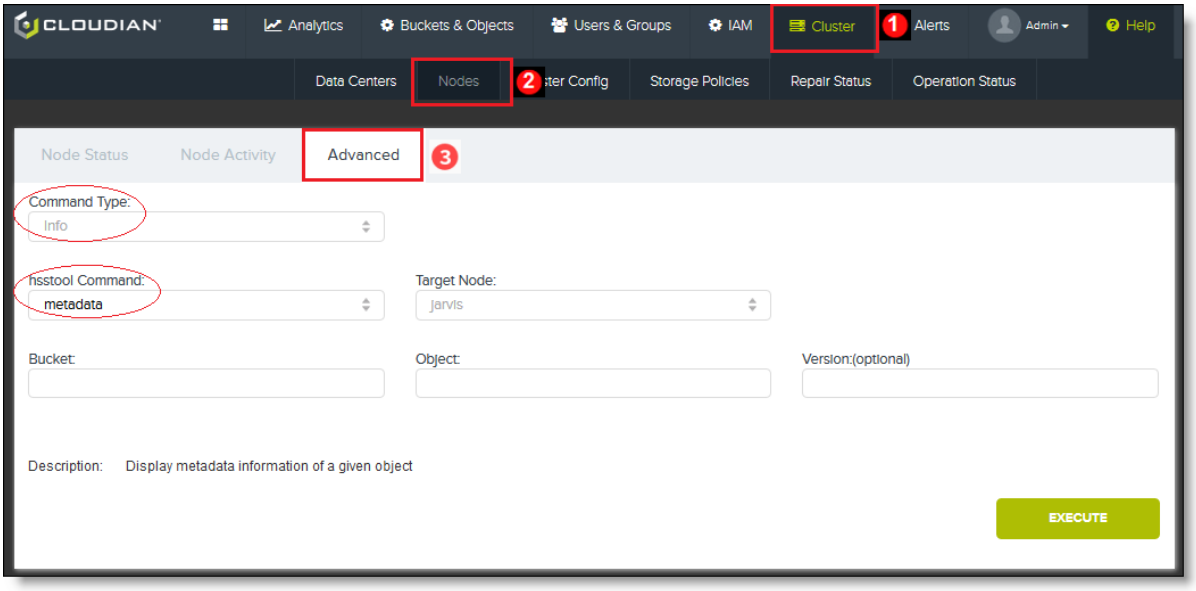
(Optional) Version ID of the object, if versioning has been used for the object. Versions are identified by *timeuuid* values in hexadecimal format (for example, "fe1be647-5f3b-e87f-b433-180373cf31f5"). If versioning has been used for the object but you do not specify a version number in this field, the operation returns metadata for the most recent version of the object.

### 10.1.5.2. Usage Notes

This **hsstool** command returns metadata for a specified S3 object, such as the object size and the date-time that the object was last accessed by an S3 client application.

#### CMC Support for This Command

As an alternative to running *hsstool metadata* on the command line, you can run it through the CMC UI:




## 10.1.5.3. Command/Response Example

The *metadata* command example below returns the metadata for the specified object.

```
# hsstool -h cloudian-node1 metadata hsfsbn/so
Key: hsfsbn/so
Policy ID: 43c3e277945403c98e5b8f9441d85e16
Version: null
Compression: NONE
Create Time: 2020-07-02T06:16:54.681Z
Last Modified: 2020-07-02T06:16:54.681Z
Last Access Time: 2020-07-02T06:16:54.681Z
Digest: 7bcec86b667ff2be2982f637b67e4942
Size: 1048576
Region: region1

CLOUDIAN_METADATA metadata: {"Path": "hsfsbn/so", "Type": "FILE", "Etag":
"7bcec86b667ff2be2982f637b67e4942", "Locale": null, "GroupId": "CloudianTest1","UserId":
"77600d5e7dfb7e0f46a198fbded86fe3", "CreatorId": "77600d5e7dfb7e0f46a198fbded86fe3",
"Ttl": null, "CreateTime": "2020-07-02T06:16:54.681Z", "ModifyTime": null, "ContentType":
null, "HttpHeaders": null, "Size": 1048576, "UserMetadata": null, "Part": 0, "Acl": null,
"Version": null, "DeleteMarker": false, "Uri": "file://", "PartSize": 10485760, "UploadId":
null, "PartInfo": null, "Policy": null, "PublicUrl": null, "HyperStoreVersion": "4",
"BlobVersion": null, "TotalSize": "1048578", "websiteIndex": null, "websiteError": null,
"websiteRedirect": null, "encryptionKey": null, "encryptionInitVec": null, "Lifecycle": null,
"Compression": null, "LastAccessTime": null, "TransitionState": null, "Replication": null,
"ReplicationState": null, "Tagging": null, "WriteTime": "1593670614681614929-0A140151",
"DeleteMarkerExpired": null, "TransitionedVersion": null, "EventNotification": null,
"chunkLevelHashValues": null, "AccumulateSize": null, "LockInfo": null}

CLOUDIAN_OBJMETADATA metadata: {"Path": "hsfsbn/so", "Type": "FILE", "Etag":
"7bcec86b667ff2be2982f637b67e4942", "Locale": null, "GroupId": "CloudianTest1",
"UserId": "user1", "CreatorId": "77600d5e7dfb7e0f46a198fbded86fe3", "Ttl":
"CHUNK_SIZE=10485760", "CreateTime": "2020-07-02T06:16:54.681Z", "ModifyTime": null,
```

```
"ContentType": "application/octet-stream", "HttpHeaders": "", "Size": 1048576,
"UserMetadata": "", "Part": 0, "Acl": null, "Version": null, "DeleteMarker": null,
"Uri": "file://", "PartSize": 10485760, "UploadId": null, "PartInfo": null, "Policy":
null, "PublicUrl": null, "HyperStoreVersion": "4", "BlobVersion": null, "TotalSize":
"1048578", "websiteIndex": null, "websiteError": null, "websiteRedirect": null,
"encryptionKey": null, "encryptionInitVec": null, "Lifecycle": null, "Compression":
null, "LastAccessTime": null, "TransitionState": null, "Replication": null,
"ReplicationState": null, "Tagging": null, "WriteTime": "1593670614681614929-0A140151",
"DeleteMarkerExpired": null, "TransitionedVersion": null, "EventNotification": null,
"chunkLevelHashValues": [{"bytes": "{ÎÈkf\u007Fò¾)\u0082ö7¶~IB"}], "AccumulateSize":
1048576, "LockInfo": null}
```


## Response Items

*Key*

Key that uniquely identifies the S3 object, in format <*bucketname*>/<*objectname*>.

*PolicyID*

System-generated identifier of the storage policy that applies to the bucket in which this object is stored.

*Version*

Object version, if versioning has been used for the object. Versions are identified by *timeuuid* values in hexadecimal format. If versioning has not been used for the object, the Version field displays "Null".

*Compression*

Compression type applied to the object, if any.

*Create Time*

Timestamp for the original creation of the object. Format is ISO 8601 and the time is in Coordinated Universal Time (UTC).

*Last Modified*

Timestamp for last modification of the object. Format is ISO 8601 and time is in UTC.

*Last Access Time*

Timestamp for last access of the object. Last Access Time is subject to updating only for objects in buckets for which a lifecycle is configured. For such objects, Last Access Time is updated if the object is accessed either for retrieval (GET or HEAD) or modification (PUT/POST/Copy). Format is ISO 8601 and time is in UTC.

*Digest*

MD5 digest of the object. This will be a 32 digit hexadecimal number. This digest is used in a variety of operations including data repair.

*Size*

The object's size in bytes.

*Region*

The HyperStore service region in which the object is stored.

*CLOUDIAN_METADATA and CLOUDIAN_OBJMETADATA metadata*

This is additional raw metadata for the object from the Cassandra CLOUDIAN_METADATA column family (in which object metadata is organized per bucket) and CLOUDIAN_OBJMETADATA column family (in which object metadata is organized per object). This raw object metadata may be useful if you are working Cloudian Support to troubleshoot an issue in regard to the object.

**Note** There is overlap in the content of these two sets of raw object metadata.

## 10.1.6. hsstool opctl

*Subjects covered in this section:*



### 10.1.6.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> opctl [-l] [-stop]
```

#### Options

*-l List all repair and cleanup operations*

(Optional) List all in-progress *repair*, *repairec*, *cleanup*, and *cleanupec* operations in the service region.

*-stop Stop all repair and cleanup operations*

(Optional) Terminate all in-progress *repair*, *repairec*, *cleanup*, and *cleanupec* operations in the service region.

### 10.1.6.2. Usage Notes

This **hsstool** command returns a list of all **hsstool repair**, **hsstool repairec**, **hsstool cleanup**, and **hsstool cleanupec** operations **currently running** in a service region. You can also use the command to **stop** all those in-progress operations.

The target *<host>* can be any node in the service region. The command will apply to all nodes in the service region.

With this command you must use either the *-l* option or the *-stop* option ("*hsstool opctl*" by itself doesn't do anything).

**Note** The *hsstool opctl* command is not supported in the CMC UI.

### 10.1.6.3. Command/Response Examples

The example below shows the responses to *hsstool opctl* commands. The first command lists the in-progress repair and cleanup operations in the cluster (in this example only a replica repair operation is in progress). The second command stops the in-progress operation(s).

```
# hsstool -h cloudian-node1 opctl -l
10.10.0.184:
REPAIR: rebuild=false,keyspaces=nokeyspaces,max-modified-ts=1526942350092,primary-range=false,
min-modified-ts=0,logging=true,full-repair=true,check-metadata=true,cmdno=1,merkletree=true,
computedigest=false
```

```
# hsstool -h cloudian-node1 opctl -stop
Aborted REPAIR on 10.10.0.184
```

## 10.1.7. hsstool opstatus

*Subjects covered in this section:*



### 10.1.7.1. Command Syntax

#### Synopsis

```
# hsstool [-h <host>] opstatus [<operation>] [-a] [-q history] [-d <operation>]
```

#### Options

*<operation>  Operation type*

(Optional) Type of operation for which to retrieve status. For example:

```
hsstool -h <host> opstatus repair
```

Valid operation types are listed below. If you do not specify a type, status is returned for all supported operation types.

- cleanup
- cleanupec
- decommissionreplicas
- decommissionec
- proactiverebalance
- proactiverepair
- proactiverepairec
- rebalance
- repair
- repaircassandra
- repairec

Note that:

- The CMC does not support the "<operation>" option. In the CMC if you run opstatus it will return status for all operation types.
- The *decommission* operation is automatically invoked by the CMC's "Uninstall" feature if you uninstall a node that's "live" in the Cassandra ring. Status reporting on a decommission operation is broken out to *decommissionreplicas* (for replicated data) and *decommissionec* (for erasure coded data).
- The *repaircassandra* operation is invoked not only when **"hsstool repaircassandra"** (page 553) is explicitly run but also when *hsstool repair* is run either with its default behavior (which includes a repair of user data keyspaces in Cassandra) or with the "allkeyspaces" option (which includes a repair of user data keyspaces and service metadata keyspaces in Cassandra).

*-a Verbose output*

(Optional) Verbose status output for a *repair* or *repairec* or *rebalance* operation. The additional status detail that this option provides can be helpful if you are working with Cloudian Support to troubleshoot a repair or rebalance problem.

For example:

```
hsstool -h <host> opstatus repair -a
```

**Note** The CMC does not support the "-a" option.

*-q history Get operation history*

(Optional) Use *hsstool -h <host> opstatus -q history* to retrieve a history of rebalance, repair, and cleanup options performed on the target node. By default this history spans the past 90 days. This period is configurable by *mts.properties.erb*: **"monitoring.ophistory.ttl"** (page 457).

If you want just a history of a particular repair or cleanup operation type, use:

```
hsstool -h <host> opstatus -q history <operation>
```

where <operation> is *rebalance*, *repair*, *repairec*, *cleanup*, or *cleanupec*. For example:

```
hsstool -h <host> opstatus -q history repairec
```

**Note** The CMC does not support the "-q history" option.

*-d <operation> Delete operation status*

Only use this option if directed to do so by Cloudian Support. This may be needed in the unlikely event that an erroneous or obsolete operation status left over in the Metadata DB is preventing you from initiating a new operation.

### 10.1.7.2. Usage Notes

This **hsstool** command returns the status of the most recent runs of repair, cleanup, rebalance, or decommission operations that have been performed on a specified node. For each operation type:

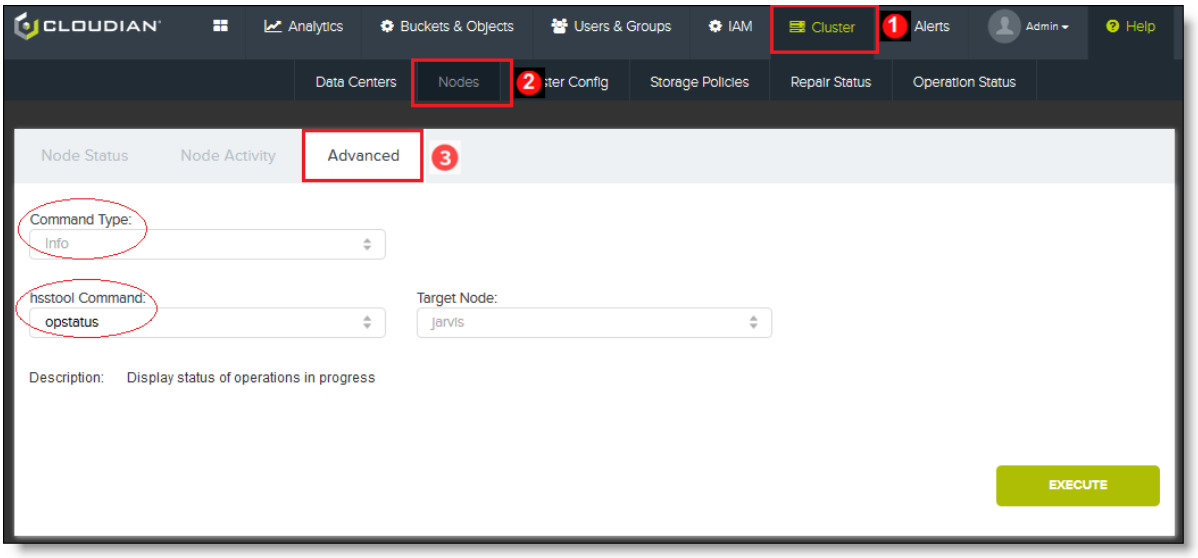
- If a run of the operation is **in progress** on the node, then that's the run for which status is returned.
- If the operation is not currently in progress on the node, then status is returned for the **most recent** run of that operation on the node, in the time since the last restart of the node.

For checking the status of repair runs other than the most recent run, *opstatus* also supports a command line option to return a **90-day history** of repairs performed on the target node.

> **Note** For operations that you've launched through the CMC UI, a convenient way to check operation status is through the CMC's **Operation Status** page, which displays the same output as *opstatus* (**Cluster -> Operation Status**). This CMC page does not report on operations that you've launched on the command line -- for such operations, explicitly running *hsstool opstatus* is your only option for checking status.

### CMC Support for This Command

As an alternative to running *hsstool opstatus* on the command line, you can run it through the CMC UI:




## 10.1.7.3. Status metrics for *cleanup* and *cleanupec*

The following shows sample output for *opstatus cleanup*. See further below for status metric descriptions. (For more information about the operation type that this is reporting on, see **"hsstool cleanup"** (page 504)).

```
# hsstool -h cloudian-node1 opstatus cleanup
optype: CLEANUP cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true deletedata=true
deleteobject-without-policy=false keyspaces=UserData no-delete-out-of-range=false logging=true
relocate-objects=false cmdno=1 delete-only-outofrange-objects=false
operation ID: 765de07e-8f27-162d-a913-000c2938a438
start: Mon Nov 21 10:11:20 PST 2022
end: Mon Nov 21 10:32:58 PST 2022
duration: 1297.241 sec
progress percentage: 100%
time remaining: 0 ms
cassandra cleanup time: 0.029 sec
task count: 1313587
completed count: 1313587
Number of files deleted count: 10
failed count: 0
```

```
skipped count: 1313577
rcvd connection count: 0
open connection count: 0
job threadpool size: 10
message threadpool active: 0
total size of cleanup objects: 100.00 MB
timer:  batch.copies.find.timer: count=442 mean=43036.53462127773; batch.fragments.find.timer:
count=0 mean=NaN; batch.cleanup.file.timer: count=442 mean=4.490631563368095E9;
total amount of object data deleted: 102.50 MB
```

The following shows same output for *opstatus cleanupec*. See further below for status metric descriptions. (For more information about the operation type that this is reporting on, see **"hsstool cleanupec"** (page 509)).

```
# hsstool -h cloudian-node1 opstatus cleanupec
optype: CLEANUPEC cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true deletedata=true
deleteobject-without-policy=false no-delete-out-of-range=false logging=true
relocate-objects=false cmdno=1 delete-only-outofrange-objects=false
operation ID: 407d4d16-d08b-1a22-88ed-0a1769c69819
start: Tue Feb 15 03:19:08 UTC 2022
end: Tue Feb 15 03:25:01 UTC 2022
duration: 5 minutes 53 seconds
progress percentage: 100%
time remaining: 0 ms
task count: 200000
completed count: 200000
Number of files deleted count: 10000
failed count: 0
skipped count: 190000
rcvd connection count: 0
open connection count: 0
job threadpool size: 10
message threadpool active: 0
total size of cleanup objects: 2.44 GB
timer:  batch.copies.find.timer: count=0 mean=NaN; batch.fragments.find.timer: count=68
mean=85069.69420885573; batch.cleanup.file.timer: count=68 mean=5.230084050540501E9;
total amount of object data deleted: 221.00 MB
```

### Metric Descriptions for *cleanup* and *cleanupec*

Unless otherwise noted these metrics apply to both *cleanup* and *cleanupec.*

*optype*

The type of *hsstool* operation.

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For high-level information about object cleanup successes and failures (if any), see the other fields in the *cleanup* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *cleanup* response. For details on any FAILED operation you can also scan *cloudian-hyperstore.log* for error messages from the period during which the operation was running.

A TERMINATED status means that the cleanup run was terminated by an operator, using *cleanup -stop*, or by some other interruption such as the HyperStore Service doing down on the target node or the target node being rebooted.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear.

*operation ID*

Globally unique identifier of the *cleanup* run. This may be useful if Cloudian Support is helping you troubleshoot a repair failure.

> **Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

*start*

Start time of the operation.

*end, duration*

For a completed operation, the end time and duration of the operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*time remaining*

For an in-progress operation, the estimated time remaining.

*estimated completion time*

For an in-progress operation, the estimated completion time.

*cassandra cleanup time (cleanup only)*

The time spent cleaning metadata in Cassandra (the Metadata DB). This is applicable only to a *cleanup* operation, not a *cleanupec* operation.

*task count*

The number of object replicas (in the case of *cleanup*) or erasure code fragments (in the case of *cleanupec*) on the node, which the operation must evaluate to determine whether they correctly belong on the node.

*completed count*

The number of object replicas (in the case of *cleanup*) or erasure code fragments (in the case of *cleanupec*) that the operation evaluated to determine whether they correctly belong on the node.

*Number of files deleted count*

The number of object replicas (in the case of *cleanup*) or erasure code fragments (in the case of *cleanupec*) that the operation successfully deleted from the node because they don't belong on the node.

> **Note** If you use the *-n* option when you run the *cleanup* command, this response item will be "Number of files to be deleted count" rather than "Number of files deleted count".

*failed count*

The number of object replicas (in the case of *cleanup*) or erasure code fragments (in the case of *cleanupec*) that the operation tried to delete (because they don't belong on the node), but failed.

*skipped count*

The number of object replicas (in the case of *cleanup*) or erasure code fragments (in the case of *cleanupec*) that the operation left on the node because they belong on the node.

*rcvd connection count*

In support of implementing the operation, the current number of open TCP connections incoming to the target node (the node on which you ran the *cleanup* or *cleanupec*command) from the other nodes in the cluster.

*open connection count*

In support of implementing the operation, the current number of open TCP connections outgoing from the target node (the node on which you ran the *cleanup* or *cleanupec*command) to the other nodes in the cluster.

*job threadpool size*

The size of the job threadpool for the operation. This determines how many cleanup 'jobs' can run concurrently on a node. For more information about how *cleanup* and *cleanupec* work is divided into high-level 'jobs' see the description of the configuration property **"cleanupjobs.threadpool.corepoolsize"** (page 436).

*message threadpool active*

In support of implementing the operation, the current number of active threads managing communications between the target node (the node on which you ran the *cleanup* or *cleanupec*command) and the other nodes in the cluster.

*timer*

This shows detailed timing metrics for various parts of the operation. These timing metrics may be useful if Cloudian Support is working with you to troubleshoot *cleanup* or *cleanupec* performance issues in your environment.

*total amount of object data deleted*

The total amount of object data deleted from the target node (the node on which you ran the *cleanup* or *cleanupec*command) as a result of the operation.

In the case of a *cleanup* operation (cleaning replica data from the target node), for a hypothetical object that is 10MB in size, if one of that object's replicas is cleaned from the target node, that's 10MB removed from the node (since each replica of a 10MB object is 10MB).

In the case of a *cleanupec* operation (cleaning erasure coded data from the target node), for a hypothetical object that is 10MB in size -- and supposing a 4+2 erasure coding storage policy is being used -- if one of that

object's erasure coded fragments is cleaned from the target node, that's 2.5MB removed from the node (since each erasure coded fragment of a 10MB object is 2.5MB if 4+2 erasure coding is being used).

### 10.1.7.4. Status metrics for *repair* and *repairec*

The following shows sample output for *opstatus repair*. See further below for status metric descriptions. (For more information about the operation type that this is reporting on, see **"hsstool repair"** (page 548)).

```
# hsstool -h cloudian-node1 opstatus repair
optype: REPAIR cmdno#: 2 status: COMPLETED
operation ID: 765d84e0-8f27-162d-a913-000c2938a438
start: Thu Feb 23 05:58:09 PST 2023
end: Thu Feb 23 05:58:11 PST 2023
duration: 1 second
progress percentage: 100%
time remaining: 0 ms
arguments: rebuild=false keyspaces=UserData max-modified-ts=1677160689935 primary-range=false
min-modified-ts=0 logging=true full-repair=true check-metadata=true cmdno=2 merkletree=true
computedigest=false
total range count: 6
executed range count: 6
failed range count: 0
keyspace count: 1
repair file count: 2
failed count: 0
repaired count: 2
pr queued count: 0
completed count: 23
total bytes streamed: 305.93 KiB
scan time: 0 ms
metadata time: 0 ms
stream time: 0 ms
```

The following shows sample output for *opstatus repairec*. See further below for status metric descriptions. (For more information about the operation type that this is reporting on, see **"hsstool repairec"** (page 555)).

```
# hsstool -h cloudian-node1 opstatus repairec
optype: REPAIREC cmdno#: 1 status: COMPLETED
arguments: mountPoint=null logging=true range=null cmdno=1 computedigest=false oldRepairec=false
operation ID: f06d79e2-af53-1ef7-858a-0a749a422815
start: Thu Jul 14 09:04:26 UTC 2022
end: Thu Jul 14 10:36:50 UTC 2022
duration: 1 hour 32 minutes 23 seconds
progress percentage: 100%
time remaining: 0 ms
total ranges: 0
object count: 1000018
task count: 1000000
completed count: 1000000
repaired count: 667531
failed count: 0
skipped count: 332469
rcvd connection count: 0
open connection count: 0
```

```
message threadpool active: 0
timer:  cassandra.iterating.timer: count=1000021 mean=1263.4357607049558; metrics.batch.executing:
count=500 mean=1.4656684591620892E11; metrics.batch.idle: count=500 mean=1.7746020372798523E11;
count.estimator.timer: count=1 mean=5.005086883E9; metrics.batch.waiting: count=500
mean=539844.2383711435;
```

### Metric Descriptions for *repair*

*optype*

The type of *hsstool* operation.

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED.

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For example in the case of repair, a COMPLETED status means that all objects in the scope of the operation were checked to see if they needed repair. It does not mean that all objects determined to need repair were successfully repaired. For high-level information about object repair successes and failures (if any), see the other fields in the *repair* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *repair* response. For details on any FAILED operation you can also scan *cloudian-hyperstore.log* for error messages from the period during which the operation was running. More details can also be had by running *hsstool -h <host> opstatus repair -a* on the command line.

A TERMINATED status means that the repair run was terminated by an operator, using *repair -stop*, or by some other interruption such as the HyperStore Service doing down on the target node or the target node being rebooted.

*operation ID*

Globally unique identifier of the *repair* run. This may be useful if Cloudian Support is helping you troubleshoot a repair failure.

> **Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

*start*

Start time of the operation.

*end, duration*

End time and duration of a completed operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*time remaining*

Approximate time remaining for the operation. This is a rough estimate.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear. For example, *hsstool repair* command-line syntax supports a "-pr" option, and within "arguments" response item the use or non-use of this option is indicated as "primary-range=true" or "primary-range=false".

*total range count*

The total number of token ranges that will be repaired during this repair operation. The repair operation will encompass not only the token ranges assigned to the target repair node but also certain token ranges on other nodes in the cluster. These are token ranges in which are stored replicas of the same objects that are on the target repair node. As a simplified example, if objects residing in token range "c to d" on the target repair node are also replicated in ranges "d to e" and "e to f" on other nodes in the cluster, then ranges "d to e" and "e to f" will be among the ranges repaired by the repair operation (as well as "c to d"). Consequently the total number of ranges to be repaired will be larger than the number of tokens assigned to the node that is the target of the repair.

The exception is if the "-pr" option was used when the *hsstool repair* operation was executed, in which case the repair operation addresses only the target node's "primary ranges". In this case the "total range count" value will equal the number of tokens assigned to the node.

**Note** For repair status detail for each token range, run *hsstool -h <host> opstatus repair -a*. This detailed, per-range status information can be helpful if you are working with Cloudian Support to troubleshoot repair problems.

*executed range count*

For each of the token ranges in the "total range count" metric, the repair thread pool schedules a sub-job. The "executed range count" metric indicates the number of sub-jobs scheduled. This does not necessarily mean that all these ranges were successfully repaired -- only that the per-range sub-jobs were scheduled by the thread pool.

*failed range count*

The number of token ranges for which the range repair sub-job did not run successfully. For example if range repair sub-jobs are scheduled by the thread pool but then you subsequently terminate the *repair* operation (using the *-stop* option) or reboot the node, this will result in some failed ranges. Problems with the endpoint nodes or the disks impacted by repair of particular token ranges are other factors that may result in failed ranges.

For information about such failures, on the target node for the repair you can scan */var/log/cloudian/cloudian-hyperstore-repair.log* for the time period during which the repair operation was running. Running *hsstool -h <host> opstatus repair -a* on the command line will also provide useful details about repair failures.

*keyspace count*

Number of Cassandra keyspaces repaired. With the default repair behavior this will equal the number of storage policies that are in your HyperStore system. There is one Cassandra *UserData_<policyid>* keyspace for

each storage policy. This is where object metadata is stored.

*repair file count*

Of all the replica files evaluated by the repair operation, this is the number of files that were determined to be in need of repair. This figure may include files on other nodes as well as files on the target repair node. For example, if an object is correct on the target node but one of the object's replicas on a different node is missing and needs repair, then that counts as one toward the repair file count. For a second example, if two of an object's three replicas are found to be out-dated, that counts as two toward the "repair file count".

*failed count*

Of the files that were found to be in need of repair, the number of files for which the attempted repair failed. For information about such failures, on the target node for the repair you can scan */var/log/cloudian/cloudian-hyperstore-repair.log* for the time period during which the repair operation was running. Running *hsstool -h <host> opstatus repair -a* on the command line will also provide useful details about repair failures.

If possible, files for which the repair attempt fails are added to the proactive repair queue (see "pr queued count" below).

The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

> **Note** One thing that can increment the "failed count" is if the operation entails writing data to a disk that is in a **stop-write** condition (which by default occurs when a disk is 90% full). Such write attempts will fail.

*repaired count*

Of the files that were found to be in need of repair, the number of files for which the repair succeeded. The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

*pr queued count*

The number of files that the *hsstool repair* operation adds to the proactive repair queue, to be fixed by the next proactive repair run. If the *hsstool repair* operation fails to repair a file that requires repair, it adds the file to the proactive repair queue. Proactive repair is a different type of repair operation and may succeed in cases where regular *hsstool repair* failed. By default proactive repair runs every 60 minutes.

The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

*completed count*

The total number of files that were assessed to see if they were in need of repair. This number reflects replication across the cluster — for example, if an object is supposed to be replicated three times (with one replica on the target repair node and two replicas on other nodes), then repair assessment of that object counts as three files toward the "completed count".

Note that "completed" here does not necessarily mean that all object repair attempts succeeded. For more information on success or failure of object repair attempts, see the other status metrics.

*total bytes streamed*

Total number of bytes streamed to the target repair node or other nodes in order to implement repairs. For example, if a 50000 byte object on the target repair node is found to be out-dated, and an up-to-date replica of the object is streamed in from a different node, that counts as 50000 toward "total bytes streamed". As a second example, if a 60000 byte object exists on the target node, and replicas of that object are supposed to exist on

two other nodes but are found to be missing, and the repair streams the good object replica from the target repair node to the two other nodes — that counts as 120000 toward "total bytes streamed".

*scan time*

Total time that it took to scan the file systems and build the Merkle Tree that is used to detect discrepancies in object replicas across nodes.

*metadata time*

Total time that it took to retrieve from the Metadata DB the metadata needed to perform the operation.

*stream time*

Total time that it took to stream replicas across nodes, in order to implement needed repairs.

### Metric Descriptions for *repairec*

*optype*

The type of *hsstool* operation.

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED.

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For example in the case of repair, a COMPLETED status means that all objects in the scope of the operation were checked to see if they needed repair. It does not mean that all objects determined to need repair were successfully repaired. For high-level information about object repair successes and failures (if any), see the other fields in the *repairec* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *repairec* response. For details on any FAILED operation you can also scan *cloudian-hyperstore-repair-failure.log* for error messages from the period during which the *repairec* operation was running.

A TERMINATED status means that the repair run was terminated by an operator, using *repairec -stop*, or by some other interruption such as the HyperStore Service doing down on the target node or the target node being rebooted.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear. For example, *hsstool repair* command-line syntax supports a "-pr" option, and within "arguments" response item the use or non-use of this option is indicated as "primary-range=true" or "primary-range=false".

*operation ID*

Globally unique identifier of the *repairec* run. This may be useful if Cloudian Support is helping you troubleshoot a repair failure.

**Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

*start*

Start time of the operation.

*end, duration*

End time and duration of a completed operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the approximate percentage of work that has been completed so far.

*time remaining*

Estimated time remaining to complete the operation.

*total ranges*

The total number of token ranges for which data is being evaluated to determine if it needs repair.

*task count*

In the HyperStore File System, objects are stored as "chunks". Objects smaller than or equal to the chunk size threshold (10MB) are stored as a single chunk. Objects larger than the chunk size threshold are broken into and stored as multiple chunks, with no chunk exceeding the threshold in size. In the case of large objects that S3 client applications upload to HyperStore by the Multipart Upload method, HyperStore breaks the individual parts into chunks if the parts exceed the chunk size threshold. In the context of erasure coding storage policies, after objects (or object parts) are broken into chunks, each object chunk is erasure coded.

In the *repairec* operation, the evaluation of a single chunk -- to determine whether all of its erasure coded fragments are present on the nodes on which they should be stored -- constitutes a single "task". For example, the evaluation of a 100MB object that has been broken into 10 chunks -- each of which has been erasure coded using a 4+2 erasure coding scheme -- would count as 10 "tasks", with one task per chunk.

The "task count" metric, then, is the total number of chunks that are being evaluated to determine whether any of them are in need of repair.

*completed count, repaired count, failed count, skipped count*

The "completed count" is the number of tasks completed so far, from among the tasks in the "task count" (for the definition of a "task" see the description of "task count" above). A "completed" task means that an erasure coded object chunk was evaluated and, if it needs repair, an attempt was made to repair it.

A completed task has one of three possible results: a successful repair, a failed repair attempt, or the determination that the chunk does not need repair (i.e., all of the chunk's fragments are in the proper locations within the cluster). These results are tallied by other *repairec* response metrics:

- "repaired count" -- The number of erasure coded object chunks for which a repair was found to be necessary and was successfully executed.
- "failed count" -- The number of erasure coded object chunks for which a repair was found to be necessary, but the repair attempt failed.

- "skipped count" -- The number of erasure coded object chunks that were evaluated and determined not to need repair.

The "repaired count", "failed count", and "skipped count" should add up to equal the "completed count".

**Note** Notice from the descriptions above that a failed repair attempt counts as a "completed" task. In other words, "completed" in this context does not necessarily mean success. It means only that the *repairec* operation has finished its processing of that chunk, resulting in one of the three outcomes described above.

**Note** If the "completed count" is less than the "task count" this means that the repair was interrupted in such a way that some erasure coded object chunks were identified by a scan of object metadata in Cassandra (and thus counted toward the "task count") but were not yet evaluated or repaired.

To get more detailed metrics about **repair failures** -- in the event that the "failed count" in the *opstatus repairec* response is non-zero -- you can run *hsstool -h <host> opstatus repairec -a* (the *-a* is the verbose output flag).This will return the same status metrics that are returned by *opstatus repairec*, followed by a categorization of repair failures (if any) into various failure types. The response excerpt below is for an operation in which no failures occurred.

```
...
Reason: CONNECTION_ERROR Count: 0
Reason: UNKNOWN Count: 0
Reason: CREATE_TASK_FAILED Count: 0
Reason: EC_DECODE_FAILED Count: 0
Reason: EC_ENCODE_FAILED Count: 0
Reason: REPAIR_TASK_EXPIRED Count: 0
Reason: REPAIR_CYCLE_EXCEEDED Count: 0
Reason: REPAIR_MESSAGE_REQUEST_FAILED Count: 0
Reason: CASSANDRA_CHECK_FAILED Count: 0
Reason: NODE_DOWN Count: 0
```

These same metrics for failure types are logged in *cloudian-hyperstore.log*, as part of the standard logging of *repairec* operations.

Also, the log file *cloudian-hyperstore-repair.log* records an entry for each object that the *repairec* operation was able to repair; and *cloudian-hyperstore-repair-failure.log* records an entry for each object that the *repairec* operation was unable to repair.

For more information about these logs see **"HyperStore Service Logs"** (page 354).

*rcvd connection count*

In support of implementing the repair operation, the current number of open TCP connections incoming to the target node (the node on which you ran the *repairec* command) from the other nodes in the cluster.

*open connection count*

In support of implementing the repair operation, the current number of open TCP connections outgoing from the target node (the node on which you ran the *repairec* command) to the other nodes in the cluster.

*message threadpool active*

In support of implementing the repair operation, the current number of active threads managing

communications between the target node (the node on which you ran the *repairec* command) and the other nodes in the cluster.

*timer*

This shows detailed timing metrics for various parts of the *repairec* operation. These metrics may be useful if Cloudian Support is working with you to troubleshoot *repairec* performance issues in your environment.

Note that even more detailed timing information for a completed *repairec* operation is available in the **HyperStore Service application log**. The timing metrics lines in the log are preceded by a line that says "Performance meters".

### 10.1.7.5. Status metrics for *rebalance*

The following shows sample output for *opstatus rebalance*. See further below for status metric descriptions. (For more information about the operation type that this is reporting on, see **"hsstool rebalance"** (page 543)).

```
# hsstool -h cloudian-node1 opstatus rebalance
optype: REBALANCE cmdno#: 1 status: COMPLETED
arguments: cleanup=false logging=false cmdno=1 cleanupFile=true
operation ID: 763d04a2-f8bc-1901-a07c-5254000f88dc
start: Thu Nov 14 09:54:33 CST 2022
end: Thu Nov 14 10:02:43 CST 2022
duration: 489.969 sec
progress percentage: 100%
time remaining: 0 ms
task count: 2033
completed count: 2033
streamed count: 1283
failed count: 0
skipped count: 0
rcvd connection count: 0
open connection count: 0
prqueued count: 750
job threadpool size: 4
message threadpool active: 0
stream jobs total: 8
stream jobs completed: 8
stream jobs failed: 0
streamed bytes: 1491288352
delete count: /10.20.2.135=186  /10.20.2.172=342  /10.20.2.173=328
/10.20.2.174=319  /10.20.2.136=0  /10.20.2.171=108
```

#### Metric Descriptions for *rebalance*

*optype*

The type of *hsstool* operation.

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED.

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For additional status detail see the other fields in the operation status response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the operation status response. For details on any FAILED operation you can also scan *cloudian-hyperstore.log* for error messages from the period during which the operation was running.

A TERMINATED status means that the rebalance run was terminated by an operator using *rebalance -stop*, or by some other interruption such as the HyperStore Service doing down on the target node or the target node being rebooted.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear.

*operation ID*

Globally unique identifier of the *rebalance* run. This may be useful if Cloudian Support is helping you troubleshoot a rebalance failure. Note that when *rebalance* is run, the REBALANCE part of the response (for replica data) and REBALANCEEC part of the response (for erasure coded data) will both have the same operation ID.

> **Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

*start*

Start time of the operation.

*end, duration*

End time and duration of a completed operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*time remaining*

Estimated time remaining to complete the operation.

*task count*

From the existing cluster, the total number of files that are evaluated for possible streaming (copying) to the newly added node, based on the newly added node's assigned tokens. Each such file constitutes a "task".

*completed count*

From the total task count, the number of tasks that have been completed so far -- that is, the number of files that the system has evaluated and if appropriate has attempted to stream to the new node. Each "completed" task results in the incrementing of either the "streamed count", the "failed count", the "skipped count", or the "prqueued count".

At the end of the operation the "completed count" should equal the "task count".

*streamed count*

The number of files successfully streamed (copied) from the existing nodes to the new node.

*failed count*

The number of files for which the attempt to stream the file to the new node fails, and the resulting attempt to insert the file stream job into the proactive repairqueue fails also.

> **Note** If the stream attempt for a file fails, but the stream job for that file is successfully added to the proactive repair queue, that file is counted toward the "prqueued count" -- not the "failed count".

For detail about rebalance streaming failures, on the target node for the rebalance operation (the new node) you can scan */var/log/cloudian/cloudian-hyperstore.log* for error messages from the time period during which the rebalance operation was running.

*skipped count*

For rebalancing of replicated object data: Replicas of each object to be streamed (copied) to a newly added node will typically exist on multiple existing nodes. For example, in a 3X replication environment, for a given object that should be streamed to the new node (based on the new node's token ranges), typically a replica file will reside on three of the existing nodes. The evaluation and processing of each such replica file counts towards the "task count" (so, 3 toward the task count in our example). But once a replica is streamed from one existing node to the new node, it doesn't need to be streamed from the other two existing nodes to the new node. On those other two existing nodes the replica file is "skipped".

*rcvd connection count*

In support of implementing the rebalance operation, the current number of open TCP connections incoming to the target node (the new node on which you ran the *rebalance* command) from the other nodes in the cluster.

*open connection count*

In support of implementing the rebalance operation, the current number of open TCP connections outgoing from the target node (the new node on which you ran the *rebalance* command) to the other nodes in the cluster.

*prqueued count*

The number of files for which the attempt to stream the file to the new node fails (even after automatic retries), but the stream job for that file is successfully added to the proactive repair queue. These stream jobs will then be automatically executed by the next run of the hourly proactive repair process.

*job threadpool size*

The size of the threadpool dedicated to processing "jobs" on the target node (the new node on which you ran the *rebalance* command). The threadpool size determines how many jobs can run concurrently on the new node. For more information about rebalance jobs see the description of "stream jobs total" below.

*message threadpool active*

In support of implementing the rebalance operation, the current number of active threads managing communications between the target node (the new node on which you ran the *rebalance* command) and the other nodes in the cluster.

*stream jobs total*

The system breaks the overall rebalance streaming workload down into multiple "jobs", with each job entailing streaming (copying) a subset of the data from an existing node to the new node. This metric shows the total number of jobs into which the rebalance streaming workload has been divided. The total depends on factors such as the number of storage policies in the system, the number of existing nodes in the cluster to which the new node is being added, and the number of token ranges involved in the rebalancing.

**Note** For rebalance status detail for each individual job, run *hsstool -h <host> opstatus rebalance -a*. This detailed information can be helpful if you are working with Cloudian Support to troubleshoot rebalance problems.

*stream jobs completed*

The number of stream jobs completed so far. Note that "completed" means that work on the job has ended and does not necessarily mean success -- if any of the stream jobs ended in failure, they will count toward the "stream jobs failed" metric.

*stream jobs failed*

The number of stream jobs that failed.

**Note** If the rebalance operation as a whole has finished with status COMPLETED, but the 'stream jobs failed' metric indicates that some stream jobs failed, you can run *hsstool -h <host> rebalance -retry* to have the system try again to execute those stream jobs. For more information see the description of the **-retry** option.

*streamed bytes*

The number of bytes of object data streamed to the new node.

*streamed bytes*

The number of bytes of object data streamed to the new node.

*delete count*

This metric is included in the opstatus response only if you included the "**-cleanupfiles**" option when you ran *hsstool rebalance*. It indicates how many files were deleted from each of your existing nodes, after those files were successfully copied to the new node.

## 10.1.8. hsstool proactiverepairq

*Subjects covered in this section:*

### 10.1.8.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> proactiverepairq [-a] [-delete <host>]
[-start [-type replicas|ec]] [-stop] [-enable true|false]
```

#### Options

*-a  Show verbose output*

(Optional) If you use *hsstool -h <host> proactiverepairq* -- **without the *-a* flag** or other options -- the command will return the number of nodes that are in need of proactive repair, the IP addresses of those nodes, and an **estimate** of the number of objects in need of proactive repair on those nodes.

If you use *hsstool -h <host> proactiverepairq* ***-a***, the command will return the number of nodes that are in need of proactive repair and the IP addresses of those nodes, and precise information about the count and total size of objects that are in need of proactive repair on each node. When you use the *-a* option, a scan of Cassandra metadata is done. This is a much more resource intensive operation than if you omit the *-a* option.

> **Note** To see how much proactive repair work **has already been completed** on a given node, use the **"hsstool opstatus"** (page 522) command on that node.

*-delete <host>  Delete a specified node's proactive repair queue*

(Optional) Use this option to completely clear out a specified node's current proactive repair queue. If you do so, the objects that had been in the proactive repair queue will not be fixed by proactive repair, and instead you will need to fix them by running **"hsstool repair"** (page 548) and **"hsstool repairec"** (page 555) on the node.

Under normal circumstances you should not need to use the *-delete <host>* option. You might however use this option if you are working with Cloudian Support to troubleshoot problems on a node.

> **Note** The CMC interface does not support the *-delete <host>* option. This option is supported only on the command line.

*-start [-type replicas|ec|rebalance]  Start proactive repair now*

(Optional) Use *hsstool -h <host> proactiverepairq -start* to immediately initiate proactive repair on the specified host. This applies **only to the specified host**.

Optionally you can restrict the immediate proactive repair to a particular category of proactive repair: proactive repair of replica data for an existing node; proactive repair of erasure coded data for an existing node; or proactive repair of object data streaming failures from a recently completed rebalance operation for a newly added node. For example, use *hsstool -h <host> proactiverepairq -start -type rebalance* to immediately initiate proactive repair on a new host after a rebalance operation that reported failures for some objects.

If you do not include the *-type* option, then using *-start* will immediately initiate proactive repair for all types of proactive repairs that are currently needed the target node.

> **Note** Proactive repair is triggered automatically every hour (by default configuration; see *hyperstore-server.properties.erb:***"hyperstore.proactiverepair.poll_time"** (page 437)), on all nodes that are in

need of proactive repair, for all proactive repair types. No operator action is required. So there is no need to use the *-start* option unless for some reason you want proactive repair to begin immediately on a particular node rather than waiting for the next automatic hourly run of proactive repair.

*-stop  Stop proactive repair*

(Optional) Use *hsstool -h <host> proactiverepairq -stop* to immediately stop any in-progress proactive repair on the specified host. This applies **only to the specified host**. Remaining repair tasks that are still in the proactive repair queue for that host will be processed the next time that proactive repair is run.

If proactive repair is in progress on multiple hosts and you want to stop all the proactive repairs, you must submit a *hsstool -h <host> proactiverepairq -stop* command separately for each of those hosts.

**Note** Unlike the *-start* option, the *-stop* option does not support specification of a proactive repair type. Instead the *-stop* option stops all types of in-progress proactive repair on the target host.

*-enable true|false  Enable or disable proactive repair throughout the service region*

(Optional) Enable or disable the proactive repair feature. By default the feature is enabled.

If you use *hsstool -h <host> proactiverepairq -enable false* to disable the proactive repair feature, this applies to **all nodes in the service region of the specified host**. So, it doesn't matter which host you specify in the command as long as it's in the right service region. Likewise *hsstool -h <host> proactiverepairq -enable true* re-enables the proactive repair feature for all nodes in the service region.

Disabling the proactive repair feature **does not abort in-progress proactive repairs**. Rather, it prevents any additional proactive repairs from launching. To stop in-progress proactive repairs on a particular node use the *-stop* option.

**IMPORTANT !** The proactive repair feature is important for maintaining data integrity in your system. Do not leave it disabled permanently.

**Note** The proactive repair feature can also be disabled and re-enabled by using *hsstool repairqueue -h <host> -enable true|false*. The difference is that the **"hsstool repairqueue"** (page 563) approach disables and re-enables scheduled auto-repairs and also proactive repairs, whereas the *hsstool proactiverepairq* approach disables and re-enables only proactive repair.

If you use both types of commands, then whichever command you used most recently will be operative in regard to the proactive repair feature. For example if you run *hsstool repairqueue -h <host> -enable false* and then subsequently you run *hsstool -h <host> proactiverepairq -enable true*, then proactive repair will be enabled in the cluster (while the scheduled auto-repair feature will remain disabled).

### 10.1.8.2. Usage Notes

This **hsstool** command returns information about nodes that are in need of automated proactive repair. This includes nodes for which automated proactive repair is in progress as well as nodes for which automated proactive repair will begin shortly. You can also use the command to immediately start proactive repair (rather

than waiting for the automatic hourly run); or to stop in-progress proactive repairs; or to temporarily disable the proactive repair feature (and to re-enable it after having disabled it).

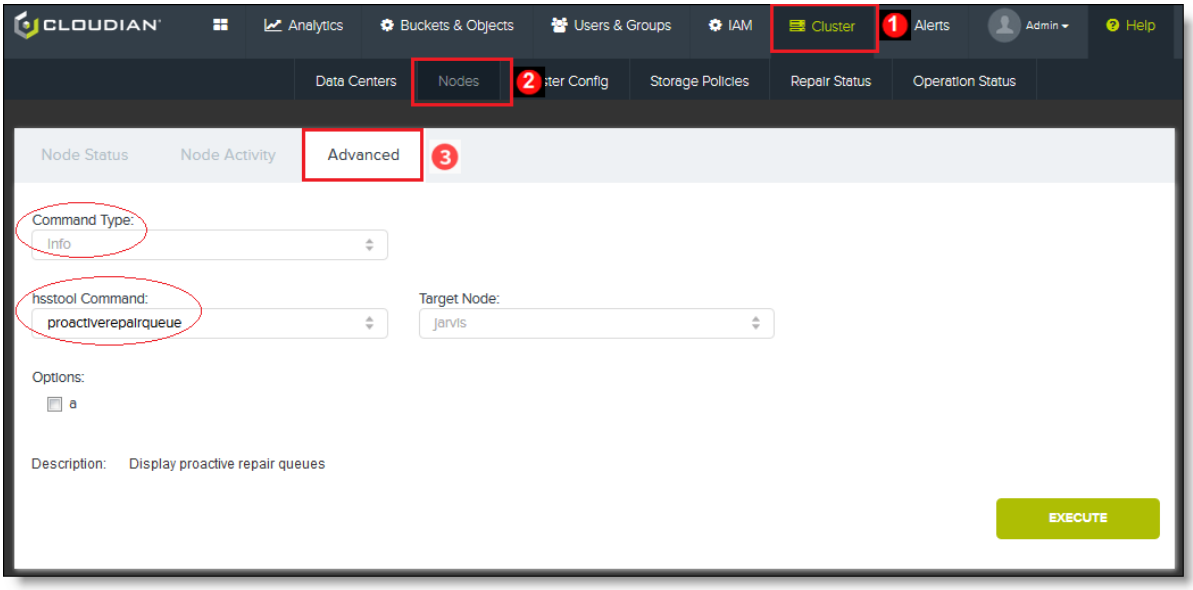
For retrieving proactive repair queue information (*hsstool -h <host> proactiverepairq*) or for disabling or re-enabling the proactive repair feature (*hsstool -h <host> proactiverepairq -enable true|false*), the *<host>* can be any host in your cluster. The command applies to all nodes in the service region of the host that you specify.

For starting or stopping a proactive repair (*hsstool -h <host> proactiverepairq -start* or *hsstool -h <host> proactiverepairq -stop*), the *<host>* is the host on which you want to start or stop a proactive repair.

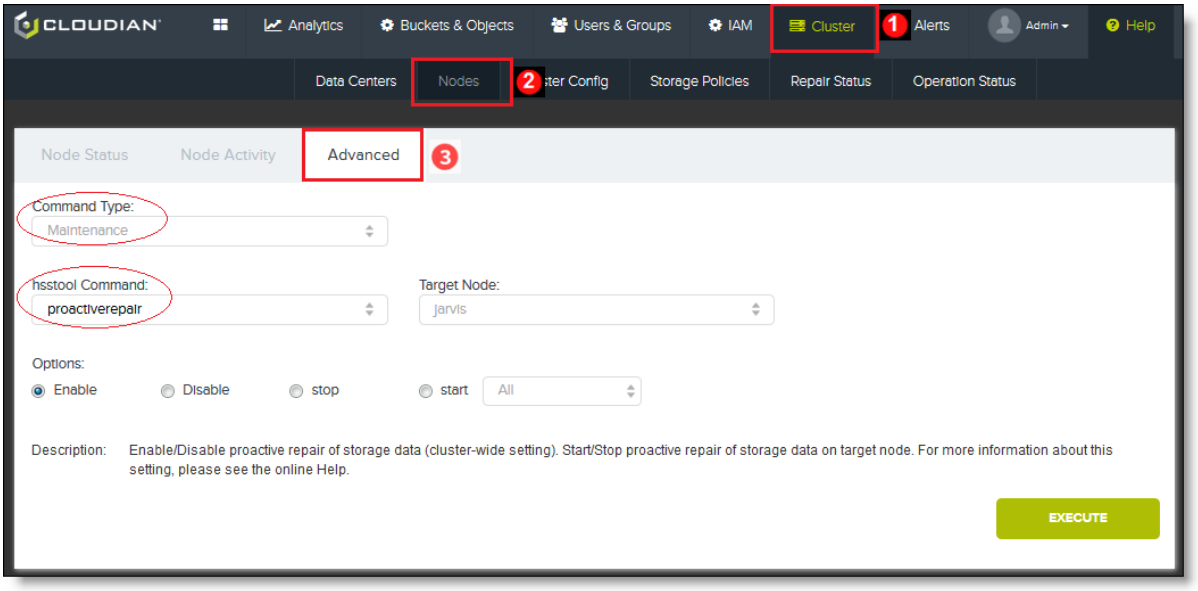
For more information about the HyperStore proactive repair feature see Automatic Data Repair Feature Overview.

## CMC Support for This Command

As an alternative to using the command line, in the CMC UI you can run the *hsstool proactiverepairq* command's proactive repair queue status reporting function through this interface:




The functions for disabling or re-enabling proactive repair, immediately starting a proactive repair, or stopping an in-progress proactive repair have their own separate CMC interface and the command is there renamed as "proactiverepair" (although *hsstool proactiverepairq* is being invoked behind the scenes):




### 10.1.8.3. Command/Response Examples

The *proactiverepairq* command example below shows that one node in the cluster is in need of proactive repair, and that an estimated 100 objects are queued for proactive repair on that node. This proactive repair occurs automatically; no operator action is required. The repair is either already underway or will be triggered at the next interval (default is hourly).

```
# hsstool -h localhost proactiverepairq
Proactive repair: true
Number of nodes to repair: 1
Estimated number of PR events per node:
cloudian7(10.20.2.57) 100
```

The *proactiverepairq -a* command example below shows that one node in the cluster is in need of proactive repair, and that the needed repair involves eight object replicas totaling about 1.6 MBs. This proactive repair occurs automatically; no operator action is required. The repair is either already underway or will be triggered at the next interval (default is hourly).

```
# hsstool -h cloudian-node1 proactiverepairq -a
Proactive repair: true
Number of nodes to repair: 1
cld01(10.20.2.123):  REPLICAS[count = 8 bytes = 1602141] EC[count = 0 bytes = 0] REBALANCE[count =
0 bytes = 0]
```

**Note** The *proactiverepairq -a* option provides more precisely accurate information about the number of queued objects than does the *proactiverepairq* command without the *-a* option -- but using the *-a* option is resource intensive.

#### Response Items

*Proactive repair*

This indicates whether the proactive repair feature is enabled, *true* or *false*. By default proactive repair is enabled.

*Number of nodes to repair: <#>*

Number of nodes that are in need of proactive repair.

*Estimated number of PR events per node*

This is an **estimate** of the number of objects queued for proactive repair, on each node that currently has objects in its proactive repair queue. This gives you a general idea of the number of queued objects but it is not an exact count.

An exact count is available if you use the *proactiverepairq* ***-a*** option, but using that option is resource intensive.

*<hostname>(<IPAddress>)*

Hostname and IP address of a node that is in need of proactive repair. The results will show one line for each node that needs proactive repair, with each such line starting with the node's IP address.

*REPLICAS[count = <#> bytes = <#>]*

For the node identified by <IP Address>, the number of object replicas that need to be written to the node by proactive repair, and the total aggregate size of those replicas in bytes.

If the proactive repair is in progress, these numbers indicate how much is left to do.

*EC[count = <#> bytes = <#>]*

For the node identified by <IP Address>, the number of erasure coded object fragments that need to be written to the node by proactive repair, and the total aggregate size of those replicas in bytes.

If the proactive repair is in progress, this number indicates how much is left to do.

*REBALANCE[count = <#> bytes = <#>]*

Starting in HyperStore version 7.2, rebalance retries are no longer included automatically within proactive repair runs, so if this appears in the response the count should be 0.

## 10.1.9. hsstool rebalance

*Subjects covered in this section:*



### 10.1.9.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> rebalance [-cleanupfile] [-retry] [-list] [-l <true|false>]
[-stop] [-resume] [-newrun]
```

#### Options

*-cleanupfile  Remove from old nodes the data successfully copied to new node*

(Optional) When you add a new node to your cluster, the new node takes over some portions of the token space from the existing nodes in the cluster. Based on the new token space allocation, the *rebalance* operation

copies certain object replicas and/or erasure coded fragments to the new node, from the existing nodes. Then, having been copied to the new node, replicas and/or fragments can be deleted from existing nodes on which they no longer belong (as a result of some portions of the token space having been taken over by the new node). This "cleanup" action frees up storage space on those existing nodes.

To have the system delete each replica or fragment from the appropriate existing node as soon as it is successfully copied to the new node, use the *-cleanupfile* option when you run *rebalance*. This option also cleans up the corresponding metadata in Cassandra.

**Note If you do not use the *-cleanupfile* option when you run *rebalance*,** then after the *rebalance* operation completes for the new node -- or after *rebalance* operations complete for all new nodes if you are adding multiple new nodes -- you will need to run **hsstool cleanup** on each of the older nodes, one node at a time, in order to remove data that no longer belongs on those nodes and free up storage space on those nodes.

*-retry Retry failed stream jobs from a previous COMPLETED operation*

(Optional) If an initial run of *hsstool rebalance* finishes with status COMPLETED but indicates that some "stream jobs" failed, you can run *hsstool -h <host> rebalance -retry* to have the system try again to execute those stream jobs.

This **opstatus rebalance** response shows an example where a *rebalance* operation has finished with status COMPLETED but indicates failed stream jobs.

```
hsstool -h cloudian-1 opstatus rebalance
optype: REBALANCE cmdno#: 1 status: COMPLETED
...
...
stream jobs total: 1132
stream jobs completed: 1132
stream jobs failed: 49
...
...
```

When running *hsstool -h <host> rebalance -retry*, you do not need to include any other options in the command (such as the *-cleanupfile* option). Instead, the retry operation will automatically implement the same option(s) that you used in the original rebalance run, if any.

**IMPORTANT ! If an *hsstool rebalance* operation finishes with status FAILED** (rather than status COMPLETED) **do not run *hsstool -h <host> rebalance -retry*.** A FAILED operation status indicates that the operation errored out and exited prematurely. In this scenario, run *hsstool -h <host> rebalance* again from scratch. Do not use the *-retry* option, since this option works only for the limited purpose of retrying failed stream jobs from a previously COMPLETED rebalance operation.

*-list List new nodes and their rebalance status*

(Optional) If you use this option the command returns a list of newly added nodes in the service region and their status in regard to the rebalance operation.

- A node status of "REQUIRED" means that the node has been added to the cluster (through the CMC's **Add Node** operation) but that you have not yet run *hsstool rebalance* on the new node.

- A node status of "DONE" means that rebalance has been successfully completed for the node.
- A node status of "FAILED" means that the rebalance operation failed for one or more token ranges.

**Note** This option is supported only on the command line -- not in the CMC UI.

Sample command and response, where one new node has been added to the cluster and rebalance has been successfully completed on the node:

```
[root@vm151 bin]# ./hsstool -h localhost rebalance -list
vm124(10.50.41.49):DC1:ca-sm3    DONE
```

The response format is *host*(*IPaddress*):*datacenter*:*regionstatus*

*-l <true|false>  Verbose logging*

(Optional, defaults to true) If this option is true, entries for each object rebalanced by this operation will be written to the *cloudian-hyperstore-repair.log* file on the newly added node. Object replicas will be logged with the string "REBR" and object erasure coded fragments will be logged with the string "REBEC".

This option defaults to true, so you only need to specify the *-l* option if you do **not** want rebalanced object logging (in which case you'd specify *-l false*).

**Note** This option is supported only on the command line -- not in the CMC UI.

*-stop  Stop an in-progress rebalance operation*

(Optional) Use *hsstool -h <host> rebalance -stop* to stop an in-progress rebalance on the specified node. The system will keep track of which token ranges have been processed so far, and then when you resume the rebalance operation it will continue with the ranges that have not yet been processed.

You can subsequently use the **"hsstool opstatus"** (page 522) command to confirm that the rebalance has been stopped (status = TERMINATED) and to see how much rebalance progress had been made before the stop. **It may take a while for the operation to become TERMINATED**, since the operation will first complete the processing of whichever token range it was working on when you executed the *-stop* option.

When you are ready to resume a rebalance that you stopped, use the *rebalance -resume* option.

The system allows stopping multiple rebalance operations concurrently, on different nodes. So if you have added multiple nodes to your cluster and are running rebalance operations on the new nodes concurrently, if you wish you can concurrently stop the rebalance operations on each node -- so that no rebalance operations are running in your cluster --and then later, concurrently resume the rebalance operations on each node. (For limitations on rebalance concurrency see **"Usage Notes"** (page 546).)

**Note** During a rebalance operation the system automatically disables the **auto-repair and proactive repair** features, and the system will not automatically re-enable these features until all rebalance operations on all new nodes in the service region have been fully completed. So if you stop a rebalance operation, auto-repair and proactive repair will remain disabled during the time that the rebalance operation is stopped. Auto-repair and proactive repair will remain disabled until the rebalance operation has been resumed and completed (and rebalance operations on any other new nodes in the service region have been completed).

*-resume  Resume a stopped rebalance operation (or retry failed stream jobs)*

(Optional) Use *hsstool -h <host> rebalance -resume* to resume a rebalance operation that you previously

stopped. The rebalance operation will continue with the token ranges that have not yet been processed.

Another use for this option is if an initial run of *hsstool rebalance* finishes with status COMPLETED but indicates that some "stream jobs" failed. You can run *hsstool -h <host> rebalance -resume* to have the system try again to execute those failed stream jobs. (In earlier releases of HyperStore this function was filled by a *-retry* option, but in HyperStore 7.5.1 and later this function is covered by the *-resume* option.)

When running *hsstool -h <host> rebalance -resume*, you do not need to include any other options in the command (such as the *-cleanupfile* option). Instead, the resume operation will automatically implement the same option(s) that you used in the original rebalance run, if any.

**IMPORTANT ! If an *hsstool rebalance* operation finishes with status FAILED** do not run *hsstool -h <host> rebalance -resume*. A FAILED operation status indicates that the operation errored out and exited prematurely. In this scenario, run *hsstool -h <host> rebalance* again from scratch. Do not use the *-resume* option.

*-newrun* *Force a new rebalance operation rather than resuming a prior one*

(Optional) Use *hsstool -h <host> rebalance -newrun* if a prior rebalance run on the same host was terminated -- by use of the *rebalance -stop* option or by some other interruption such as the target node being rebooted -- and for some reason you need to run a new rebalance operation on the node rather than resuming the prior rebalance operation. This might happen if for example the prior rebalance operation is 'stuck' in some way and not able to be resumed. Typically you would use the *-newrun* option only if advised to do so by Cloudian Support.

If you run a fresh rebalance operation on the node in this way, it will not re-stream to the new node any data that had already been streamed there by the prior rebalance operation (there will not be a duplication of data on the new node). Instead the new rebalance will detect that this data is already in place on the new node and will not re-stream it.

**Note** The *-newrun* option is supported only on the command line -- not in the CMC UI.

## 10.1.9.2. Usage Notes

This **hsstool** command copies S3 object data from your existing nodes to a specified new node that you have added to your HyperStore cluster. The *rebalance* operation populates the new node with its share of S3 object replica data and erasure coded data, based on the token ranges that the system automatically assigned the new node when you added it to the cluster.

The target *<host>* must be a newly added node, unless the *-list* option is used in which case the target *<host>* can be any node.

All other nodes in the cluster must be up and running when you run *rebalance* on a new node.

### When to Use *hsstool rebalance*

The only time to use this command is when you have added a new node or nodes to an existing data center. You will then run *rebalance* on the new node(s).

For complete instructions on adding new nodes including the proper use of the *rebalance* command within the context of the procedure, see:

- **"Adding Nodes"** (page 264)

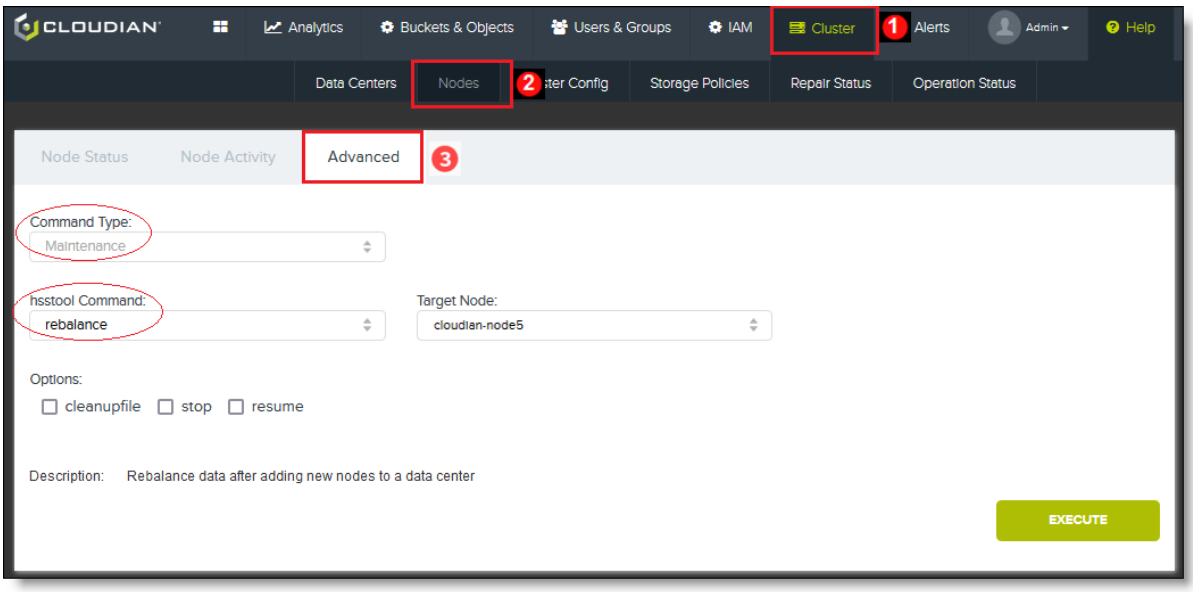
The rebalance is a background operation that may take many hours or days, depending on your HyperStore cluster size and stored data volume.

You can run *rebalance* on multiple new nodes concurrently. However, **do not run *rebalance* concurrently on a number of new nodes that exceeds one-third the number of your already existing nodes** in that same data center. For example, if you have 12 existing nodes in a DC and you are adding 6 new nodes to that DC, do not run *rebalance* on more than 4 of the new nodes concurrently (one-third of 12). In this scenario you would run *rebalance* on 4 of the new nodes concurrently, and then when the *rebalance* operation completes on at least 2 of those nodes, you can start *rebalance* on the other 2 new nodes. In this way *rebalance* is never running on more than 4 nodes at a time (never running on more than one-third the number of the pre-existing nodes in the DC).

In the event that the rebalance operation fails for some objects that are supposed to be copied to the new node, these failures will subsequently be corrected automatically by the Proactive Repair feature.

### CMC Support for This Command

As an alternative to running *hsstool rebalance* on the command line, you can run it through the CMC UI:




**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's **Operation Status** page (**Cluster -> Operation Status**). This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the **hsstool opstatus** command.

## 10.1.9.3. Checking Operation Status

The *rebalance* operation is a long-running operation. Its duration will depend on multiple factors including the amount of data in your system and the number of nodes in your system. For information about checking the status of an in-progress or recently completed *rebalance* operation see **"hsstool opstatus"** (page 522).

## 10.1.10. hsstool repair

*Subjects covered in this section:*



### 10.1.10.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> repair [allkeyspaces|nokeyspaces] [-computedigest] [-pr]
[-m <true|false>] [-d <mount-point> -rebuild] [-range <start-token,end-token>]
[-t <min-timestamp,max-timestamp>] [-l <true|false>] [-stop] [-resume]
```

#### Options

*allkeyspaces | nokeyspaces* *Set scope of metadata repair*

(Optional) You have three alternatives for choosing which Cassandra metadata keyspaces to repair, while also cleaning replicated S3 object data in the HyperStore File System (HSFS):

- Use *allkeyspaces* to repair replicated objects in the HSFS and also repair all the Cassandra keyspaces. Cassandra repair will be completed first, then HSFS replica repair. The Cassandra keyspaces that will be repaired are: UserData_<storage-policy-ID> keyspaces; AccountInfo; Reports; Monitoring; and ECKeyspace. (For more information see the overview of **Cassandra keyspaces for HyperStore**).
- Use *nokeyspaces* to repair only replicated objects in the HyperStore File System, and not any Cassandra keyspaces.
- If you specify neither *allkeyspaces* nor *nokeyspaces* then the default behavior is to repair replicated objects in the HSFS and also to repair the Cassandra *UserData_<storage-policy-ID>* keyspaces (which store object metadata). Cassandra repair will be completed first, then HSFS replica repair.

*-computedigest* *Repair corrupted replicas as well as missing replicas*

(Optional) Use this option if you want to check for and repair not only missing replicas but also any replicas that are present but corrupted. When doing the repair with the *-computedigest* option, the system will compute a fresh digest for each replica rather than using cached digests. If the re-computed digest of a given replica does not match the original digest (stored in a file alongside the object data) or does not match the re-computed digests of the other replicas of that same object (on other nodes), the replica is considered to be corrupted and is replaced by a copy of a correct replica streamed in from a different node.

**This way of running *repair* is resource-intensive.**

> **Note** If you wish, you can have some or all of the **scheduled auto-repairs** of replica data use the "-computedigest" option to combat bit rot. This aspect of auto-repair is controlled by the **"auto_repair_computedigest_run_number"** (page 395) setting in *common.csv*. By default "-computedigest" is not used in auto-repair runs.

*-pr* *Repair only the node's primary ranges*

(Optional) Only repair objects that fall within the target node's primary ranges (objects for which the hash of the

object name falls into one of the token ranges assigned to the node). Do not repair objects that fall into other nodes' primary ranges and that are replicated on to the target node. For background information on replication in HyperStore, see **"How vNodes Work"** (page 80).

If each node in the cluster is being repaired in succession, using this option makes the successive repair operations less redundant and more efficient.

> **Note** The HyperStore auto-repair feature -- which automatically runs node repairs on a schedule -- uses the *-pr* option when it initiates the *hsstool repair* operation. For more on this feature see Automated Data Repair Feature Overview.

*-m <true|false>  Repair only objects for which metadata exists*

(Optional, defaults to true) If this option is true, then before repairing a given object the repair process will verify that metadata for the object exists in Cassandra. This prevents attempts to repair objects that had been intentionally deleted by users, as indicated by the absence of corresponding object metadata in Cassandra. Defaults to true, so you only need to specify an *-m* option if you do **not** want the object metadata check to be performed (in which case you'd specify *-m false*).

*-d <mountpoint> -rebuild  Repair only a specified mountpoint*

(Optional) If you use this option the repair is performed only for objects mapped to the vNodes that are assigned to the specified HyperStore data mount point (for example *hsstool repair -d /cloudian1 -rebuild*), either as primary replicas or as secondary or tertiary replicas. This option may be useful in circumstances where data is known or suspected to be missing or incorrect on a particular disk.

**You cannot repair the replica data on multiple disks concurrently.** The repair of one disk must complete before you can start the repair of a different disk, even if the different disk is on a different node.

> **Note** If you use the *-d <mount-point> -rebuild* option do not use the *-pr* option or the *-range <start-token,end-token>* option. The system does not support using these options in combination.

*-range <start-token,end-token>  Repair only a specified token range*

(Optional) If you use this option the repair is performed only for objects mapped to your specified token range. You specify the range by indicating the start token and end token (an example of a token is 18315119863185730105557340630830311535). The repair operation will repair all objects that fall within the token range bounded by the start and end tokens.

The **end-token must be a token that is assigned to the target host**. To see what tokens are assigned to each node you can use the **"hsstool ring"** (page 567) command. The **start-token must be the next-lower token in the ring** from the end-token.

This option may be useful if a previous full node repair failed for particular ranges. You can obtain Information about range repair failures (including the start and end tokens that bound any failed ranges) by running *hsstool -h <host> opstatus repair -a* on the command line.

> **Note** If you use the *-range <start-token,end-token>* option do not use the *-pr* option or the *-d <mount-point> -rebuild* option. The system does not support using these options in combination.

*-t <min-timestamp,max-timestamp>  Repair only objects last modified within a specified time period*

(Optional) If you use this option the repair is performed only for objects that have a last-modified timestamp

equal to or greater than *min-timestamp* and less than *max-timestamp*. When using this option, use Unix milliseconds as the timestamp format.

**Note** This option is not supported in the CMC interface -- only on the command line.

*-l <true|false>  Log a list of repaired objects*

(Optional, defaults to true) If this option is true, write to a log file a list of all the objects that were repaired. Defaults to true, so you only need to specify an *-l* option if you do **not** want repair object logging (in which case you'd specify *-l false*).

The log is named *cloudian-hyperstore-repair.log* and is written into the Cloudian HyperStore log directory of the target node. Activity associated with a particular instance of a command run is marked with a unique command number.

*-stop  Stop an in-progress repair*

(Optional) Use *hsstool -h <host> repair -stop* to stop an in-progress repair on the specified node.

If **Cassandra repair is in-progress** — that is, if *repair* was launched with the default behavior or the *allkeyspaces* option, and the Cassandra part of the repair is still in-progress — the Cassandra repair is terminated. The HSFS replica repair — which would normally launch after the Cassandra repair — is canceled.

If **HSFS replica repair is in-progress** — that is, if *repair* was launched with the *nokeyspaces* option, or if it was launched with the default behavior or the *allkeyspaces* option and the Cassandra part of the repair has already completed and HSFS replica repair is underway — the HSFS replica repair is terminated.

You can subsequently use the **"hsstool opstatus"** (page 522) command to confirm that the repair has been stopped (status = TERMINATED) and to see how much repair progress had been made before the stop.

If you subsequently want to resume a repair that you stopped, you can use the *repair -resume* option.

**Note** The *-stop* option stops a single in-progress repair on a single node. It does **not** disable the HyperStore **scheduled auto-repair** feature.

*-resume  Resume a terminated repair*

(Optional) Use this option if you want to resume a previous repair operation that did not complete. This will repair token ranges that were not repaired by the previous repair.

You may want to resume an incomplete repair in any of these circumstances:

- The repair was interrupted by the *repair -stop* command.
- The repair was interrupted by a restart of the target node or a restart of the HyperStore Service on the target node
- The repair reported failed token ranges

**Note If during the incomplete repair run you used the *-pr* option or the *-d <mount-point>* option**, you do not need to re-specify the option when running *repair -resume*. The system will automatically detect that one of those options was used in the previous repair and will use the same option when resuming the repair.

**If during the incomplete repair run you used the *-range* option**, then resuming the repair is not

supported. Repair resumption works by starting from whichever token ranges were not repaired by the previous repair. Since a repair that uses the *-range* option targets just one token range, resuming such a repair would not be any different than running that same repair over again. If you want to run the repair over again, do *repair -range <start-token,end-token>* again, not *repair -resume*.

**Note** If you run *repair -resume*, then in the command results the Arguments section will include "resuming-cmd=<n>", where <n> is the number of times that *repair -resume* has been run on the node since the last restart of the HyperStore Service. This Argument string -- which serves to distinguish *repair -resume* result output from regular *repair* result output -- also appears in **"hsstool opstatus"** (page 522) results for *repair -resume* operations.

### 10.1.10.2. Usage Notes

Use this **hsstool** command to check whether a physical node has all of the replicated data that it is supposed to have (based on the node's assigned tokens and on replication settings for the system); and to replace or update any data that is missing or out-of-date. Replacement or update of data is implemented by retrieving correct and current replica data from other nodes in the system.

The system will not allow you to run *hsstool repair*:

- On a node on which **hsstool cleanup** is currently running.
- On any node if there is a **disabled disk** on any node in the same service region.
- On any node if *hsstool repair* is already running on a different node in the same service region. The exception to this rule is if you use the *-pr* option with each repair run (you can run *hsstool repair -pr* on multiple nodes concurrently).

#### When to Use *hsstool repair*

The HyperStore system automatically uses a combination of **read repair, proactive repair, and scheduled auto-repair** to keep the replica data on each node complete and current. Consequently, you should rarely need to manually initiate a full *repair* operation.

However, there are these uncommon circumstances when you should manually initiate *repair* on a specific node:

- If you are removing a "dead" node from your cluster. In this circumstance, after removing the dead node you will run repair on each of the remaining nodes, one node at a time. See **"Removing a Node"** (page 289) for details.
- If a node is unavailable for longer than the configurable **"hyperstore.proactiverepair.queue.max.time"** (page 470) in *mts.properties.erb* (default = 4 hours). In this case then the metadata required for implementing **proactive repair** on the node will stop being written to Cassandra, and an alert will display in the CMC's **Alerts** page (**Alerts -> Alerts**). When the node comes back online you will need to:
    1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until it completes. You can check the CMC's **Repair Status** page (**Cluster -> Repair Status**) periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the objects from during the period when proactive repair

metadata was still being written to the Cassandra for the node.

2. After proactive repair on the node completes, manually initiate a full repair of the node (using *hsstool repair* and, if appropriate for your environment, **hsstool repairec**). This will repair objects that were written after the proactive repair queueing time maximum was reached.

**Note** The *repair* operation will fail if the HyperStore service is down on any of the nodes affected by the operation (the nodes storing the affected token ranges). The operation will also fail if any disk storing affected token ranges is disabled or **more than 95% full**.

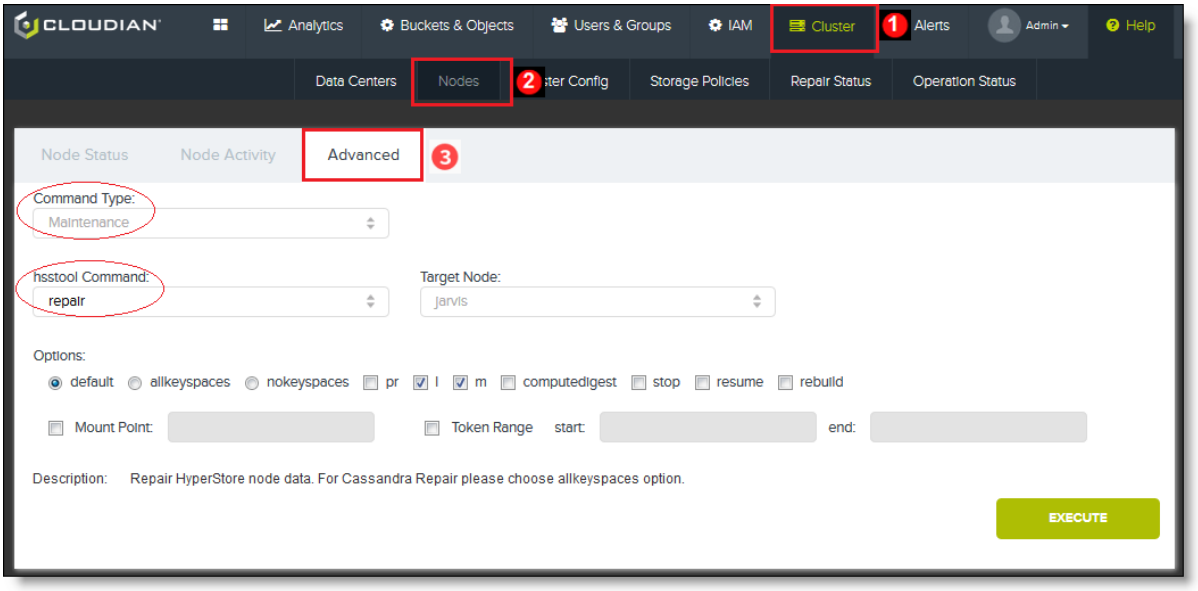
### Problems Remedied by *repair* and *repair computedigest*

The table below lists data problem cases and shows whether or not they are remedied by regular *repair* and by *repair* that uses the *computedigest* option. Although regular *repair* can handle some cases of corruption, if corruption is suspected on a node and you're not certain exactly which data is corrupted, it's best to use *repair computedigest*.

| Case | Will *repair* fix it? | Will *repair computedigest* fix it? |
|---|---|---|
| Missing blob file | yes | yes |
| Missing digest | yes | yes |
| Missing blob file and digest | yes | yes |
| Corrupted blob file | no | yes |
| Corrupted digest | yes | yes |
| Corrupted blob file and digest | yes | yes |

### CMC Support for This Command

As an alternative to running *hsstool repair* on the command line, you can run it through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's **Operation Status** page (**Cluster -> Operation Status**). This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the **hsstool opstatus** command.

#### 10.1.10.3. Checking Operation Status

The *repair* operation is a long-running operation. Its duration will depend on multiple factors including the amount of data in your system and the number of nodes in your system. For information about checking the status of an in-progress or recently completed *repair* operation see **"hsstool opstatus"** (page 522).

### 10.1.11. hsstool repaircassandra

*Subjects covered in this section:*



#### 10.1.11.1. Command Syntax

**Synopsis**

```
# hsstool -h <host> repaircassandra [-keyspace <keyspacename> [<columnfamily> <columnfamily>...]]
[-pr] [-local] [-dc <dcname>] [-stop [enforce]]
```

**Options**

*-keyspace <keyspacename> [<columnfamily> <columnfamily>...]  Set scope of metadata repair*

(Optional) Use the *-keyspace* option if you want to repair only a specific keyspace in Cassandra (rather than repairing all keyspaces, which is the default behavior). You can also optionally specify one or more column families within that keyspace, if you want to repair just that column family or those column families. If you do not specify a column family then all column families in the specified keyspace will be repaired.

You can only specify one *-keyspace* option per *hsstool repaircassandra* run. Specifying multiple keyspaces is not supported. Note again that the default behavior of *hsstool repaircassandra* -- if you omit the *-keyspace* option -- is to repair all of the Cassandra keyspaces.

Example of using the *-keyspace* option together with a target column family name:

```
... -keyspace UserData_a36d2c44fbfcc12222e9587b3a42997f CLOUDIAN_OBJMETADATA
```

**Note** The *-keyspace* option is supported only on the command line, not in the CMC UI.

*-pr  Repair only the node's primary ranges*

(Optional) Only repair data that falls within the target node's primary ranges (data for which the hash of the data row key falls into one of the token ranges assigned to the node). Do not repair data that falls into other nodes'

primary ranges and that is replicated on to the target node.

If each node in the cluster is being repaired in succession, using this option makes the successive repair operations less duplicative and more efficient.

*-local  Repair only the nodes in the local data center*

(Optional) Only repair data on nodes in the same data center as the target node.

If you use neither the *-local* option nor the *-dc <dcname>* option, then Cassandra data in all of your HyperStore system's data centers will be repaired.

> **Note** The *-local* option is supported only on the command line, not in the CMC UI.

*-dc <dcname>  Repair only the nodes in the specified data center*

(Optional) Only repair data on nodes in the specified data center. For example if you specify *-dc boston* then only the Cassandra data on the nodes in the data center named *boston* will be repaired. The *<dcname>* must be a valid data center name from your HyperStore system configuration.

If you use neither the *-local* option nor the *-dc <dcname>* option, then Cassandra data in all of your HyperStore system's data centers will be repaired.

> **Note** The *-dc <dcname>* option is supported only on the command line, not in the CMC UI.

*-stop [-enforce]  Stop an in-progress repair*

(Optional) Use *hsstool -h <host> repaircassandra -stop* to terminate an in-progress Cassandra repair that was initiated via *hsstool* (whether by you or automatically by the system, such as with auto-repair).

If you need to terminate an in-progress Cassandra repair that was initiated via the native Cassandra utility *nodetool*, use *hsstool -h <host> repaircassandra -stop -enforce*. Note that using *nodetool* to initiate a Cassandra repair is not recommended.

> **Note** In the CMC UI the *-stop -enforce* option is called "force stop".

The *hsstool repaircassandra* command does not support a 'resume' option. If you stop an in-progress Cassandra repair, to do the repair again use the *hsstool repaircassandra* command again and the repair will start over.

> **Note** The *-stop* option stops a single in-progress Cassandra repair on a single node. It does **not** disable the HyperStore **scheduled auto-repair** feature.

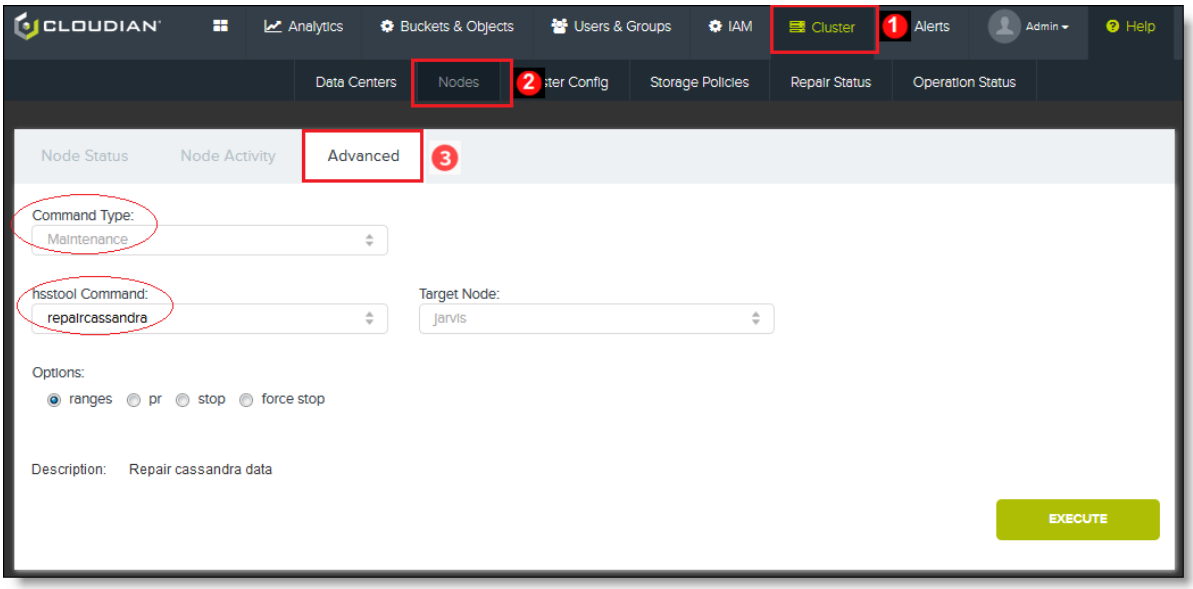
### 10.1.11.2. Usage Notes

Use this **hsstool** command to repair only the Cassandra data on a node (the system metadata and object metadata stored in Cassandra) and **not** the object data on the node. Under normal circumstances you should not need to use this command, but you might use it when in a troubleshooting or recovery situation.

**IMPORTANT !** If you do need to initiate a Cassandra-only repair (with no repair of the object data in the HyperStore File System), use this command rather than using the native Cassandra utility *nodetool* to initiate the repair. Using *hsstool repaircassandra* has multiple advantages over using *nodetool repair*, including that with *hsstool repaircassandra* you can track the repair's progress with **hsstool opstatus** and you can stop the repair if you need to for some reason.

### CMC Support for This Command

As an alternative to running *hsstool repaircassandra* on the command line, you can run it through the CMC UI:




**Note** As is shown in the CMC interface for *hsstool repaircassandra* and in the status response when you run the command, the command applies a "**ranges**=true" argument by default (the status response includes "ranges=true" in the list of arguments). With this method of Cassandra repair, each impacted token range is repaired one range at a time, sequentially. This approach improves the performance for Cassandra repair. Prior to HyperStore release 7.1.5 using this method was optional, but starting with release 7.1.5 it became the default repair behavior.

### 10.1.11.3. Checking Operation Status

The *repaircassandra* operation is a long-running operation. Its duration will depend on multiple factors including the amount of metadata in your system and the number of nodes in your system. For information about checking the status of an in-progress or recently completed *repaircassandra* operation see **"hsstool opstatus"** (page 522).

## 10.1.12. hsstool repairec

*Subjects covered in this section:*

- ***"Checking Operation Status"*** *(page 562)*

### 10.1.12.1. Command Syntax

#### Synopsis

```
# hsstool -h <host> repairec [-computedigest] [[-reb] -d <mountpoint>] [-range <start-token,end-token>]
[-f <input-file> [-opid <operationId>]] [-l <true|false>] [-stop] [-resume]
```

#### Options

*-computedigest  Repair corrupted fragments as well as missing fragments*

(Optional) Use this option if you want to check for and repair not only missing erasure coded fragments but also any fragments that are present but corrupted. When doing the repair with the *-computedigest* option, the system computes a fresh digest for each fragment rather than using cached digests. The re-computed digest of each fragment is compared to the original digest of those fragment (stored alongside the fragment data) and if there is a mismatch the fragment is considered to be corrupted. The object is then decoded from healthy fragments and then re-encoded, and the corrupted fragment is replaced by a new and correct fragment.

**This way of running *repairec* is resource-intensive.**

> **Note** If you wish, you can have some or all of the **scheduled auto-repairs** of erasure coded data use the "-computedigest" option to combat bit rot. This aspect of auto-repair is controlled by the **"auto_repair_computedigest_run_number"** (page 395) setting in *common.csv*. By default "-computedigest" is not used in auto-repair runs.

*[-reb] -d <mountpoint>  Repair only a specified mountpoint*

(Optional) If you use this option the repair is performed only for object fragments mapped to the token ranges that are assigned to the specified HyperStore data mount point on the target node -- for example *hsstool -h localhost repairec -d /cloudian1* . This option may be useful in circumstances where data is known or suspected to be missing or incorrect on a particular disk.

If you are performing the disk repair after an "Add Node" operation has successfully completed and before rebalance has completed for all the new nodes , use the *-reb* option together with *-d <mountpoint>* -- -- for example *hsstool -h localhost repairec -reb -d /cloudian1*. In all other circumstances, omit the *-reb* option.

You can repair disks on multiple nodes concurrently**, if there is only erasure coded data in your system.** Repairing data on multiple disks concurrently is **not** supported in either of these circumstances:

- You have replication storage policies in your system (storage policies that use object replication rather than erasure coding). Note that in a multi-DC environment, "Replicated EC" is considered an erasure coding storage policy not a replication storage policy.
- You are using the new "hybrid" storage policies introduced in HyperStore 7.5.1 (which use replication for small objects and erasure coding for large objects).

Also, you cannot repair multiple disks on the same node concurrently. You can only repair multiple disks concurrently if the disks are on different nodes.

> **Note**
> * If you use the *-d <mountpoint>* option do not use the *-range <start-token,end-token>* option or the *-f*

*<input-file>*. The system does not support combining mountpoint repair with those other repair options.
* The HyperStore *replaceDisk* function (see **"Replacing a HyperStore Data Disk"** (page 332)) automatically invokes the *hsstool repairec -d <mountpoint>* command, after automatically performing other tasks required for bringing a replacement disk back into service. If you use the *replaceDisk* function after adding nodes to your cluster and before rebalancing has been completed, the function automatically invokes the *hsstool repairec -reb -d <mountpoint>* command.

*-range <start-token,end-token>  Repair only a specified token range*

(Optional) If you use this option the repair is performed only for objects mapped to your specified token range. You specify the range by indicating the start token and end token (an example of a token is 183151198631857301055573406308303115 35). The repair operation will repair all objects that fall within the token range bounded by the start and end tokens.

The **end-token must be a token that is assigned to the target host**. To see what tokens are assigned to each node you can use the **"hsstool ring"** (page 567) command. The **start-token must be the next-lower token in the ring** from the end-token.

This option may be useful if a previous full node repair reported failures for a particular token range.

*-f <input-file> [-opid <operationId>]  Repair only the objects listed in an input file*

(Optional) See **"Repairing Objects Specified in a File"** (page 560).

**Note** The CMC does not support the *-f <input-file> [-opid <operationId>]* option. This option is only supported if you run *hsstool repairec* on the command line.

*-l <true|false>  Log a list of repaired objects*

(Optional, defaults to true) If this option is true, write to a log file a list of all the objects that were repaired. Defaults to true, so you only need to specify an *-l* option if you do **not** want repair object logging (in which case you'd specify *-l false*).

The log is named *cloudian-hyperstore-repair.log* and is written into the Cloudian HyperStore log directory of the target node.

Failures to repair particular erasure coded object chunks are logged in a separate log file, named *cloudian-hyperstore-repair-failure.log*.

For more information about these logs see **"HyperStore Service Logs"** (page 354).

*-stop  Stop an in-progress repair*

(Optional) Use *hsstool -h <host> repairec -stop* to stop an in-progress erasure coded data repair. This stops the repair immediately. You can subsequently use the **"hsstool opstatus"** (page 522) command to confirm that the repair has been stopped (status = TERMINATED) and to see how much repair progress had been made before the stop.

Note that:

- If you want to stop a repair of the erasure coded data on a particular mountpoint or a repair of a specified token range, you can stop the repair operation with simply *hsstool -h <host> repairec -stop*. You do not need to specify the mountpoint or token range along with the *-stop* option. Resumption of these

operations is supported as per the description of the *-resume* option below.

- If you want to stop a repair of the objects listed in an input file, you can stop the repair operation with simply *hsstool -h <host> repairec -stop* -- but resumption of this type of repair is not supported.
- The *-stop* option stops a single in-progress repair. It does not disable the HyperStore **scheduled auto-repair** feature.

*-resume <options> Resume a stopped or interrupted repair*

(Optional) Use *hsstool -h <host> repairec -resume <options>* to resume the most recent *repairec* run, if you had stopped that run (with the *-stop* command option) or if the run was interrupted (by the target host going down during the repair, for instance).The repair will resume from the point of progress at which it was when the repair run was stopped or interrupted.

Note that:

- If you want to resume a repair of a particular mountpoint or a repair of a specified token range, you must specify the same mountpoint (or token range) options along with the *-resume* option. For example, *hsstool -h <host> repairec -d /cloudian1 -resume*

  **Note** This is different than the behavior of the *-resume* option for the **"hsstool repair"** (page 548) and **"hsstool rebalance"** (page 543) operations. With those operations you do not need to specify the options from the original operation run when resuming an operation.

- Resuming a repair of the objects listed in an input file is not supported.

### 10.1.12.2. Usage Notes

Use this **hsstool** command to evaluate and repair erasure coded object data. When you run *hsstool repairec* on a target node, the scope of the repair depends on the **storage policy or policies** that you are using in your system:

- For an **erasure coding storage policy confined to a single data center**, the *hsstool repairec* operation repairs all erasure coded data on all nodes in the data center in which the target node resides. So, to repair all the data associated with this policy you only need to run *hsstool repairec* on any one node in the data center.
- For a **distributed erasure coding storage policy spanning multiple data centers**, the *hsstool repairec* operation repairs all erasure coded data on all nodes in all of the data centers included in the storage policy. To repair all the data associated with this type of storage policy you only need to run *hsstool repairec* on one node in any one of the data centers included in the storage policy.
- For a **replicated erasure coding storage policy spanning multiple data centers**, the *hsstool repairec* operation repairs all erasure coded data on all nodes in the data center in which the target node resides. To repair all the data associated with this type of storage policy you must run *hsstool repairec* on one node in each of the data centers included in the storage policy.

The repair process entails replacing erasure coded object fragments that are missing, outdated, or corrupted. Replacement of a missing or bad object fragment is implemented by using the object's good fragments to decode the object, re-encoding the object, and then re-writing the missing or bad fragment to the correct end-point node. To repair an erasure coded object in this manner, there must be at least "k" good fragments present for the object within the system.

The system will not allow you to run a full *hsstool repairec*:

- On any node if a full *hsstool repairec* is already running on a different node in the same service region. Note however that you can run a mountpoint-specific repair (with the *-d <mountpoint>* option) on multiple hosts concurrently, if multiple hosts have disks that are in need of repair.
- On any node if there is a **disabled disk** on any node in the same service region.
- On a node on which **hsstool cleanupec** is currently running.

If the HyperStore Service or Cassandra Service is down on a node in the same data center as the target node, the system does allow you to run *hsstool repairec* but the repair will skip all objects that have a fragment on the node on which the HyperStore Service or Cassandra Service is down. Those objects will not be added to the proactive repair queue -- instead, those objects will remain un-repaired until you subsequently run *hsstool repairec* again with the HyperStore Service and Cassandra Service both up on that node.

Note that:

- In a large cluster with high data volume *hsstool repairec* is a long-running operation that may take multiple weeks to complete.
- HyperStore 7.5.1 and later supports Hybrid Storage Policies whereby objects larger than a **configurable size threshold** are erasure coded and objects at or smaller than that threshold are replicated. The *hsstool repairec* operation will evaluate and repair the erasure coded objects associated with such storage policies, but not the replicated objects. To evaluate and repair the replicated objects use **"hsstool repair"** (page 548).

## When to Use *hsstool repairec*

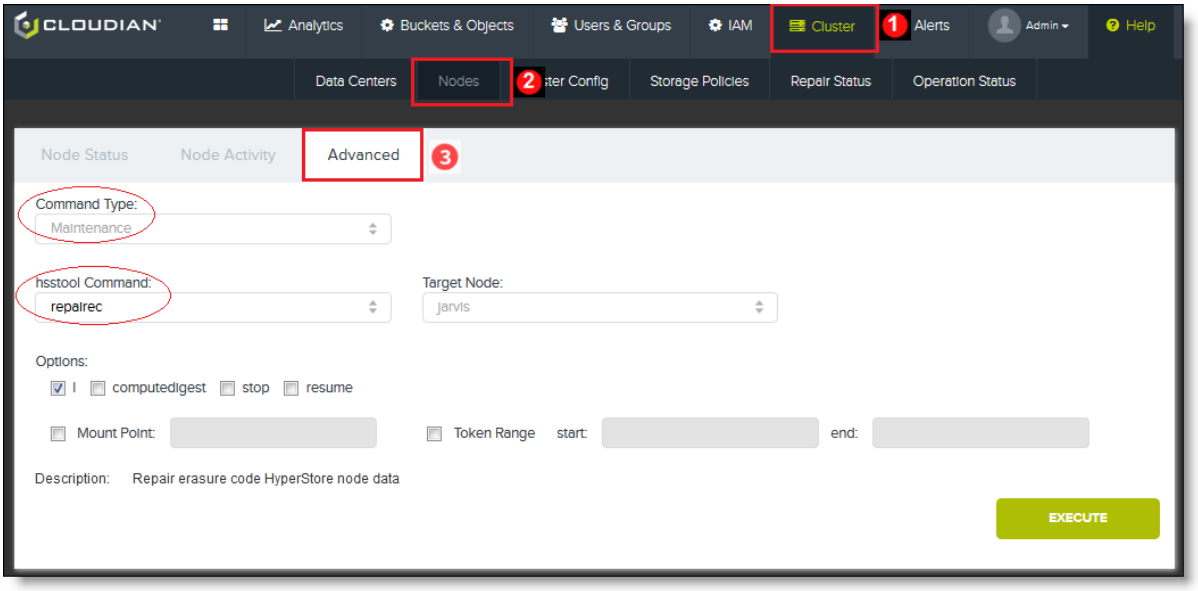
The HyperStore system automatically uses a combination of **read repair, proactive repair, and scheduled auto-repair** to keep the erasure coded data on each node complete and current. Consequently, you should rarely need to manually initiate a full *repairec* operation.

However, if you use erasure coding in your system, there are these uncommon circumstances when you should manually initiate a *repairec*operation:

- If you are removing a "dead" node from your cluster. In this circumstance, after removing the dead node you will run *repairec* on one node in each of your data centers. See **"Removing a Node"** (page 289) for details.
- If a node is unavailable for longer than the configurable **"hyperstore.proactiverepair.queue.max.time"** (page 470) in *mts.properties.erb* (default = 4 hours). In this case then the metadata required for implementing **proactive repair** on the node will stop being written to Cassandra, and an alert will display in the CMC's **Alerts** page (**Alerts -> Alerts**). When the node comes back online you will need to:
    1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until it completes.You can check the CMC's **Repair Status** page (**Cluster -> Repair Status**) periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the objects from during the period when proactive repair metadata was still being written to the Cassandra for the node.
    2. After proactive repair on the node completes, manually initiate a full repair of the node (using *hsstool repairec* and **hsstool repair**). This will repair objects that were written after the proactive repair queueing time maximum was reached.

## CMC Support for This Command

You can also run the *hsstool repairec* command through the CMC UI:




**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's **Operation Status** page (**Cluster -> Operation Status**). This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the **hsstool opstatus** command.

## Repairing Objects Specified in a File

When run on the command line, the *hsstool repairec* command supports an option for repairing one or more specific objects that are listed in an input file:

```
# hsstool -h <host> repairec -f <input-file> [-opid <operationId>]
```

For *<input-file>*, specify the full absolute path to the input file (including the file name). Both the input file and the directory in which it is located must be readable by the 'cloudian' user or else the repair command will immediately fail and report an error.

The target *<host>* can be any node that is utilized by the erasure coding storage policies used by the objects that are listed in the input file.

You have two options for obtaining or creating the input file that lists the objects to repair:

- Use the dedicated repair failure log file that the system generates automatically when *hsstool repairec* is run and repair fails for some objects.
- Create the input file yourself, to target a specific object or small set of objects.

These methods are described in the sections below.

### Using the Repair Failure Log as the Input File

Each time *hsstool repairec* is run, if repair fails for any object -- or more precisely, if repair fails for any object "chunk" (see **"Creating an Input File"** (page 561) for background information about chunks) -- the failures are written to a dedicated repair failure log, on the node(s) on which the failures occur. In this log file there is an entry for each chunk that fails to be repaired. If an *hsstool repairec* run results in chunk repair failures on mul-

tiple nodes, then a repair failure log will be written on each node on which failures occurred, and on each such node the repair failure log entries will identify the chunks for which repair failed on that node.

On each node, the location and name of the current repair failure log is:

```
/var/log/cloudian/cloudian-hyperstore-repair-failure.log
```

The format of entries to the repair failure log is:

```
ChunkName|Path|OperationId|yyyy-mm-dd HH:mm:ss,SSS|FailReason[|TaskType]
```

> **Note** For more information about this log, including sample entries and log rotation policy, see **"HyperStore Service Logs"** (page 354).

Within the log entries, the OperationId field indicates a system-generated unique ID for the *hsstool repairec* run that resulted in the chunk repair failure.

You can use the repair failure log file as input to the *hsstool repairec -f <input-file>* command, and if you wish you can use the *-opid <operationId>* option to limit the repair to the chunk repair failures that resulted from a particular *hsstool repairec* run. If you do not use the *-opid <operationId>* option, then the repair will be performed for all chunks listed in the log file.

Here is an example of the command syntax, including use of the *-opid* option:

```
hsstool -h 10.50.200.161 repairec -f /var/log/cloudian/cloudian-hyperstore-repair-failure.log -opid
a2e2b92a-78a0-1206-983d-000c29b774a0
```

Note that:

- Since *repairec* is a long-running operation, and the *cloudian-hyperstore-repair-failure.log* is rotated at the end of each day (or when the log file reaches 10MB in size, whichever occurs first), the log entries associated with a particular *repairec* operation may be spread across multiple files -- for example, the entries may be in the current, live *cloudian-hyperstore-repair-failure.log* file and also in one or more rotated log files. You can *grep* for the operation ID of interest to see if entries associated with the operation appear in multiple log files. If so, you will need to run *hsstool repairec -f <input-file> [-opid <operationId>]* separately for each file. Wait for the *repairec* run to complete for one file before moving on to run it again for the next file (do not run multiple *repairec* operations concurrently).
- Rotated *cloudian-hyperstore-repair-failure.log* files are Gzipped. You must unzip them before using them as input to an *hsstool repairec -f <input-file> [-opid <operationId>]* run.
- The *-opid <operationId>* option is supported only if the repair failure log is the input file. The *-opid <operationId>*option is not supported if you use an input file that you've created (as described below).

### Creating an Input File

In the HyperStore File System on each of your nodes, objects are stored as "chunks". Objects smaller than or equal to the chunk size threshold (10MB) are stored as a single chunk. Objects larger than the chunk size threshold are broken into and stored as multiple chunks, with no chunk exceeding the threshold size. In the case of large objects that S3 client applications upload to HyperStore by the Multipart Upload method, HyperStore breaks the individual parts into chunks if the parts exceed the chunk size threshold.

In the context of erasure coding storage policies, after objects are broken into chunks, each object chunk is erasure coded.

The *hsstool repairec -f <input-file>* operation acts on individual chunks, and so in the input file each line must specify a chunk name and the full path to the chunk file, in the following format:

```
ChunkName|Path
```

There is no limit on the number of lines that you can include in the file (no limit on the number of chunks that you can specify in the file).

Here is an example in which the object is smaller than the chunk size threshold and so the object is stored as just one chunk. In this case the chunk name is simply in the form of *<bucketname>/<objectname>*. (For background information about chunk **file paths** within the HyperStore File System see **"HyperStore Service and the HSFS"** (page 63).)

```
bucket1/HyperFileInstallGuide_v-3.6.pdf|/cloudian1/ec/std8ZdRJDskcPvmOg4/
0bb5332b429ccb76466e05bee2915d34/074/156/907217638635412080725392490999110784 58.
155413078661679516 3-0A3232C9
```

> **Note** Although the example above and those that follow below break to multiple lines in this documentation, in the actual input file each *ChunkName*|*Path* combination constitutes just one line in the file.

Here is a second example, for an object that was uploaded by a regular S3 PUT operation (not a Multipart Upload) but which is larger than the chunk size threshold and so has been broken into multiple chunks. The example shows an input file entry for one of those chunks. Note that the chunk name here includes a chunk number suffix (shown in bold).

```
bucket1/HyperFileAdminGuide_v-3.6.pdf..0001|/cloudian1/ec/std8ZdRJDskcPvmOg4/
0bb5332b429ccb76466e05bee2915d34/087/073/130803952224141276815735834848732625 19.
155412466201967052 9-0A3232C9
```

In this third example, the object has been uploaded via an S3 Multipart Upload operation. In this example that specifies one of the object's chunks, the chunk name includes a prefix based on the upload ID, as well as a chunk number suffix and part number suffix (all shown in bold).

```
m-MDA1NTE5NjExNTU0MTIzODU3Mjg1/bucket1/cloudian-hyperfile_v-3.6.tar.gz..0001.2|
/cloudian1/ec/std8ZdRJDskcPvmOg4/0bb5332b429ccb76466e05bee2915d34/082/100/
39535768889303436494640495599026926454.1554123857285865726-0A3232C9
```

**You have two options for obtaining the chunk name and chunk path** for chunks that you want to target for repair. One option is to use the **hsstool whereis** command for each object that you want to repair. The *whereis* response includes the chunk name(s) and chunk path(s) for the object that you specify. You can copy the chunk name and path from the *whereis* response into your input file.

> **Note** The *whereis* response has information for each erasure coded fragment, on each node on which the fragments reside. For a given object chunk, each erasure coded fragment has the same chunk name and chunk path, so you can get this information from any of the fragments, regardless of which node a particular fragment is stored on.

The other option is to copy one or more entries from the log file *cloudian-hyperstore-repair-failure.log* (described in **"Using the Repair Failure Log as the Input File"** (page 560)) into a separate file, and use that separate file as the input file. You could do this if you wanted to target a particular object's chunks for another repair attempt, rather than targeting all failures from a preceding repair run.

### 10.1.12.3. Checking Operation Status

The *repairec* operation is a long-running operation. Its duration will depend on multiple factors including the amount of data in your system and the number of nodes in your system. For information about checking the

status of an in-progress or recently completed *repairec* operation see **"hsstool opstatus"** (page 522).

## 10.1.13. hsstool repairqueue

*Subjects covered in this section:*

- ***"Command Syntax"*** *(page 563)*
- ***"Usage Notes"*** *(page 564)*
- ***"Command/Response Example "*** *(page 566)*

### 10.1.13.1. Command Syntax

#### Synopsis

```
# hsstool [-h <host>] repairqueue [-enable true|false] [-t replicas|ec|cassandra] [-inc]
```

#### Options

*-enable true|false  Enable or disable auto-repair*

(Optional) Enable or disable the auto-repair feature. By default the feature is enabled.

If you use *hsstool -h <host> repairqueue -enable false* to disable the auto-repair feature, this applies to **all nodes in the service region of the specified host**. So, it doesn't matter which host you specify in the command as long as it's in the right service region. Likewise *hsstool -h <host> repairqueue -enable true* re-enables the auto-repair feature for all nodes in the service region.

If you do not use the optional *-t* flag (described below) to specify an auto-repair type, then the disabling or re-enabling applies to **all auto-repair types and also to the proactive repair feature**. (If you want to disable or re-enable only proactive repair without impacting the scheduled auto-repair feature see **"hsstool proactiverepairq"** (page 538).)

Note that disabling the auto-repair feature **does not abort in-progress auto-repairs**. Rather, it prevents any additional scheduled auto-repairs from launching. (For information about stopping in-progress repairs, see **"hsstool repair"** (page 548) and **"hsstool repairec"** (page 555)).

**IMPORTANT !** The scheduled auto-repair feature is important for maintaining data integrity in your system. Do not leave it disabled permanently.

**Note** In the CMC UI the enable/disable option is presented as part of a **Maintenance -> autorepair** command rather than the **Info -> repairqueue** command.

*-t replicas|ec|cassandra  Apply command only to a specified data type*

(Optional) You can use this option if you want to restrict the *repairqueue* command action to a particular type of auto-repair -- *replicas* or *ec* or *cassandra*:

- In combination with the *-enable true|false* option, you can use the *-t* option to disable or re-enable just a particular type of auto-repair. For example, use *hsstool -h <host> repairqueue -enable false -t ec* to disable auto-repairs of erasure coded object data. In this example auto-repairs would continue to be enabled for replicated object data and for Cassandra metadata.

> **Note** If you do not use the optional *-t* flag to specify an auto-repair type, then the disabling or re-enabling applies to **all auto-repair types and also to the proactive repair feature**. (If you want to disable or re-enable only proactive repair without impacting the scheduled auto-repair feature see **"hsstool proactiverepairq"** (page 538).)

- Without the *-enable true|false* option, you can use the *-t* option to retrieve scheduling information for just a particular type of scheduled auto-repair. For example, use *hsstool -h <host> repairqueue -t replicas* to retrieve scheduling information for auto-repairs of replicated object data. Using *hsstool -h <host> repairqueue* by itself with no *-t* flag will retrieve scheduling information for all auto-repair types.

*-inc* *Restrict '-t cassandra' option to incremental repair*

(Optional) You can use this option in combination with the *-enable true|false -t cassandra* option if you want the enabling or disabling action to apply only to Cassandra incremental auto-repair. For Cassandra, two types of auto-repair are executed on schedule for each node: incremental repair (once a day) and full repair (once a week). For further information see Auto-Repair Schedule Settings.

If you use the *-enable true|false -t cassandra* option without the *-inc* option then your enabling or disabling action applies to both types of Cassandra auto-repair (incremental and full).

> **Note** The *-inc* option is only supported on the command line, not in the CMC.

### 10.1.13.2. Usage Notes

The HyperStore "auto-repair" feature implements a periodic automatic repair of replicated object data, erasure coded object data, and Cassandra metadata on each node in your system. With the *hsstool repairqueue* command you can:

- **Check on the upcoming auto-repair schedule** as well as the status from the most recent auto-repair runs.
- **Temporarily disable auto-repair** for a particular repair type or all types.
- **Re-enable auto-repair**, if it has previously been disabled by the *hsstool repairqueue* command.

For background information on the auto-repair feature, see Automatic Data Repair Feature Overview.

> **Note** You cannot enable or disable auto-repair for just one particular node — the auto-repair feature is either enabled or disabled for the cluster as a whole. Therefore when running the command the target *<host>* can be any node.

#### When to Use *hsstool repairqueue*

With the *hsstool repairqueue* command you can disable (and subsequently re-enable) the HyperStore auto-repair feature. You should disable auto-repair before performing the following cluster operation:

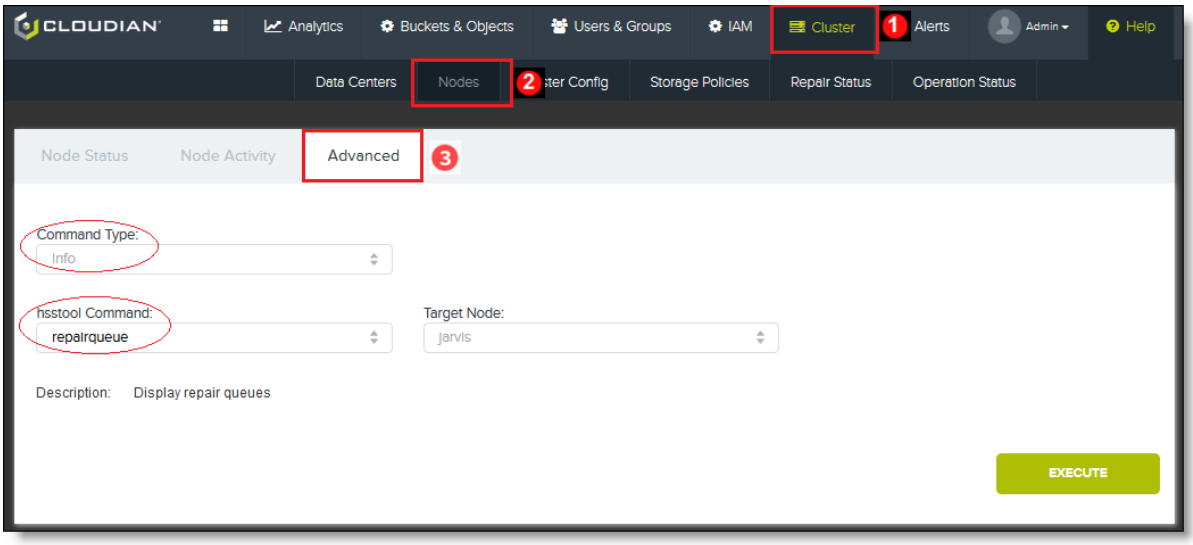
- **"Removing a Node"** (page 289)

> **IMPORTANT !** If you disable auto-repair in order to perform an operation, be sure to re-enable it afterward.

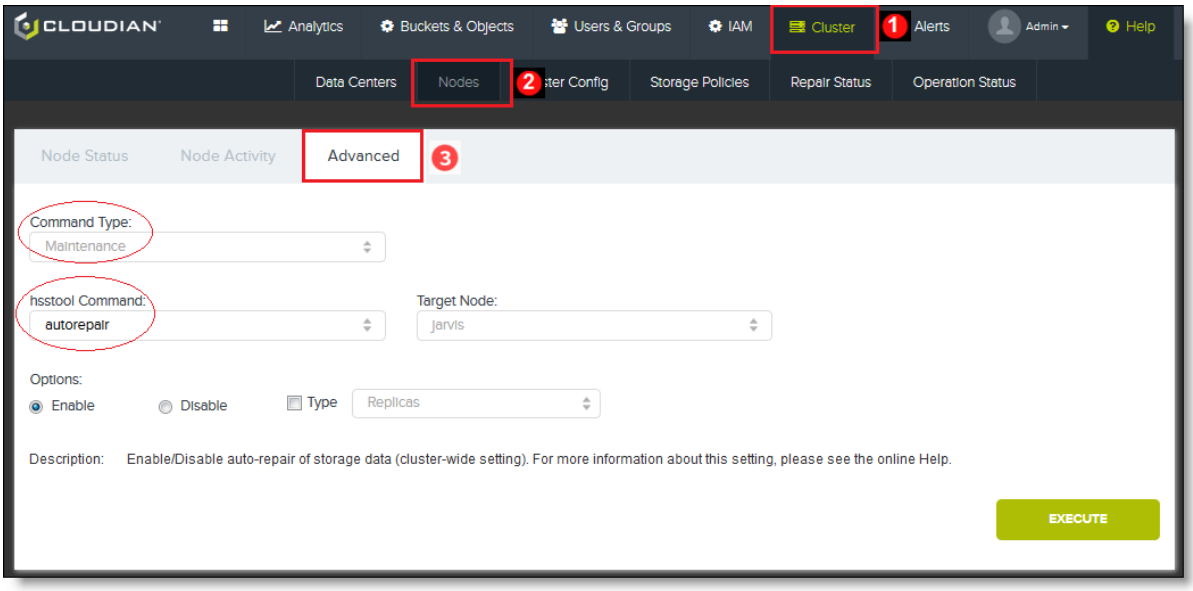
**Note** The system automatically disables the auto-repair feature when you upgrade your HyperStore software version or when you add nodes to your cluster; and the system automatically re-enables auto-repair after these operations are completed. You do not need to use *hsstool repairqueue* when you perform those operations.

### CMC Support for this Command

In the CMC UI you can run the *hsstool repairqueue* command's auto-repair queue status reporting function through this interface:




The function for disabling and re-enabling auto-repair has its own separate CMC interface and the command is there renamed as "autorepair" (although *hsstool repairqueue* is being invoked behind the scenes):




**Note** If you use this command to disable or re-enable auto-repair and you do not specify a repair type, then the disabling or re-enabling applies also to the "proactive repair" feature. (If you want to disable or

re-enable only proactive repair without impacting the scheduled auto-repair feature see **hsstool proactivrepairq**.)

### 10.1.13.3. Command/Response Example

The first *repairqueue* example below shows the "replicas" auto-repair queue status for a recently installed six-node cluster.

```
# hsstool -h cloudian-node1 repairqueue -t replicas

Queue: replicas Auto-repair: enabled #endpoints: 6
1: endpoint: 192.168.204.201 next repair at: Tue Mar 17 22:32:57 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
2: endpoint: 192.168.204.202 next repair at: Tue Mar 17 22:32:58 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
3: endpoint: 192.168.204.203 next repair at: Tue Mar 17 22:32:59 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
4: endpoint: 192.168.204.204 next repair at: Tue Mar 17 22:33:01 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
5: endpoint: 192.168.204.205 next repair at: Tue Mar 17 22:33:02 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
6: endpoint: 192.168.204.206 next repair at: Tue Mar 17 22:33:03 PDT 2015 last repair status:
INIT interval: 43200 mins count: 0
```

**Note** The response attributes for the "ec" and "cassandra" auto-repair queues would be the same as for the "replicas" queue ("next repair at", "last repair status", and so on) -- except that for the "cassandra" repair queue the response also includes a "repairScope" attribute which distinguishes between "INCREMENTAL" (for Cassandra incremental repairs) and "DEFAULT" (for Cassandra full repairs).

The next example command disables "replicas" auto-repair. Note that this disables replicated object data auto-repair for the **whole cluster**. It does not matter which node you submit the command to.

```
# hsstool -h cloudian-node1 repairqueue -enable false -t replicas
Auto replicas repair disabled
```

#### Response Items

When you use the *repairqueue* command to retrieve auto-repair queue information, the command results have three sections — one for each repair type. Each section consists of the following items:

*Queue*

Auto-repair type — either "replicas", "ec", or "cassandra"

*Auto-repair*

Enabled or disabled

*#endpoints*

Number of nodes in the cluster. Each node is separately scheduled for repair, for each repair type.

*<Queue position>*

This is an integer that indicates the position of this node within the cluster-wide queue for auto-repairs of this type. The node at the head of the queue has queue position "1" and is listed first in the command results.

*endpoint*

IP address of a node

*next repair at*

For each repair type, each node's *next repair at* value is determined by adding the configurable auto-repair *interval* for that repair type to the start-time of the last repair of that type done on that node. It's important to note that *next repair at* values are used to **order** the cluster-wide queues for each repair type, but the next repair of that type on that node won't necessarily start at that exact time. This is because the queue processing logic takes into account several other considerations along with the scheduled repair time.

**Note** For erasure coded (EC) object repair, the "next repair at" values are not relevant. Ignore these values. This is because auto-repair for erasure coded objects is run against just one randomly selected target host in each data center each 29-day auto-repair period (and this results in repair of all EC objects in the whole data center).

*last repair status*

This can be INIT (meaning no repair has been performed on that node yet), INPROGRESS, COMPLETED, TERMINATED (meaning that an in-progress repair was aborted by the operator using the "stop" option for **"hsstool repair"** (page 548) or **"hsstool repairec"** (page 555)), or FAILED.

If a node restart interrupts a repair, that repair job is considered FAILED and it goes to the head of the queue.

*interval*

The **configurable interval** at which this type of repair is automatically initiated on each node, in number of minutes. (Note though the qualifiers indicated in the "next repair at"description above).

*count*

The number of repairs of this type that have been executed on this node since HyperStore was installed on the node.

## 10.1.14. hsstool ring

*Subjects covered in this section:*

- ***"Command Syntax"*** *(page 567)*
- ***"Usage Notes"*** *(page 568)*
- ***"Command/Response Example"*** *(page 568)*

### 10.1.14.1. Command Syntax

#### Synopsis

```
# hsstool [-h <host>] ring
```

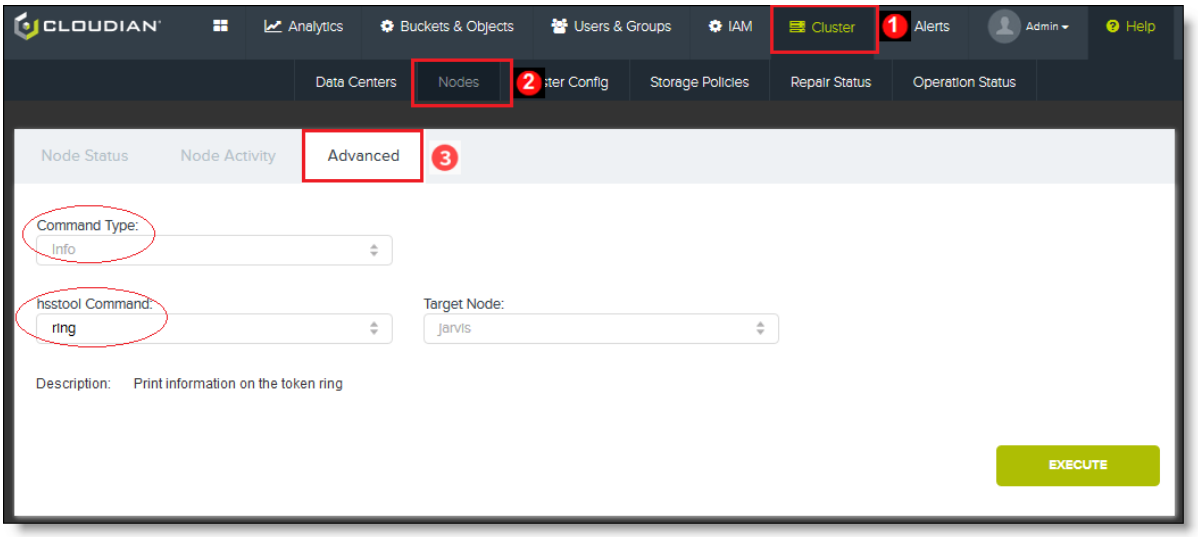
## 10.1.14.2. Usage Notes

This **hsstool** command provides status information for each of the dozens or hundreds of **virtual nodes** (vNodes) in your storage cluster. It is very granular and verbose. In most circumstances you will find more value in using the **hsstool info** or **hsstool status** commands rather than *ring*.

Since this command retrieves information for the whole cluster, the target <*host*> can be any node.

### CMC Support for This Command

As an alternative to running *hsstool ring* on the command line, you can run it through the CMC UI:




## 10.1.14.3. Command/Response Example

The *ring* command results display a status line for each virtual node (vNode) in the storage cluster. In a typical cluster the *ring* command may return hundreds of lines of information. The returned information is sorted by ascending vNode token number.

The example below is an excerpt from a *ring* command response for a four node HyperStore system that spans two data centers. Each of the four physical nodes has 32 vNodes, so the full response has 128 data lines. The list is sorted by ascending vNode token number. Note that although the command is submitted to a particular node ("cloudian-node1"), it returns information for the whole cluster. It doesn't matter which node you submit the command to.

```
# hsstool -h cloudian-node1 ring
Address         DC   Rack Cassandra Cassandra-Load  Hss  State  Token
105.236.130.70 DC1  RAC1   Up       2.05 MB        Up  Normal 21462031172012658791503332843230686l8
105.236.130.70 DC1  RAC1   Up       2.05 MB        Up  Normal 55423285286546307257765329278638683383
105.236.218.176 DC2 RAC1   Up       1.35 MB        Up  Normal
120005232879821718033775l4999547780254
105.236.218.176 DC2 RAC1   Up       1.35 MB        Up  Normal
138783654882411451567357278478650471l0
198.199.106.194 DC1 RAC1   Up       1.9 MB         Up  Normal
183151198631857301055573406308303115l5
...
...
```

### Response Items

*Address*

IP address of the physical node on which the vNode resides.

*DC*

Data center in which the vNode resides.

*Rack*

Rack in which the vNode resides.

*Cassandra*

Cassandra Service status of the vNode. Will be one of: "Up", "Down", "Joining" (in the process of joining the cluster), "Leaving" (in the process of decommissioning or being removed from the cluster), or "?" (physical host cannot be reached). All vNodes on a physical node will have the same Cassandra status.

*Cassandra-Load*

Cassandra load (quantity of data stored in Cassandra) for the physical host on which the vNode resides. There will be some Cassandra load even if all S3 objects are stored in the HyperStore File System or the erasure coding file system. For example, Cassandra is used for storage of object metadata and service usage data, among other things. Note that Cassandra load information is available only for the physical node as a whole; it is not available on a per-vNode basis.

*HSS*

HyperStore Service status for the vNode. Will be one of: "Up", "Down", or "?" (physical host cannot be reached). All vNodes on a physical node will have the same HSS status.

*State*

HyperStore Service state for the vNode. Will be one of: "Normal" or "Decommissioning". All vNodes on a given physical node will have the same HSS state.

*Token*

The vNode's token (from an integer token space ranging from 0 to $2^{127}$ -1). This token is the top of the token range that constitutes the vNode. Each vNode's token range spans from the next-lower token (exclusive) in the cluster up to its own token (inclusive).

## 10.1.15. hsstool status

*Subjects covered in this section:*



### 10.1.15.1. Command Syntax

### Synopsis

```
# hsstool [-h <host>] status
```

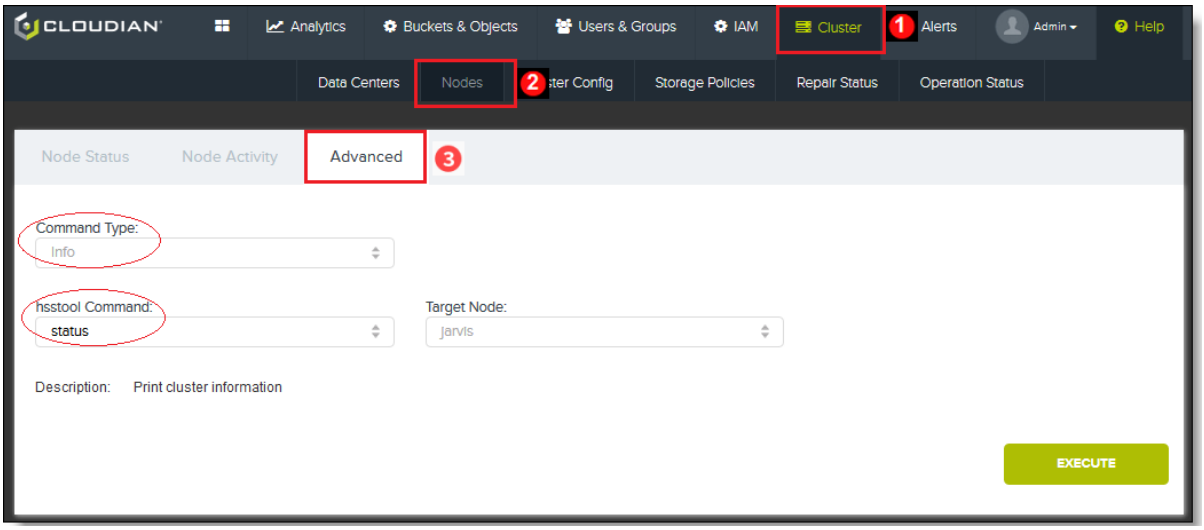
## 10.1.15.2. Usage Notes

This **hsstool** command returns status information for the storage cluster as a whole.

Since this command retrieves information for the whole cluster, the target *<host>* can be any node.

### CMC Support for This Command

As an alternative to running *hsstool status* on the command line, you can run it through the CMC UI:




## 10.1.15.3. Command/Response Example

The *status* command example below retrieves the status of a four-node cluster.

```
# hsstool -h localhost status
Datacenter: DC1
===============
Address     DC   Rack   Cassandra Tokens Cassandra-Load Hss State       Hyperstore-Disk     Host-ID
   Hostname
10.20.2.54 DC1  RAC1  Up       32     0.24118415     Up  Normal 427.66MB/7.75GB(5.39%) cfa851df-4170-
4006-a516-d5b56f99a820 cld05-04
10.20.2.57 DC1  RAC1  Down     32     0.2239672      Up  Normal 79.45MB/11.17GB(0.69%) 6bf98091-
5318-4ae6-acc3-f1524f5e7da3 cld05-07
10.20.2.55 DC1  RAC1  Up       32     0.23458646     Up  Normal 1.1GB/20.48GB(5.36%) c77ff93a-5579-
4f9f-b5a3-9e22d68fab84 cld05-05
10.20.2.52 DC1  RAC1  Up       32     0.3002622      Up  Normal 692.7MB/12.61GB(5.37%) 914117cb-
3ec5-4c79-9dff-5a9f33002ce3 cld05-02
```

*Address*

IP address of the physical node on which the vNode resides.

*DC*

Data center in which the node resides.

*Rack*

Rack in which the node resides.

*Cassandra*

Cassandra Service status of the node. Will be one of: "Up", "Down", "Joining" (in the process of joining the cluster), "Leaving" (in the process of decommissioning or being removed from the cluster), or "?" (physical host cannot be reached).

*Cassandra-Load*

Of the Cassandra data in the system as a whole, the portion (as a decimal) that is stored on each node.

*HSS*

HyperStore Service status for the node. Will be one of: "Up", "Down", or "?" (physical host cannot be reached).

*State*

HyperStore Service state for the node. Will be one of: "Normal" or "Decommissioning".

*HyperStore-Disk*

HyperStore data disk utilization on the node

*Host-ID*

System-generated unique ID for the node

*Hostname*

Hostname of the node

## 10.1.16. hsstool trmap

*Subjects covered in this section:*

- ***"Command Syntax"*** *(page 571)*
- ***"Usage Notes"*** *(page 572)*
- ***"Command/Response Examples"*** *(page 573)*

### 10.1.16.1. Command Syntax

#### Synopsis

```
# /opt/cloudian/bin/hsstool -h <host> trmap [list [-a]] [show <snapshotId>]
[set <snapshotId> <active/disabled>] [delete <snapshotId>]
```

#### Options

*list [-a]  List token range map snapshot IDs*

(Optional) Using *list***without** the optional *-a* flag returns only a list of active token range map snapshot IDs. These are token range map snapshots for which the associated rebalancing operation has not yet completed.

If you use the *-a* flag -- that is, *trmap list -a* -- then the command returns the IDs of **all** snapshots -- the active snapshots and also the disabled snapshots (snapshots for which the associated rebalancing operation has completed). When you use the *-a* flag the return includes a status field for each snapshot, to distinguish active snapshots from disabled snapshots. For example:

```
[root]# /opt/cloudian/bin/hsstool -h localhost trmap list -a
59af5410-b303-41bd-94a0-31ce92058e11 Tue Jun 26 09:03:56 EDT 2018 DISABLED
061e9190-6d62-429d-85dc-e4baec07f49e Wed Jun 19 15:26:07 EDT 2019 ACTIVE
22351e62-e921-4130-afaf-1ca3183046e1 Thu Jun 20 14:17:04 EDT 2019 ACTIVE
1c84d57e-dccb-47b0-af6d-015e823eab07 Wed Jun 19 21:42:42 EDT 2019 ACTIVE
9e096b8f-5914-42fc-8dc7-063de85deb4a Thu Jun 20 06:56:17 EDT 2019 ACTIVE
3bf72d7e-a2a1-4551-972c-1223a4485335 Mon Jun 25 17:23:32 EDT 2018 DISABLED
dbf98852-2557-4cfd-8708-0398009fc6d7 Mon Jun 25 18:38:47 EDT 2018 DISABLED
```

*show <snapshotId>  Show snapshot content*

(Optional) This returns the content of the specified token range map snapshot. This option is only supported on the command line, not in the CMC interface.

*set <snapshotId> <active/disabled>  Change snapshot status*

(Optional) **Do not use this option unless instructed to do so by Cloudian Support.** This sets the status of the specified token range map snapshot to *active* (rebalancing in connection with this snapshot still needs to be completed) or *disabled* (rebalancing in connection with this snapshot has been completed). This option is only supported on the command line, not in the CMC interface.

*delete <snapshotId>  Delete snapshot*

(Optional) **Do not use this option unless instructed to do so by Cloudian Support.** This deletes the specified token range map snapshot. This option is only supported on the command line, not in the CMC interface.

## 10.1.16.2. Usage Notes

This **hsstool** command returns a list of token range map snapshot IDs along with information about each snapshot such as the snapshot creation time. You can also use the command to return the contents of a specified token range map snapshot.

The system creates a token range map snapshot each time you **add a new node to your cluster**. The token range map identifies, for each storage policy in your system, the nodes (endpoints) that store data from each token range. The data from each token range will be stored on multiple nodes, with the number of nodes depending on the storage policy (for example, in a 3X replication storage policy each token range would be mapped to three storage endpoints). When you've added new nodes to your cluster, the system uses token range maps to manage the rebalancing of S3 object data from existing nodes to the new nodes.

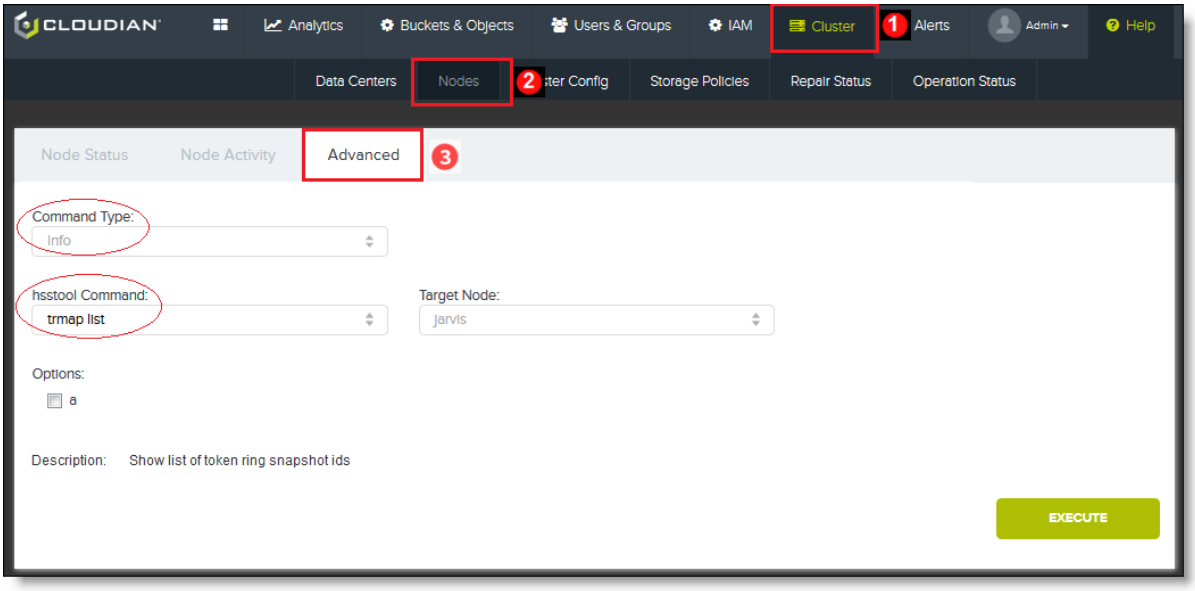
**Typically you should not need to use this command** unless you are working with Cloudian Support to troubleshoot a failed attempt to add nodes to your HyperStore cluster or a failed attempt to rebalance the cluster after adding nodes.

As the target <*host*> you can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**IMPORTANT !** Do not use the *set* or *delete* options unless instructed to do so by Cloudian Support.

### CMC Support for This Command

As an alternative to running *hsstool trmap* on the command line, you can run it through the CMC UI:




## 10.1.16.3. Command/Response Examples

In this first example a list of active token range map snapshot IDs is retrieved. These are token range map snapshots for which the associated rebalancing operation has not yet completed.

```
# /opt/cloudian/bin/hsstool -h localhost trmap list
061e9190-6d62-429d-85dc-e4baec07f49e Wed Jun 19 15:26:07 EDT 2019
22351e62-e921-4130-afaf-1ca3183046e1 Thu Jun 20 14:17:04 EDT 2019
1c84d57e-dccb-47b0-af6d-015e823eab07 Wed Jun 19 21:42:42 EDT 2019
9e096b8f-5914-42fc-8dc7-063de85deb4a Thu Jun 20 06:56:17 EDT 2019
```

In this next example, a token range map snapshot is retrieved (this is from a different system and is not one of the snapshots listed in the first example). The map is in JSON format. The response is very large, and is truncated below. In practice, if you use this command you should redirect the output to a text file.

```
# /opt/cloudian/bin/hsstool -h cloudian-node1 trmap show fbcdf5be-7c15-40a4-be59-
955df70e7fb2
{
  "id" : "fbcdf5be-7c15-40a4-be59-955df70e7fb2",
  "version" : "1",
  "timestamp" : 1477266709616,
  "rebalance" : "REQUIRED",
  "policies" : {
    "5871e62dae4ccfd618112ca3403b3251" : {
      "policyId" : "5871e62dae4ccfd618112ca3403b3251",
      "keyspaceName" : "UserData_5871e62dae4ccfd618112ca3403b3251",
      "replicationScheme" : {
        "DC2" : "1",
        "DC1" : "1",
        "DC3" : "1"
       },
      "ecScheme" : null,
      "ecMap" : null,
      "replicasMap" : [ {
        "left" : 163766745962987615139826681359528839019,
```

```
      "right" : 164582023203604297970082254372849331112,
      "endPointDetails" : [ {
        "endpoint" : "10.10.10.114",
        "datacenter" : "DC3",
        "rack" : "RAC1"
      }, {
        "endpoint" : "10.10.10.115",
        "datacenter" : "DC2",
        "rack" : "RAC1"
      }, {
        "endpoint" : "10.10.10.111",
        "datacenter" : "DC1",
        "rack" : "RAC1"
      } ]
    }, {
      "left" : 63416606637620968312905411887124449 13,
      "right" : 64497377786602168777274729714048571 43,
      "endPointDetails" : [ {
        ...
        ...
```

## Response Items

*id*

System-generated unique identifier of this token range map snapshot.

*version*

Version of the token range map snapshot. This integer is incremented each time a new snapshot is created.

*timestamp*

Timestamp indicating when the token range map snapshot was created.

*rebalance*

Status of the *hsstool rebalance* operation in regard to this token range map snapshot, such as REQUIRED or COMPLETED. Each time you add a node to your system (using the CMC's function for adding a node, in the **Data Centers** page), the system automatically generates a token range snapshot. After adding a node, you then must run *hsstool rebalance* on the new node (which you can do from the CMC's **Nodes Advanced** page). The rebalance operation utilizes the token range map snapshot. For complete instructions on adding nodes, see **"Adding Nodes"** (page 264).

*policies*

This marks the beginning of the per-policy token range map information. The token range map will have separate token range map information for each of your storage policies.

*policyId*

System-generated unique identifier of a storage policy. Note that this ID appears three times: at the outset of the policy block, then again as the *policyId* attribute, then again within the *keyspaceName*.

*keyspaceName*

Name of the Cassandra keyspace in which object metadata is stored for this storage policy. The name is in

format *UserData_<policyId>*.

*replicationScheme*

Specification of the replication scheme, if this policy block is for a replication storage policy. In the example the policy's replication scheme calls for one replica in each of three data centers.

*ecScheme*

Specification of the erasure coding scheme, if this policy block is for an erasure coding storage policy. In the example, the policy block is for a replication policy so the *ecScheme* value is null.

*ecMap*

Content of the token range map for the policy scheme, if this policy block is for an erasure coding storage policy. In the example, the policy block is for a replication policy so the *ecMap* value is null.

*replicasMap*

Content of the token range map for the policy scheme, if this policy block is for a replication storage policy. The map consists of lists of endpoints per token range.

*left*

Token at the low end of the token range (exclusive). This is from the consistent hashing space of 0 to $2^{127}$ from which HyperStore generates tokens for the purpose of allocating data across the cluster.

*right*

Token at the high end of the token range (inclusive). This is from the consistent hashing space of 0 to $2^{127}$ from which HyperStore generates tokens for the purpose of allocating data across the cluster.

*endPointDetails*

Endpoint mapping information for this particular token range, for this storage policy. This is a list of endpoints (nodes), with each endpoint identified by IP address as well as data center name and rack name. The number of endpoints per token range will depend on the storage policy scheme. In the example the policy is a 3X replication policy, so there are three endpoints listed for each token range. Objects for which the object token (based on a hash of the bucket name / object name combination) falls into this token range will have replicas placed on each of these nodes.

## 10.1.17. hsstool whereis

*Subjects covered in this section:*



### 10.1.17.1. Command Syntax

#### Synopsis

```
# hsstool [-h <host>] whereis <bucket>/<object> [-v <version>] [-ck] [-a]
```

### Options

*<bucket>/<object>  Bucket and object name*

(Mandatory unless using the *-a* option) Bucket name, followed by a forward slash, followed by the full object name (including "folder path", if any). For example, *mybucket/file1.txt* or *mybucket/Videos/Vacation/Italy_2021-06-27.mpg*.

If the object name has spaces in it, enclose the *bucket/object* name pair in quotes. For example, *"mybucket/big document.doc".*

The *bucket/object* name is case-sensitive.

> **Note** In the CMC UI implementation of this command, you enter the bucket name and the full object name (including folder path) in separate fields. For example, bucket name *mybucket* and full object name *Videos/Vacation/Italy_2021-06-27.mpg*.

*-v <version>  Object version*

(Optional) If versioning is enabled on the bucket that contains the object, you can optionally specify the version ID of a particular version of the object. Version IDs are system-generated hexadecimal values (for example, *fe1be647-5f3b-e87f-b433-180373cf31f5*). If versioning has been used for the object but you do not specify a version ID, location information will be retrieved for the most recent version of the object.

*-ck  Check for object corruption*

(Optional) Use the *-ck* option if you want the *whereis* results to indicate whether any of the target object's replicas or fragments are corrupted. When you use this option, the *whereis* operation detects corruption by comparing a freshly computed MD5 hash of each replica or fragment to what the MD5 hash of each replica or fragment should be, based on the object's stored metadata.

*-a  Write location info for all objects to a log file*

(Optional) Use the *-a* option if you want a full list of **all replicas and fragments of all objects in the entire service region** . This information will be written to a log file. For more information on the log file, see the section that follows the command/response example.

If you use the *-a* option do not use the <*bucket/object*> parameter or the *-v* <*version*> parameter. Run the command simply as *hsstool -h <host> whereis -a*

> **Note** The CMC does not support the *-a* option. To use this option you need to use *hsstool whereis* on the command line.

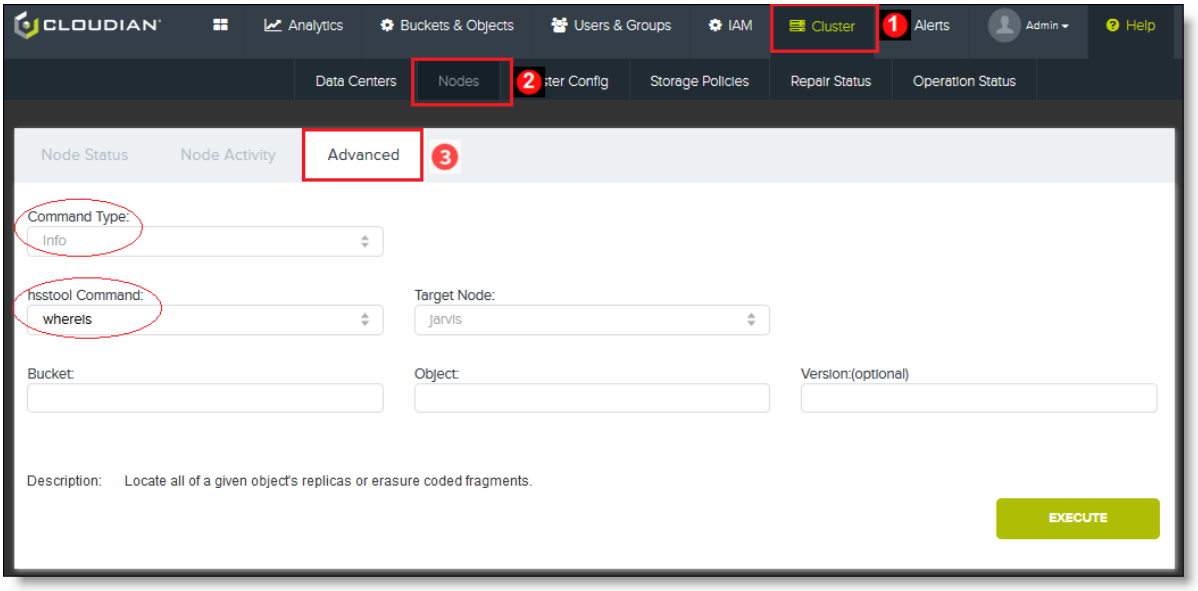
## 10.1.17.2. Usage Notes

This **hsstool** command returns the current storage location of each replica of a specified S3 object (or in the case of erasure coded objects, the location of each of the object's fragments). The command response also shows the specified object's metadata such as last modified timestamp and object digest.

Since this command returns information from across the cluster, you can specify any node as the target <*host*>.

### CMC Support for This Command

As an alternative to running *hsstool whereis* on the command line, you can run it through the CMC UI:




### Log File for "whereis -a"

When you use the *whereis -a* command, information about all replicas and erasure coded fragments of all objects in the entire service region is written to a log file. The log file is written on the HyperStore host that you connect to when you run *whereis -a*, and by default the log file path is:

```
/var/log/cloudian/whereis.log
```

In the *whereis.log* file, the start and completion of the output from a single run of *whereis -a* is marked by "START"and "END" timestamps. Within those timestamps, the output is organized by user. The start of output for a particular user is marked by "#user:<canonical UID>". This line is then followed by lines for the user's buckets and objects, with the same object detail information as described in the *whereis* command results documentation above. Users who do not have any buckets will not be included in the log file.

The output of multiple runs of *whereis -a* may be written to the same log file, depending on the size of the output. Because the output of *whereis -a* may be very large, it's also possible that the output of a single run may be spread across multiple log files, if maximum file size is reached and log rotation occurs.

By default this log is rotated if it reaches 10MB in size or at the end of the day, whichever occurs first. The oldest rotated *whereis* log file is automatically deleted if it reaches 180 days in age or if the aggregate size of all rotated *whereis* log files (after compression) reaches 100MB. These rotation settings are configurable in the *RollingRandomAccessFile name="APP"* section of the */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hsstool.xml.erb* file. For information about changing these settings see **"Log Configuration Settings"** (page 373).

## 10.1.17.3. Command/Response Examples

The first *whereis* command example below retrieves location information for an object named "Guide.pdf". This is from a single-node HyperStore system, so there is just one replica of the object. The location detail information for the object replica is truncated in this example.

```
# hsstool -h sirius whereis bucket2/Guide.pdf
Key: bucket2/Guide.pdf
Policy ID: c4a276180b0c99346e2285946f60e59c
Version: null
Compression: NONE
```

```
Create Time: 2017-02-20T16:38:09.783Z
Last Modified: 2017-02-20T16:38:09.783Z
Last Access Time: 2017-02-20T16:38:10.273Z
Size: 7428524
Type: REPLICAS
Region: region1
[DC1 10.50.10.21 sirius] bucket2/Guide.pdf file://10.50.10.21:/var/lib/cloudian/hsfs/1L1t...
```

The second *whereis* command example retrieves location information for an object named "obj1.txt". This is from a two data center HyperStore system, using replicated 2+1 erasure coding. The location detail information for each found fragment is truncated in this example.

```
# hsstool -h cloudian-node3 whereis bucket1/obj1.txt
Key: bucket1/obj1
Policy ID: 1d140a90b17285cf2d1502cbd424d621
Version: null
Compression: NONE
Create Time: 2018-01-29T23:07:52.626Z
Last Modified: 2018-01-29T23:07:52.626Z
Last Access Time: 2018-01-29T23:07:52.626Z
Size: 11231
Type: EC
Region: region1
6 out of 6 fragments were found
[DC1 2 10.100.186.47 cloudian-node3] bucket1/obj1.txt ec://10.100.186.47:/var/lib/cloudian/ec/...
[DC1 3 10.100.76.90 cloudian-node4] bucket1/obj1.txt ec://10.100.76.90:/var/lib/cloudian/ec/...
[DC2 1 10.100.61.17 cloudian-node6] bucket1/obj1.txt ec://10.100.61.17:/var/lib/cloudian/ec/...
[DC2 2 10.100.80.35 cloudian-node7] bucket1/obj1.txt ec://10.100.80.35:/var/lib/cloudian/ec/...
[DC2 3 10.100.218.23 cloudian-node8] bucket1/obj1.txt ec://10.100.218.23:/var/lib/cloudian/ec/...
[DC1 1 10.100.186.42 cloudian-node1] bucket1/obj1.txt ec://10.100.186.42:/var/lib/cloudian/ec/...
```

**Note** For objects for which the Type is "TRANSITIONED" (auto-tiered), the response will also include a *URL* field.

## Response Items

*Key*

Key that uniquely identifies the S3 object, in format <*bucketname*>/<*objectname*>. For example, *bucket1/Documents/Meetings_2021-06-27.docx*.

*PolicyID*

System-generated identifier of the storage policy that applies to the bucket in which this object is stored.

*Version*

Object version, if versioning has been used for the object. Versions are identified by timeuuid values in hexadecimal format. If versioning has not been used for the object, the Version field displays "null".

*Compression*

Type of server-side compression applied to the object, if any. Possible values are NONE, SNAPPY, ZLIB, or LZ4. The type of compression applied depends on the storage policy used by the bucket. Each storage policy has its own configuration as to whether compression is used and the compression type.

*Create Time*

Timestamp for the original creation of the object. Format is ISO 8601 and the time is in Coordinated Universal Time (UTC).

*Last Modified*

Timestamp for last modification of the object. Format is ISO 8601 and time is in UTC.

*Last Access Time*

Timestamp for last access of the object. Last Access Time is subject to updating only for objects in buckets for which a lifecycle is configured. For such objects, Last Access Time is updated if the object is accessed either for retrieval (GET or HEAD) or modification (PUT/POST/Copy). Format is ISO 8601 and time is in UTC.

*Size*

The object's size in bytes.

*Type*

One of:

- REPLICAS — The object is replicated in the HyperStore File System (HSFS).
- EC — The object is erasure coded in the HSFS.
- TRANSITIONED — The object has been transitioned (**auto-tiered**) to a different storage system such as Amazon S3.

*Region*

The HyperStore service region in which the object is stored.

*URL (transitioned objects only*

This field appears only in the case of TRANSITIONED objects. For such objects, this field shows the URL that identifies the location of the object in the tiering destination system. For example:

```
http://s3.amazonaws.com/bucket2.mdazyjgxnjyxndu2ody4mji1nty3/notes.txt
```

In this example, the tiering destination is Amazon S3; the bucket name in the destination system is *bucket2.mdazyjgxnjyxndu2ody4mji1nty3* (which is the HyperStore source bucket name — *bucket2* in this case — appended by a 28 character random string); and the object name is *notes.txt*. Note that the URL field will specify the transfer protocol as *http*, whereas to actually access the object in the destination system the protocol would typically be *https*.

*Fragment count summary (erasure coded objects only)*

For erasure coded objects, the location detail section of the *hsstool whereis* response starts with a line stating the number of fragment locations that have been identified for the object -- for example "6 out of 6 fragments were found" for an object for which there are supposed to be 6 fragments. This statement does not necessarily mean that all the fragments were found in their expected locations -- it means only that 6 expected fragment locations were identified. If the fragment itself is found at the expected location, in the location detail section the last field of the entry for that location will show the actual size of the found fragment. **If the fragment is missing from its expected location, the location entry's last field will show a size of "-1".**

*Location detail (including indicator of whether the replica or fragment is found at the expected location)*

For objects stored locally (objects that are not of type TRANSITIONED), the lower part of the response shows

the location of **each object replica** (for replicated objects) or of **each erasure coded object fragment** (for EC objects).

For HSFS replicated objects each location is specified as:

```
[<datacenter> <IP-address> <hostname>] <bucket>/<objectname>
file://<IP-address>:<mountpoint>/hsfs/<base62-encoded-vNode-token>/<policyid>/
<000-255>/<000-255>/<filename> <last-modified> <version> <digest> <size>
```

For erasure coded object fragments each location is specified as:

```
[<datacenter> <key-suffix-digit> <IP-address> <hostname>] <bucket>/<objectname>
ec://<IP_address>:<mountpoint>/ec/<base62-encoded-vNode-token>/<policyid>/
<000-255>/<000-255>/<filename> <last-modified> <version> <digest> <size>
```

- For EC objects only, the *<key_suffix_digit>* at the beginning of each location is a digit that the system generates and uses to ensure that each fragment goes to a different node.
- The *<base62-encoded-vNode-token>* is a base-62 encoding of the token belonging to the vNode to which the object instance or fragment is assigned.
- The *<policyid>* segment is the unique identifier of the storage policy applied to the bucket in which the object is stored.
- The two *<000-255>* segments of the path are based on a hash of the *<filename>*, normalized to a 255*255 number.
- The *<filename>* is a dot-separated concatenation of the object's hash token and the object's Last Modified Time timestamp. The timestamp is formatted as *<UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>* (the last element is the IP address -- in hexadecimal format -- of the S3 Service node that processed the object upload request). Note: For objects last modified prior to HyperStore version 6.1, the timestamp is simply Unix time in milliseconds. This was the timestamp format used in HyperStore 6.0.x and older..
- If the replica or fragment is found at the expected location, the *<size>* field shows the size of the replica or fragment . If the replica or fragment is not found at the expected location, the size field shows "-1". **A size of "-1" indicates that the replica or fragment is missing from a location where it is supposed to be.** If the digest is also missing, the digest will be absent from the location detail entry.

**Note** For multipart objects (large objects uploaded via the S3 multipart upload method), storage location detail is shown **for each part**.

## 10.2. cloudianInstall.sh

The *cloudianInstall.sh* tool (also known as "the installer") serves several purposes including:

- Installation of a HyperStore cluster
- Implementing advanced, semi-automated system configuration changes
- Pushing configuration file edits to the cluster and restarting services to apply the changes

The *cloudianInstall.sh* tool is in your installation staging directory on your Configuration Master node. To perform advanced configurations, or to push configuration file changes to the system and restart services, you would launch the tool simply like this, without using additional command line options:

```
# ./cloudianInstall.sh
```

## 10.2.1. Command Line Options When Using cloudianInstall.sh for HyperStore Cluster Installation

To perform a HyperStore cluster installation you typically would launch the script either like this:

```
# ./cloudianInstall.sh -s survey.csv
```

Or like this if you are not using your DNS environment to resolve HyperStore service endpoints and you want to use the bundled tool *dnsmasq* instead (which is not appropriate for production systems):

```
# ./cloudianInstall.sh -s survey.csv configure-dnsmasq
```

However the script does support additional command line options. The syntax is as follows:

```
# ./cloudianInstall.sh [-s <survey-filename>] [-k <ssh-private-key-filename>]
[-d] [-h] [no-hosts] [configure-dnsmasq] [no-firewall] [force] [uninstall]
```

**Note** If you use multiple options, on the command line place options that start with a "-" (such as *-s <survey-filename>* or *-d*) before options that do not (such as *no-hosts* or *configure-dnsmasq*).

*If you are using the **HyperStore Shell***

If you are using the HyperStore Shell (HSH) as a Trusted user, from any directory on the Configuration Master node you can launch the installer with this command:

```
$ hspkg install
```

The installer's options are the same regardless of whether it is launched from the HSH command line or the OS command line.

**Note** After using the installer, exit the installer when you're done. Do not leave it running. Certain automated system tasks invoke the installer and cannot do so if it is already running.

The supported command line options are:

- *[-s <survey-filename>]* — Name of your cluster survey file (including the full path to the file). If you do not specify the survey file name argument, the script will prompt you for the file name during installation.
- *[-k <ssh-private-key-filename>]* — The Configuration Master employs SSH for secure communication with the rest of your HyperStore installation nodes. By default the install script automatically creates an SSH key pair for this purpose. But if instead you would prefer to use your own existing SSH key pair for this purpose, you can use the installer's *-k <ssh-private-key-filename>* option to specify the name of the private key file (including the full path to the file). When you run the install script it will copy the private key and corresponding public key to the installation staging directory, and in the staging directory the key file will be renamed to *cloudian-installation-key*. Then from the staging directory, the public key file *cloudian-installation-key.pub* will be copied to each node on which you are installing HyperStore.
- *[-d]* — Turn on debugging output.
- *[-h]* — Display usage information for the install tool. This option causes the tool to print a usage message and exit.

  **Note** This usage information mentions more command line options than are described here in this Help topic. This is because the usage information includes installer options that are meant for HyperStore internal system use, such as options that are invoked by the CMC when you use

the CMC to add nodes to your cluster or remove nodes from your cluster. You should perform such operations through the CMC, not directly through the installer. The CMC implements automations and sanity checks beyond what is provided by the install script alone.

- *[no-hosts]* — Use this option if you do not want the install tool to append entries for each HyperStore host on to the */etc/hosts* file of each of the other HyperStore hosts. By default the tool appends to these files so that each host is resolvable to the other hosts by way of the */etc/hosts* files.
- *[configure-dnsmasq]* — Use this option if you want the install tool to install and configure **dnsmasq**, a lightweight utility that can provide domain resolution services for testing a small HyperStore system. If you use this option the installer installs *dnsmasq* and automatically configures it for resolution of HyperStore service domains. If you did not create DNS entries for HyperStore service domains as described in **"DNS Set-Up"** (page 50), then you must use the *configure-dnsmasq* option in order for the system to be functional when you complete installation. Note that using *dnsmasq* is not appropriate in a production environment.

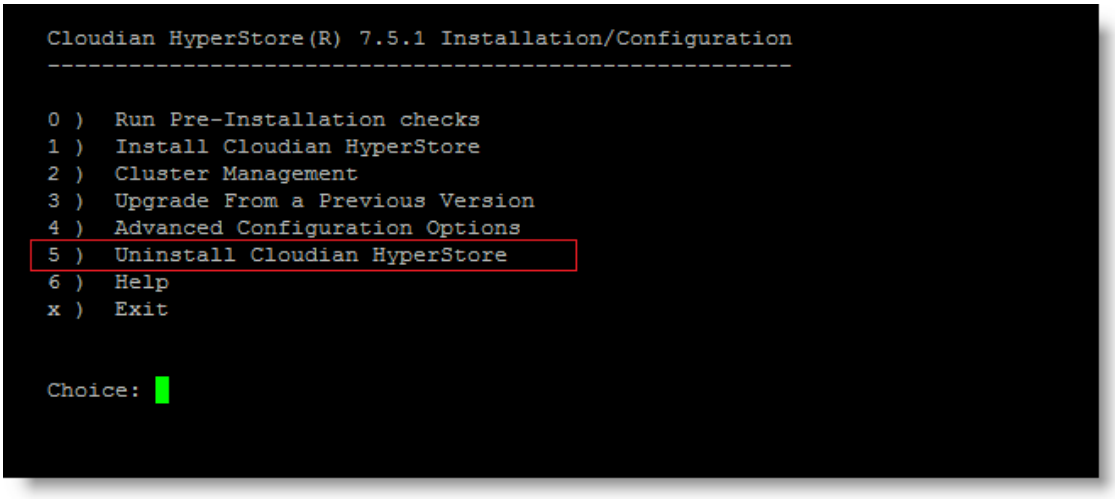
  **Note** If you do not have the installer install *dnsmasq* during HyperStore installation, and then later you decide that you do want to use *dnsmasq* for your already installed and running HyperStore system, do not use the *configure-dnsmasq* command line option when you re-launch the installer. Instead, re-launch the installer with no options and use the **"Installer Advanced Configuration Options"** (page 377) menu to enable *dnsmasq* for your system.

- [*no-firewall*] — If this option is used, the HyperStore firewall will **not** be enabled upon HyperStore installation. By default the HyperStore firewall will be enabled upon completion of a fresh HyperStore installation.
- *[force]* — By default the installer performs certain prerequisite checks on each node on which you are installing HyperStore and aborts the installation if any of your nodes fails a check. By contrast, if you use the *force* option when you launch the installer, the installer will output warning messages to the terminal if one or more nodes fails a prerequisite check but the installation will continue rather than aborting. The prerequisite checks that this feature applies to are:
  - CPU has minimum of 8 cores
  - RAM is at least 128GB
  - System Architecture is x86 64-bit
  - SELinux is disabled
  - firewalld is disabled
  - iptables is not running

  **Note** If you specify the *force* option when running the installer, the *force*option will "stick" and will be used automatically for any subsequent times the installer is run to install additional nodes (such as when you do an "Add Node" operation via the Cloudian Management Console, which invokes the installer in the background). To turn the *force*option off so that it is no longer automatically used when the installer is run to add more nodes, launch the installer and go to the Advance Configuration Options. Then choose option **t** for **Configure force behavior** and follow the prompts.

> **Note** Even if the *force* option is used the installer will abort if it detects an error condition on the host that will prevent successful installation.

- *[uninstall]* — If you use this option when launching the installer, the installer main menu will include an additional menu item -- "Uninstall Cloudian HyperStore".




Use this menu option only if you want to **delete the entire HyperStore system, on all nodes, including any metadata and object data** stored in the system. You may want to use this Uninstall Cloudian HyperStore option, for example, after completing a test of HyperStore -- if you do not want to retain the test system.

> **IMPORTANT !** Do not use this option to uninstall a single node from a HyperStore system that you want to retain (such as a live production system).

## 10.3. system_setup.sh

The *system_setup.sh* tool is for setting up nodes on which you will install HyperStore software, either during initial cluster installation or during cluster expansion. For basic information about using *system_setup.sh*, change into the installation staging directory and run the following command:

```
# ./system_setup.sh --help
```

## 10.4. search-mgr.sh

The *search-mgr.sh* tool is relevant only if you have deployed the optional HyperStore Search Service. For information about this tool see **"Using the search-mgr.sh Tool For On-Demand Indexing"** (page 205).

# 10.5. Redis Monitor Commands

The HyperStore Redis Monitor Service monitors Redis Credentials DB and Redis QOS DB cluster health and implements automatic failover of the master node role in each cluster. The Redis Monitor runs on two nodes, with one instance being the primary and the other being the backup. When you submit Redis Monitor commands **you must submit them to the primary node**, not the backup. To check which of your nodes is running the primary Redis Monitor, go to the CMC's **Cluster Information** page (**Cluster -> Cluster Config -> Cluster Information**).

You can submit commands to the Redis Monitor primary host through the Redis Monitor CLI. A couple of the more useful commands can also be executed through the CMC.

- **To initiate a Redis Monitor CLI session**, use *netcat* to connect to port 9078 on the node on which the primary Redis Monitor is running:

  ```
  # nc <redismon_primary_host> 9078
  ```

  Specify the hostname or IP address (do not use 'localhost'). Once connected, you can then use any of the Redis Monitor CLI commands listed below. When you're done using Redis Monitor commands, enter **quit** to end your Redis Monitor CLI session and then enter ***<ctrl>-d*** to end your *netcat* session and return to the terminal prompt.

  *If you are using the **HyperStore Shell***

  Because 'nc' is not a supported command for the HyperStore Shell, you cannot initiate a Redis Monitor CLI session through the HyperStore Shell. Your only access to Redis Monitor commands is through the CMC as noted below.

- **To access Redis Monitor commands in the CMC**, go to the **Node Advanced** page (**Cluster -> Nodes -> Advanced**) and from the "Command Type" drop-down list select "Redis Monitor Operations". Note that only a small number of Redis Commands are available through the CMC (specifically *get cluster* and *set master*).

The Redis Monitor supports the following commands:

## 10.5.1. get cluster

### get cluster

Use this command to retrieve basic status information that the Redis Monitor currently has for a specified Redis DB cluster. The cluster status information includes:

- The identity of the master node within the cluster
- Whether monitoring of the cluster by the Redis Monitor is enabled
- Whether the sending of cluster status notifications to clients of the cluster is enabled
- A list of cluster member nodes, with an indication of the status (UP/DOWN) of each node
- A list of cluster clients (which write to and/or read from this Redis database), with an indication of the status (UP/DOWN) of each client. The clients will include S3 Service instances (identified by JMX listening socket <host>:19080), IAM Service instances (<host>:19084), Admin Service instances (<host>:19081), and HyperStore Service instances (<host>:19082). These are the clients to which the Redis Monitor sends notifications regarding the Redis cluster's status.

- "state" information that includes a timestamp indicating the last date and time that the master role status was updated (if there have been any fail-overs of the master role, the timestamp is the time of the last fail-over -- otherwise, it's the time of the last start-up of the Redis Monitor)

### 10.5.1.1. Command Line Syntax

```
get cluster redis.credentials|redis.qos.<region>
```

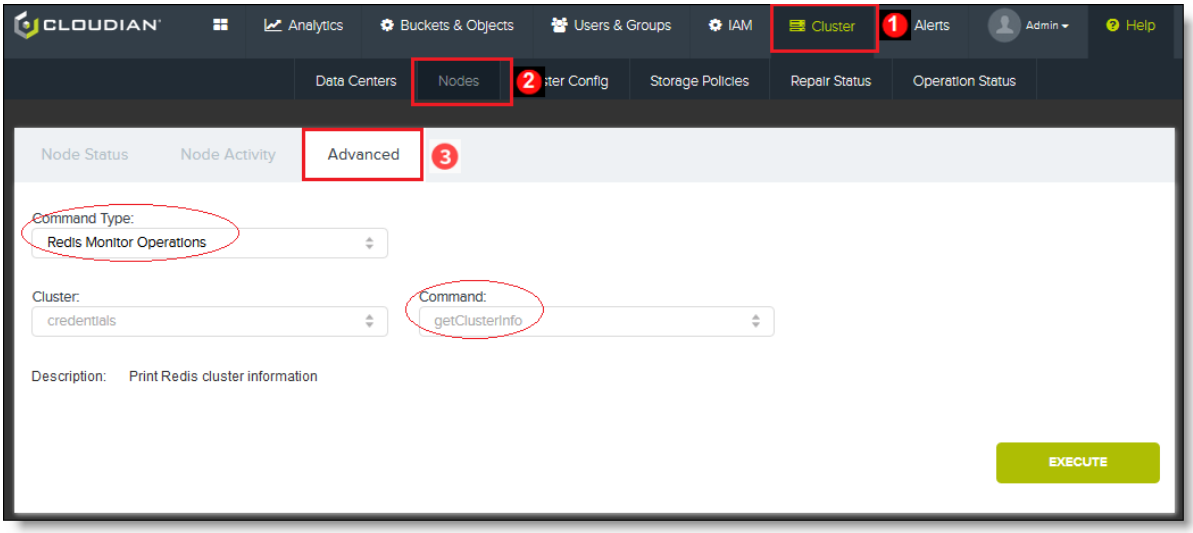
> **Note** For this and all other Redis Monitor commands, the Redis QOS cluster identifier includes the name of the service region in which the cluster resides. This is necessary since in a multi-region HyperStore system each region has its own Redis QoS cluster. By contrast the Redis Credentials cluster, since it is global (extending across all service regions), does not include a region name in its identifier.

Example:

```
# nc 10.50.20.12 9078
get cluster redis.credentials
OK master: store1(10.50.20.1):6379, monitoring: enabled, notifications: enabled
nodes: [[store1(10.50.20.1):6379,UP,master], [store4(10.50.20.4):6379,UP,slave],
[store5(10.50.20.5):6379,UP,slave]]
clients: [[store1(10.50.20.1):19080,UP,store1], [store2(10.50.20.2):19080,UP,store1],
[store3(10.50.20.3):19080,UP,store1], [store4(10.50.20.4):19080,UP,store1],
[store5(10.50.20.5):19080,UP,store1], [store6(10.50.20.6):19080,UP,store1],
[store1(10.50.20.1):19081,UP,store1], [store2(10.50.20.2):19081,UP,store1],
[store3(10.50.20.3):19081,UP,store1], [store4(10.50.20.4):19081,UP,store1],
[store5(10.50.20.5):19081,UP,store1], [store6(10.50.20.6):19081,UP,store1],
[store1(10.50.20.1):19082,UP,store1], [store2(10.50.20.2):19082,UP,store1],
[store3(10.50.20.3):19082,UP,store1], [store4(10.50.20.4):19082,UP,store1],
[store5(10.50.20.5):19082,UP,store1], [store6(10.50.20.6):19082,UP,store1]]
state: redis.credentials: master= 10.50.20.1 updatetime= Fri Sep 21 16:17:23 PDT 2018
```

### 10.5.1.2. CMC UI

Path: **Cluster** → **Nodes** → **Advanced**



## 10.5.2. get master

### get master

Use this command to retrieve from the Redis Monitor the identity of the current master node within a specified Redis cluster.

#### 10.5.2.1. Command Line Syntax

```
get master redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
get master redis.qos.region1
OK store2(10.50.20.2):6380
```

## 10.5.3. get nodes

### get nodes

Use this command to retrieve from the Redis Monitor a list of all current members of a specified Redis cluster. The command response also indicates the status (UP/DOWN) and role (master/slave> of each member node.

#### 10.5.3.1. Command Line Syntax

```
get nodes redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
get nodes redis.credentials
OK [[[store1(10.50.20.1):6379,UP,master], [store4(10.50.20.4):6379,UP,slave],
[store5(10.50.20.5):6379,UP,slave]]]
```

## 10.5.4. get clients

### get clients

Use this command to retrieve from the Redis Monitor a list of clients of a specified Redis cluster (client nodes that write to and/or read from the Redis database). These are the clients to which the Redis Monitor sends notifications regarding the Redis cluster's status. For example, if the cluster's master role changes from one node to another, the Redis Monitor will notify these clients of the change.

The clients will include S3 Service instances (identified by JMX listening socket <host>:19080), IAM Service instances (<host>:19084), Admin Service instances (<host>:19081), and HyperStore Service instances (<host>:19082). The command response also indicates the status (UP/DOWN) of each client, and for each client it shows which node the client thinks is the Redis master.

#### 10.5.4.1. Command Line Syntax

```
get clients redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
get clients redis.credentials
OK [[[store1(10.50.20.1):19080,UP,store1], [store2(10.50.20.2):19080,UP,store1],
[store3(10.50.20.3):19080,UP,store1], [store4(10.50.20.4):19080,UP,store1],
[store5(10.50.20.5):19080,UP,store1], [store6(10.50.20.6):19080,UP,store1],
[store1(10.50.20.1):19081,UP,store1], [store2(10.50.20.2):19081,UP,store1],
[store3(10.50.20.3):19081,UP,store1], [store4(10.50.20.4):19081,UP,store1],
[store5(10.50.20.5):19081,UP,store1], [store6(10.50.20.6):19081,UP,store1],
[store1(10.50.20.1):19082,UP,store1], [store2(10.50.20.2):19082,UP,store1],
[store3(10.50.20.3):19082,UP,store1], [store4(10.50.20.4):19082,UP,store1],
[store5(10.50.20.5):19082,UP,store1], [store6(10.50.20.6):19082,UP,store1]]]
```

In the above example, "store1" is the current Redis Credentials master node. All the clients correctly have this information.

## 10.5.5. enable monitoring

### enable monitoring

Use this command to enable monitoring of a specified Redis cluster by the Redis Monitor.

> **Note** Monitoring is enabled by default. This command is relevant only if you have previously disabled monitoring.

#### 10.5.5.1. Command Line Syntax

```
enable monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
enable monitoring redis.credentials
OK enabled
```

## 10.5.6. disable monitoring

### disable monitoring

Use this command if you want to temporarily disable Redis Monitor's monitoring of a specified Redis cluster — for example if you are performing maintenance work on the Redis cluster. (You can subsequently use the *enable monitor* command re-enable monitoring of that cluster.)

#### 10.5.6.1. Command Line Syntax

```
disable monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
disable monitoring redis.qos.region1
OK disabled
```

## 10.5.7. enable notifications

### enable notifications

Use this command to enable Redis Monitor's sending of notifications to the clients of a specified Redis cluster. (The clients are the S3 Service instances, IAM Service instances, Admin Service instances, and HyperStore Service instances that write to and/or read from that Redis cluster).

The Redis Monitor sends notifications to inform clients of the identity of the Redis cluster's master node, in either of these circumstances:

- The Redis master role has switched from one host to another. (This could happen if the original master goes down and Redis Monitor detects this and fails the master role over to one of the slave nodes; or if an operator uses the Redis Monitor CLI to move the master role from one node to another).
- The Redis Monitor in its regular polling of cluster clients' status detects that one of the clients has incorrect information about the identity of the Redis cluster master node. In this case the Redis Monitor notifies the client to give it the correct information.

**Note** Notifications are enabled by default. This operation is relevant only if you have previously disabled notifications using the *disable notifications* command.

#### 10.5.7.1. Command Line Syntax

```
enable notifications redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
enable notifications redis.credentials
OK enabled
```

## 10.5.8. disable notifications

### disable notifications

Use this command to temporarily disable Redis Monitor's sending of Redis cluster status notifications to the clients of that cluster. For more information on the notification feature see **"enable notifications"** (page 588).

#### 10.5.8.1. Command Line Syntax

```
disable notifications redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
disable notifications redis.qos.region1
OK disabled
```

## 10.5.9. set master

### set master

Use this command to assign the Redis master role to a different node within a specified Redis cluster. **The node to which you assign the master role must be one of the current slaves within the same Redis cluster**.

The Redis master node within a cluster is the node to which Redis clients submit writes. The writes are asynchronously replicated to the slave(s) within that cluster. Redis clients read from the slave(s).

An example of when you would move the Redis master role is if you want to remove the current Redis master host from your cluster.

Using this command is part of a broader procedure for moving a Redis master role to a slave. For the full procedure including the use of this command within the procedure, see **"Move the Credentials DB Master Role or QoS DB Master Role"** (page 317).

### 10.5.9.1. Command Line Syntax

```
set master redis.credentials|redis.qos.<region> [<host:redisPort>]
```

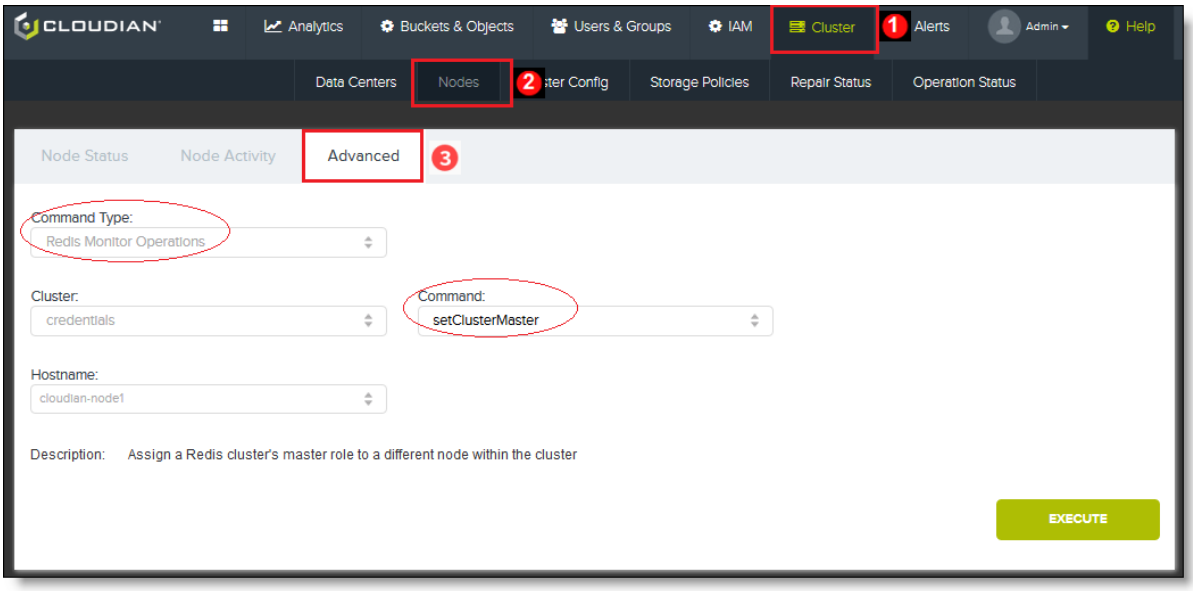
Example:

```
# nc 10.50.20.12 9078
set master redis.credentials store5:6379
OK set new master store5(10.50.20.5):6379
```

**Note** If you do not specify a *<host:redisPort>* value, the Redis Monitor chooses a slave node at random (from within the cluster) to elevate to the master role.

### 10.5.9.2. CMC UI

Path: **Cluster → Nodes → Advanced**



In the CMC UI, use the "Hostname" field to specify the host to which you want to move the Redis master role.

## 10.5.10. add node

### add node

Use this command to add a Redis node to the list of nodes that Redis Monitor is monitoring, for a specified Redis cluster. This would be if you have used the installer (*cloudianInstall.sh* in the installation directory on your Configuration Master node) to activate Redis on a HyperStore node that wasn't previously running Redis. In this circumstances you have two options to make Redis Monitor aware of the new member of the Redis cluster:

- Restart Redis Monitor

  OR

- Use this command.

#### 10.5.10.1. Command Line Syntax

```
add node redis.credentials|redis.qos.<region> <host:redisPort>
```

Example:

```
# nc 10.50.20.12 9078
add node redis.qos.region1 store3:6380
OK added node store3(10.50.20.3):6380 to redis
```

## 10.5.11. add client

### add client

This command can be used to add a new S3 Service, IAM Service, Admin Service, and/or HyperStore Service node to the list of clients to which Redis Monitor will send notifications regarding the status of a specified Redis cluster.

In normal circumstances you should not have to use this command. If you add a new node to your HyperStore cluster (as described in **"Adding Nodes"** (page 264)), the system automatically makes Redis Monitor aware of the new clients of the Redis Credentials and Redis QoS clusters.

If you do use this command, add only one client per command run. A client is identified by its JMX socket (for example "cloudian12:19080" for an S3 Service instance running on host cloudian12, or "cloudian12:19081" for an Admin Service instance running on host cloudian12).

#### 10.5.11.1. Command Line Syntax

```
add client redis.credentials|redis.qos.<region> <host:JMXport>
```

Example:

```
# nc 10.50.20.12 9078
add client redis.credentials store7:19080
OK added client store7(10.50.20.7):19080 to redis
```

## 10.5.12. test dc partition

### test dc partition

If your HyperStore system includes multiple data centers (DCs), you can use this command to check whether or not there is a DC partition in the specified Redis cluster. A Redis cluster is considered to have a DC partition if the Redis Monitor -- from its location within one of the DCs -- cannot reach any of that cluster's Redis nodes or any of that cluster's Redis clients (S3, IAM, Admin, HyperStore) in one of the other DCs.

**Note** The Redis Monitor **automatically** checks for a DC partition once every five seconds, and if a partition is detected an alert is logged in *cloudian-redismon.log* on the Redis Monitor node and is displayed in the CMC's **Alerts** page. So under normal circumstances you should not need to manually trigger a DC partition check by using this command.

### 10.5.12.1. Command Line Syntax

```
test dc partition redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
test dc partition redis.credentials
OK
Redis cluster, DC: us-east-2a # of Nodes: 1 Failure nodes: 0
Client side, DC: us-east-2a # of Clients: 9 Failure Clients: 0
Redis cluster, DC: us-east-1a # of Nodes: 3 Failure nodes: 0
Client side, DC: us-east-1a # of Clients: 16 Failure Clients: 0
Redis cluster, DC: us-east-1b # of Nodes: 2 Failure nodes: 0
Client side, DC: us-east-1b # of Clients: 12 Failure Clients: 0
Has Partition: false
```

## 10.5.13. test split brain

### test split brain

You can use this command to check whether or not there is a "split brain" condition in the specified Redis cluster. A Redis cluster is consider to have a "split brain" if two different Redis nodes within the cluster have the master role at the same time. For proper cluster operation and metadata integrity, each Redis cluster should have just one master node at any given point in time. A split brain can occur, for instance, if the master role fails over from one node to another, and then the original master node comes back up and -- due to connectivity failures or some other system problem -- starts acting as master again rather than rejoining the cluster as a slave.

**Note** The Redis Monitor **automatically** checks for a "split brain" condition once every five seconds, and if a split brain condition is detected an alert is logged in *cloudian-redismon.log* on the Redis Monitor node and is displayed in the CMC's **Alerts** page. So under normal circumstances you should not need to manually trigger a split brain check by using this command.

#### 10.5.13.1. Command Line Syntax

```
test split brain redis.credentials|redis.qos.<region>
```

Example #1:

```
# nc 10.50.20.12 9078
test split brain redis.credentials
OK
Number of Master in cluster redis.credentials: 1
Has No Brain: false
Has Split Brain: false
```

Example #2:

```
# nc 10.50.20.12 9078
test split brain redis.credentials
OK
Number of Master in cluster redis.credentials: 2
Has No Brain: false
Has Split Brain: true
```

## 10.5.14. disable dc partition monitoring

### disable dc partition monitoring

This command affects how the Redis Monitor behaves **after it has detected a data center partition** in a Redis cluster. (For a description of how the Redis Monitor determines that a Redis cluster is in a DC partition condition, see **"test dc partition"** (page 591)).

If DC partition monitoring is **enabled**, then in the circumstance where DC partition has been detected and the Redis master node is in the unreachable DC, the Redis Monitor will continue with its normal master role monitoring and managing behavior by **promoting a reachable slave in a different DC to the master role** (in other words, failover of the master role will be executed).

If DC partition monitoring is **disabled**, then in the circumstance where DC partition has been detected and the Redis master node is in the unreachable DC, the Redis Monitor will discontinue its normal master role monitoring behavior and will **not promote a reachable slave in a different DC to the master role** (in other words, failover of the master role will **not** be executed). Once the unreachable DC becomes reachable again -- which will be detected by the Redis Monitor -- then the Redis Monitor will resume its normal monitoring behavior and will execute failover of the master role if the existing master node goes down.

**By default DC partition monitoring/failover is disabled**. This is controlled by the setting **"redis-.monitor.skip.dc.monitoring"** (page 460) in **mts.properties.erb** (which defaults to true, so that DC partition monitoring/failover is "skipped" [disabled]).

Since DC partition monitoring/failover is disabled by default, the only circumstances in which you might want to use the *disable dc partition monitoring* command is if you have previously changed the *redis-.monitor.skip.dc.monitoring* configuration property (so that DC partition monitoring/failover is enabled by configuration) or if you have previously used the **enable dc partition monitoring** command (so that DC partition monitoring/failover is enabled in the current session of the Redis Monitor).

**Note** If the Redis Monitor (or its host) is restarted, it will revert to using the value of the *redis-.monitor.skip.dc.monitoring* configuration property to determine whether DC partition monitoring/failover is enabled or disabled. Note also that the configuration property applies to all Redis clusters in the system, while the enable/disable commands apply only to the Redis cluster that you specify when you run the command.

**Note** If a Redis cluster DC partition occurs an alert will be written to *cloudian-redismon.log* on the Redis Monitor node and an alert will display in the CMC. You can also confirm that the condition exists by using the **test dc partition** command. For additional guidance on managing and recovering from a Redis cluster DC partition condition consult with Cloudian Support.

#### 10.5.14.1. Command Line Syntax

```
disable dc partition monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
disable dc partition monitoring redis.credentials
OK
Skip Monitoring when DC Partition detected
```

## 10.5.15. enable dc partition monitoring

### enable dc partition monitoring

This command affects how the Redis Monitor behaves **after it has detected a data center partition** in a Redis cluster. (For a description of how the Redis Monitor determines that a Redis cluster is in a DC partition condition, see **"test dc partition"** (page 591)).

If DC partition monitoring is **enabled**, then in the circumstance where DC partition has been detected and the Redis master node is in the unreachable DC, the Redis Monitor will continue with its normal master role monitoring and managing behavior by **promoting a reachable slave in a different DC to the master role** (in other words, failover of the master role will be executed).

If DC partition monitoring is **disabled**, then in the circumstance where DC partition has been detected and the Redis master node is in the unreachable DC, the Redis Monitor will discontinue its normal master role monitoring behavior and will **not promote a reachable slave in a different DC to the master role** (in other words, failover of the master role will **not** be executed). Once the unreachable DC becomes reachable again -- which will be detected by the Redis Monitor -- then the Redis Monitor will resume its normal monitoring behavior and will execute failover of the master role if the existing master node goes down.

**By default DC partition monitoring/failover is disabled**. This is controlled by the setting **"redis-.monitor.skip.dc.monitoring"** (page 460) in **mts.properties.erb** (which defaults to true, so that DC partition monitoring/failover is "skipped" [disabled]).

**Note** If the Redis Monitor (or its host) is restarted, it will revert to using the value of the *redis-.monitor.skip.dc.monitoring* configuration property to determine whether DC partition monitoring/failover is enabled or disabled. Note also that the configuration property applies to all Redis

clusters in the system, while the enable/disable commands apply only to the Redis cluster that you specify when you run the command.

**Note** If a Redis cluster DC partition occurs an alert will be written to *cloudian-redismon.log* on the Redis Monitor node and an alert will display in the CMC. You can also confirm that the condition exists by using the **test dc partition** command. For additional guidance on managing and recovering from a Redis cluster DC partition condition consult with Cloudian Support.

#### 10.5.15.1. Command Line Syntax

```
enable dc partition monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
enable dc partition monitoring redis.credentials
OK
Keep Monitoring when DC Partition detected
```

## 10.5.16. disable split brain monitoring

### disable split brain monitoring

This command affects how the Redis Monitor behaves **after it has detected a "split brain" condition** (two simultaneous masters) in a Redis cluster.

If split brain monitoring is **enabled**, then in the circumstance where split brain has been detected the Redis Monitor will continue with its normal master role monitoring and managing behavior by **demoting one of the masters to a slave role**. Of the two masters that constitute the "split brain", Redis Monitor will demote the one that most recently became a master. The node that had been master for a longer period of time will be left as the one master.

If split brain monitoring is **disabled**, then in the circumstance where split brain has been detected the Redis Monitor will discontinue its normal master role monitoring behavior and will **not automatically demote one of the masters to a slave role**. Instead it will be left to you to resolve the split brain condition by using the **resolve split brain** command (which will let you choose which node should remain as the one master).

**By default split brain monitoring/resolution is enabled**. This is controlled by the setting **"redis-.monitor.skip.brain.monitoring"** (page 460) in **mts.properties.erb** (which defaults to false, so that split brain monitoring/resolution is enabled [is not "skipped"]).

**Note** If the Redis Monitor (or its host) is restarted, it will revert to using the value of the *redis-.monitor.skip.brain.monitoring* configuration property to determine whether split brain monitoring/resolution is enabled or disabled. Note also that the configuration property applies to all Redis clusters in the system, while the enable/disable commands apply only to the Redis cluster that you specify when you run the command.

**Note** If a Redis cluster "split brain" condition occurs an alert will be written to *cloudian-redismon.log* on the Redis Monitor node and an alert will display in the CMC. For additional guidance on managing and recovering from a Redis cluster split brain condition consult with Cloudian Support.

### 10.5.16.1. Command Line Syntax

```
disable split brain monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
disable split brain monitoring redis.credentials
OK
Skip Monitoring when Split Brain detected
```

## 10.5.17. enable split brain monitoring

### enable split brain monitoring

This command affects how the Redis Monitor behaves **after it has detected a "split brain" condition** (two simultaneous masters) in a Redis cluster.

If split brain monitoring is **enabled**, then in the circumstance where split brain has been detected the Redis Monitor will continue with its normal master role monitoring and managing behavior by **demoting one of the masters to a slave role**. Of the two masters that constitute the "split brain", Redis Monitor will demote the one that most recently became a master. The node that had been master for a longer period of time will be left as the one master.

If split brain monitoring is **disabled**, then in the circumstance where split brain has been detected the Redis Monitor will discontinue its normal master role monitoring behavior and will **not automatically demote one of the masters to a slave role**. Instead it will be left to you to resolve the split brain condition by using the **resolve split brain** command (which will let you choose which node should remain as the one master).

**By default split brain monitoring/resolution is enabled**. This is controlled by the setting **"redis-.monitor.skip.brain.monitoring"** (page 460) in **mts.properties.erb** (which defaults to false, so that split brain monitoring/resolution is enabled [is not "skipped"]).

Since split brain monitoring/resolution is enabled by default, the only circumstances in which you might want to use the *enable split brain monitoring* command is if you have previously changed the *redis-.monitor.skip.brain.monitoring* configuration property (so that split brain monitoring/resolution is disabled by configuration) or if you have previously used the **disable split brain monitoring** command (so that split brain monitoring/resolution is disabled in the current session of the Redis Monitor).

**Note** If the Redis Monitor (or its host) is restarted, it will revert to using the value of the *redis-.monitor.skip.brain.monitoring* configuration property to determine whether split brain monitoring/resolution is enabled or disabled. Note also that the configuration property applies to all Redis clusters in the system, while the enable/disable commands apply only to the Redis cluster that you specify when you run the command.

**Note** If a Redis cluster "split brain" condition occurs an alert will be written to *cloudian-redismon.log* on the Redis Monitor node and an alert will display in the CMC. For additional guidance on managing and recovering from a Redis cluster split brain condition consult with Cloudian Support.

#### 10.5.17.1. Command Line Syntax

```
enable split brain monitoring redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.50.20.12 9078
enable split brain monitoring redis.credentials
OK
Keep Monitoring when Split Brain detected
```

## 10.5.18. resolve split brain

### resolve split brain

This command is applicable **only if you have previously disabled automatic split brain resolution** (either by having changed the *redis.monitor.skip.brain.monitoring* configuration property to "true" in *mts.properties.erb*, or by having run the **disable split brain monitoring** command).

If a Redis cluster is in a "split brain" condition and automatic split brain resolution is disabled, you can use the *resolve split brain* command to resolve the condition. When you run the command, it will show you how long each of the current masters has been acting as a master, and you will be prompted to choose one of the masters to continue as master. The other master will be demoted to slave.

**Note** If a Redis cluster "split brain" condition occurs an alert will be written to *cloudian-redismon.log* on the Redis Monitor node and an alert will display in the CMC. You can also confirm that the condition exists by using the **test split brain** command. For additional guidance on managing and recovering from a Redis cluster split brain condition consult with Cloudian Support.

#### 10.5.18.1. Command Line Syntax

```
resolve split brain redis.credentials|redis.qos.<region>
```

Example:

```
# nc 10.112.2.12 9078
resolve split brain redis.credentials
OK
1. ch-us-east-1-us-east-1a-2-251(10.112.2.251):6379
ch-us-east-1-us-east-1a-2-251(10.112.2.251):6379: Consecutive time being master: 2.51 min
2. ch-us-east-1-us-east-1b-2-33(10.112.2.33):6379
ch-us-east-1-us-east-1b-2-33(10.112.2.33):6379: Consecutive time being master: 8.73 min
Please select which redis instance as master:
1
Forced new Master: ch-us-east-1-us-east-1a-2-251(10.112.2.251):6379
```