CLOUDIAN®

Cloudian HyperStore Admin API Reference

Version 8.2

Contents

Chapter 1. Introduction 9

1.1. HyperStore Admin API Introduction 9

1.1.1. Admin API Behavior in Multi-Region Systems 9

1.1.2. Admin API Logging 10

1.2. Admin API Methods List 10

1.3. Common Status Codes and Headers 13

1.3.1. Common Response Status Codes 13

1.3.2. Common Request and Response Headers 14

1.4. cURL Examples 15

1.5. HTTP and HTTPS for Admin API Access 15

1.5.1. If Your Original HyperStore Install Was Older Than Version 6.0.2: 16

1.6. HTTP(S) Basic Authentication for Admin API Access 16

1.6.1. Checking the Admin API's Current HTTP(S) Basic Auth Password 16

1.6.2. Changing the Admin API's HTTP(S) Basic Auth Password 17

1.7. Role-Based Access to Admin API Operations 17

1.7.1. Comparing the Admin API to the IAM API with RBAC Extensions 17

1.7.2. Administrative Actions Supported by the IAM API 18

1.7.3. Giving Administrative Action Privileges to IAM Users 20

1.7.4. Using admin_client.py to Call the IAM Service Extensions for Administrative Actions 22

Chapter 2. allowlist 25

2.1. GET /allowlist 25

2.2. POST /allowlist 26

2.3. POST /allowlist/list 28

Chapter 3. billing 31

3.1. GET /billing 31

3.2. POST /billing 37

Chapter 4. bppolicy 43

4.1. GET /bppolicy/bucketsperpolicy 43

4.2. GET /bppolicy/listpolicy 44

Chapter 5. bucketops 47

5.1. GET /bucketops/cleanUpdatesInRange/status 47

5.2. GET /bucketops/id 48

5.3. GET /bucketops/gettags 49
5.4. POST /bucketops/cleanUpdatesInRange/go 51
5.5. POST /bucketops/cleanUpdatesInRange/stop 53
5.6. POST /bucketops/purge 54

Chapter 6. group 59

6.1. DELETE /group 59
6.2. GET /group 60
6.3. GET /group/list 62
6.4. GET /group/ratingPlanId 65
6.5. POST /group 66
6.6. POST /group/ratingPlanId 67
6.7. PUT /group 68

Chapter 7. monitor 75

7.1. DELETE /monitor/notificationrule 75
7.2. GET /monitor/events 76
7.3. GET /monitor/nodelist 80
7.4. GET /monitor/host 81
7.5. GET /monitor 93
7.6. GET /monitor/history 100
7.7. GET /monitor/notificationrules 102
7.8. POST /monitor/acknowledgeevents 103
7.9. POST /monitor/notificationruleenable 105
7.10. POST /monitor/notificationrule 106
7.11. PUT /monitor/notificationrule 107

Chapter 8. permissions 115

8.1. GET /permissions/publicUrl 115
8.2. POST /permissions/publicUrl 117

Chapter 9. qos 121

9.1. DELETE /qos/limits 121
9.2. GET /qos/limits 122
9.3. POST /qos/limits 127

Chapter 10. ratingPlan 131

10.1. DELETE /ratingPlan 131
10.2. GET /ratingPlan 132
10.3. GET /ratingPlan/list 134
10.4. POST /ratingPlan 135
10.5. PUT /ratingPlan 136

Chapter 11. system 141

11.1. GET /system/audit 141
11.2. GET /system/bucketcount 144
11.3. GET /system/bucketlist 145
11.4. GET /system/bucketusage 147
11.5. GET /system/bytecount 149
11.6. GET /system/bytestiered 151
11.7. GET /system/dcnodelist 152
11.8. GET /system/groupbytecount 152
11.9. GET /system/groupobjectcount 155
11.10. GET /system/license 157
11.11. GET system/objectcount 162
11.12. GET /system/objectlockenabled 164
11.13. GET /system/token/challenge 165
11.14. GET /system/version 166
11.15. POST /system/processProtectionPolicy 167
11.16. POST /system/repairusercount 168

Chapter 12. tiering 171

12.1. DELETE /tiering/credentials 171
12.2. DELETE /tiering/azure/credentials 172
12.3. DELETE /tiering/spectra/credentials 172
12.4. GET /tiering/credentials 173
12.5. GET /tiering/credentials/src 174
12.6. GET /tiering/azure/credentials 175
12.7. GET /tiering/spectra/credentials 176
12.8. POST /tiering/credentials 176
12.9. POST /tiering/azure/credentials 178
12.10. POST /tiering/spectra/credentials 179

Chapter 13. usage 181

13.1. DELETE /usage 181
13.2. GET /usage 183
13.3. POST /usage/bucket 199
13.4. POST /usage/repair 201
13.5. POST /usage/repair/bucket 202
13.6. POST /usage/repair/dirtyusers 203
13.7. POST /usage/repair/user 205
13.8. POST /usage/rollup 206
13.9. POST /usage/storage 207
13.10. POST /usage/storageall 208

Chapter 14. user 209

14.1. DELETE /user 210

14.2. DELETE /user/credentials 211
14.3. DELETE /user/deleted 212
14.4. DELETE /user/mfa/deleteDevice 213
14.5. GET /user 215
14.6. GET /user/credentials 217
14.7. GET /user/credentials/list 219
14.8. GET /user/credentials/list/active 222
14.9. GET /user/islocked 225
14.10. GET /user/list 226
14.11. GET /user/mfa/list 230
14.12. GET /user/password/verify 231
14.13. GET /user/ratingPlan 232
14.14. GET /user/ratingPlanId 235
14.15. POST /user 236
14.16. POST /user/credentials 237
14.17. POST /user/credentials/status 238
14.18. POST /user/mfa/createDevice 239
14.19. POST /user/mfa/deactivateDevice 241
14.20. POST /user/mfa/enableDevice 242
14.21. POST /user/mfa/resyncDevice 244
14.22. POST /user/mfa/verify 245
14.23. POST /user/password 247
14.24. POST /user/ratingPlanId 248
14.25. POST /user/unlock 249
14.26. PUT /user 251
14.27. PUT /user/credentials 256

Chapter 1. Introduction

1.1. HyperStore Admin API Introduction

Cloudian® HyperStore® provides a RESTful HTTP API through which you can provision users and groups, manage rating plans and quality of service (QoS) controls, retrieve monitoring data, and perform other administrative tasks. This Admin API is implemented by the HyperStore Admin Service.

By default the HTTPS listening port for the Admin Service is 19443 and the HTTP port is 18081. In HyperStore systems for which the first installed version was 6.0.2 or later, the Admin Service supports **only HTTPS** connections, and clients are required to use Basic Authentication. (For more detail see **"HTTP and HTTPS for Admin API Access"** (page 15) and **"HTTP(S) Basic Authentication for Admin API Access"** (page 16)).

The Cloudian Management Console (CMC) accesses the Admin API to implement its provisioning and reporting functions. You also have the option of accessing the Admin API directly, using a command line tool such as cURL or a REST client application of your own creation. For information about using cURL see **"cURL Examples"** (page 15).

HyperStore Admin API response payloads are JSON encoded. For POST or PUT requests that require a request payload, the request payloads must be JSON encoded as well.

IMPORTANT ! The Admin API is not designed to be exposed to end users of the Cloudian HyperStore storage service. It is intended to be accessed only within an internal network -- by the CMC and by system administrators using other types of clients (such as cURL). Do not expose the Admin Service to an external network.

1.1.1. Admin API Behavior in Multi-Region Systems

If your HyperStore system has multiple service regions, then:

- The Admin Service in the **default service region** supports executing **all** of the Admin API operations in this document.
- The Admin Service in **regions other than the default region** supports executing only the following subset of Admin API operations:
  - **POST /usage/storage**
  - **POST /usage/storageall**
  - **POST /usage/rollup**
  - **POST /usage/repair/dirtyusers**
  - **POST /bucketops/purge**

If in a non-default region you send your local Admin Service a request to execute an operation other than those listed above, you will receive a 403:Forbidden response.

Consequently, in a multi-region system your DNS configuration**must resolve the Admin Service endpoint to nodes in the default service region**. The CMC will use this endpoint to submit requests to the Admin API. And if you access the Admin API directly -- through a command line tool or a client application of your own creation

-- you must submit the requests to nodes in the default service region (with the exception of the calls listed above)

For API calls that involve retrieving data from multiple regions, this is all handled by the Admin Service in the default region. For example in a **GET /usage** call submitted to the Admin Service in your default service region you can retrieve service usage data for all of your regions or for any single one of your regions.

1.1.2. Admin API Logging

The Admin Service generates an application log and also a request log that records information about every request submitted to the Admin API. For detail about these logs see "Admin Service Logs" in the Logging section of the ***Cloudian HyperStore Administrator's Guide***.

1.2. Admin API Methods List

The table below shows all of the HyperStore Admin API methods. For more detail about a method or methods, click on the corresponding Resource link.

Resource	Method	Purpose
allowlist	GET /allowlist	Get allowlist content
	POST /allowlist	Change allowlist content (by request body object)
	POST /allowlist/list	Change allowlist content (by query parameters)
billing	GET /billing	Get a bill for a user or group
	POST /billing	Create a bill for a user or group
bppolicy	GET /bppolicy/bucketsperpolicy	Get list of buckets using each storage policy
	GET /bppolicy/listpolicy	Get list of storage policy names and IDs
bucketops	GET /bucketops/id	Get a bucket's canonical ID
	GET /bucketops/gettags	Get bucket tags for users in a group
	POST /bucketops/purge	Delete all the objects in a bucket
group	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
monitor	DELETE /monitor/notificationrule	Delete a notification rule
	GET /monitor/events	Get the event list for a node
	GET /monitor/nodelist	Get the list of monitored nodes
	GET /monitor/host	Get current monitoring statistics for a node
	GET /monitor	Get current monitoring statistics for a service region

Resource	Method	Purpose
	GET /monitor/history	Get historical monitoring statistics for a node
	GET /monitor/notificationrules	Get the list of notification rules
	POST /mon-itor/acknowledgeevents	Acknowledge monitoring events
	POST /mon-itor/notificationruleenable	Enable or disable notification rules
	POST /monitor/notificationrule	Change a notification rule
	PUT /monitor/notificationrule	Create a new notification rule
permissions	GET /permissions/publicUrl	Get public URL permissions for an object
	POST /permissions/publicUrl	Create or change public URL permissions for an object
qos	DELETE /qos/limits	Delete QoS settings for a user or group
	GET /qos/limits	Get QoS settings for a user or group
	POST /qos/limits	Create QoS settings for a user or group
ratingPlan	DELETE /ratingPlan	Delete a rating plan
	GET /ratingPlan	Get a rating plan
	GET /ratingPlan/list	Get the list of rating plans in the system
	POST /ratingPlan	Change a rating plan
	PUT /ratingPlan	Create a new rating plan
system	GET /system/audit	Get summary counts for system
	GET /system/bucketcount	Get count of buckets owned by a group's members
	GET /system/bucketusage	Get stored byte and object counts for each bucket owned by a group's members
	GET /system/bucketlist	Get list of buckets owned by a group's members
	GET /system/bytecount	Get stored byte count for the system, a group, or a user
	GET /system/bytestiered	Get tiered byte count for the system, a group, or a user
	GET /system/dcnodelist	Get list of data centers and nodes
	GET /system/groupbytecount	Get stored byte counts for all of a group's users
	GET /system/groupobjectcount	Get stored object counts for all of a group's users
	GET /system/license	Get HyperStore license terms
	GET system/objectcount	Get stored object count for the system, a group, or a user
	GET /system/objectlockenabled	Get Object Lock enabled/disabled status
	GET /system/token/challenge	Get token challenge to provide to Cloudian Support
	GET /system/version	Get HyperStore system version
	POST /sys-tem/processProtectionPolicy	Process pending storage policy deletion or creation jobs
	POST /system/repairusercount	Reconcile user counts in Redis and Cassandra
"tiering" (page 171)	DELETE /tiering/credentials	Delete a tiering credential for Amazon, Google, or other S3-compliant destination

Resource	Method	Purpose
	DELETE /tier-ing/azure/credentials	Delete a tiering credential for Azure
	DELETE /tier-ing/spectra/credentials	Delete a tiering credential for Spectra
	GET /tiering/credentials	Get a tiering credential for Amazon, Google, or other S3-compliant destination
	GET /tiering/credentials/src	Check whether a bucket uses a bucket-specific or system default tiering credential
	GET /tiering/azure/credentials	Get a tiering credential for Azure
	GET /tiering/spectra/credentials	Get a tiering credential for Spectra
	POST /tiering/credentials	Post a tiering credential for Amazon, Google, or other S3-compliant destination
	POST /tiering/azure/credentials	Post a tiering credential for Azure
	POST /tier-ing/spectra/credentials	Post a tiering credential for Spectra
usage	DELETE /usage	Delete usage data
	GET /usage	Get usage data for group, user, or bucket
	POST /usage/bucket	Get raw usage data for multiple buckets
	POST /usage/repair	Repair storage usage data for group or system
	POST /usage/repair/bucket	Retrieve total bytes and total objects for a bucket
	POST /usage/repair/dirtyusers	Repair storage usage data for users with recent activity
	POST /usage/repair/user	Repair storage usage data for a user
	POST /usage/rollup	Roll up usage data
	POST /usage/storage	Post raw storage usage data for users with recent activity
	POST /usage/storageall	Post raw storage usage data for all users

Resource	Method	Purpose
user	DELETE /user	Delete a user
	DELETE /user/credentials	Delete a user's S3 security credential
	DELETE /user/deleted	Purge profile data of a deleted user or users
	DELETE /user/mfa/deleteDevice	Delete an MFA device from a user's account
	GET /user	Get a user's profile
	GET /user/credentials	Get a user's S3 secret key corresponding to a supplied access key
	GET /user/credentials/list	Get a user's list of S3 security credentials
	GET /user/credentials/list/active	Get a user's list of active S3 security credentials
	GET /user/islocked	Get a user's lock-out status
	GET /user/list	Get a list of user profiles
	GET /user/mfa/list	Get a list of a user's MFA devices
	GET /user/password/verify	Verify a user's supplied password
	GET /user/ratingPlan	Get a user's rating plan content
	GET /user/ratingPlanId	Get a user's rating plan ID
	POST /user	Change a user's profile
	POST /user/credentials	Post a user's supplied S3 credential
	POST /user/credentials/status	Deactivate or reactivate a user's S3 credential
	POST /user/mfa/createDevice	Create a virtual MFA device for a user
	POST /user-/mfa/deactivateDevice	Deactivate a user's MFA device
	POST /user/mfa/enableDevice	Activate an MFA device for a user
	POST /user/mfa/resyncDevice	Resync a user's MFA device
	POST /user/mfa/verify	Verify a user's supplied MFA code
	POST /user/password	Create or change a user's password
	POST /user/ratingPlanId	Assign a rating plan to a user
	POST /user/unlock	Unlock a locked-out user
	PUT /user	Create a new user
	PUT /user/credentials	Create a new S3 credential for a user

1.3. Common Status Codes and Headers

1.3.1. Common Response Status Codes

The following HTTP Status Codes are common to most or all Admin API methods. Each method may return these codes, in addition to method-specific status codes indicated in the method documentation.

Status Code	Description
200	OK

Status Code	Description
401	Unauthorized
403	Not allowed because only the Admin API service in the default region can execute this method. The only Admin API methods that are allowed in non-default regions of a multi-region deployment are: ◦ POST /usage/storage ◦ POST /usage/storageall ◦ POST /usage/rollup ◦ POST /usage/repair/dirtyusers ◦ POST /bucketops/purge For any other Admin API method, submitting the method request to a non-default region's Admin Service will result in the 403 response.
404	Not found **Note** URIs for the Admin API are **case-sensitive**. If you submit a request wherein the case of the specified request resource and URI parameters does not match the case documented in this Admin API Guide — or a request wherein the URI contains any other typographical error — the system will return a 404 error response.
500	Internal Server Error
503	Service Unavailable Error

1.3.2. Common Request and Response Headers

1.3.2.1. Common Request Headers

For PUT requests, the Content-Type header should be set to "application/json". For POST requests, the Content-Type header should be set to "application/json", "application/x-www-form-urlencoded", or "multipart/form-data".

Depending on the request type and the result, the response from the Admin API may be in format application/json, text/html, or text/plain. So in your requests do not use an Accept header that excludes these content types.

1.3.2.2. Common Response Headers

In responses, the Content-Type will be either "application/json", "text/html", or "text/plain" depending on the type of request being processed and the result.

1.4. cURL Examples

This Admin API documentation includes examples using the open source command-line utility **cURL**. When a JSON object is the expected response payload, the example commands pipe the output through the standard Python tool *mjson.tool* so that the JSON pretty-prints. If you wish you can copy the commands from the documentation, paste them on to your command line, customize them appropriately, and the commands should work against your HyperStore Admin Service (so long as you have cURL and Python on your local machine).

Here is a sample command, for retrieving a user group's profile:

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/group?groupId=QA | python -mjson.tool
```

With this example you would:

- Replace *sysadmin* and *password* with whatever the Admin API HTTP(S) Basic Authentication user name and password are in your HyperStore system . For information about how to find the current Basic Auth user name and password -- and about the option of using user name and password variables in your cURL commands rather than the explicit user name and password -- see **"Checking the Admin API's Current HTTP(S) Basic Auth Password"** (page 16).
- Replace *localhost* with the IP address of one of your HyperStore nodes in the default service region (unless you're running cURL from a HyperStore node in the default region, in which case you can leave it as *localhost*).
- Replace *QA* with the name of one of your user groups.

Note that the backslash in this and other examples indicates line continuation -- telling the Linux shell to ignore the newline for purposes of running the command. These are used in the examples so that a long command can be split to multiple lines in this documentation, while still allowing you to copy all the text (including the backslash) and paste it on your command line and be able to run the command.

Note By default the Admin Server uses a self-signed SSL certificate and so in the example cURL commands the "-k" flag is used to disable certificate checking.

In cases where a JSON object is required as the request payload, the examples use the cURL "-d" flag to reference the name of a text file that contains the JSON object. For example:

```
curl -X PUT -H "Content-Type: application/json" -k -u sysadmin:password \
-d @group_QA.txt https://localhost:19443/group
```

In the full documentation the content of the referenced text file is also shown.

Note that **if a request includes multiple query parameters with ampersand delimitation, the URL must be enclosed in single quotes** -- as in this example which deletes a user:

```
curl -X DELETE -k -u sysadmin:password \
'https://localhost:19443/user?userId=John&groupId=QA'
```

1.5. HTTP and HTTPS for Admin API Access

The Admin Service by default requires clients to use HTTPS and rejects attempts to connect with regular HTTP (unless your original HyperStore system was older than version 6.0.2; see the next section below). For its HTTPS implementation the Admin Service by default uses a self-signed certificate that is generated during

HyperStore installation. For information about managing SSL certificates in HyperStore, in the Cloudian HyperStore Administrator's Guide see "HTTPS Feature Overview".

The Admin Service requires Basic Authentication from connecting HTTP(S) clients. For more information including how to customize the required Basic Authentication password see **"HTTP(S) Basic Authentication for Admin API Access"** (page 16).

1.5.1. If Your Original HyperStore Install Was Older Than Version 6.0.2:

If your original HyperStore install was older than version 6.0.2, the Admin Service by default accepts regular HTTP requests (through port 18081) as well as HTTPS requests (through port 19443). Also for such systems, the CMC uses regular HTTP when submitting requests to the Admin Service.

If you want the Admin Service to accept **only HTTPS** requests from clients -- and to reject regular HTTP requests -- follow the steps below. Following these steps also has the effect of reconfiguring the CMC so that it uses exclusively HTTPS when submitting requests to the Admin Service.

1. On the Config Controller node, make this setting change:

   ```
   # hsctl config set admin.tls.required true
   ```

2. Apply your change and restart the S3 Service and the CMC Service:

   ```
   # hsctl config apply admin
   # hsctl service restart s3 cmc --nodes=ALL
   ```

1.6. HTTP(S) Basic Authentication for Admin API Access

The Admin Service requires that clients use HTTP(S) Basic Authentication credentials (user name and password) when connecting to the service. By default the required user name for this purpose is "sysadmin" and the default password is either a randomly generated password unique to your system (if your original HyperStore install was version 7.2.2 or newer) or "public" (if your original HyperStore install was older than version 7.2.2). **If the Admin API HTTP(S) Basic Authentication password in your system is "public", you should change it** to something more secure. Even if the password is a random one generated upon system install, you may still wish to change it to a password of your own creation.

1.6.1. Checking the Admin API's Current HTTP(S) Basic Auth Password

To run Admin API calls you will need to supply the Admin API's current HTTP(S) Basic Authentication user name and password. On the Config Controller node you can check to see what the current user name and password are with these *hsctl* commands:

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

```
# hsctl config get admin.auth.password
n5OiJP0d#5VOl9qJJTBn
```

Note that the current password value will only display if you are the root user.

If you are running cURL from a HyperStore node, if you wish you can use the *hsctl* commands above as variables within a call to the Admin API, and the current Basic Auth user name and password will be automatically retrieved. The example below executes the Admin API's *GET /system/version* call.

```
curl -X GET -k -u $(hsctl config get admin.auth.username):$(hsctl config get admin.auth.password) \
https://localhost:19443/system/version
8.1.1 Compiled: 2024-08-16 16:32
```

1.6.2. Changing the Admin API's HTTP(S) Basic Auth Password

To change the HTTP(S) Basic Authentication password for the Admin API, on the Config Controller node run these *hsctl* commands:

```
# hsctl config set admin.auth.password=<string>
# hsctl config apply admin
# hsctl service restart s3 --nodes=ALL
```

1.7. Role-Based Access to Admin API Operations

Subjects covered in this section:

- *Introduction (immediately below)*
- ***"Comparing the Admin API to the IAM API with RBAC Extensions"*** *(page 17)*
- ***"Administrative Actions Supported by the IAM API"*** *(page 18)*
- ***"Giving Administrative Action Privileges to IAM Users"*** *(page 20)*
- ***"Using admin_client.py to Call the IAM Service Extensions for Administrative Actions"*** *(page 22)*

The **HyperStore IAM Service** supports extensions to the IAM API that allow for role-based access control (RBAC) for read-only HyperStore Admin API operations. The IAM API extensions take the form of additions to the list of valid values that can be specified by the "Action" request parameter in a request to the HyperStore IAM Service. The supported Actions vary by the role of the requester: the IAM Service allows a HyperStore system administrator to execute a wider range of Actions than can a group administrator or a regular user.

Note For general information on HyperStore's support for the AWS Identity & Access Management (IAM) API, see the IAM section of the ***Cloudian HyperStore AWS APIs Support Reference***.

1.7.1. Comparing the Admin API to the IAM API with RBAC Extensions

The table below compares the HyperStore Admin API to the HyperStore IAM API with its extensions for admin actions.

Admin API	IAM API with RBAC Extensions for Admin Actions
Implemented by the HyperStore Admin Service • Runs on each node in each of your service regions (but with limited functionality in regions other than the default region) • Listens on ports 19443 (HTTPS) and 18081 (HTTP, optional) • Includes a bundled self-signed certificate for HTTPS • Request authentication is by HTTP Basic Authentication • Should only be exposed to internal traffic, not user traffic	Implemented by the HyperStore IAM Service • Runs on each node in your default region only • Listens on ports 16443 (HTTPS) and 16080 (HTTP) • Includes a bundled self-signed certificate for HTTPS • Request authentication is by

Admin API	IAM API with RBAC Extensions for Admin Actions
• Makes no distinctions based on role of the requester -- all access is system administrator level access	Amazon-compliant Signature v2 or v4 • Can be exposed to user traffic • Makes distinctions based on the role of the requester -- system administrators have a greater permissions scope than group administrators, who have a greater permission scope than regular users (role-based access control)
Proprietary RESTful API • GET, PUT, POST, and DELETE are all supported, and are different operations with different consequences • Request parameters are in lower camel case -- for example "canonicalUserId" and "billingPeriod" • Response bodies are JSON formatted	Compliant with Amazon's IAM API • Only GET and POST are supported and it doesn't matter which you use (what matters is the "Action" parameter) • Request parameters are in upper camel case (Pascal case) -- for example "CanonicalUserId" and "BillingPeriod" • Response bodies are XML formatted
Wide range of administrative tasks The Admin API supports more than 80 different methods for retrieving information about or making changes to the system	Narrow range of administrative tasks The IAM API extensions currently support only 16 administrative actions and these are all **read-only** (none of the supported actions make changes to the system)

1.7.2. Administrative Actions Supported by the IAM API

The table below lists the administrative Actions supported by the HyperStore IAM Service, and how the IAM Service restricts the use and implementation of these Actions according to the role (user account type) of the requester.

Also as shown by the table, each administrative Action supported by the IAM Service corresponds to an existing method in the Admin API -- in the sense that the IAM Action supports the same request parameters as the corresponding Admin API method (except the IAM version uses upper camel case for parameter names rather than lower camel case) and returns the same response body elements as the corresponding Admin API method (except the IAM version uses XML formatting for the response body rather than JSON). Consequently, for details about the request parameters and response body associated with a particular administrative IAM Action you can check the documentation of the corresponding Admin API method.

IAM Action	Permission Scope Based On Requester's Role			Corresponding Admin API Method
	System Admin	Group Admin	Regular User	
GetCloudianBill	Get any	Get bill of	Get own	GET /billing

IAM Action	Permission Scope Based On Requester's Role			Corresponding Admin API Method
	System Admin	Group Admin	Regular User	
(see Important note below table)	user's bill	any user in own group	bill	
GetCloudianGroup	Get any group's profile	Get own group's profile	Not allowed	GET /group
GetCloudianGroupList	Get list of groups	Not allowed	Not allowed	GET /group/list
GetCloudianMonitorEvents	Get event list for a node	Not allowed	Not allowed	GET /monitor/events
GetCloudianMonitorNodelist	Get list of monitored nodes	Not allowed	Not allowed	GET /monitor/nodelist
GetCloudianMonitorHost	Get monitoring stats for a node	Not allowed	Not allowed	GET /monitor/host
GetCloudianMonitorRegion	Get monitoring stats for a region	Not allowed	Not allowed	GET /monitor
GetCloudianQosLimits	Get QoS limits for any group or user	Get QoS limits for own group or users in own group	Get own QoS limits	GET /qos/limits
GetCloudianSystemLicense	Get system license info	Not allowed	Not allowed	GET /system/license
GetCloudianSystemVersion	Get current system version	Get current system version	Get current system version	GET /system/version
GetCloudianUsage	Get usage info for any group or user	Get usage info for own group or users in own group	Get own usage info	GET /usage
GetCloudianUser	Get any user's profile	Get profile of any user in own group	Get own profile	GET /user
GetCloudianUserCredentials	Get any user's S3 credential	Get S3 credential of any user in own group	Get own S3 credential	GET /user/credentials

IAM Action	Permission Scope Based On Requester's Role			Corresponding Admin API Method
	System Admin	Group Admin	Regular User	
GetCloudianUserCredentialsList	Get any user's S3 credentials list	Get S3 credentials list of any user in own group	Get own S3 credentials list	GET /user/credentials/list
GetCloudianUserCredentialsListActive	Get any user's active S3 credentials list	Get active S3 credentials list of any user in own group	Get own active S3 credentials list	GET /user/credentials/list/active
GetCloudianUserList	Get list of users in any group	Get list of users in own group	Not allowed	GET /user/list

IMPORTANT ! Before the "GetCloudianBill" Action can be used to retrieve billing data for a specified user and billing period, you must either execute the Admin API method POST /billing to generate billing data for that user and billing period, or else use the CMC's **Account Activity** page (**Users & Groups -> Account Activity**) to generate billing data for that user and billing period. There is currently no RBAC version of the *POST /billing* call that generates user billing data.

1.7.3. Giving Administrative Action Privileges to IAM Users

Just as HyperStore users can use IAM policies to grant S3 action permissions to their IAM users, so too can HyperStore users use IAM policies to grant HyperStore admin permissions to their IAM users. A typical use case would be if a HyperStore system administrator wanted to create an IAM user who is allowed to perform some of the system admin Actions but not all of them.

Note IAM users created by the default system administrative user -- the user named "admin" -- cannot create buckets or perform other S3 operations. Such IAM users can only perform role-based administrative operations, if granted permissions by the "admin" user.

At a high level this feature works as follows:

- As is the case with S3 permissions, an **IAM user by default has no admin permissions** -- an IAM user gains permissions only if she is assigned an IAM policy that specifies those permissions, and she gains only the permissions specified in the policy.
- When an IAM policy grants an IAM user permission to an administrative action, the IAM user's permission scope in respect to that action is the **same as her parent HyperStore user's permission scope** (as identified in the table above). For example:
  - If an IAM user is granted permission to the "GetCloudianGroup" action and her parent HyperStore user is a system administrator, the IAM user can get any HyperStore group's profile.

- If an IAM user is granted permission to the "GetCloudianGroup" action and her parent Hyper-Store user is a group administrator, the IAM user can (only) get that HyperStore group's profile.
- If an IAM user is granted permission to the "GetCloudianGroup" action and her parent Hyper-Store user is a regular user, the IAM Service will reject the IAM user's attempt to get any group profile. The IAM Service **will not allow an IAM user to execute an administrative action that her parent HyperStore user is not allowed to execute**.

Note When a HyperStore regular user grants his IAM users administrative action permissions that are allowed to a regular user -- such as "GetCloudianUsage" or "GetCloudianQosLimits" -- this gives the IAM users permission to perform those actions in regard to the **parent user's account**. For example an IAM user granted permission to the "GetCloudianUsage" action would be able to get usage information for the parent user account; and if granted permission to "GetCloudianQosLimits" would be able to get the QoS limits associated with the parent user account. HyperStore does not track usage, billing, or QoS information specifically for IAM users. This information is only tracked for the parent HyperStore user accounts.

When specified as an "Action" in an IAM policy document, all HyperStore administrative actions are pre-fixed by "admin: " (analogous to how S3 actions are prefixed by "s3: ") -- for example "admin:GetCloud-ianGroup" or "admin:GetCloudianMonitorEvents".

Below is an example of a simple IAM policy document for HyperStore administrative permissions:

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":"admin:GetCloudianUserList",
    "Resource":"*"
    }
  ]
}
```

Note You must include the "Resource" element and set it to "*". This is because Resource is a required element in IAM policy document syntax.

For more information on using IAM polices to grant permissions to IAM users, see:

- The IAM section of the ***Cloudian HyperStore AWS APIs Support Reference*** (for the HyperStore IAM Service's support of policy document elements)
- The online Help for the CMC's **Manage IAM Policies** page (for the CMC's support for creating IAM policies)

Note The CMC provides two tools for creating IAM policies -- a Visual Editor and a JSON Editor. The Visual Editor does not support creating a policy that contains HyperStore admin per-missions, but the JSON Editor does.

1.7.4. Using *admin_client.py* to Call the IAM Service Extensions for Administrative Actions

In the current HyperStore release, the CMC's built-in IAM client does not support calling the IAM extensions for HyperStore administrative Actions. To perform administrative Actions through the HyperStore IAM Service you can either:

- Use a third party tool that you customize to be able to support the HyperStore admin Action strings (as listed in the table above) and their associated request parameters (detailed in the corresponding Admin API links provided for each action in the table).
- Use the Python tool *admin_client.py* that comes bundled with HyperStore version 7.1 and later, as described below.

If you are using the HyperStore Shell

The HyperStore Shell (HSH) does not support using the *admin_client.py* tool.

Note To use the *admin_client.py* tool you will need to supply the tool with S3 access credentials. These can be your own credentials or those of an IAM user to whom you have granted administrative action permissions in an IAM policy.

HyperStore includes an interactive tool (written in Python) that makes it easy to call the administrative Actions that the HyperStore IAM Service supports. The tool is located in the following directory on each HyperStore node:

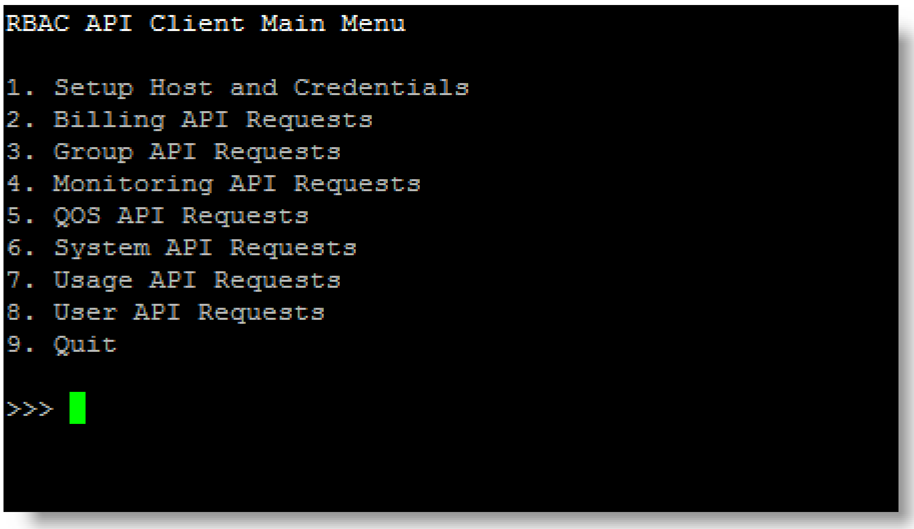
```
/opt/cloudian/tools
```

To launch the tool:

```
# ./admin_client.py
```

The first time that the tool is launched on a node, the tool automatically downloads and installs the required Python packages if they are not already present on the node (this requires outbound internet access from the node, in order for the tool to download the packages).

Once this completes, the tool's main menu displays:



```
RBAC API Client Main Menu

1. Setup Host and Credentials
2. Billing API Requests
3. Group API Requests
4. Monitoring API Requests
5. QOS API Requests
6. System API Requests
7. Usage API Requests
8. User API Requests
9. Quit

>>>
```

Use option 1 to supply the tool with your S3 access credentials and to specify the target HyperStore host information (you can connect to any HyperStore host) and the service region in which the host resides.

You can then perform admin Actions by choosing from the menus. For each request type the tool will prompt you to provide the needed parameter values (if any). The request response will display in the tool interface, in XML format.

It may be helpful to have the HyperStore Help open as you use the tool -- specifically the Admin API section of the Help. If needed you can check the documentation for the corresponding Admin API call as you provide the information required for a given request type. For example if you're using the tool to call the "GetCloudianBill" request and the tool prompts you for the "BillingPeriod", and you're not sure of the proper format for billing period -- you can check the Help for *GET /bill* to get this information. See **"Administrative Actions Supported by the IAM API"** (page 18) to see which Admin API methods correspond to the administrative Actions that the IAM Service supports.

Note Although the CMC's built-in IAM client does not currently support calling the admin Actions, the CMC does support creating an IAM user, assigning that user to an IAM group, and creating an inline policy for that group which includes admin Action permissions. The IAM user will then have those admin Action permissions. However, the IAM user will not be able to execute those admin Actions through the CMC -- he or she would need to use the Python tool, or a third party IAM client that's been customized to support the IAM extensions.

This page left intentionally blank

Chapter 2. allowlist

The Admin API methods built around the **allowlist** resource are for managing a billing "allowlist" of source IP addresses or subnets that you want to allow to have free S3 traffic with the HyperStore storage service. For background information on the allowlist feature, including how to enable the feature, see "Creating an 'Allowlist' for Free Traffic" in the ***Cloudian HyperStore Administrator's Guide***. The allowlist feature is disabled by default.

Note Prior to HyperStore version 7.4, this resource was called "whitelist". The *GET /whitelist*, *POST /whitelist*, and *POST whitelist/list* API calls have been deprecated but will still work, if you already have applications making those calls.

Note If you are using load balancers in front of the HyperStore S3 Service, the allowlist feature will only work if you use PROXY Protocol between the load balancers and the S3 Service. This protocol allows the load balancers to pass the IP addresses of originating clients to the S3 Service along with the S3 requests. For more information about enabling PROXY Protocol support on the S3 Service side, see the configuration setting *s3.enable.proxyProtocol* in the ***Cloudian HyperStore Administrator's Guide***. For guidance on configuring the load balancers consult with Cloudian Sales Engineering or Support.

Note that using the "X-Forwarded-For" HTTP header is **not** sufficient to support the allowlist feature. You must use PROXY Protocol if you have load balancers in front of the S3 Service and want to use the allowlist feature .

Methods associated with the *allowlist* resource:

- **"GET /allowlist"** (page 25)
- **"POST /allowlist"** (page 26)
- **"POST /allowlist/list"** (page 28)

2.1. GET /allowlist

GET /allowlist   Get allowlist content

2.1.1. Syntax

```
GET /allowlist?allowlistId=Default-WL
```

There is no request payload.

Note In the current version of HyperStore, only one allowlist is supported and its ID is "Default-WL".

2.1.2. Example Using cURL

The **example** below retrieves the current contents of the allowlist with ID "Default-WL".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/allowlist?allowlistId=Default-WL | python -mjson.tool
```

The response payload is a JSON-formatted *Allowlist* object, which in this example is as follows.

```
{
  "id": "Default-WL",
  "list": [
    "10.20.2.10",
    "10.20.2.11",
    "10.20.2.12"
  ],
  "name": "Default Allowlist",
  "ratingPlanId": "Whitelist-RP"
}
```

Note By default the "Default-WL" allowlist that comes with your HyperStore system is empty. The allowlist in the example above has had some IP addresses added to it.

2.1.3. Response Element Descriptions

For *Allowlist* object element descriptions see **"POST /allowlist Change allowlist content (by request body object)"** (page 26).

2.1.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Allowlist does not exist
400	Missing required parameter : allowlistId

2.2. POST /allowlist

POST /allowlist Change allowlist content (by request body object)

2.2.1. Syntax

```
POST /allowlist
```

The required request payload is a JSON-formatted *Allowlist* object. See example below.

2.2.2. Usage Notes

For billing purposes, changes that you make to the composition of the allowlist (by adding or deleting IP addresses or subnets) will take effect starting with the next hourly roll-up of HyperStore usage data.

2.2.3. Example Using cURL

The **example** below uploads allowlist content as specified in the request body. In this example the JSON-formatted *Allowlist* object is specified in a text file named *default_allowlist.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @default_allowlist.txt https://localhost:19443/allowlist
```

The *default_allowlist.txt* file content in this example is as follows.

```
{
  "id": "Default-WL",
  "list": ["10.20.2.10","10.20.2.11","10.20.2.12"],
  "name": "Default Allowlist",
  "ratingPlanId": "Whitelist-RP"
}
```

2.2.4. Request Element Descriptions

id

(Mandatory, string) Unique ID of the allowlist.

In the current HyperStore release only one allowlist is supported and its non-editable ID is "Default-WL".

Example:

```
"id": "Default-WL"
```

list

(Mandatory, list<string>) JSON array of source IP addresses and/or subnets.

To indicate an empty list, use an empty JSON array.

Example:

```
"list": ["10.20.2.10","10.20.2.11","10.20.2.12"]
```

Note IP addresses can be IPv4 or IPv6 format. For subnets, only IPv4 format is supported in the current HyperStore release. IP addresses are validated for IPv4 or IPv6 syntax, and subnets are validated for CIDR syntax.

name

(Mandatory, string) Display name of the allowlist. The default allowlist object has display name "Default Allowlist". This is editable.

Example:

```
"name": "Default Allowlist"
```

ratingPlanId

(Mandatory, string) Unique ID of the rating plan assigned to the allowlist. The default allowlist object is assigned rating plan "Whitelist-RP". This system-provided default allowlist rating plan makes all inbound and outbound traffic free of charge. (By contrast, data storage continues to be priced according to the user's regular assigned rating plan.) You can edit the "ratingPlanId" to associate a different rating plan with the default allowlist. You can also edit the "Whitelist-RP" rating plan, using the usual rating plan APIs.

Example:

```
"ratingPlanId": "Whitelist-RP"
```

2.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Allowlist does not exist
400	Missing required attributes : {id, name, ratingPlanId}
400	Invalid JSON object
400	Invalid IP Address or IPv4 Subnet CIDR: <value>

2.3. POST /allowlist/list

POST /allowlist/list Change allowlist content (by query parameters)

2.3.1. Syntax

```
POST /allowlist/list?allowlistId=string&list=string
```

There is no request payload.

2.3.2. Parameter Descriptions

allowlistId

(Mandatory, string) Unique identifier of the allowlist. In the current HyperStore release, only one allowlist is supported and its ID is "Default-WL".

list

(Mandatory, string) With a *POST /allowlist/list* request: A comma-separated list of IP addresses or subnets. This list will **overwrite** the existing allowlist contents, so be sure to specify your full desired list of addresses or subnets (not just new additions). IP addresses can be IPv4 or IPv6 format. For subnets, only IPv4 format is supported in the current HyperStore release. IP addresses are validated for IPv4 or IPv6 syntax, and subnets are validated for CIDR syntax.

2.3.3. Usage Notes

For billing purposes, changes that you make to the composition of the allowlist (by adding or deleting IP addresses or subnets) will take effect starting with the next hourly roll-up of HyperStore usage data.

2.3.4. Example Using cURL

The **example** below replaces the existing allowlist contents with a new list of IP addresses.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/allowlist/list?allowlistId=Default-WL&list=10.20.2.10,10.20.2.11'
```

2.3.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Allowlist does not exist
400	Missing required attributes : {allowlistId, list}
400	Invalid IP Address or IPv4 Subnet CIDR: {value}

Chapter 3. billing

The Admin API methods built around the **billing** resource are for generating or retrieving a billable activity report for a specified user or group. The report shows the user or group's billable activity and the charges for that activity based on the assigned **rating plan**.

For an overview of the HyperStore billing feature, see "Billing Feature Overview" in the ***Cloudian HyperStore Administrator's Guide***.

Methods associated with the *billing* resource:

- **"GET /billing"** (page 31)
- **"POST /billing"** (page 37)

3.1. GET /billing

GET /billing Get a bill for a user or group

3.1.1. Syntax

```
GET /billing?[userId=string&][groupId=string][canonicalUserId=string]&billingPeriod=string
```

There is no request payload.

3.1.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Optional, strings) Identifiers of the user or group for which to retrieve a bill.

- To retrieve a bill for a **user who currently is part of the service**, you can either use the "userId" parameter in combination with the "groupId" parameter (for example *user-Id=martinez&groupId=operations*), or use the "canonicalUserId" parameter by itself (with no "groupId" parameter).
- To retrieve a bill for a **user who has been deleted from the service**, you must use the "canon-icalUserId" parameter by itself (not the "userId" or "groupId" parameter).

  Note If you don't know the user's system-generated canonical ID, you can obtain it by using the *GET /user/list* method.

- To retrieve a bill for a **whole user group**, use the "groupId" parameter by itself (not the "userId" or "canonicalUserId" parameter).

billingPeriod

(Mandatory, string) Specifies the year and month of bill. Format is *yyyyMM* — for example "202407" for July 2024. Note that the system uses GMT time when demarcating exactly when a month begins and ends.

3.1.3. Usage Notes

This method retrieves an **existing** bill for a user or group (a bill that has already been generated by the **POST /billing** method.) You must use the **POST /billing** method for the user or group and billing period of interest before you can use this method.

3.1.4. Example Using cURL

The **example** below generates a billable activity report for the user "glad" from the "eng" group, for the month of July 2024. This is an existing billable activity report that has previously been generated by the **POST /billing** method.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/billing?userId=glad&groupId=eng&billingPeriod=202407' \
| python -mjson.tool
```

The response payload is a JSON-formatted *Bill* object, which in this example is as follows.

```
{
  "billID": "936265a2-fbd5-47c2-82ed-d62298299a1b",
  "canonicalUserId": "d47151635ba8d94efe981b24db00c07e",
  "currency": "USD",
  "endCal": 1659326400000,
  "groupId": "eng",
  "notes": null,
  "regionBills": [
    {
      "currency": "USD",
      "items": {
        "SB": {
          "item":"SB",
          "quantity":108.00,
          "rules":"1,0.14:5,0.12:0,0.10",
          "subtotal":10.94
        }
      },
      "region": "taoyuan",
      "total": 10.94,
      "whitelistItems": {},
      "whitelistTotal": 0
    }
  ],
  "startCal": 1656648000000,
  "total": 10.94,
  "userId": "glad",
  "whitelistTotal": 0
}
```

3.1.5. Response Element Descriptions

billID

(String) System-generated globally unique bill ID. Example:

```
"billID": "936265a2-fbd5-47c2-82ed-d62298299a1b"
```

canonicalUserId

(String) System-generated canonical user ID for the user. Empty if the bill is for a whole group. Example:

```
"canonicalUserId": "d47151635ba8d94efe981b24db00c07e"
```

currency

(String) Currency string. Example:

```
"currency": "USD"
```

endCal

(String) End date/time of the billing period in UTC milliseconds. Example:

```
"endCal": 1659326400000
```

groupId

(String) ID of the group to which the user belongs (or of the group for which the bill was generated, in the case of a whole group bill). Example:

```
"groupId": "eng"
```

notes

(String) Notes regarding the bill, if any. Example:

```
"notes": null
```

regionBills

(Map<string,*RegionBill*>) List of *RegionBill* objects, with one such object per service region. The *RegionBill* object consists of the following attributes and nested objects:

currency

(String) Currency string. Example:

```
"currency": "USD"
```

items

(Map<string,*BillItem*>) List of *BillItem* objects, with one such object for each activity type that’s being charged for, per the terms of the user’s rating plan. Supported activity types are "SB" (storage bytes), "BI" (bytes in), "BO" (bytes out), "HG" (HTTP Gets), "HP" (HTTP Puts), "HD" (HTTP Deletes). This list excludes activity for whitelisted IP addresses. Note that some or even most activity types may not appear, depending on the rating plan terms. For example, it may be that only storage bytes ("SB") are billed for, if that’s how the user’s rating plan is configured.

In the items list, each *BillItem* object is preceded by its activity type string, such as "SB": {*BillItem* data}. In the example only storage bytes ("SB") are charged for in the rating plan that was applied when this bill was generated.

The *BillItem* object consists of the following attributes:

item

(String) Usage type being billed for. Types are "SB" (storage bytes), "BI" (bytes in), "BO" (bytes out), "HG" (HTTP GETs), "HP" (HTTP PUTs), "HD" (HTTP DELETEs). Example:

```
"item":"SB"
```

quantity

(Number) Usage quantity during billing period. Usage quantity metrics depend on the usage type:

- For storage bytes (SB), the metric is GiB-Month (average number of GiBs of data stored for the billing month). This is calculated by summing the month’s hourly readings of stored bytes, converting to GiB, then dividing by the number of hours in the month. In the example above the usage quantity during the billing period was 108 GiB-months (that is, the user’s storage bytes volume average 108GiBs over the course of the month)
- For data transfer bytes in (BI) or out (BO), the metric is number of bytes.
- For HTTP GETs (HG), PUTs (HP), or DELETEs (HD), the metric is number of multiples of 10,000 requests. For example, if usage type is HG and quantity is 7.50, that means 75,000 HTTP GET requests.

Example:

```
"quantity":108.00
```

rules

(String) Specification of billing rules for this usage type (as configured in the user’s assigned rating plan). In the example the "rules" attribute indicates that the user’s rating plan is such that the first 1 GiB-month is charged at $0.14, the next 5 GiB-months is charged at $0.12 per GiB-month, and all GiB-months above that are charged at $0.10 per GiB-month.

Example:

```
"rules":"1,0.14:5,0.12:0,0.10"
```

subtotal

(Number) Total billing charge for the particular usage type specified by the "item" attribute. This will be in units of the currency specified by the "currency" attribute of the *RegionBill*object that contains this *BillItem* object. It’s labeled as "subtotal" because it will be added together with subtotals for other usage types (from other*BillItem*object instances within the*RegionBill*object, if any) to compute the "total" attribute for the encompassing *RegionBill* instance. In the example the $10.94 sub-total comes from applying the billing rules to the 108 GiB-months usage quantity ([1 X .14] + [5 X .12] + [102 X .10] = 10.94).

Example:

```
"subtotal":10.94
```

region

(String) Region name. Example:

```
"region": "taoyuan"
```

total

(Number) For the region, the total charges incurred — excluding activity originating from whitelisted source IP addresses. Example:

```
"total": 10.94
```

whitelistItems

(Map<string,*BillItem*>) List of *BillItem*objects, for activity originating from whitelisted IP addresses (if any). Types are "BI" (bytes in), "BO" (bytes out), "HG" (HTTP Gets), "HP" (HTTP Puts), "HD" (HTTP Deletes). Example:

```
"whitelistItems": {}
```

whitelistTotal

(Number) For the region, the total charges incurred for activity originating from whitelisted source IP addresses. Typically there are no charges for such activity. Example:

```
"whitelistTotal": 0
```

startCal

(String) Start date/time of the billing period in UTC milliseconds. Example:

```
"startCal": 1656648000000
```

total

(Number) The total charges incurred by the user during the billing period, excluding activity for whitelisted source IP addresses. Example:

```
"total": 10.94
```

userId

(String) ID of the user for whom the bill was generated. Empty if the bill is for a whole group. Example:

```
"userId": "glad"
```

whitelistTotal

(Number) The total charges incurred by the user for activity originating from whitelisted source IP addresses. Example:

```
"whitelistTotal": 0
```

3.1.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Billing data does not exist
400	User does not exist
400	Missing required parameter : {billingPeriod}
400	Conflicting parameters: {canonicalUserId, groupId, userId}

3.1.7. RBAC Version of this Method

IMPORTANT ! Before the RBAC version of this method can be used to retrieve billing data for a specified user and billing period, you must either execute the Admin API method **POST /billing** to generate billing data for that user and billing period, or else use the CMC's **Account Activity** page (**Users & Groups -> Account Activity**) to generate billing data for that user and billing period. There is currently no RBAC version of the *POST /billing* call that generates user billing data.

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianBill*
- Parameters: Same as for *GET /billing*, except:
  - *userId* and *groupId* are not supported. A user can only be specified by canonical ID, and retrieving a bill for a whole group is not supported.
  - All parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /billing* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can get a bill for any user
  - HyperStore group admin user can only get bills for users within her group
  - HyperStore regular user can only get own bill
  - IAM user can only use this method if granted *admin:GetCloudianBill* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

  Note The "GetCloudianBill" action retrieves billing data for Cloudian HyperStore user accounts, not for subsidiary IAM users. The system does not maintain billing data per IAM user. For example, if a HyperStore group administrator grants *admin:GetCloudianBill* permission to an IAM user, the IAM user will be able to retrieve billing information for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianBill* permission to an IAM user, the IAM user will be able to retrieve billing information for the **parent HyperStore user**.

- Sample request and response (abridged):

```
REQUEST


http://
localhost:16080/?Action=GetCloudianBill&CanonicalUserId=d47151635ba8d94efe981b24db00c07e
&BillingPeriod=201807

<request headers including authorization info>

RESPONSE

200 OK
```

```
<?xml version="1.0" encoding="utf-8"?>
<GetCloudianBillResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<Bill>
<billID>936265a2-fbd5-47c2-82ed-d62298299a1b</billID>
etc...
...
...
</Bill>
</GetCloudianBillResponse>
```

3.2. POST /billing

POST /billing   Create a bill for a user or group

3.2.1. Syntax

```
POST /billing?[userId=string&][groupId=string][canonicalUserId=string]&billingPeriod=string
```

There is no request payload.

3.2.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Optional, strings) Identifiers of the user or group for which to generate or retrieve a bill.

- To generate a bill for a **user who currently is part of the service**, you can either use the "userId" parameter in combination with the "groupId" parameter (for example *userId=martinez&groupId=operations*), or use the "canonicalUserId" parameter by itself (with no "groupId" parameter).
- To generate a bill for a **user who has been deleted from the service**, you must use the "canonicalUserId" parameter by itself (not the "userId" or "groupId" parameter).

  Note If you don't know the user's system-generated canonical ID, you can obtain it by using the *GET /user/list* method.

- To generate a bill for a **whole user group**, use the "groupId" parameter by itself (not the "userId" or "canonicalUserId" parameter). Note that if you generate a bill for a whole group, the bill will be based on the rating plan assigned to the group as a whole, and will not take into account any different rating plans that administrators may have assigned to specific users within the group.

billingPeriod

(Mandatory, string) Specifies the year and month of bill. Format is *yyyyMM* — for example "202407" for July 2024. The system uses GMT time when demarcating exactly when a month begins and ends. Note that the bills are derived from hourly rollup usage data which by default is only retained in the system for

65 days. So by default you cannot specify a billing period (billing month) that started more than 65 days ago.

3.2.3. Usage Notes

This method generates a user's monthly bill or a whole group’s monthly bill, and returns the bill in the response body. The billing period **must be a month that has already completed**. You cannot generate a bill for the current, in-progress month.

IMPORTANT ! Billing calculation is derived from hourly rollup usage data. The retention period for hourly rollup usage data is configured by the setting *s3.usage.reports.hourlyRollupTTL*. The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills. So be sure to have *s3.usage.reports.hourlyRollupTTL* set to a value large enough to accommodate your billing routine.

3.2.4. Example Using cURL

The **example** below generates a billable activity report for the user "glad" from the "eng" group, for the month of July 2024.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/billing?userId=glad&groupId=eng&billingPeriod=202407' \
| python -mjson.tool
```

The response payload is a JSON-formatted *Bill* object, which in this example is as follows.

```
{
  "billID": "936265a2-fbd5-47c2-82ed-d62298299a1b",
  "canonicalUserId": "d47151635ba8d94efe981b24db00c07e",
  "currency": "USD",
  "endCal": 1659326400000,
  "groupId": "eng",
  "notes": null,
  "regionBills": [
    {
      "currency": "USD",
      "items": {
        "SB": {
          "item":"SB",
          "quantity":108.00,
          "rules":"1,0.14:5,0.12:0,0.10",
          "subtotal":10.94
        }
      },
      "region": "taoyuan",
      "total": 10.94,
      "whitelistItems": {},
      "whitelistTotal": 0
    }
  ],
  "startCal": 1656648000000,
  "total": 10.94,
  "userId": "glad",
  "whitelistTotal": 0
}
```

3.2.5. Response Element Descriptions

billID

(String) System-generated globally unique bill ID. Example:

```
"billID": "936265a2-fbd5-47c2-82ed-d62298299a1b"
```

canonicalUserId

(String) System-generated canonical user ID for the user. Empty if the bill is for a whole group. Example:

```
"canonicalUserId": "d47151635ba8d94efe981b24db00c07e"
```

currency

(String) Currency string. Example:

```
"currency": "USD"
```

endCal

(String) End date/time of the billing period in UTC milliseconds. Example:

```
"endCal": 1659326400000
```

groupId

(String) ID of the group to which the user belongs (or of the group for which the bill was generated, in the case of a whole group bill). Example:

```
"groupId": "eng"
```

notes

(String) Notes regarding the bill, if any. Example:

```
"notes": null
```

regionBills

(Map<string,*RegionBill*>) List of *RegionBill* objects, with one such object per service region. The *RegionBill* object consists of the following attributes and nested objects:

currency

(String) Currency string. Example:

```
"currency": "USD"
```

items

(Map<string,*BillItem*>) List of *BillItem* objects, with one such object for each activity type that's being charged for, per the terms of the user's rating plan. Supported activity types are "SB" (storage bytes), "BI" (bytes in), "BO" (bytes out), "HG" (HTTP Gets), "HP" (HTTP Puts), "HD" (HTTP Deletes). This list excludes activity for whitelisted IP addresses. Note that some or even most activity types may not appear, depending on the rating plan terms. For example, it may be that only storage bytes ("SB") are billed for, if that's how the user's rating plan is configured.

In the items list, each *BillItem* object is preceded by its activity type string, such as "SB": {*BillItem* data}. In the example only storage bytes ("SB") are charged for in the rating plan that was applied when this bill was generated.

The *BillItem* object consists of the following attributes:

item

(String) Usage type being billed for. Types are "SB" (storage bytes), "BI" (bytes in), "BO" (bytes out), "HG" (HTTP GETs), "HP" (HTTP PUTs), "HD" (HTTP DELETEs). Example:

```
"item":"SB"
```

quantity

(Number) Usage quantity during billing period. Usage quantity metrics depend on the usage type:

- For storage bytes (SB), the metric is GiB-Month (average number of GiBs of data stored for the billing month). This is calculated by summing the month’s hourly readings of stored bytes, converting to GiB, then dividing by the number of hours in the month. In the example above the usage quantity during the billing period was 108 GiB-months (that is, the user’s storage bytes volume average 108GiBs over the course of the month)
- For data transfer bytes in (BI) or out (BO), the metric is number of bytes.
- For HTTP GETs (HG), PUTs (HP), or DELETEs (HD), the metric is number of multiples of 10,000 requests. For example, if usage type is HG and quantity is 7.50, that means 75,000 HTTP GET requests.

Example:

```
"quantity":108.00
```

rules

(String) Specification of billing rules for this usage type (as configured in the user’s assigned rating plan). In the example the "rules" attribute indicates that the user’s rating plan is such that the first 1 GiB-month is charged at $0.14, the next 5 GiB-months is charged at $0.12 per GiB-month, and all GiB-months above that are charged at $0.10 per GiB-month.

Example:

```
"rules":"1,0.14:5,0.12:0,0.10"
```

subtotal

(Number) Total billing charge for the particular usage type specified by the "item" attribute. This will be in units of the currency specified by the "currency" attribute of the *RegionBill*object that contains this *BillItem* object. It’s labeled as "subtotal" because it will be added together with subtotals for other usage types (from other*BillItem*object instances within the*RegionBill*object, if any) to compute the "total" attribute for the encompassing *RegionBill* instance. In the example the $10.94 sub-total comes from applying the billing rules to the 108 GiB-months usage quantity ([1 X .14] + [5 X .12] + [102 X .10] = 10.94).

Example:

```
"subtotal":10.94
```

region

(String) Region name. Example:

```
"region": "taoyuan"
```

total

(Number) For the region, the total charges incurred — excluding activity originating from whitelisted source IP addresses. Example:

```
"total": 10.94
```

whitelistItems

(Map<string,*BillItem*>) List of *BillItem*objects, for activity originating from whitelisted IP addresses (if any). Types are "BI" (bytes in), "BO" (bytes out), "HG" (HTTP Gets), "HP" (HTTP Puts), "HD" (HTTP Deletes). Example:

```
"whitelistItems": {}
```

whitelistTotal

(Number) For the region, the total charges incurred for activity originating from whitelisted source IP addresses. Typically there are no charges for such activity. Example:

```
"whitelistTotal": 0
```

startCal

(String) Start date/time of the billing period in UTC milliseconds. Example:

```
"startCal": 1656648000000
```

total

(Number) The total charges incurred by the user during the billing period, excluding activity for whitelisted source IP addresses. Example:

```
"total": 10.94
```

userId

(String) ID of the user for whom the bill was generated. Empty if the bill is for a whole group. Example:

```
"userId": "glad"
```

whitelistTotal

(Number) The total charges incurred by the user for activity originating from whitelisted source IP addresses. Example:

```
"whitelistTotal": 0
```

3.2.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No billing data

Status Code	Description
400	Invalid billing period
400	Missing required parameter : {billingPeriod}
400	Conflicting parameters: {canonicalUserId, groupId, userId}

Chapter 4. bppolicy

The Admin API methods built around the **bppolicy** resource are for retrieving certain information about Hyper-Store storage policies (also known as bucket protection policies).

For an overview of the HyperStore storage policy feature, see "Storage Policies Feature Overview" in the ***Cloud-ian HyperStore Administrator's Guide***. **To create or change storage policies** use the CMC's **Storage Policies** page (**Cluster -> Storage Policies**).

Methods associated with the *bppolicy* resource:

- **"GET /bppolicy/bucketsperpolicy"** (page 43)
- **"GET /bppolicy/listpolicy"** (page 44)

4.1. GET /bppolicy/bucketsperpolicy

GET /bppolicy/bucketsperpolicy Get list of buckets using each storage policy

4.1.1. Syntax

```
GET /bppolicy/bucketsperpolicy
```

There is no request payload.

4.1.2. Usage Notes

If you have a storage policy in your system that was created prior to the release of HyperStore version 5.2 (when support for multiple storage policies was introduced), the *GET /bppolicy/bucketsperpolicy* method does not work for listing buckets that use that storage policy. This is true even for buckets that were created after HyperStore version 5.2, if the buckets use that legacy storage policy. In the *GET /bppolicy/bucketsperpolicy*response, the policy ID for such a legacy storage policy will be *DEFAULT_ <regionName>* and the bucket list for the storage policy will be empty.

4.1.3. Example Using cURL

The **example** below retrieves the list of buckets using each storage policy.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/bppolicy/bucketsperpolicy | python -mjson.tool
```

The response payload is a JSON-formatted list of *BucketsInPolicy* objects, which in this example is as follows.

```
[
  {
    "buckets": [
      "qa.tests",
      "dev.specs"
    ],
    "policyId": "b06c5f9213ae396de1a80ee264092b56",
    "policyName": "Replication-3X"
  },
  {
    "buckets": [
      "release.packages.archive",
      "techpubs.manuals.archive"
    ],
    "policyId": "af37905a8523d8d403d993c4f2e2c1a1",
    "policyName": "EC-4-2"
  }
]
```

4.1.4. Response Element Descriptions

buckets

(List<string>) List of buckets that use the storage policy. Example:

```
"buckets": ["qa.tests","dev.specs"]
```

policyId

(String) System-generated unique identifier of the storage policy. Example:

```
"policyId": "b06c5f9213ae396de1a80ee264092b56"
```

policyName

(String) Storage policy name. Example:

```
"policyName": "Replication-3X"
```

4.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

4.2. GET /bppolicy/listpolicy

GET /bppolicy/listpolicy Get list of storage policy names and IDs

4.2.1. Syntax

```
GET /bppolicy/listpolicy[?region=string][&groupId=string[&status=enum]
```

There is no request payload.

4.2.2. Parameter Descriptions

region

(Optional, string) If you use this parameter, then only policies associated with the specified service region will be returned.

groupId

(Optional, string) If you use this parameter, then only policies that are available to the specified group will be returned. This includes system default storage policies (which are available to all groups) as well as storage policies that are explicitly made available to the specified group.

status

(Optional, enum) If you use this parameter, then only policies that have the specified status will be returned. The supported statuses are:

- *pending* — The policy is in the process of being created in the system. In this state the policy is not yet available to be used.
- *active* — The policy is currently available to users when they create a new bucket.
- *disabled* — The policy is no longer available to users when they create a new bucket. However, the policy still exists in the system and is still being applied to any buckets to which the policy was assigned during the period when it was active.
- *deleted* — The policy has been marked for deletion and is no longer available to users. However, the policy has not yet been purged from the system by the daily cron job.
- *failed* — During the policy creation, the policy failed to be fully set up in the system. Though a BucketProtectionPolicy JSON object exists and can be retrieved, the policy is not actually set up in the system and is not usable.

4.2.3. Usage Notes

Use this method if you want to retrieve the system-generated policy ID for each of your storage policies.

If you use more than one of the three optional filters -- *region*, *groupId*, and *status* -- then the returned list of storage policy IDs will be for storage policies that match all of your specified filters. For example if you specify a *region* and a *groupId*, then the returned list will consist only of policies that are both associated with that region and available to that group.

4.2.4. Example Using cURL

The **example** below retrieves the list of all storage policies currently in the system.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/bppolicy/listpolicy | python -mjson.tool
```

The response payload is a JSON-formatted list of *BucketProtectionPolicy* objects, with one such object for each storage policy. Among the attributes for each policy is the "policyId" and "policyName". In the example that follows there are two storage policies in the system, and the response payload is truncated so as to show only the policy ID and policy name attributes.

```
[
  {
    ...
    ...
    "policyId": "b06c5f9213ae396de1a80ee264092b56",
    "policyName": "Replication-3X",
    ...
    ...
  },
  {
    ...
    ...
    "policyId": "af37905a8523d8d403d993c4f2e2c1a1",
    "policyName": "EC-4-2",
    ...
    ...
  }
]
```

4.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

Chapter 5. bucketops

Methods associated with the *bucketops* resource:

- **"GET /bucketops/cleanUpdatesInRange/status"** (page 47)
- **"GET /bucketops/id"** (page 48)
- **"GET /bucketops/gettags"** (page 49)
- **"POST /bucketops/cleanUpdatesInRange/go"** (page 51)
- **"POST /bucketops/cleanUpdatesInRange/stop"** (page 53)
- **"POST /bucketops/purge"** (page 54)

5.1. GET /bucketops/cleanUpdatesInRange/status

GET /bucketops/cleanUpdatesInRange/status Check status of a 'cleanUpdatesInRange' operation

5.1.1. Syntax

```
GET /bucketops/cleanUpdatesInRange/status?bucketName=string
```

There is no request payload.

5.1.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the bucket for which to check the status of a 'cleanUpdatesInRange' operation.

5.1.3. Usage Notes

This Admin API call checks the status of an in-progress 'cleanUpdatesInRange' operation on a specified bucket. For information about the 'cleanUpdatesInRange' operation see **"POST /bucketops/cleanUpdatesInRange/go"** (page 51).

5.1.4. Example Using cURL

The **example** below checks 'cleanUpdatesInRange' operation status for a bucket named "bucket1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/bucketops/cleanUpdatesInRange/status?bucketName=bucket1
```

The response indicates that 48 "columns" (objects) have been processed so far:

```
Bucket: bucket1 cleanUpdatesInRange processed 48 columns
```

If no 'cleanUpdatesInRange' operation is in progress on the specified bucket, the response will indicate "CleanUpdatesInRange job not found for bucket: <bucketName>".

5.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: bucketName

5.2. GET /bucketops/id

GET /bucketops/id   Get a bucket's canonical ID

5.2.1. Syntax

```
GET /bucketops/id?bucketName=string
```

There is no request payload.

5.2.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the bucket for which to retrieve a canonical ID.

5.2.3. Usage Notes

This operation returns a bucket's canonical ID, if one exists. A bucket will have a canonical ID (a system-generated unique identifier) if either of the following applies:

- The bucket was created in HyperStore 7.0 or later.
- The bucket has been subjected to a successful *POST /bucketops/purge* operation.

After a successful *POST /bucketops/purge* operation a bucket will have a different canonical ID than the one it had before (if it had any) but will have the same bucket name.

5.2.4. Example Using cURL

The **example** below retrieves the canonical ID of a bucket named "bucket1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/bucketops/id?bucketName=bucket1
```

The response payload is the bucket's canonical ID in plain text, which in this example is as follows:

```
40cc2eba37fd82df4ce04bce2bc35a94
```

5.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: bucketName

5.3. GET /bucketops/gettags

GET /bucketops/gettags   Get bucket tags for users in a group

5.3.1. Syntax

```
GET /bucketops/gettags?groupId=string[&limit=integer][&userId=string]
```

There is no request payload.

5.3.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve a list of users and their bucket tags.

limit

(Optional, integer) The maximum number of users to return (along with those users' bucket tags) per operation.

Users are retrieved in alphanumeric order. If the number of users in the group exceeds the number specified by *limit*, in the response to the first *GET /bucketops/gettags* operation the *nextUserId* attribute will indicate the user ID of the next available user in the alphanumeric ordering (the alphanumerically first of the users that has not yet been returned). That user ID can then be used as the *userId* parameter value in a subsequent *GET /bucketops/gettags* operation. That operation will again return up to the number of users specified by *limit*; and if there are more remaining users beyond that, the return will again use the *nextUserId* attribute to indicate the next available user's ID; and so on.

Admin API client applications can use the *limit* and *userId* parameters in combination to support pagination of results.

Defaults to 10. Maximum allowed value for *limit* is 100.

userId

(Optional, string) The alphanumerically first user to retrieve. See the description of *limit* above for more detail about how the *userId* and *limit* parameters can be used to support pagination.

In the first *GET /bucketops/gettags* request for a group the client should omit the *userId* parameter. If *userId* is omitted from the request, the operation's returned list of users starts with the alphanumerically first user in the group.

Note The *userId* parameter is to be used only for pagination. You cannot use this parameter to retrieve bucket tags for just one user of your choosing.

5.3.3. Usage Notes

For each user in the specified group this operation returns the bucket tags associated with the user's buckets (after such tags have been created by the S3 API method *PutBucketTagging*; for more information on this method see the S3 section of the ***Cloudian HyperStore AWS APIs Support Reference***). Pagination of the response is supported by use of the optional *limit* and *userId* settings. By default a maximum of 10 users is returned per request.

5.3.4. Example Using cURL

The **example** below returns the bucket tags for the buckets owned by users in the group "Cloudian".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/bucketops/gettags?groupId=Cloudian \
| python -mjson.tool
```

The response payload is a JSON-formatted *BucketTags* object, which in this example is as follows.

```
{
"groupId":"Cloudian",
"nextUserId":null,
"userBucket":
  {"kthompson":
    {"bbucket":{"Project":"Project1","Manager":"jsmith"},
    "cbucket":{"security":"public"}},
  "gwashington":
    {"dbucket":{"security":"public"}}
  }
}
```

5.3.5. Response Element Descriptions

groupId

(String) The group for which users and bucket tags have been retrieved. Example:

```
"groupId":"Cloudian"
```

nextUserId

(String) Users are retrieved in alphanumeric order. If the number of users in the group exceeds the number specified by the query parameter *limit* (which defaults to 10), in the *GET /bucketops/gettags* response the *nextUserId* attribute will indicate the user ID of the next available user in the alphanumeric ordering (the alphanumerically first of the users that has not yet been returned). That user ID can then be used as the *userId* query parameter value in a subsequent *GET /bucketops/gettags* operation. That operation will again return up to the number of users specified by *limit*; and if there are more remaining users beyond that, the return will again use the *nextUserId* attribute to indicate the next available user's ID.

If alphanumerically there are no additional users beyond those returned in the current response, the *nextUserId* attribute value will be null.

Example:

```
"nextUserId":null
```

userBucket

(Map<String, Map<String, Map<String, String>>>) This entity contains the map of users, their owned buckets that have bucket tags, and the bucket tags. The format is as follows:

{"*userId*": {"*bucketName*":{"*tagName*":"*tagValue*"},{"*tagName2*":"*tagValue2*"}...}, {"*bucketName2*": {"*tagName*":"*tagValue*"},{"*tagName2*":"*tagValue2*"}...},... "*userId2*": {"*bucketName*"...} }

Example:

```
"userBucket":
  {"kthompson":
    {"bbucket":{"Project":"Project1","Manager":"jsmith"},
    "cbucket":{"security":"public"}},
  "gwashington":
    {"dbucket":{"security":"public"}}
  }
```

Note Only buckets that have bucket tags are listed in the response. Buckets that do not have bucket tags are excluded from the response.

5.3.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : {groupId}

5.4. POST /bucketops/cleanUpdatesInRange/go

POST /bucketops/cleanUpdatesInRange/go   Delete object versions created within specified time range

5.4.1. Syntax

```
POST /bucketops/cleanUpdatesInRange/go?bucketName=string&fromTimestamp=string&toTimestamp=string
```

There is no request payload.

IMPORTANT ! When objects are uploaded to the HyperStore system, they are assigned creation stamps indicating the current time **in the time zone in which the HyperStore system is located**. So this should be taken into consideration when specifying *fromTimestamp* and *toTimestamp* values to indicate the range of object versions to delete, based on object version creation time. The specified time range should be expressed as times in the time zone in which the system is located.

5.4.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the bucket for which to delete object versions created within specified time range.

fromTimestamp

(Optional, string) Timestamp in ISO 8601 format, indicating the starting date-time (inclusive) of the interval from which to delete newly created object versions.

Although the *fromTimestamp* parameter and *toTimestamp* are each optional individually, you must use at least one of the parameters:

- If you use only the *fromTimestamp* parameter, all object versions created at or after that timestamp will be deleted.
- If you use only the *toTimestamp* parameter, all object versions created before that timestamp will be deleted
- If you use the *fromTimestamp* parameter and also the *toTimestamp* parameter, all object versions created at or after the *fromTimestamp* and before the *toTimestamp* will be deleted.

toTimestamp

(Optional, string) Timestamp in ISO 8601 format, indicating the ending date-time (exclusive) of the interval from which to delete newly created object versions. For more information see the description of the *fromTimestamp* parameter.

5.4.3. Usage Notes

This Admin API call initiates an operation that deletes from a specified bucket all object versions created within a specified time period. This operation is supported only for buckets that have versioning enabled (and had it enabled during the specified time period). The primary use case for this operation is to recover from a cybersecurity incident in which a malicious actor changes (creates new versions of) objects in a bucket -- for example, by making it so that the most current versions of objects are encrypted -- and/or deletes (creates delete markers for) objects in a bucket. In the wake of such an attack, once the time period of the attack has been definitively determined, the 'cleanUpdatesInRange' operation enables you to delete all object versions (including delete markers) created during that time period. In this way you can "roll back" to the bucket's condition prior to the attack.

Note that:

- The success response to this API call indicates only that the operation has been initiated. The operation may subsequently take some time to complete. You can check operation status with the **GET /bucketops/cleanUpdatesInRange/status** call. If for some reason you need to stop the in-progress operation you can do so with the **POST /bucketops/cleanUpdatesInRange/stop** call.
- Only one 'cleanUpdatesInRange' operation can be run on a bucket at a time.

IMPORTANT ! When objects are uploaded to the HyperStore system, they are assigned creation stamps indicating the current time **in the time zone in which the HyperStore system is located**. So this should be taken into consideration when specifying *fromTimestamp* and *toTimestamp* values to indicate the range of object versions to delete, based on object version creation time. The specified time range should be expressed as times in the time zone in which the system is located.

5.4.4. Example Using cURL

The **example** below starts a 'cleanUpdatesInRange' operation for a bucket named "bucket1".

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/bucketops/cleanUpdatesInRange/go?bucketName=bucket1
&fromTimestamp=2025-02-26T00:00:00Z&toTimestamp=2025-02-27T00:00:00Z'
```

5.4.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: bucketName
400	Missing required parameter: from Timestamp or toTimestamp
400	Invalid timestamp: from Timestamp
400	Invalid timestamp: toTimestamp
409	CleanUpdatesInRange job already started for bucket: {bucketName}

5.5. POST /bucketops/cleanUpdatesInRange/stop

POST /bucketops/cleanUpdatesInRange/stop Stop a 'cleanUpdatesInRange' operation

5.5.1. Syntax

```
POST /bucketops/cleanUpdatesInRange/stop?bucketName=string
```

There is no request payload.

5.5.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the bucket for which to stop an in-progress 'cleanUpdatesInRange' operation.

5.5.3. Usage Notes

This Admin API call stops an in-progress 'cleanUpdatesInRange' operation on a specified bucket. For information about the 'cleanUpdatesInRange' operation see **"POST /bucketops/cleanUpdatesInRange/go"** (page 51).

5.5.4. Example Using cURL

The **example** below stops an in-progress 'cleanUpdatesInRange' operation on a bucket named "bucket1".

```
curl -X POST -k -u sysadmin:password \
https://localhost:19443/bucketops/cleanUpdatesInRange/stop?bucketName=bucket1
```

The response indicates that the operation is now stopped on that bucket:

```
Bucket: bucket1 cleanUpdatesInRange stopped
```

5.5.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: bucketName

5.6. POST /bucketops/purge

POST /bucketops/purge   Delete all the objects in a bucket

5.6.1. Syntax

```
POST /bucketops/purge?bucketName=string[&token=string][&deleteBucket=boolean]
```

There is no request payload.

5.6.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the bucket for which to purge all objects.

token

(Optional, string) The OTP token (one-time password token) provided to you by Cloudian Support for the purpose of allowing a purge of this bucket. This is applicable **only if the target bucket is configured with Object Lock and your system is licensed for Compatible Object Lock** (not Certified Object Lock). You must include this token in order to purge an Object Locked bucket. For information about how to acquire this token from Cloudian Support see **"Purging an Object Locked Bucket"** (page 56) below.

deleteBucket

(Optional, boolean) This option is supported **only if the target bucket is configured with Object Lock and your system is licensed for Compatible Object Lock** (not Certified Object Lock). If those conditions are met you can use *deleteBucket=true* to have the system delete the bucket itself after it purges all data from the bucket. If the bucket owner has File Services enabled, and has created an SMB or NFS file share corresponding to this bucket, when the system deletes the bucket it will also delete the corresponding file share.

Note If you are purging a bucket that is **not** Object Locked you cannot use the *deleteBucket* option with the purge operation. Instead, after the data is purged from the bucket you can delete the bucket in the normal way through the HyperStore S3 interface, if you wish. You can do this either in the CMC (if your system is configured to allow administrators to view and manage users' data in the CMC) or by using a third party S3 application to submit a *DeleteBucket* call to the HyperStore S3 Service.

5.6.3. Usage Notes

The *POST /bucketops/purge* operation results in the system marking all the objects in the bucket as having been deleted. However the actual deletion of object data from disk will not occur until the next automatic running of the object deletion batch processing job. By default this batch processing of object data deletes runs hourly on each node. (The frequency with which the batch processing job runs is configurable by the setting *s3.batchOperations.delete.queuePollInterval*.)

The *POST /bucketops/purge* operation does not invoke the S3 API's *DeleteObject* or *DeleteObjects* calls and does not create the Cassandra tombstone issues that can sometimes be caused by mass delete operations that use HyperStore's S3 interface.

If the bucket is an Object Locked bucket and you use the *deleteBucket* option, the bucket itself is deleted along with all its contents. Otherwise the bucket itself continues to exist after the operation. If you run an S3 *ListObjects* call on the bucket -- or get the bucket in the CMC -- after executing the *POST /bucketops/purge* operation, the *ListObjects* response will indicate that the bucket is empty even though the actual deletion of objects may not have been completed by the cron job yet.

Note that:

- In a multi-region system, the *POST /bucketops/purge* call **must be submitted to the Admin API service in the region in which the target bucket is located**. If you have a multi-region system and you're not sure which region the bucket is located in, you have several options for determining the bucket's region:
    - If your system is configured to allow administrators to view and manage users' data in the CMC, you can view the bucket and its region location in the CMC.
    - If you have the bucket owner's S3 credentials you can submit an S3 *GetBucketLocation* call to the HyperStore S3 Service.
    - You can use the Admin API to run these two calls in succession -- the first call to check what storage policy the bucket is using and to see the *policyId* of that storage policy; and the second call to get that storage policy's attributes including what region the policy is located in:
        - *GET /bppolicy/bucketsperpolicy* (for more information see **"GET /bppolicy/bucketsperpolicy Get list of buckets using each storage policy"** (page 43))
        - *GET /bppolicy?policyId=*string
- If you have versioning configured on the bucket, the *POST /bucketops/purge* operation will purge all versions of all objects in the bucket.
- If you have auto-tiering configured on the bucket, any objects that have been tiered from the bucket to the remote tiering destination will also be deleted (at the next running of the hourly system cron job mentioned above).
- Any S3 multipart upload operations in-progress for the bucket at the time that you execute the *POST /bucketops/purge* operation will be aborted.

5.6.4. Purging an Object Locked Bucket

If your system is licensed for **Compatible** Object Lock you can purge an Object Locked bucket as described below. You cannot purge an Object Locked bucket if your system is licensed for Certified Object Lock.For background information on the two types of Object Lock (Compatible and Certified), including how to tell which type your system is licensed for, see "Object Lock Feature Overview" in the ***Cloudian HyperStore Administrator's Guide***.

IMPORTANT ! This procedure requires you to provide a token challenge to Cloudian Support, and in response Cloudian Support will provide you an OTP token (one-time password token) that you will use when executing the purge of the locked bucket. In your planning for purging the bucket, **please take into account that Cloudian Support will need time to review and process your case**. This may take up to several days.

To purge an Object Locked bucket:

1. Use the Admin API call *GET /system/token/challenge?action=purge&params=bucketName:string* to generate a token challenge. For more information about this API call see **"GET /system/token/challenge Get token challenge to provide to Cloudian Support"** (page 165).

   Here is an example of a token challenge generated by the *GET /system/token/challenge* API call.

   ```
   A.fa868eff0219b2ecdf58d2aff6fa355584b090a2a6ff4073d28a2245bbc23f09
   ```

2. In the Cloudian Support Portal open a support case with subject "Token Request", and copy-paste the token challenge (that you generated in Step 1) into the case. Indicate that your system is licensed for Compatible Mode Object Lock and you want to purge an Object Locked bucket.

3. After reviewing the case and verifying your token challenge, through the support case Cloudian Support will provide you an OTP token that is specifically for the purpose of allowing a purge of the bucket for which you generated the token challenge in Step 1.

   Here is an example of an OTP token such as Support will provide you through the support case:

   ```
   2024091704.A.ed3596d9d7b725a19e6e0f6df89fa9751c0c4403effe04712197c3acb34c37fb
   ```

   The first segment in the three-segment token is the token expiration date-time in format YYYYMMDDHH (2024091704 in the example above).

   IMPORTANT ! Once you open the "Token Request" support case, watch for the email indicating that Cloudian Support has updated the case and provided you the OTP token. **The OTP token expires and becomes invalid 24 hours after Cloudian Support creates it.** When you receive the OTP token from Cloudian Support use it promptly, as described in the next step below.

4. Use the Admin API call *POST /bucketops/purge?bucketName=string&token=string [&deleteBucket=boolean]* to purge the bucket. For the *&token=string* value use the **entire** three-segment token that Cloudian Support provided you. For more information about this API call see **"POST /bucketops/purge Delete all the objects in a bucket"** (page 54).

   Note If you do not delete the bucket by using the *deleteBucket=true* option, the empty bucket will continue to exist and will still have Object Lock enabled.

Note that you can only use the OTP token for this one Object Locked bucket, and only before the token's expiration date-time. If you need to purge this same Object Locked bucket again in the future, or if you need to purge another Object Locked bucket, you will need to acquire a different token from Cloudian Support by following the steps above.

5.6.4.1. Logging of Bucket Purge Requests

Requests to purge a locked bucket are -- like any other Admin API request -- logged to the Admin Service request log (*cloudian-admin-request-info.log*). Such requests are **not** logged to the Object Lock / WORM audit log (*s3-worm.log*).

For example, in *cloudian-admin-request-info.log:*

```
2024-09-08 15:31:42,119|[0:0:0:0:0:0:0:1]|POST|/bucketops/purge|
bucketName:objlocked,token:ONERESPONSE|917|403
```

Additional details regarding the purge request processing are logged in the Admin Service application log (*cloudian-admin.log*). For example:

```
2024-09-08 15:31:42,119  INFO[qtp1530295320-2528] BucketOpsResource:Cannot purge bucket
objlocked as provided token ONERESPONSE is invalid or expired.
```

5.6.5. Example Using cURL

The **example** below purges the contents of a bucket named "bucket1". This is not an Object Locked bucket so the *token* parameter is not used.

```
curl -X POST -k -u sysadmin:password \
https://localhost:19443/bucketops/purge?bucketName=bucket1
```

The response indicates that the bucket contents have been successfully purged (marked for deletion):

```
Bucket: bucket1 purged.
```

5.6.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : bucketName
400	Purge request must be sent to region in which bucket is located
400	deleteBucket is supported only for locked buckets
403	Compatible Object Lock type license is required to purge a locked bucket
403	Valid token from Cloudian Support is required to purge a locked bucket

This page left intentionally blank

Chapter 6. group

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.

Methods associated with the *group* resource:

- **"DELETE /group"** (page 59)
- **"GET /group"** (page 60)
- **"GET /group/list"** (page 62)
- **"GET /group/ratingPlanId"** (page 65)
- **"POST /group"** (page 66)
- **"POST /group/ratingPlanId"** (page 67)
- **"PUT /group"** (page 68)

6.1. DELETE /group

DELETE /group   Delete a group

6.1.1. Syntax

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

There is no request payload.

6.1.2. Parameter Descriptions

groupId

(Mandatory, string) Unique identifier of the group.

6.1.3. Usage Notes

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

6.1.4. Example Using cURL

The **example** below deletes the "QA" group.

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

6.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {groupId}
400	Group does not exist
409	Cannot delete. Group is not empty.

6.2. GET /group

GET /group   Get a group's profile

6.2.1. Syntax

```
GET /group?groupId=string
```

There is no request payload.

6.2.2. Parameter Descriptions

groupId

(Mandatory, string) Unique identifier of the group.

> **Note** The System Admin group's "groupId" is "0".

6.2.3. Example Using cURL

The **example** below retrieves the "QA" group.

```
curl -X GET -k -u sysadmin:password https://localhost:19443/group?groupId=QA \
| python -mjson.tool
```

The response payload is a JSON-formatted *GroupInfo* object, which in this example is as follows.

```
{
  "active": "true",
  "groupId": "QA",
  "groupName": "Quality Assurance Group",
  "ldapEnabled": false,
  "ldapGroup": "",
  "ldapMatchAttribute": "",
  "ldapSearch": "",
  "ldapSearchUserBase": "",
  "ldapServerURL": "",
  "ldapUserDNTemplate": "",
  "s3endpointshttp": ["ALL"],
  "s3endpointshttps": ["ALL"],
  "s3websiteendpoints": ["ALL"]
}
```

6.2.4. Response Element Descriptions

For description of the elements that comprise the *GroupInfo* object, see **"PUT /group Create a new group"** (page 68).

6.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Group does not exist
400	Missing required parameters : {groupId}

6.2.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianGroup*
- Parameters: Same as for *GET /group*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /group* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can get any group
  - HyperStore group admin user can only get his own group
  - HyperStore regular user cannot use this method
  - IAM user can only use this method if granted *admin:GetCloudianGroup* permission by policy, and subject to the same restriction as the parent HyperStore user

  Note The "GetCloudianGroup" action retrieves group profile data for Cloudian HyperStore groups, not for HyperStore users' subsidiary IAM groups.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianGroup&GroupId=QA

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
```

```
<GetCloudianGroupResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<GroupInfo>
<active>true</active>
etc...
...
...
</GroupInfo>
</GetCloudianGroupResponse>
```

6.3. GET /group/list

GET /group/list Get a list of group profiles

6.3.1. Syntax

```
GET /group/list[?prefix=string][&limit=integer][&offset=string]
```

There is no request payload.

6.3.2. Parameter Descriptions

prefix

(Optional, string) A group ID prefix to use for filtering. For example, if you specify "prefix=usa" then only groups whose group ID starts with "usa" would be retrieved.

Defaults to empty string (meaning that no prefix-based filtering is performed).

limit

(Optional, integer) For purposes of pagination, the maximum number of groups to return in one response. If more than this many groups meet the filtering criteria, then the actual number of groups returned will be "limit plus 1". The last group returned — the "plus 1" — is an indicator that there are more matching groups than could be returned in the current response (given the specified "limit" value). That group’s ID can be used as the "offset" value in the next request. Note that if the offset group happens to be the last group in the entire set of matching groups, the subsequent query using the offset will return no groups.

Defaults to 100.

offset

(Optional, string) The group ID with which to start the response list of groups for the current request, sorted alphanumerically. The "offset" parameter can be used for purposes of pagination within a large result set that is being retrieved via multiple sequential requests. See the description of "limit" above for more information.

If "offset" is not specified, the first group in the response list will be the alphanumerically first group from the entire result set.

6.3.3. Example Using cURL

The **example** below retrieves the profiles of all groups currently in the system.

```
curl -X GET -k -u sysadmin:password https://localhost:19443/group/list \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *GroupInfo* objects, which in this example is as follows.

```
[
  {
    "active": "true",
    "groupId": "QA",
    "groupName": "Quality Assurance Group",
    "ldapEnabled": false,
    "ldapGroup": "",
    "ldapMatchAttribute": "",
    "ldapSearch": "",
    "ldapSearchUserBase": "",
    "ldapServerURL": "",
    "ldapUserDNTemplate": "",
    "s3endpointshttp": ["ALL"],
    "s3endpointshttps": ["ALL"],
    "s3websiteendpoints": ["ALL"]
  },
  {
    "active": "true",
    "groupId": "Support",
    "groupName": "Technical Support Group",
    "ldapEnabled": false,
    "ldapGroup": "",
    "ldapMatchAttribute": "",
    "ldapSearch": "",
    "ldapSearchUserBase": "",
    "ldapServerURL": "",
    "ldapUserDNTemplate": "",
    "s3endpointshttp": ["ALL"],
    "s3endpointshttps": ["ALL"],
    "s3websiteendpoints": ["ALL"]
  },
  {
    "active": "true",
    "groupId": "engineering",
    "groupName": "Engineering Group",
    "ldapEnabled": false,
    "ldapGroup": "",
    "ldapMatchAttribute": "",
    "ldapSearch": "",
    "ldapSearchUserBase": "",
    "ldapServerURL": "",
    "ldapUserDNTemplate": "",
    "s3endpointshttp": ["ALL"],
    "s3endpointshttps": ["ALL"],
    "s3websiteendpoints": ["ALL"]
  }
]
```

6.3.4. Response Element Descriptions

For description of the elements that comprise the *GroupInfo* object, see **"PUT /group   Create a new group"** (page 68).

6.3.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Limit should be greater than zero

6.3.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianGroupList*
- Parameters: Same as for *GET /group/list*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /group/list* except the data is formatted in XML rather than JSON
- Role-based restrictions:
    - HyperStore system admin user can use this method
    - HyperStore group admin user cannot use this method
    - HyperStore regular user cannot use this method
    - IAM user can only use this method if granted *admin:GetCloudianGroupList* permission by policy, and subject to the same restriction as the parent HyperStore user

    Note The "GetCloudianGroupList" action retrieves a list of Cloudian HyperStore groups, not a list of HyperStore users' subsidiary IAM groups.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianGroupList

<request headers including authorization info>

RESPONSE

200 OK
```

```
<?xml version="1.0" encoding="utf-8"?>
<GetCloudianGroupListResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<groupInfo>
<active>true</active>
etc...
...
...
</groupInfo>
<groupInfo>
etc...
...
...
</groupInfo>
</ListWrapper>
</GetCloudianGroupListResponse>
```

6.4. GET /group/ratingPlanId

GET /group/ratingPlanId Get a group's rating plan ID

6.4.1. Syntax

```
GET /group/ratingPlanId?groupId=string[&region=string]
```

There is no request payload.

6.4.2. Parameter Descriptions

groupId

(Mandatory, string) Unique identifier of the group.

Note The System Admin group's "groupId" is "0".

region

(Optional, string) Region from which to retrieve rating plan IDs. Defaults to the default service region if not specified.

6.4.3. Example Using cURL

The **example** below retrieves the ID of the rating plan assigned to the "QA" group.

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

The response payload is the rating plan identifier in plain text, which in this example is as follows.

```
Default-RP
```

6.4.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Rating Plan does not exist
400	Missing Required parameters : {groupId}
400	Region {region} is not valid

6.5. POST /group

POST /group Change a group's profile

6.5.1. Syntax

```
POST /group
```

The required request payload is a JSON-formatted *GroupInfo* object.

6.5.2. Usage Notes

When editing the *GroupInfo* object before doing the POST operation you can edit any attribute except for the "groupId" attribute. The "groupId" attribute must remain the same, so that you're modifying an existing group rather than creating a new one. For an example *GroupInfo* object see **PUT /group**.

For **LDAP authentication to work for system admin users** when they log into the HyperStore Shell, along with enabling LDAP for the System Admin group by editing the group's profile you must also perform this additional configuration step:

1. Log in to the Config Controller 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
   ```

6.5.3. Example Using cURL

The **example** below modifies the group profile that was created in the **PUT /group** example. Again the *GroupInfo* object is specified in a text file named *group_QA.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @group_QA.txt https://localhost:19443/group
```

There is no response payload.

6.5.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Group does not exist
400	Missing required attribute : {groupId}
400	Invalid Active Status for Post Group
400	Invalid JSON Object

6.6. POST /group/ratingPlanId

POST /group/ratingPlanId Assign a rating plan to a group

6.6.1. Syntax

```
POST /group/ratingPlanId?groupId=string&ratingPlanId=string[&region=string]
```

There is no request payload.

6.6.2. Parameter Descriptions

groupId

(Mandatory, string) Unique identifier of the group.

ratingPlanId

(Mandatory, string) Unique identifier of the rating plan to assign to the group.

region

(Optional, string) If your service deployment has multiple service regions, rating plan assignment is on a per-region basis. With the *POST /group/ratingPlanId* method, use the "region" parameter to indicate the service region in which to apply the specified rating plan. For example, if *groupId=Engineering&ratingPlanId=Gold&region=East*, then the Gold rating plan will be applied to the Engineering group's service activity in the East region. Defaults to the default service region if not specified.

6.6.3. Example Using cURL

The **example** below assigns the "Gold" rating plan to the "QA" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/group/ratingPlanId?groupId=QA&ratingPlanId=Gold'
```

6.6.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {groupId, ratingPlanId}
400	Region {region} is not valid

6.7. PUT /group

PUT /group Create a new group

6.7.1. Syntax

```
PUT /group
```

The required request payload is a JSON-formatted *GroupInfo* object. See example below.

6.7.2. Example Using cURL

The **example** below creates a new group with "QA" as its unique identifier. In this example the JSON-formatted *GroupInfo* object is specified in a text file named *group_QA.txt* which is then referenced as the data input to the cURL command.

```
curl -X PUT -H "Content-Type: application/json" -k -u sysadmin:password \
-d @group_QA.txt https://localhost:19443/group
```

The *group_QA.txt* file content in this example is as follows.

```
{
  "active": "true",
  "groupId": "QA",
  "groupName": "Quality Assurance Group",
  "ldapEnabled": false,
  "ldapGroup": "",
  "ldapMatchAttribute": "",
  "ldapSearch": "",
  "ldapSearchUserBase": "",
  "ldapServerURL": "",
  "ldapUserDNTemplate": "",
  "s3endpointshttp": ["ALL"],
  "s3endpointshttps": ["ALL"],
  "s3websiteendpoints": ["ALL"]
}
```

Note If you set the "ldapEnabled" attribute to "false" for a group that you are creating, you do not need to include the other "ldap*" attributes in the *GroupInfo* object. However they are shown above for completeness. The S3 endpoint attributes are also optional.

There is no response payload.

6.7.3. Request Element Descriptions

active

(Optional, string) Whether the group is enabled ("true") or disabled ("false") in the system. The users associated with a disabled group will be unable to access HyperStore data storage or to log in to the Cloudian Management Console. On a PUT of a new group, the "active" attribute defaults to "true" if not explicitly set. If not specified in a POST update of an existing group, the group retains its existing active or inactive status.

Example:

```
"active": "true"
```

groupId

(Mandatory, string) Group ID. Only letters, numbers, dashes, and underscores are allowed. Length must be at least 1 character and no more than 64 characters.

Example:

```
"groupId": "QA"
```

Note The System Admin group's "groupId" is "0".

Note In the CMC interface the field "Group Name" maps to the "groupId" attribute in the Admin API.

groupName

(Optional, string) Group name. Maximum length 64 characters. Example:

```
"groupName": "Quality Assurance Group"
```

Note In the CMC interface the field "Group Description" maps to the "groupName" attribute in the Admin API.

ldapEnabled

(Optional, boolean) Whether LDAP authentication is enabled for members of this group, *true* or *false*. Defaults to *false*. If LDAP authentication is enabled for the group, then by default when users from this group log into the CMC, the CMC will check against an LDAP system (with details as specified by other *GroupInfo* attributes, below) in order to authenticate the users. You can override this behavior on a per-user basis -- that is, you can configure certain users within the group so that they are authenticated by reference to a CMC-based password rather than an LDAP system. For more high-level information about HyperStore's support for LDAP-based user authentication, see "LDAP Integration" in the ***Cloudian HyperStore Administrator's Guide***.

Example:

```
"ldapEnabled": false
```

Note If you enable LDAP Authentication for an existing group to which users have already been added 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.

ldapGroup

(Optional, string) The group's name from the LDAP system. This would typically be the group's "ou" (Organization Unit) value in the LDAP system, but could also be for example the "l" (Location) value or "memberOf" value -- depending on which LDAP attribute is to be used to identify users' group membership when the CMC authenticates them against the LDAP system.

If you use the variable *{groupId}* in any of the other LDAP authentication configuration attributes, when implementing LDAP authentication HyperStore will automatically replace the variable with the ldapGroup value.

```
"ldapGroup": "Quality Assurance (U.S.)"
```

ldapMatchAttribute

(Optional, string) For background information about the purpose of this attribute, see the description of the *ldapSearch* attribute below.

Use the *ldapMatchAttribute* setting to specify an LDAP attribute value against which LDAP-enabled users in this group must match in order to be authorized to log into the CMC. Use this format: *<attribute>=<value>*.

Example:

```
"ldapMatchAttribute": "l=California"
```

Example:

```
"ldapMatchAttribute": "memberOf=Sales"
```

ldapSearch

(Optional, string) If this is an LDAP-enabled group, and if you want to establish a LDAP-based user authorization filter to complement the user authentication template that you set with the *ldapUserDNTemplate* attribute, then use the *ldapSearch*, *ldapSearchUserBase*, and *ldapMatchAttribute* attributes to configure the filter. If you do so, then LDAP-enabled users from this group when logging in to the CMC will need to meet the requirements of the authentication template and also the requirements of the filter.

Use the *ldapSearch* attribute to specify the user identifier type that you used in the *ldapUserDNTemplate*, in format "*(<LDAP_user_identifier_attribute>={userId})*". This is used to retrieve a user’s LDAP record in order to apply the filtering.

Example:

```
"ldapSearch": "(uid={userId})"
```

Example:

```
"ldapSearch": "(userPrincipalName={userId})"
```

ldapSearchUserBase

(Optional, string) For background information about the purpose of this attribute, see the description of the *ldapSearch* attribute above.

Use the *ldapSearchUserBase* attribute to specify the LDAP search base from which the CMC should start when retrieving the user's LDAP record in order to apply filtering. .

Example:

```
"ldapSearchUserBase": "dc=my-company,dc=com"
```

Example:

```
"ldapSearchUserBase": "uid={userId},ou=engineering,dc=my-company,dc=com"
```

ldapServerURL

(Optional, string) If this is an LDAP-enabled group, use this attribute to specify the URL that the CMC should use to access the LDAP Server when authenticating users in this group.

Note that if you use *ldaps* (LDAP secured by SSL/TLS), the LDAP server must use a CA-verified certificate not a self-signed certificate. HyperStore does not support connecting to an LDAP server that's using a self-signed SSL certificate.

Example:

```
"ldapServerURL": "ldap://my.ldap.server:389"
```

Example:

```
"ldapServerURL": "ldap://my.ldap.server:389/o=MyCompany"
```

ldapUserDNTemplate

(Optional, string) If this is an LDAP-enabled group, use this attribute to specify how users within this group will be authenticated against the LDAP system when they log into the CMC. It is a template that defines how user names supplied during CMC login will be mapped to user-identifying information in the LDAP system. Two typical ways of configuring this template are:

- **Distinguished Name**. With this approach the template specification would include the LDAP attribute "uid" set to equal the CMC token "{userId}" (exactly as shown below), the LDAP attribute "ou" set to equal the group's organizational unit value from the LDAP system, and the domain components from LDAP. For example:

  ```
  "ldapUserDNTemplate": "uid={userId},ou=engineering,dc=my-company,dc=com"
  ```

  With the DN template above, LDAP-enabled users from this group will log in with their LDAP *uid* value as their CMC user ID. During login the CMC will also verify that the *ou* value in the user's LDAP record matches against the *ou* value from the template.

  > **Note** If you are configuring LDAP authentication for the System Admin group, use the Distinguished Name approach for the user DN template.

- **userPrincipalName**. With this approach the template would simply map the LDAP attribute "userPrincipalName" to the CMC variable "{userId}", like this:

  ```
  "ldapUserDNTemplate": "userPrincipalName={userId}"
  ```

With the approach above LDAP-enabled users from this group will log in with their LDAP *user-PrincipalName* value (such as *<user>@<domain>*) as their CMC user ID. Optionally, to implement additional LDAP-based authorization filters such as the users' group or location, you can use the *ldapSearch*, *ldapSearchUserBase*, and *ldapMatchAttribute* attributes (all described earlier in this topic) when you create the group in HyperStore.

s3endpointshttp

(Optional, list<string>) The S3 HTTP service endpoint(s) that will be displayed to this group's users when those users log into the CMC and go to the **Security Credentials** page. The value can be:

- A single endpoint
- Multiple endpoints in a comma-separated list
- The string "ALL" (to indicate that this group's users will be able to see all of the system's configured S3 HTTP endpoints in the CMC's **Security Credentials** page)
- The string "NONE" (to indicate that this group's users will not be able to see any S3 HTTP endpoints in the CMC's **Security Credentials** page)

If the *s3endpointshttp* attribute is omitted from the *GroupInfo* object in a *PUT /group* request, the attribute defaults to ["ALL"].

Example:

```
"s3endpointshttp": ["ALL"]
```

Note This attribute and the other S3 endpoint attributes do not impact a group's users' authorization to access S3 endpoints. They only impact what S3 endpoint information is displayed to users in the CMC's **Security Credentials** page.

s3endpointshttps

(Optional, list<string>) The S3 HTTPS service endpoint(s) that will be displayed to this group's users when those users log into the CMC and go to the **Security Credentials** page. The value can be:

- A single endpoint
- Multiple endpoints in a comma-separated list
- The string "ALL" (to indicate that this group's users will be able to see all of the system's configured S3 HTTPS endpoints in the CMC's **Security Credentials** page)
- The string "NONE" (to indicate that this group's users will not be able to see any S3 HTTPS endpoints in the CMC's **Security Credentials** page)

If the *s3endpointshttps* attribute is omitted from the *GroupInfo* object in a *PUT /group* request, the attribute defaults to ["ALL"].

Example:

```
"s3endpointshttps": ["ALL"]
```

s3websiteendpoints

(Optional, list<string>) The S3 website service endpoint(s) that will be displayed to this group's users when those users log into the CMC and go to the **Security Credentials** page. The value can be:

- A single endpoint
- Multiple endpoints in a comma-separated list

- The string "ALL" (to indicate that this group's users will be able to see all of the system's configured S3 website endpoints in the CMC's **Security Credentials** page)
- The string "NONE" (to indicate that this group's users will not be able to see any S3 website endpoints in the CMC's **Security Credentials** page)

If the *s3websiteendpoints* attribute is omitted from the *GroupInfo* object in a *PUT /group* request, the attribute defaults to ["ALL"].

Example:

```
"s3websiteendpoints": ["ALL"]
```

6.7.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required attribute : {groupId}
400	Invalid JSON Object
400	Invalid Group ID
400	Invalid Active Status for Add Group
409	Unique constraint violation : {groupId}

Chapter 7. monitor

The Admin API methods built around the **monitor** resource are for monitoring the health and performance of your HyperStore system. There are methods for retrieving system and node statistics and for implementing system alert functionality.

Methods associated with the *monitor* resource:

- **"DELETE /monitor/notificationrule"** (page 75)
- **"GET /monitor/events"** (page 76)
- **"GET /monitor/nodelist"** (page 80)
- **"GET /monitor/host"** (page 81)
- **"GET /monitor"** (page 93)
- **"GET /monitor/history"** (page 100)
- **"GET /monitor/notificationrules"** (page 102)
- **"POST /monitor/acknowledgeevents"** (page 103)
- **"POST /monitor/notificationruleenable"** (page 105)
- **"POST /monitor/notificationrule"** (page 106)
- **"PUT /monitor/notificationrule"** (page 107)

7.1. DELETE /monitor/notificationrule

DELETE /monitor/notificationrule   Delete a notification rule

7.1.1. Syntax

```
DELETE /monitor/notificationrule?ruleId=string[&region=string]
```

There is no request payload.

7.1.2. Parameter Descriptions

ruleId

(Mandatory, string) The system-generated unique ID for the notification rule. For the default notification rules that come packaged with the HyperStore system this will be a simple integer like "1", "2", or "14". For rules that you create yourself the system will generate a ruleId in the form of a UUID string like "8e4c-c533-360a-4dd5-bfe4-6b5f5b6c40da".

If you do not know the ruleId, you can retrieve it by using the *GET /monitor/notificationrules* method. That method returns a list of notification rules which includes each rule’s ruleId.

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.1.3. Example Using cURL

The **example** below deletes a notification rule.

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/monitor/notificationrule?ruleId=8ef63b63-4961-4e17-88c7-d53c966557db
```

7.1.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {ruleId}
400	Notification rule does not exist : {ruleId}

7.2. GET /monitor/events

GET /monitor/events Get the event list for a node

7.2.1. Syntax

```
GET /monitor/events?nodeId=string[&showAck=bool][&limit=integer][&region=string]
```

There is no request payload.

7.2.2. Parameter Descriptions

nodeId

(Mandatory, string) The hostname of the target node.

showAck

(Optional, boolean) Whether to return acknowledged events as well as unacknowledged events, true or false.

If not specified in the request, defaults to false (only unacknowledged events are returned).

limit

(Optional, integer) The maximum number of events to return in the response.

If not specified in the request, the default limit is 100 events.

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.2.3. Usage Notes

Alert lists in the CMC are retrieved by this API method, but the CMC interface uses the term "alerts" rather than "events".

7.2.4. Example Using cURL

The **example** below retrieves the list of unacknowledged events for the node that has hostname "store1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/monitor/events?nodeId=store1 | python -mjson.tool
```

The response payload is a JSON-formatted list of *MonitoringEvent* objects, which in this example is as follows.

```
[
  {
    "ack": false,
    "condition": "<",
    "conditionVal": "0.15",
    "count": 1,
    "eventType": "13|/dev/mapper/vg0-root",
    "nodeId": "store1",
    "severityLevel": 2,
    "statId": "diskInfo",
    "timestamp": "1502797442785",
    "value": "/dev/mapper/vg0-root: 0.11198006761549427"
  },
  {
    "ack": false,
    "condition": "",
    "conditionVal": "",
    "count": 1,
    "eventType": "14|",
    "nodeId": "store1",
    "severityLevel": 0,
    "statId": "repairCompletionStatus",
    "timestamp": "1502794743351",
    "value": "REPAIR cmdno#: 610 status: COMPLETED"
  }
]
```

7.2.5. Response Element Descriptions

ack

(Boolean) Whether the event has been acknowledged, true or false. This will be false unless you explicitly retrieved acknowledged events when you executed the *GET /monitor/events* method. Example:

```
"ack": false
```

condition

(String) From the notification rule that triggered this event, the condition comparison type in the rule definition. This will be "=", "<", or ">". Example:

```
"condition": "<"
```

conditionVal

(String) From the notification rule that triggered this event, the condition value against which to compare. For example, this may be a numerical threshold value, or a service status such as "SVC_DOWN", or a log message level such as "LOG_ERR". Example:

```
"conditionVal": "0.15"
```

count

(Integer) Number of times that the event has occurred without being acknowledged. Example:

```
"count": 1
```

eventType

(String) From the notification rule that triggered this event, the integer <ruleId> value . (Or, for log events, a concatenation of "<ruleId>|<logCategory>". The logCategory is derived from the line of code that generates the specific log message.) Example:

```
"eventType": "13|/dev/mapper/vg0-root"
```

nodeId

(String) Hostname of the node on which the event occurred. Example:

```
"nodeId": "store1"
```

severityLevel

(Integer) Severity level of the event, as configured in the notification rule for the event. This is an integer with meaning as follows:

- 0 = Low
- 1 = Medium
- 2 = High
- 3 = Critical

Example:

```
"severityLevel": 2
```

statId

(String) From the notification rule that triggered this event, the statId. See for a list of supported statIds.

Example:

```
"statId": "diskInfo"
```

Note The "svcS3" statId encompasses events pertaining to auto-tiering and cross-region replication, as well as events pertaining to providing S3 service to clients.

timestamp

(String) Timestamp of latest event occurrence in UTC milliseconds. Example:

```
"timestamp": "1502797442785"
```

value

(String) On the node, the statistic value that triggered this event. For example, if the event was triggered by a statistic rising to a threshold-exceeding value, this attribute would indicate that value. In the case of a log event, the "value" is the log message.

Example:

```
"value": "/dev/mapper/vg0-root: 0.11198006761549427"
```

7.2.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {nodeId}
400	Invalid region : {region}
400	Invalid limit

7.2.7. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianMonitorEvents*
- Parameters: Same as for *GET /monitor/events*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /monitor/events* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can use this method
  - HyperStore group admin user cannot use this method
  - HyperStore regular user cannot use this method
  - IAM user can only use this method if granted *admin:GetCloudianMonitorEvents* permission by policy, and subject to the same restriction as the parent HyperStore user
- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianMonitorEvents&NodeId=store1

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianMonitorEventsResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
```

```
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<monitoringEvent>
<ack>false</ack>
etc...
...
...
</monitoringEvent>
<monitoringEvent>
etc...
...
...
</monitoringEvent>
</ListWrapper>
</GetCloudianMonitorEventsResponse>
```

7.3. GET /monitor/nodelist

GET /monitor/nodelist Get the list of monitored nodes

7.3.1. Syntax

```
GET /monitor/nodelist[?region=string]
```

There is no request payload.

7.3.2. Parameter Descriptions

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.3.3. Example Using cURL

The **example** below retrieves the list of monitored nodes in the default service region.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/monitor/nodelist | python -mjson.tool
```

The response payload is a JSON-formatted list of hostnames, which in this example is as follows.

```
[
  "store1",
  "store2",
  "store3"
]
```

7.3.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

7.3.5. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianMonitorNodeList*
- Parameters: Same as for *GET /monitor/nodelist*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /monitor/nodelist* except the data is formatted in XML rather than JSON
- Role-based restrictions:
    - HyperStore system admin user can use this method
    - HyperStore group admin user cannot use this method
    - HyperStore regular user cannot use this method
    - IAM user can only use this method if granted *admin:GetCloudianMonitorNodeList* permission by policy, and subject to the same restriction as the parent HyperStore user
- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianMonitorNodeList

<request headers including authorization info>

RESPONSE

200 OK


<?xml version="1.0" encoding="utf-8"?>
<GetCloudianMonitorNodeListResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<String>
hyperstore1
</String>
</GetCloudianMonitorNodeListResponse>
```

7.4. GET /monitor/host

GET /monitor/host   Get current monitoring statistics for a node

7.4.1. Syntax

```
GET /monitor/host?nodeId=string[&region=string]
```

There is no request payload.

7.4.2. Parameter Descriptions

nodeId

(Mandatory, string) The hostname of the target node.

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.4.3. Example Using cURL

The **example** below retrieves current monitoring statistics for the node that has hostname "store1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/monitor/host?nodeId=store1 | python -mjson.tool
```

The response payload is a JSON-formatted *MonitorNodeInfo* object, which in this example is as follows.

```
{
  "adminHeapMax": {
    "timestamp": "1502799543355",
    "value": "409075712"
  },
  "adminHeapUsed": {
    "timestamp": "1502799543355",
    "value": "109849080"
  },
  "cassCMSGCCount": {
    "timestamp": "1502799543355",
    "value": "3"
  },
  "cassCMSGCTime": {
    "timestamp": "1502799543355",
    "value": "151"
  },
  "cassCopyGCCount": null,
  "cassCopyGCTime": null,
  "cassHeapMax": {
    "timestamp": "1502799543355",
    "value": "2086666240"
  },
  "cassHeapUsed": {
    "timestamp": "1502799543355",
    "value": "1233215776"
  },
  "cassParNewGCCount": {
    "timestamp": "1502799543355",
    "value": "4882"
  },
  "cassParNewGCTime": {
    "timestamp": "1502799543355",
    "value": "87598"
  },
  "cpu": {
    "timestamp": "1502799663530",
    "value": "0.06"
  },
  "diskAvailKb": {
    "timestamp": "1502799543355",
    "value": "21912316"
  },
  "diskIORead": {
    "timestamp": "1502799663530",
    "value": "0"
  },
  "diskIOWrite": {
    "timestamp": "1502799663530",
    "value": "93811"
  },
  "diskTotalKb": {
    "timestamp": "1502799543355",
    "value": "36056096"
  },
  "diskUsedKb": {
    "timestamp": "1502799543355",
    "value": "13330724"
  },
  "disksInfo": {
    "disks": [
      {
        "deviceName": "/dev/mapper/vg0-root",
        "diskAvailKb": "1776316",
        "diskIORead": "724419584",
        "diskIOWrite": "471087837184",
        "diskTotalKb": "15874468",
        "diskUsedKb": "13285096",
        "mountPoint": "/",
        "status": "OK",
        "storageUse": [
          "CASSANDRA",
          "REDIS",
          "LOG"
        ]
      },
      {
        "deviceName": "/dev/vdb1",
        "diskAvailKb": "20135940",
        "diskIORead": "3163136",
        "diskIOWrite": "50221056",
        "diskTotalKb": "20181628",
        "diskUsedKb": "45688",
        "mountPoint": "/cloudian1",
        "status": "OK",
        "storageUse": [
          "HS"
        ]
      }
    ],
    "timestamp": "1502799663530"
  },
  "hyperStoreHeapMax": {
    "timestamp": "1502799543355",
    "value": "1635909632"
  },
  "hyperStoreHeapUsed": {
    "timestamp": "1502799543355",
    "value": "139187600"
  },
  "ioRx": {
    "timestamp": "1502799663530",
    "value": "17216"
  },
  "ioTx": {
    "timestamp": "1502799663530",
    "value": "28179"
  },
  "s3GetLatency": null,
  "s3GetTPS": null,
  "s3GetThruput": {
    "timestamp": "1502799543355",
    "value": "0"
  },
  "s3HeapMax": {
    "timestamp": "1502799543355",
    "value": "818020352"
  },
  "s3HeapUsed": {
    "timestamp": "1502799543355",
    "value": "164786136"
  },
  "s3PutLatency": {
    "timestamp": "1502799543355",
    "value": "18.4"
  },
  "s3PutTPS": {
    "timestamp": "1502799543355",
    "value": "0.0"
  },
  "s3PutThruput": {
    "timestamp": "1502799543355",
    "value": "0"
  },
  "status": {
    "ipaddr": "",
    "status": [
      "LOG_WARN"
    ],
    "timestamp": "1502799663530",
    "value": "[LOG_WARN]"
  },
"svcAdmin": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcCassandra": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcHyperstore": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcRedisCred": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcRedisMon": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcRedisQos": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
  "timestamp": "1502799663530",
  "value": "[OK]"
},
"svcS3": {
  "ipaddr": "10.10.2.91",
  "status": [
    "OK"
  ],
    "timestamp": "1502799663530",
    "value": "[OK]"
  }
}
```

7.4.4. Response Element Descriptions

Note Within the *MonitorNodeInfo* object:
* All statistics for which the data type (in the descriptions below) is *MonitorStat* are formatted as *"<statName>": {"timestamp": "<UTCMilliseconds>","value": "<statValue>"}*.
* All statistics for which the data type is *ServiceStatus* are formatted as *{"ipaddr": "<nodeIPAdress>","status": [<list of one or more of "OK","SVC_DOWN", "LOG_WARN", or "LOG_ERR">],"timestamp": "<UTCMilliseconds>","value": "<statusValueFormattedAsString>"}*

adminHeapMax

(MonitorStat) The maximum JVM heap size allocated to the Admin Service, in bytes. Example:

```
"adminHeapMax": {"timestamp": "1502799543355","value": "409075712"}
```

adminHeapUsed

(MonitorStat) The Admin Service's current JVM heap memory usage on the node, in bytes. This is measured each five minutes. Example:

```
"adminHeapUsed": {"timestamp": "1502799543355","value": "109849080"}
```

cassCMSGCCount

(MonitorStat) The number of concurrent mark-sweep (CMS) garbage collections executed since the last start-up of the Cassandra service on this node. This collection type targets old-generation objects. Example:

```
"cassCMSGCCount": {"timestamp": "1502799543355","value": "3"}
```

cassCMSGCTime

(MonitorStat) The aggregate time (in milliseconds) spent on executing CMC garbage collections since the last start-up of the Cassandra service on this node. Example:

```
"cassCMSGCTime": {"timestamp": "1502799543355","value": "151"}
```

cassCopyGCCount

(MonitorStat) The number of Copy garbage collections executed since the last start-up of the Cassandra service on this node. Example:

```
"cassCopyGCCount": null
```

cassCopyGCTime

(MonitorStat) The aggregate time (in milliseconds) spent on executing Copy garbage collections since the last start-up of the Cassandra service on this node. Example:

```
"cassCopyGCTime": null
```

cassHeapMax

(MonitorStat) The maximum JVM heap size allocated to the Cassandra Service, in bytes. Example:

```
"cassHeapMax": {"timestamp": "1502799543355","value": "2086666240"}
```

cassHeapUsed

(MonitorStat) The Cassandra Service’s current JVM heap memory usage on the node, in bytes. This is measured each five minutes. Example:

```
"cassHeapUsed": {"timestamp": "1502799543355","value": "1233215776"}
```

cassParNewGCCount

(MonitorStat) The number of parallel new-generation (ParNew) garbage collections executed since the last start-up of the Cassandra service on this node. Example:

```
"cassParNewGCCount": {"timestamp": "1502799543355","value": "4882"}
```

cassParNewGCTime

(MonitorStat) The aggregate time (in milliseconds) spent on executing ParNew garbage collections since the last start-up of the Cassandra service on this node. Example:

```
"cassParNewGCTime": {"timestamp": "1502799543355","value": "87598"}
```

cpu

(MonitorStat) Current CPU utilization percentage on the node. This is measured once per every five minutes. Example:

```
"cpu": {"timestamp": "1502799663530","value": "0.06"}
```

diskAvailKb

(MonitorStat) On the node, the total mounted disk space that's still available for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory). Reported as a number of kibibytes.

The following are deducted from the total amount of unused disk space to arrive at the "available" disk space amount that is reported by the *diskAvailKb* statistic:

- Each disk’s "reserved-blocks-percentage" (the portion of the disk that’s reserved for privileged processes)
- Each HyperStore data disk's "stop-write" buffer (10% of capacity by default). For more information on the stop-write feature see the "Automated Disk Management" section in the ***Cloudian HyperStore Administrator's Guide***.

Example:

```
"diskAvailKb": {"timestamp": "1502799543355","value": "21912316"}
```

Note Because the reserved-blocks-percentage and the stop-write buffer percentage are not counted as available space, the *diskUsedKb* and *diskAvailKb* will add up to less than the *diskTotalKb*.

diskIORead

(MonitorStat) Across all the node’s disks that are being used for S3 object storage or Cassandra metadata storage, the average disk read throughput in bytes per second. This stat is recalculated each minute, based on the most recent minute of activity. Example:

```
"diskIORead": {"timestamp": "1502799663530","value": "0"}
```

diskIOWrite

(MonitorStat) Across all the node’s disks that are being used for S3 object storage or Cassandra metadata storage, the average disk write throughput in bytes per second. This stat is recalculated each minute, based on the most recent minute of activity. Example:

```
"diskIOWrite": {"timestamp": "1502799663530","value": "93811"}
```

diskTotalKb

(MonitorStat) On the node, the total size of the disks mounted for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory). Reported as a number of kibibytes. Example:

```
"diskTotalKb": {"timestamp": "1502799543355","value": "36056096"}
```

diskUsedKb

(MonitorStat) On the node, the total disk space that's been consumed for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory) Reported as a number of kibibytes. Example:

```
"diskUsedKb": {"timestamp": "1502799543355","value": "13330724"}
```

disksInfo

(DiskMonitorStat) Current information about each disk on the node. Formatted as {"disks": List<*DiskInfo*>,"timestamp": "<UTCMilliseconds>"}. There is one nested *DiskInfo* object for each disk on the node. Each *DiskInfo* object consists of the following attributes:

deviceName

(String) Disk drive device name. Example:

```
"deviceName": "/dev/mapper/vg0-root"
```

diskAvailKb

(String) Total remaining free space on the disk in number of KiBs. In calculating the available space, a disk’s "reserved-blocks-percentage" (the portion of the disk space that’s reserved for privileged processes) is considered to be unavailable. By default in Linux systems the configurable "reserved-blocks-percentage" for a file system is 5% of disk capacity. If this is a HyperStore data disk, then the "stop-write" buffer (10% of disk capacity by default) is also considered to be unavailable.

Example:

```
"diskAvailKb": "1776316"
```

diskIORead

(String) The average disk read throughput for this disk, in bytes per second. This stat is recalculated each minute, based on the most recent minute of data. Example:

```
"diskIORead": "724419584"
```

diskIOWrite

(String) The average disk write throughput for this disk, in bytes per second. This stat is recalculated each minute, based on the most recent minute of data. Example:

```
"diskIOWrite": "471087837184"
```

diskTotalKb

(String) Total capacity of the disk in number of KiBs. Example:

```
"diskTotalKb": "15874468"
```

diskUsedKb

(String) Amount of used space on the disk in number of KBs. Example:

```
"diskUsedKb": "13285096"
```

mountPoint

(String) File system mount point for the disk. Example:

```
"mountPoint": "/
```

status

(EDiskStatus) Disk status string. One of "OK", "ERROR", or "DISABLED". For description of these disk statuses, while on the CMC's **Node Status** page click **Help**.

Example:

```
"status": "OK"
```

storageUse

(EStorageType) List of storage type strings, indicating what type of data is being stored on the disk. One or more of:

- "CASSANDRA" — System metadata and S3 object metadata in Cassandra.
- "REDIS" — System metadata in Redis.
- "LOG" — Application logs
- "HS" — Replicated S3 object data.
- "EC" — Erasure coded S3 object data.
- "NOTAVAIL" — Storage usage information cannot be retrieved for this disk.

Example:

```
"storageUse": ["CASSANDRA","REDIS","LOG"]
```

hyperStoreHeapMax

(MonitorStat) The maximum JVM heap size allocated to the HyperStore Service, in bytes. Example:

```
"hyperStoreHeapMax": {"timestamp": "1502799543355","value": "1635909632"}
```

hyperStoreHeapUsed

(MonitorStat) The HyperStore Service’s current JVM heap memory usage on the node, in bytes. This is measured each five minutes. Example:

```
"hyperStoreHeapUsed": {"timestamp": "1502799543355","value": "139187600"}
```

ioRx

(MonitorStat) The aggregate network bytes per second received by the node, for all types of network traffic including but not limited to S3 request traffic. For nodes with multiple network interfaces, this stat is an aggregation across the multiple interfaces. This stat is recalculated each minute, based on the most recent minute of activity.

Example:

```
"ioRx": {"timestamp": "1502799663530","value": "17216"}
```

ioTx

(MonitorStat) The aggregate network bytes per second transmitted by the node, for all types of network traffic including but not limited to S3 request traffic. For nodes with multiple network interfaces, this stat is an aggregation across the multiple interfaces. This stat is recalculated each minute, based on the most recent minute of activity. Example:

```
"ioTx": {"timestamp": "1502799663530","value": "28179"}
```

s3GetLatency

(MonitorStat) On the node, the 95th percentile request latency value for S3 GET transactions, in milliseconds. New statistic values are calculated each five minutes, based on the most recent approximately 1000 GET transactions. The s3GetLatency value indicates that of the last 1000 GET transactions, 95% completed in that many milliseconds or less.

Example:

```
"s3GetLatency": null
```

Note HEAD transactions are counted toward this stat also.

s3GetTPS

(MonitorStat) On the node, the number of S3 GET transactions processed per second. This stat is recalculated each five minutes, based on the most recent five minutes of activity. HEAD transactions are counted toward this stat also. Example:

```
"s3GetTPS": null
```

s3GetThruput

(MonitorStat) On the node, the data throughput for S3 GET transactions, expressed as bytes per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. HEAD transactions are counted toward this stat also. Example:

```
"s3GetThruput": {"timestamp": "1502799543355","value": "0"}
```

s3HeapMax

(MonitorStat) The maximum JVM heap size allocated to the S3 Service, in bytes. Example:

```
"s3HeapMax": {"timestamp": "1502799543355","value": "818020352"}
```

s3HeapUsed

(MonitorStat) The S3 Service's current JVM heap memory usage on the node, in bytes. This is measured each five minutes. Example:

```
"s3HeapUsed": {"timestamp": "1502799543355","value": "164786136"}
```

s3PutLatency

(MonitorStat) On the node, the 95th percentile request latency value for S3 PUT transactions, in milliseconds. New statistic values are calculated each five minutes, based on the most recent approximately 1000 PUT transactions. The s3PutLatency value indicates that of the last 1000 PUT transactions, 95% completed in that many milliseconds or less.

Example:

```
"s3PutLatency": {"timestamp": "1502799543355","value": "18.4"}
```

Note POST transactions are counted toward this stat also.

s3PutTPS

(MonitorStat) On the node, the number of S3 PUT transactions processed per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. POST transactions are counted toward this stat also. Example:

```
"s3PutTPS": {"timestamp": "1502799543355","value": "0.0"}
```

s3PutThruput

(MonitorStat) On the node, the data throughput for S3 PUT transactions, expressed as bytes per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. POST transactions are counted toward this stat also. Example:

```
"s3PutThruput": {"timestamp": "1502799543355","value": "0"}
```

status

(ServiceStatus) Overall status of the node.

Formatted as *{"ipaddr": "<empty>", "status": [<list of one or more of "OK","SVC_DOWN","LOG_WARN", or "LOG_ERR">], "timestamp": "<UTCMilliseconds>","value": "<statusValueFormattedAsString>",}.*

The "ipaddr" value will be empty or null here; an IP address is specified only in the service-specific status attributes (such as "svcCassandra") described below.

Possible values within the "status" list are:

- "OK" — All HyperStore services are up and running on the node, and the node has no unacknowledged events.
- "SVC_DOWN" — One or more HyperStore services (Admin, Cassandra, HyperStore, Redis QoS, Redis Credentials, Redis Monitor, or S3) is down on the node
- "LOG_WARN" — There are unacknowledged warnings in an application log on the node (such as the S3 Service application log or the Cassandra application log).
- "LOG_ERR" — There are unacknowledged errors in an application service log.

The "value" attribute will be identical to the "status" attribute, except formatted as a single string rather than a list of strings.

Example:

```
"status": {"ipaddr": "","status": ["SVC_DOWN","LOG_WARN"],"timestamp":
"1502799663530","value": "[SVC_DOWN, LOG_WARN]"}
```

svcAdmin

(ServiceStatus) Admin service status on the node.

Formatted as *{"ipaddr": "<nodeIPAdress>","status": [<list of one or more of "OK","SVC_DOWN", "LOG_WARN", or "LOG_ERR">],"timestamp": "<UTCMilliseconds>","value": "<statusValueFormattedAsString>"}*.

Example:

```
"svcAdmin": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

The other service status attributes that follow below (*svcCassandra* and so on) are formatted in the same way.

svcCassandra

(ServiceStatus) Cassandra service status on the node. Example:

```
"svcCassandra": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

svcHyperstore

(ServiceStatus) HyperStore service status on the node. Example:

```
"svcHyperstore": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

svcRedisCred

(ServiceStatus) Redis Credentials service status on the node. Example:

```
"svcRedisCred": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

svcRedisMon

(ServiceStatus) Redis Monitor service status on the node. Example:

```
"svcRedisMon": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

svcRedisQos

(ServiceStatus) Redis QoS service status on the node. Example:

```
"svcRedisQos": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

svcS3

(ServiceStatus) S3 service status on the node. Example:

```
"svcS3": {"ipaddr": "10.10.2.91","status": ["OK"],"timestamp":
"1502799663530","value": "[OK]"}
```

7.4.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {nodeId}

7.4.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianMonitorHost*
- Parameters: Same as for *GET /monitor/host*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /monitor/host* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can use this method
  - HyperStore group admin user cannot use this method
  - HyperStore regular user cannot use this method
  - IAM user can only use this method if granted *admin:GetCloudianMonitorHost* permission by policy, and subject to the same restriction as the parent HyperStore user
- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianMonitorHost

<request headers including authorization info>

RESPONSE

200 OK


<?xml version="1.0" encoding="utf-8"?>
<GetCloudianMonitorHostResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<MonitorNodeInfo>
<adminHeapMax>
<timestamp>1534534923619</timestamp>
<value>1538260992</value>
</adminHeapMax>
etc...
...
...
</MonitorNodeInfo>
</GetCloudianMonitorHostResponse>
```

7.5. GET /monitor

GET /monitor   Get current monitoring statistics for a service region

7.5.1. Syntax

```
GET /monitor?[region=string]
```

There is no request payload.

7.5.2. Parameter Descriptions

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.5.3. Example Using cURL

The **example** below retrieves current monitoring statistics for the default service region.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/monitor | python -mjson.tool
```

The response payload is a JSON-formatted *MonitorSystemInfo* object, which in this example is as follows.

```
{
  "diskAvailKb": {
    "timestamp": "1502799843254",
    "value": "61855592"
  },
  "diskTotalKb": {
    "timestamp": "1502799843254",
    "value": "88115680"
  },
  "diskUsedKb": {
  "timestamp": "1502799843254",
  "value": "23814368"
  },
  "nodeStatuses": [
    {
      "hostname": "store1",
      "ipaddr": null,
      "status": [
        "LOG_WARN"
      ],
      "timestamp": "1502799843254",
      "value": "[LOG_WARN]"
    },
    {
      "hostname": "store2",
      "ipaddr": null,
      "status": [
        "LOG_WARN"
      ],
      "timestamp": "1502799843254",
      "value": "[LOG_WARN]"
    },
    {
      "hostname": "store3",
      "ipaddr": null,
      "status": [
        "LOG_WARN"
      ],
      "timestamp": "1502799843254",
      "value": "[LOG_WARN]"
    }
  ],
  "s3GetLatency": null,
  "s3GetTPS": null,
  "s3GetThruput": {
    "timestamp": "1502799843254",
    "value": "0"
  },
  "s3PutLatency": {
    "timestamp": "1502799843254",
    "value": "130.1"
  },
  "s3PutTPS": {
    "timestamp": "1502799843254",
    "value": "0.0"
  },
  "s3PutThruput": {
    "timestamp": "1502799843254",
    "value": "0"
  },
  "status": {
    "ipaddr": "",
    "status": [
      "OK"
    ],
    "timestamp": "1502799843254",
    "value": "[OK]"
  }
}
```

7.5.4. Response Element Descriptions

Note Within the *MonitorSystemInfo* object, all statistics for which the data type (in the descriptions below) is *MonitorStat* are formatted as *"<statName>":{"timestamp":"<UTCMilliseconds>","value":"<statValue>"}*.

diskAvailKb

(MonitorStat) Across the whole service region, the total mounted disk space that's still available for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory). Reported as a number of kibibytes.

The following are deducted from the total amount of unused disk space to arrive at the "available" disk space amount that is reported by the *diskAvailKb* statistic:

- Each disk’s "reserved-blocks-percentage" (the portion of the disk that’s reserved for privileged processes)
- Each HyperStore data disk's "stop-write" buffer (10% of capacity by default). For more information on the stop-write feature see the "Automated Disk Management" section in the ***Cloudian HyperStore Administrator's Guide***.

Example:

```
"diskAvailKb": {"timestamp": "1502799843254","value": "61855592"}
```

Note Because the reserved-blocks-percentage and the stop-write buffer percentage are not counted as available space, the *diskUsedKb* and *diskAvailKb* will add up to less than the *diskTotalKb*.

Note The *diskAvailKb*value can potentially be larger than a 64 bit integer can hold.

diskTotalKb

(MonitorStat) Across the whole service region, the total size of the disks mounted for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory). Reported as a number of kibibytes.

Example:

```
"diskTotalKb": {"timestamp": "1502799843254","value": "88115680"}
```

Note This value can potentially be larger than a 64 bit integer can hold.

diskUsedKb

(MonitorStat) Across the whole service region, on disks that are mounted for S3 object storage (HyperStore data directories) or Cassandra metadata storage (Cassandra data directory), the total disk space that's used. Reported as a number of kibibytes.

Example:

```
"diskUsedKb": {"timestamp": "1502799843254","value": "23814368"}
```

Note This value can potentially be larger than a 64 bit integer can hold.

nodeStatuses

(List<*NodeStatus*>) List of *NodeStatus* objects, one for each node in the service region. Each nested *NodeStatus* object consists of the following attributes:

hostname

(String) Hostname of the node. Example:

```
"hostname": "store1"
```

ipaddr

(String) This attribute will have value *null*. Example:

```
"ipaddr": null
```

status

(List<string>) Status of the node. A list of one or more of the following strings: "OK","SVC_ DOWN","LOG_WARN", or "LOG_ERR". The meanings are:

- "OK" — All HyperStore services are up and running on the node, and the node has no unacknowledged events.
- "SVC_DOWN" — One or more HyperStore services (Admin, Cassandra, HyperStore, Redis QoS, Redis Credentials, Redis Monitor, or S3) is down on the node
- "LOG_WARN" — There are unacknowledged warnings in an application log on the node (such as the S3 Service application log or the Cassandra application log).
- "LOG_ERR" — There are unacknowledged errors in an application service log.

Example:

```
"status": ["SVC_DOWN","LOG_WARN"]
```

timestamp

(String) Status timestamp in UTC milliseconds. Example:

```
"timestamp": "1502799843254"
```

value

(String) The "value" attribute will be the same as the "status" attribute, except formatted as a single string rather than a list of strings. Example:

```
"value":"[SVC_DOWN, LOG_WARN]"
```

s3GetLatency

(MonitorStat) Across the whole service region, the 95th percentile request latency value for S3 GET transactions, in milliseconds. New statistic values are calculated each five minutes, based on the most recent approximately 1000 GET transactions. The s3GetLatency value indicates that of the last 1000 GET transactions, 95% completed in that many milliseconds or less.

Example:

```
"s3GetLatency": null
```

Note HEAD transactions are counted toward this stat also.

s3GetTPS

(MonitorStat) Across the whole service region, the number of S3 GET transactions processed per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. HEAD transactions are counted toward this stat also.

Example:

```
"s3GetTPS": null
```

s3GetThruput

(MonitorStat) Across the whole service region, the data throughput for S3 GET transactions, expressed as bytes per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. HEAD transactions are counted toward this stat also.

Example:

```
"s3GetThruput": {"timestamp": "1502799843254","value": "0"}
```

s3PutLatency

(MonitorStat) Across the whole service region, the 95th percentile request latency value for S3 PUT transactions, in milliseconds. New statistic values are calculated each five minutes, based on the most recent approximately 1000 PUT transactions. The s3PutLatency value indicates that of the last 1000 PUT transactions, 95% completed in that many milliseconds or less.

Example:

```
"s3PutLatency": {"timestamp": "1502799843254","value": "130.1"}
```

Note POST transactions are counted toward this stat also.

s3PutTPS

(MonitorStat) Across the whole service region, the number of S3 PUT transactions processed per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. POST transactions are counted toward this stat also.

Example:

```
"s3PutTPS": {"timestamp": "1502799843254","value": "0.0"}
```

s3PutThruput

(MonitorStat) Across the whole service region, the data throughput for S3 PUT transactions, expressed as bytes per second. This stat is recalculated each five minutes, based on the most recent five minutes of transaction data. POST transactions are counted toward this stat also.

Example:

```
"s3PutThruput": {"timestamp": "1502799843254","value": "0"}
```

status

(ServiceStatus) High-level service status for the system as a whole. Formatted as {"ipaddr": "<empty>","status": [<one of "OK" or "SVC_DOWN">],"timestamp": "<UTCMilliseconds>","value": "<statusValueFormattedAsString>"}.

The "ipaddr" value will be empty or null.

The "status" will be formatted as a list but with just one member string. Status string meanings are:

- "OK" — All HyperStore services are up and running on all nodes in the service region.
- "SVC_DOWN" — A HyperStore service (Admin, Cassandra, HyperStore, Redis QoS, Redis Credentials, Redis Monitor, or S3) is down on one or more nodes in the service region.

The "value" attribute will be identical to the "status" attribute, except formatted as a string rather than as a list.

Example:

```
"status": {"ipaddr": "","status": ["OK"],"timestamp": "1502799843254",
"value": "[OK]"}
```

7.5.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Invalid region : {region}

7.5.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianMonitorRegion*
- Parameters: Same as for *GET /monitor*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /monitor* except the data is formatted in XML rather than JSON
- Role-based restrictions:
    - HyperStore system admin user can use this method
    - HyperStore group admin user cannot use this method
    - HyperStore regular user cannot use this method
    - IAM user can only use this method if granted *admin:GetCloudianMonitorRegion* permission by policy, and subject to the same restriction as the parent HyperStore user
- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianMonitorRegion

<request headers including authorization info>

RESPONSE

200 OK
<?xml version="1.0" encoding="utf-8"?>
<GetCloudianMonitorRegionResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<MonitorSystemInfo>
<status>
<timestamp>1534535223489</timestamp>
etc...
...
...
</MonitorSystemInfo>
</GetCloudianMonitorRegionResponse>
```

7.6. GET /monitor/history

GET /monitor/history   Get historical monitoring statistics for a node

7.6.1. Syntax

```
GET /monitor?nodeId=string[&region=string]&statId=enum&startTime=string&endTime=string
```

There is no request payload.

7.6.2. Parameter Descriptions

nodeId

(Mandatory, string) The hostname of the target node.

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

statId

(Mandatory, enum) The statistic for which to retrieve a history. The supported statistic IDs are listed in the table below. Depending on the statistic, the returned history will include either one data point (one instance of the statistic value) per minute or one data point per five minutes, across the time interval bounded by the *startTime* and *endTime* specified in the request.

The *GET /monitor/history* call only supports one statistic ID per request. You cannot request multiple or all statistics IDs in a single request.

For more information about a particular statistic, see **"GET /monitor/host   Get current monitoring statistics for a node "** (page 81).

StatId	Data Point Frequency
diskIORead	Every minute
diskIOWrite	
ioTx	
ioRx	
cpu	Every five minutes
s3GetTPS	
s3PutTPS	
s3GetThruput	
s3PutThruput	
s3GetLatency	
s3PutLatency	
adminHeapUsed	
cassHeapUsed	
hyperStoreHeapUsed	
s3HeapUsed	

startTime

(Mandatory, string) The start time of the interval for which to retrieve the statistic history, in format *yyyyMMddHHmm* (for example 202407200000). The system interprets this as a **GMT time**, so when specifying your desired start time do it in terms of the GMT time zone -- not the local time.

endTime

(Mandatory, string) The end time of the interval for which to retrieve the statistic history, in format *yyyyMMddHHmm* (for example 202407201200). The system interprets this as a **GMT time**, so when specifying your desired end time do it in terms of the GMT time zone -- not the local time.

7.6.3. Example Using cURL

The **example** below retrieves history for the "cpu" statistic, for a one hour period (2024 July 20th, midnight to 1AM). Note that the system interprets the start and end times as **GMT** times.

```
curl -X GET -k -u sysadmin:password \

'https://localhost:19443/monitor/history?nodeId=
hs1&statId=cpu&startTime=202407201200&endTime=202407201300' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of timestamp/value pairs (with the timestamps as UTC milliseconds). Note that the response body does **not** include the statistic ID. Depending on the statistic, there will be either one timestamp/value pair per minute or one timestamp/value pair per five minutes, throughout the requested startTime / endTime interval. In this truncated example, it's one per five minutes.

```
[
  {
    "timestamp": "1563624003885",
    "value": "0.21"
  },
  {
    "timestamp": "1563624303754",
    "value": "0.21"
  },
  {
    "timestamp": "1563624603451",
    "value": "0.21"
  },
  ...
  }
]
```

7.6.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing parameter : {parameter}

Status Code	Description
400	Both regionId and nodeId are empty
400	Invalid region : {region}
400	Invalid parameter : {parameter}

7.7. GET /monitor/notificationrules

GET /monitor/notificationrules   Get the list of notification rules

7.7.1. Syntax

```
GET /monitor/notificationrules[?region=string]
```

There is no request payload.

7.7.2. Parameter Descriptions

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.7.3. Example Using cURL

The **example** below retrieves the current list of notification rules for the default service region.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/monitor/notificationrules | python -mjson.tool
```

The response payload is a JSON-formatted list of *NotificationRule* objects, which in this example is as follows. The example is truncated so that only a few rules are shown.

```
[
  {
    "condition": ">",
    "conditionVal": "0.9",
    "email": "default",
    "enabled": true,
    "region": "",
    "ruleId": "12",
    "severityLevel": 1,
    "snmpTrap": false,
    "statId": "cpu"
  },
  {
    "condition": "",
    "conditionVal": "",
    "email": "default",
    "enabled": true,
    "region": "",
    "ruleId": "19",
    "severityLevel": 3,
    "snmpTrap": false,
    "statId": "currentFailDiskInfo"
  },
  {
    "condition": "<",
    "conditionVal": "0.1",
    "email": "default",
    "enabled": true,
    "region": "",
    "ruleId": "11",
    "severityLevel": 2,
    "snmpTrap": false,
    "statId": "diskAvail"
  },
  ...
  ...
]
```

7.7.4. Response Element Descriptions

For description of *NotificationRule* object elements, see **"PUT /monitor/notificationrule Create a new notification rule"** (page 107).

7.7.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

7.8. POST /monitor/acknowledgeevents

POST /monitor/acknowledgeevents Acknowledge monitoring events

7.8.1. Syntax

```
POST /monitor/acknowledgeevents?nodeId=string[&region=string]
```

The required request payload is a JSON-formatted *EventsAck* object. See example below.

7.8.2. Parameter Descriptions

nodeId

(Mandatory, string) The hostname of the target node.

region

(Optional, string) Service region. If you do not specify a region, the default region is assumed.

7.8.3. Example Using cURL

The **example** below acknowledges two monitoring events from the node with hostname "store1" (the same two events that were retrieved in the **GET /monitor/events** example). In this example the JSON-formatted *EventsAck* object is specified in a text file named *event_acknowledge.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @event_acknowledge.txt \
https://localhost:19443/monitor/acknowledgeevents?nodeId=store1
```

The *event_acknowledge.txt* file content in this example is as follows.

```
{
  "eventTypes":["13|/dev/mapper/vg0-root","14|"],
  "nodeId":"store1"
}
```

7.8.4. Request Elements

eventTypes

(Mandatory, list<string>) List of the eventTypes being acknowledged. For non-log events (such as a service down event or a threshold crossing event), the eventType is the integer *ruleId* value from the notification rule that triggered the event. For log events (events triggered by the writing of an application log message), the eventType is formatted as "<ruleId>|<logCategory>". The specific eventTypes currently present on a node can be retrieved with the *GET /monitor/events* method.

Example:

```
"eventTypes": ["13|/dev/mapper/vg0-root","14|"]
```

nodeId

(Mandatory, string) Hostname of the node on which the event(s) occurred. Example:

```
"nodeId": "store1"
```

regionId

(Optional, string) Name of service region in which the event(s) occurred. Defaults to the default service region. Example:

```
"regionId": "southwest"
```

delete

(Optional, boolean) If this attribute is included and set to *true* then the events specified by the "eventTypes" attribute will be immediately deleted from the system.

If this attribute is omitted or set to *false* then the events specified by the "eventTypes" attribute will be marked as acknowledged but will not yet be deleted from system. Such acknowledged events will instead be deleted automatically after 86400 seconds (one day).

Example:

```
"delete": true
```

7.8.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : {nodeId, events}
400	Invalid region : {region}
400	Invalid JSON object

7.9. POST /monitor/notificationruleenable

POST /monitor/notificationruleenable   Enable or disable notification rules

7.9.1. Syntax

```
POST /monitor/notificationruleenable
```

The required request payload is a JSON-formatted *NotificationRulesEnable* object. See example below.

7.9.2. Usage Notes

You can use this method to disable notification rules or to re-enable rules that you've previously disabled. When a notification rule is disabled the rule will not trigger system event notifications.

Note To disable or re-enable just one notification rule, you can use either the *POST /monitor/notificationruleenable* method or the **POST /monitor/notificationrule** method. To disable or re-enable multiple notification rules in one operation use the *POST /monitor/notificationruleenable* method.

7.9.3. Example Using cURL

The **example** below disables two notification rules. In this example the JSON-formatted *NotificationRulesEnable* object is specified in a text file named *rule_disable.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @rule_disable.txt https://localhost:19443/monitor/notificationruleenable
```

The *rule_disable.txt* file content in this example is as follows.

```
{
  "enable":false,
  "regionId":"",
  "ruleList":["836da4bf-c6cc-4f73-afa3-9854ce407ca6","8ef63b63-4961-4e17-88c7-d53c966557db"]
}
```

There is no response payload.

7.9.4. Request Element Descriptions

enable

(Mandatory, boolean) Set to *true* to enable the rule(s). Set to *false* to disable the rule(s). Example:

```
"enable":false
```

regionId

(Optional, string) Service region in which the rules are configured. If you do not specify a region, the default region is assumed. Example:

```
"regionId":""
```

ruleList

(Mandatory, list<string>) List of ruleId(s) of the notification rule(s) to enable or disable.

If you do not know the ruleIds of the rules that you want to enable/disable, you can retrieve them by using the **GET /monitor/notificationrules** method. That method returns a list of rules, which includes each rule's ruleId.

Example:

```
"ruleList":["836da4bf-c6cc-4f73-afa3-9854ce407ca6",
"8ef63b63-4961-4e17-88c7-d53c966557db"]
```

7.9.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {ruleId}
400	Notification rule does not exist : {ruleId}

7.10. POST /monitor/notificationrule

POST /monitor/notificationrule Change a notification rule

7.10.1. Syntax

```
POST /monitor/notificationrule
```

The required request payload is a JSON-formatted *NotificationRule* object. See example below.

7.10.2. Example Using cURL

The **example** below changes an existing notification rule (the rule that was created in the **PUT /monitor/notificationrule** description). In this example the JSON-formatted *NotificationRule* object is specified in a text file named *rule_s3GetLatency.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @rule_s3GetLatency.txt https://localhost:19443/monitor/notificationrule
```

The *rule_s3GetLatency.txt* file content in this example is as follows.

```
{
  "condition":">",
  "conditionVal":"150",
  "email":"default",
  "enabled":true,
  "region":"",
  "ruleId":"836da4bf-c6cc-4f73-afa3-9854ce407ca6",
  "severityLevel": 2,
  "snmpTrap":false,
  "statId":"s3GetLatency"
}
```

Note Unlike when you create a new notification rule, when you change an existing rule you must specify the rule's "ruleId" value in the *NotificationRule* object. If you're not sure of a rule's ID you can retrieve it using the **GET /monitor/notificationrules** method.

7.10.3. Request Element Descriptions

For description of *NotificationRule* object elements, see **"PUT /monitor/notificationrule Create a new notification rule"** (page 107).

7.10.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Invalid JSON object
400	Notification rule Id does not exist : {ruleId}

7.11. PUT /monitor/notificationrule

PUT /monitor/notificationrule Create a new notification rule

7.11.1. Syntax

```
PUT /monitor/notificationrule
```

The required request payload is a JSON-formatted *NotificationRule* object. See example below.

7.11.2. Usage Notes

The HyperStore system comes with many pre-configured notification rules. Before creating new rules, you can review the existing pre-configured rules by going to the CMC's **Alert Rules** page.

> **Note** The CMC interface uses the term "alert rules" rather than "notification rules".

7.11.3. Example Using cURL

The **example** below creates a new notification rule. In this example the JSON-formatted *NotificationRule* object is specified in a text file named *rule_s3GetLatency.txt* which is then referenced as the data input to the cURL command.

```
curl -X PUT -H "Content-Type: application/json" -k -u sysadmin:password \
-d @rule_s3GetLatency.txt https://localhost:19443/monitor/notificationrule
```

The *rule_s3GetLatency.txt* file content in this example is as follows.

```
{
  "condition":">",
  "conditionVal":"150",
  "email":"default",
  "enabled":true,
  "region":"",
  "ruleId":"",
  "severityLevel": 1,
  "snmpTrap":false,
  "statId":"s3GetLatency"
}
```

7.11.4. Request Element Descriptions

condition

(Mandatory, string) Comparator used in defining this rule: can be ">", "<", or "=". Example:

```
"condition": ">"
```

conditionVal

(Mandatory, string) Value against which to compare. The value of the statistic specified by statId will be compared to the conditionVal to determine whether a notification is called for. This statistic monitoring occurs on each node.

For example for a rule that triggers notifications if CPU utilization on any individual node exceeds 90%, the "statId" would be "cpu", the "condition" would be ">", and the "conditionVal" would be ".9".

Example:

```
"conditionVal": "0.9"
```

email

(Optional, string) Comma-separated list of email addresses to receive notifications. Or to use the default email address list (as configured on the CMC's **Configuration Settings** page [**Cluster -> Cluster Config**

-> Configuration Settings]), set this to the string "default".

To not have email notifications as part of the rule (for instance, if the rule is only meant to trigger SNMP traps), set the "email" attribute to empty ("").

This defaults to empty ("") if the attribute is omitted in a *PUT /monitor/notificationrule* request.

Example:

```
"email": "default"
```

enabled

(Mandatory, boolean) Whether the rule is enabled, *true* or *false*.

This attribute defaults to *false* if the attribute is omitted in a *PUT /monitor/notificationrule* request.

Example:

```
"enabled": true
```

region

(Optional, string) In a *PUT /monitor/notificationrule* request, use this attribute to specify the service region in which to create the notification rule. To create the rule in the default region set this attribute to the default region name or to empty (""). If you omit the "region" attribute in a PUT, then by default the notification rule is created in the default region.

In a *GET /monitor/notificationrules* response, the "region" attribute will be present but set to empty. The client application will be aware of what region the retrieved rules are from because the desired region is specified in the GET request line.

Example:

```
"region": ""
```

ruleId

(Mandatory, string) System-generated unique ID for this rule. For the default notification rules that come packaged with the HyperStore system this will be a simple integer like "1", "2", or "14". For rules that you create yourself the system will generate a ruleId in the form of a UUID string like "8e4cc533-360a-4dd5-bfe4-6b5f5b6c40da".

In a PUT (when you are creating a new rule), include the "ruleId" attribute and set it to empty (""). The system will subsequently generate a ruleId upon rule creation.

In a POST (when you are updating an existing rule), set the "ruleId" attribute to the ruleId of the rule you want to update. To find out what the ruleId is for a particular rule, use the *GET /monitor/notificationrules* method.

Example:

```
"ruleId": "12"
```

severityLevel

(Mandatory, integer) Severity level to assign to the event. This is an integer with meaning as follows:

- 0 = Low
- 1 = Medium

- 2 = High
- 3 = Critical

Example:

```
"severityLevel": 1
```

snmpTrap

(Optional, boolean) Whether to transmit an SNMP trap as part of the notification when this rule is triggered, *true* or *false*. If a trap is sent it is sent to the destination configured on the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

This defaults to false if the attribute is omitted in a *PUT /monitor/notificationrule* request.

Example:

```
"snmpTrap": false
```

statId

(Mandatory, string) ID of the node statistic being checked for this rule. The table below lists statistics for which notification rules can be defined. These statistics are monitored on **each node**. Note that the sample "conditionVal" column is not intended to suggest suitable values upon which to base notification rules but simply to illustrate the applicable data format. The right-most column indicates whether a rule for that *statId* already exists, as part of the default set of notification rules that come pre-packaged with the HyperStore system.

Example:

```
"statId": "cpu"
```

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
s3GetTPS	Number of S3 GET transactions per second on the node.	">"	"100"	No
s3PutTPS	Number of S3 PUT transactions per second on the node.	">"	"100"	No
s3PutThruput	Number of bytes of throughput per second for S3 PUT operations on the node.	">"	"100000"	No
s3GetThruput	Number of bytes of throughput per second for S3 GET operations on the node.	">"	"100000"	No
s3PutLatency	Recent average latency for S3 PUT operations on the node, in number of milliseconds.	">"	"100"	No
s3GetLatency	Recent average latency for S3 GET operations on the node, in number of milliseconds.	">"	"100"	No

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
diskAvail	Of the node's total disk space allocated to HyperStore data storage, the portion of disk space that's still free. Expressed as a decimal value.	"<"	".1"	Yes, for < .1
diskInfo	On each individual disk that is allocated to HyperStore data storage, the portion of disk space that's still free. Expressed as a decimal. This rule triggers a notification if any individual disk on the node crosses the defined threshold.	"<"	".15"	Yes, for < .15
repairCompletionStatus	When this type of rule is set, a notification is triggered any time that an auto-repair completes. The notification includes the auto-repair's final status: COMPLETED, FAILED, or TERMINATED.	Set to empty ("")	Set to empty ("")	Yes
cpu	Current CPU utilization level as a decimal value.	">"	".9"	Yes, for > .9
ioRx	Total network bytes per second received by the node (S3 traffic plus any other network traffic to the node).	">"	"100000000"	No
ioTx	Total network bytes per second transmitted by the node (S3 traffic plus any other network traffic from the node).	">"	"100000000"	No
diskIORead	Bytes per second for disk reads on the node.	">"	"1000000"	No
diskIOWrite	Bytes per second for disk writes on the node.	">"	"1000000"	No
svcAdmin	Admin service status. One of {SVC_DOWN, LOG_WARN, LOG_ERR}. Note that for this and the other "svc<ServiceType>" statIds, you have the option of creating multiple rules — for example, one rule for status "SVC_DOWN" and	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_ERR

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
	a second separate rule for status "LOG_ERR". Do not specify multiple service values in a single notification rule.			
svcCassandra	Cassandra service status. One of {SVC_DOWN, LOG_WARN, LOG_ERR}.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_ERR
svcHyperStore	HyperStore service status. One of {SVC_DOWN, LOG_WARN, LOG_ERR}.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_ERR
svcRedisCred	Redis Credentials service status. One of {SVC_DOWN, LOG_WARN}.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_WARN
svcRedisQos	Redis QoS service status. One of {SVC_DOWN, LOG_WARN}.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_WARN
svcRedisMon	Redis Monitor service status. Only supported value is "SVC_DOWN".	"="	"SVC_DOWN"	Yes, for SVC_DOWN
svcS3	S3 service status. One of {SVC_DOWN, LOG_WARN, LOG_ERR}.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
	Note Along with log warnings and errors pertaining to providing S3 service to clients, the S3 service alerts category includes log warnings and errors pertaining to auto-tiering and cross-region replication.			rule for LOG_ ERR

7.11.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Invalid JSON object

This page left intentionally blank

Chapter 8. permissions

The Admin API methods built around the **permissions** resource are for creating, changing, or retrieving public URL permissions for an object that's stored in the HyperStore system. When you create a public URL for an object, the object can then be accessed at that URL by a regular web browser.

Methods associated with the *permissions* resource:

- **"GET /permissions/publicUrl"** (page 115)
- **"POST /permissions/publicUrl"** (page 117)

8.1. GET /permissions/publicUrl

GET /permissions/publicUrl Get public URL permissions for an object

8.1.1. Syntax

```
GET /permissions/publicUrl?userId=string&groupId=string&bucketName=string&objectName=string
```

There is no request payload.

8.1.2. Parameter Descriptions

userId

(Mandatory, string) User ID for user who owns the object.

groupId

(Mandatory, string) Group ID for user who owns the object.

bucketName

(Mandatory, string) S3 bucket that contains the object. Note that the bucket’s owner must be the same as the object owner or the request will be rejected with a 400 error response.

objectName

(Mandatory, string) Name of the object for which the public URL is being retrieved.

8.1.3. Usage Notes

Use this method to retrieve existing public URL permissions for an object (public URL permissions that have already been created with the **POST /permissions/publicUrl** method).

8.1.4. Example Using cURL

The **example** below retrieves an existing public URL for an object named *ob*.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/permissions/publicUrl?userId=u1&groupId=g1&bucketName=b1&objectName=ob' \
| python -mjson.tool
```

The response payload is a JSON-formatted *PublicUrlAccess* object, which in this example is as follows.

```
{
  "allowRead": true,
  "currentDownloads": 0,
  "expiryTime": "1517385600000",
  "maxDownloadNum": 1000,
  "secure": true,
  "url": "https://s3-region1.mycloudianhyperstore.com/b1/ob?
AWSAccessKeyId=00b3ec480eb5c844fb88&Expires=1517385600&Signature=
rxxJnEWoUusrj1kQ02A9PMcFQ4U%3D&x-amz-pt=MDAzMTE1NjIxNTE2MTE4NzIxMDc1"
}
```

8.1.5. Response Element Descriptions

allowRead

(Optional, boolean) Whether a public URL is enabled for the associated object, *true* or *false*. Defaults to *true*. Example:

```
"allowRead": true
```

currentDownloads

(Optional, number) Current total number of times that the object has been downloaded via public URL. This value is set by the system, not by the client. Starts at 0 for a new public URL. Example:

```
"currentDownloads": 0
```

expiryTime

(Mandatory, string) Expiration date-time of the public URL in UTC milliseconds. After this date-time the public URL will no longer work. Example:

```
"expiryTime": "1517385600000"
```

maxDownloadNum

(Optional, number) Maximum number of times that the system will allow the object to be downloaded via public URL, by all users in total. To allow unlimited downloads, set this to "-1". Defaults to 1000. Example:

```
"maxDownloadNum": 1000
```

secure

(Optional, boolean) Whether the object’s public URL should use HTTPS rather than HTTP, *true* or *false*. Defaults to *true*. Example:

```
"secure": true
```

url

(Optional, string) System-generated public URL for accessing the object.

With a POST request:

- To create a new public URL for an object do not include the "url" attribute in the request body.
- To change permission attributes for an existing public URL, use the "url" attribute in the request body to specify the existing public URL.

The public URL for an object will have the following format:

```
http[s]://<bucketName>.<S3Domain>/<objectName>?AWSAccesKeyId=
<accessKeyOfObjectOwner>&Expires=<expiryTime>&Signature=<signatureString>&
x-amz-pt=<internalTrackingCode>
```

This format follows the AWS specification for signed URLs.

Example:

```
"url": "https://s3-region1.mycloudianhyperstore.com/bkt1/Cloudian.pdf?
AWSAccessKeyId=00b3ec480eb5c844fb88&Expires=1517385600&Signature=
rxxJnEWoUusrj1kQ02A9PMcFQ4U%3D&x-amz-pt=MDAzMTE1NjIxNTE2MTE4NzIxMDc1"
```

8.1.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : {userId, groupId, bucketName, objectName}
400	User does not exist

8.2. POST /permissions/publicUrl

POST /permissions/publicUrl   Create or change public URL permissions for an object

8.2.1. Syntax

```
POST /permissions/publicUrl?userId=string&groupId=string&bucketName=string&objectName=string
```

The required request payload is a JSON-formatted *PublicUrlAccess* object. See "Example Using cURL" below.

8.2.2. Parameter Descriptions

userId

(Mandatory, string) User ID for user who owns the object.

groupId

(Mandatory, string) Group ID for user who owns the object.

bucketName

(Mandatory, string) S3 bucket that contains the object. Note that the bucket's owner must be the same as the object owner or the request will be rejected with a 400 error response.

objectName

(Mandatory, string) Name of the object for which a public URL is being generated.

8.2.3. Usage Notes

Use this method to create or update public URL permissions for an object that's stored in the HyperStore system. See the Example section below for the distinction between creating a new public URL and updating an existing one.

For this method to work, the object owner must also be the bucket owner. Also, the method will not work if the object has been encrypted using a user-managed encryption key (SSE-C).

The public URL for an object will have the following format:

```
http[s]://<bucketName>.<S3Domain>/<objectName>?AWSAccesKeyId=<accessKeyOfObjectOwner>
&Expires=<expiryTime>&Signature=<SignatureString>&x-amz-pt=<>
```

This format follows the AWS specification for signed URLs.

8.2.4. Example Using cURL

The **example** below creates a public URL for an object named *ob*. In this example the JSON-formatted *PublicUrlAccess* object is specified in a text file named *postPublicUrlAccess.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @postPublicUrlAccess.txt \
'https://localhost:19443/permissions/publicUrl?userId=u1&groupId=g1&bucketName=b1&objectName=ob'
```

The *postPublicUrlAccess.txt* file content in this example is as follows.

```
{
  "allowRead": true,
  "expiryTime": "1517385600000",
  "maxDownloadNum": 1000,
  "secure": true
}
```

Note If the *PublicUrlAccess* JSON object supplied in the POST request body does not include a "url" attribute -- as it does not in the example above -- the POST request is processed as a request to generate a **new** public URL. If a "url" attribute value is included in the *PublicUrlAccess* object and set to equal an existing public URL, the POST request is processed as an update to the permissions associated with the existing public URL (such as an update of the expiration date-time or the maximum allowed downloads limit).

To retrieve a public URL that you've just created or that you've created previously, use the **GET /permissions/publicUrl** method.

There is no response payload.

8.2.5. Request Element Descriptions

allowRead

(Optional, boolean) Whether a public URL is enabled for the associated object, *true* or *false*. Defaults to *true*. Example:

```
"allowRead": true
```

expiryTime

(Mandatory, string) Expiration date-time of the public URL in UTC milliseconds. After this date-time the public URL will no longer work. Example:

```
"expiryTime": "1517385600000"
```

maxDownloadNum

(Optional, number) Maximum number of times that the system will allow the object to be downloaded via public URL, by all users in total. To allow unlimited downloads, set this to "-1". Defaults to 1000. Example:

```
"maxDownloadNum": 1000
```

secure

(Optional, boolean) Whether the object's public URL should use HTTPS rather than HTTP, *true* or *false*. Defaults to *true*. Example:

```
"secure": true
```

8.2.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : {userId, groupId, bucketName, objectName}
400	User does not exist
400	Invalid JSON object

This page left intentionally blank

Chapter 9. qos

The Admin API methods built around the **qos** resource are for managing HyperStore quality of service (QoS) controls. These controls set limits on service usage by user groups and by individual users. There are API methods for assigning, retrieving, or deleting QoS settings for specified users or groups.

For an overview of the HyperStore quality of service feature, see "Quality of Service (QoS) Feature Overview" in the ***Cloudian HyperStore Administrator's Guide***.

Methods associated with the *qos* resource:

- **"DELETE /qos/limits"** (page 121)
- **"GET /qos/limits"** (page 122)
- **"POST /qos/limits"** (page 127)

9.1. DELETE /qos/limits

DELETE /qos/limits   Delete QoS settings for a user or group

9.1.1. Syntax

```
DELETE /qos/limits?userId=string&groupId=string[&region=string]
```

There is no request payload.

9.1.2. Parameter Descriptions

userId

(Mandatory, string) User ID of the user to whom the QoS settings apply. Supported options are a specific user ID, "ALL", or "*".

See the "groupId" description below for information about how to use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings.

groupId

(Mandatory, string) Group ID of the group to which the QoS settings apply.

You can use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings:

- User-level QoS for a specific user (userId=<*userId*>&groupId=<*groupId*>)
- Default user-level QoS for a specific group (userId=*ALL*&groupId=<*groupId*>)
- Group-level QoS for a specific group (userId=*&groupId=<*groupId*>)

region

(Optional, string) Service region to which the QoS settings apply. If not specified, the default region is assumed.

9.1.3. Usage Notes

Use this method to:

- Delete QoS limits that have been assigned to a specific user. If you delete user-specific QoS limits, the system will automatically assign the user the default user-level QoS limits associated with the group to which the user belongs.
- Delete QoS limits that have been assigned to a specific group. If you delete group-specific QoS limits, the system will automatically assign the group the regional default QoS limits.

Essentially, this method allows you to clear user-specific or group-specific QoS overrides so that default QoS settings are used instead.

9.1.4. Example Using cURL

The **example** below deletes QoS settings for the "Dev" group. With these group-specific settings deleted, the default group QoS settings for the service region will be applied to the Dev group.

```
curl -X DELETE -k -u sysadmin:password \
'https://localhost:19443/qos/limits?userId=*&groupId=Dev'
```

9.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {userId, groupId}
400	Region {region} is invalid

9.2. GET /qos/limits

GET /qos/limits   Get QoS settings for a user or group

9.2.1. Syntax

```
GET /qos/limits?userId=string&groupId=string[&region=string]
```

There is no request payload.

9.2.2. Parameter Descriptions

userId

(Mandatory, string) User ID of the user to whom the QoS settings apply. Supported options are a specific user ID, "ALL", or "*".

See the *groupId* description below for information about how to use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings.

groupId

(Mandatory, string) Group ID of the group to which the QoS settings apply. Supported options are a specific group ID, "ALL", or "*".

You can use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings:

- User-level QoS for a specific user (userId=<*userId*>&groupId=<*groupId*>)
- Default user-level QoS for a specific group (userId=*ALL*&groupId=<*groupId*>)
- Default user-level QoS for the whole region (userId=*ALL*&groupId=*)
- Group-level QoS for a specific group (userId=*&groupId=<*groupId*>)
- Default group-level QoS for the whole region (userId=*&groupId=*ALL*)

region

(Optional, string) Service region to which the QoS settings apply. If not specified, the default region is assumed.

9.2.3. Example Using cURL

The **example** below retrieves current QoS settings for the user "cody" in the "Dev" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/qos/limits?userId=cody&groupId=Dev' \
| python -mjson.tool
```

The response payload is a JSON-formatted *QosLimitSettings* object, which in this example is as follows.

```
{
  "groupId": "Dev",
  "labelId": "qos.userQosOverrides.title",
  "qosLimitList": [
    {
      "type": "STORAGE_QUOTA_KBYTES_LW",
      "value": -1
    },    {
      "type": "STORAGE_QUOTA_KBYTES_LH",
      "value": 1073741824
    },
    {
      "type": "REQUEST_RATE_LW",
      "value": -1
    },
    {
      "type": "REQUEST_RATE_LH",
      "value": -1
    },
    {
      "type": "DATAKBYTES_IN_LW",
      "value": -1
    },
    {
      "type": "DATAKBYTES_IN_LH",
      "value": -1
    },
    {
      "type": "DATAKBYTES_OUT_LW",
      "value": -1
    },
    {
      "type": "DATAKBYTES_OUT_LH",
      "value": -1
    },
    {
      "type": "STORAGE_QUOTA_COUNT_LW",
      "value": -1
    },
    {
      "type": "STORAGE_QUOTA_COUNT_LH",
      "value": -1
    }
  ],
  "userId": "cody"
}
```

9.2.4. Response Element Descriptions

groupId

(String) Group ID. This will be either a specific group ID, or "ALL", or "*". For details of how this is used, see **"Parameter Descriptions"** (page 122).

Example:

```
"groupId": "Dev"
```

labelId

(String) This attribute is used by the CMC to display the correct title on a group or user QoS configuration screen. Example:

```
"labelId": "qos.userQosOverrides.title"
```

qosLimitList

(List<*QosLimit*>) List of *QosLimit* objects. There will be one *QosLimit* object for each of the eight QoS limit types. Each *QosLimit* object consists of the following attributes:

type

(String) One of the following QoS limit types:

- STORAGE_QUOTA_KBYTES (possible legacy value from systems originally installed as 7.4.x or older; equivalent to STORAGE_QUOTA_KBYTES_LH)
- STORAGE_QUOTA_KBYTES_LW
- STORAGE_QUOTA_KBYTES_LH
- STORAGE_QUOTA_COUNT (possible legacy value from systems originally installed as 7.4.x or older; equivalent to STORAGE_QUOTA_COUNT_LH)
- STORAGE_QUOTA_COUNT_LW

- STORAGE_QUOTA_COUNT_LH
- REQUEST_RATE_LW
- REQUEST_RATE_LH
- DATAKBYTES_IN_LW
- DATAKBYTES_IN_LH
- DATAKBYTES_OUT_LW
- DATAKBYTES_OUT_LH

For descriptions of these QoS limits see **"POST /qos/limits Create QoS settings for a user or group"** (page 127), specifically the Parameter Descriptions section. Note that as request parameters these limits have slightly different names than they have as JSON object attributes. For example the limit named "hlStorageQuotaKBytes" as a request parameter for the *POST /qos/limits* call is named "STORAGE_QUOTA_KBYTES_LH" as a response body element for the *GET /qos/limits* call.

Note The limits having to do with data size are in number of kibibytes (KiBs), not number of kilobytes (KBs).

Example:

```
"type": "STORAGE_QUOTA_KBYTES_LH"
```

value

(Number) The value assigned to this QoS limit type. A value of -1 indicates that the limit is disabled. Example:

```
"value": 1073741824
```

userId

(String) User ID. This will be either a specific user ID, or "ALL", or "*". For details of how this is used, see **"Parameter Descriptions"** (page 122). Example:

```
"userId": "cody"
```

9.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	User does not exist
400	Missing required parameter : {userId, groupId}
400	Region {region} is invalid

9.2.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianQosLimits*
- Parameters: Same as for *GET /qos/limits*, except parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /qos/limits* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can get QoS limits for any group or user
  - HyperStore group admin user can only get QoS limits for own group or users within own group
  - HyperStore regular user can only get own QoS limits
  - IAM user can only use this method if granted *admin:GetCloudianQosLimits* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

  Note The "GetCloudianQosLimits" action retrieves QoS limit information for Cloudian HyperStore user accounts, not for subsidiary IAM users. The system does not maintain QoS limits per IAM user. For example, if a HyperStore group administrator grants *admin:GetCloudianQosLimits* permission to an IAM user, the IAM user will be able to retrieve QoS limits for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianQosLimits* permission to an IAM user, the IAM user will be able to retrieve QoS limits for the **parent HyperStore user**.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianQosLimits&UserId=cody&GroupId=Dev

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianQosLimitsResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<UserQosOverrides>
<groupId>Pubs</groupId>
etc...
...
...
</UserQosOverrides>
<GetCloudianQosLimitsResponse>
```

9.3. POST /qos/limits

POST /qos/limits Create QoS settings for a user or group

9.3.1. Syntax

```
POST /qos/limits?userId=string&groupId=string&wlStorageQuotaKBytes=integer
&hlStorageQuotaKBytes=integer&wlStorageQuotaCount=integer&hlStorageQuotaCount=integer
&wlRequestRate=integer&hlRequestRate=integer&wlDataKBytesIn=integer&hlDataKBytesIn=integer
&wlDataKBytesOut=integer&hlDataKBytesOut=integer[&region=string]
```

There is no request body payload.

9.3.2. Parameter Descriptions

userId

(Mandatory, string) User ID of the user to whom the QoS settings apply. Supported options are a specific user ID, "ALL", or "*".

See the *groupId* description below for information about how to use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings.

groupId

(Mandatory, string) Group ID of the group to which the QoS settings apply. Supported options are a specific group ID, "ALL", or "*".

You can use the "userId" and "groupId" parameters in combination to designate different levels of QoS settings:

- User-level QoS for a specific user (userId=<*userId*>&groupId=<*groupId*>)
- Default user-level QoS for a specific group (userId=*ALL*&groupId=<*groupId*>)
- Default user-level QoS for the whole region (userId=*ALL*&groupId=*)
- Group-level QoS for a specific group (userId=*&groupId=<*groupId*>)
- Default group-level QoS for the whole region (userId=*&groupId=*ALL*)

wlStorageQuotaKBytes

(Mandatory, integer) Storage quota warning threshold ("soft limit"), in number of KiBs

Implementation detail:

- *For user QoS* — If a user’s total stored data reaches this limit, a WARN level message is written to the S3 Service's application log.
- *For group QoS* — If a group’s total stored data reaches this limit, a WARN level message is written to the S3 Service's application log.

hlStorageQuotaKBytes

(Mandatory, integer) Storage quota maximum ("hard limit"), in number of KiBs

Implementation detail:

- *For user QoS* — If a user’s total stored data reaches this limit, the user will be blocked from uploading additional data until she deletes some of her currently stored data. Also, a WARN level message is written to the S3 Service's application log.
- *For group QoS* — If a group’s total stored data reaches this limit, all of the group’s users will be blocked from uploading additional data until some of their currently stored data is deleted. Also, a WARN level message is written to the S3 Service's application log.

wlStorageQuotaCount

(Mandatory, integer) Storage quota warning threshold ("soft limit"), in total number of objects. Note that folders count as objects, as well as files.

Implementation detail:

- *For user QoS* — If a user’s total stored data reaches this limit, a WARN level message is written to the S3 Service's application log.
- *For group QoS* — If a group’s total stored data reaches this limit, a WARN level message is written to the S3 Service's application log.

hlStorageQuotaCount

(Mandatory, integer) Storage quota maximum ("hard limit"), in total number of objects. Note that folders count as objects, as well as files.

Implementation detail:

- *For user QoS* — If a user’s total stored data reaches this limit, the user will be blocked from uploading additional data until she deletes some of her currently stored data. Also, a WARN level message is written to the S3 Service's application log.
- *For group QoS* — If a group’s total stored data reaches this limit, all of the group’s users will be blocked from uploading additional data until some of their currently stored data is deleted. Also, a WARN level message is written to the S3 Service's application log.

wlRequestRate

(Mandatory, integer) Request rate warning threshold ("soft limit"), in total number of HTTP requests per minute.

Implementation detail:

- *For user QoS* — On receipt of a first HTTP request from a user, a 60 second timer is started for that user. If during the 60 seconds the total number of requests reaches the Request Rate Warning Limit, a WARN level message is written to the S3 Service’s application log, and requests from the user continue to be fulfilled. At the end of the 60 seconds, the request counter for the user is reset. Subsequently, the next request that comes in from the user triggers the start of a new 60 second interval, and the process repeats.
- *For group QoS* — The implementation is the same as for user QoS, except that it applies to requests from all users in the group, in the aggregate. For example, a request from any user in the group triggers the start of the 60 second timer, and subsequent requests from any user in the group are counted toward the per-minute total.

HTTP DELETE requests are not counted toward Request Rate controls.

hlRequestRate

(Mandatory, integer) Request rate maximum ("hard limit"), in total number of HTTP requests per minute.

Implementation detail:

- *For user QoS* — On receipt of a first request from a user, a 60 second timer is started for that user (the same timer described in Request Rate Warning Limit). If during the 60 seconds the number of requests reaches Request Rate High Limit, the system temporarily blocks all requests from the user (and also writes a WARN level message to the S3 Service’s application log). At the end of the 60 seconds the block on requests is released and the request counter is reset. Subsequently, the next request that comes in from the user triggers the start of a new 60 second interval, and the process repeats.
- *For group QoS* — The implementation is the same as for user QoS, except that it applies to requests from all users in the group, in the aggregate. For example, a request from any user in the group triggers the start of the 60 second timer, and subsequent requests from any user in the group are counted toward the per-minute total. If a block is triggered by the high limit being reached, the block applies to all users in the group.

wlDataKBytesIn

(Mandatory, integer) Inbound data rate warning threshold ("soft limit"), in KiBs per minute.

This works the same as described for the Request Rate Warning Limit, except what’s counted during each timed 60 second interval is inbound kibibytes of data.

hlDataKBytesIn

(Mandatory, integer) Inbound data rate maximum ("hard limit"), in KiBs per minute.

This works the same as described for the Request Rate High Limit, except what’s counted during each timed 60 second interval is inbound kibibytes of data. Note that if a block is triggered by the Data Bytes IN High Limit being reached, the block applies to all HTTP request types (not just PUTs.)

wlDataKBytesOut

(Mandatory, integer) Outbound data rate warning threshold ("soft limit"), in KiBs per minute.

This works the same as described for the Request Rate Warning Limit, except what’s counted during each timed 60 second interval is outbound kibibytes of data.

wlDataKBytesOut

(Mandatory, integer) Outbound data rate maximum ("hard limit"), in KiBs per minute.

This works the same as described for the Request Rate High Limit, except what’s counted during each timed 60 second interval is outbound kibibytes of data. Note that if a block is triggered by the Data Bytes OUT High Limit being reached, the block applies to all HTTP request types (not just GETs.)

region

(Optional, string) Service region to which the QoS settings apply. If not specified, the default region is assumed.

9.3.3. Usage Notes

This method creates user-level or group-level QoS settings. User-level QoS settings place an upper limit on the storage utilization and transaction activities of individual users, while group-level QoS settings place such limits on entire user groups.

You must include each of the QoS type query parameters, even those types for which you do not want to set a limit. To disable a type set it to "-1".

In a multi-region HyperStore deployment, you must establish QoS limits separately for each region. The QoS limits that you establish for a region will be applied only to activity in that region.

IMPORTANT ! By default the HyperStore system's enforcement of QoS restrictions is **disabled**. For information about enabling the QoS feature -- and information about the option of having the system generate alerts when users hit QoS thresholds -- see "Enabling QoS Enforcement and Alerting" in the "Setting Up Quality of Service Controls" section in the ***Cloudian HyperStore Administrator's Guide***.

Note The system does not notify users who have exceeded a QoS Warning Limit.

9.3.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {userId, groupId, wlStorageQuotaKBytes, hlStorageQuotaKBytes, wlStorageQuotaCount, hlStorageQuotaCount, wlRequestRate, hlRequestRate, wlDataKBytesIn, hlDataKBytesIn, wlDataKBytesOut, hlDataKBytesOut}
400	Invalid parameter
400	Region {region} is invalid

Chapter 10. ratingPlan

The Admin API methods built around the **ratingPlan** resource are for managing HyperStore rating plans. Rating plans assigning pricing to various types and levels of service usage, in support of billing users or charging back to an organization's business units. There are methods for creating, changing, and deleting rating plans.

Note By default the system only supports billing based on number of stored bytes. If you want your rating plans to include billing based on request rates or data transfer rates you must enable the "Track-/Report Usage for Request Rates and Data Transfer Rates" setting in the CMC's **Configuration Settings** page (**Cluster -> Cluster Config -> Configuration Settings**).

Methods associated with the *ratingPlan* resource:

- **"DELETE /ratingPlan"** (page 131)
- **"GET /ratingPlan"** (page 132)
- **"GET /ratingPlan/list"** (page 134)
- **"POST /ratingPlan"** (page 135)
- **"PUT /ratingPlan"** (page 136)

10.1. DELETE /ratingPlan

DELETE /ratingPlan   Delete a rating plan

10.1.1. Syntax

```
DELETE /ratingPlan?ratingPlanId=string
```

There is no request payload.

10.1.2. Parameter Descriptions

ratingPlanId

(Mandatory, string) Unique identifier of the rating plan.

10.1.3. Example Using cURL

The **example** below deletes a rating plan with ID "Plan-6".

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/ratingPlan?ratingPlanId=Plan-6
```

10.1.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Rating Plan does not exist
400	Missing required parameter : {ratingPlanId}

10.2. GET /ratingPlan

GET /ratingPlan   Get a rating plan

10.2.1. Syntax

```
GET /ratingPlan?ratingPlanId=string
```

There is no request payload.

10.2.2. Parameter Descriptions

ratingPlanId

(Mandatory, string) Unique identifier of the rating plan.

10.2.3. Example Using cURL

The **example** below retrieves the system default rating plan, which has ID "Default-RP".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/ratingPlan?ratingPlanId=Default-RP python -mjson.tool
```

The response payload is a JSON-formatted *RatingPlan* object, which in this example is as follows.

```
{
  "currency": "USD",
  "id": "Default-RP",
  "mapRules": {
    "BI": {
      "ruleclassType": "BYTES_IN",
      "rules": [
        {
          "first": "1",
          "second": "0.20"
        },
        {
          "first": "0",
          "second": "0.10"
        }
      ]
    },
    "BO": {
      "ruleclassType": "BYTES_OUT",
      "rules": [
        {
          "first": "1",
          "second": "0.20"
        },
        {
          "first": "0",
          "second": "0.10"
        }
      ]
    },
    "HD": {
      "ruleclassType": "HTTP_DELETE",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HG": {
      "ruleclassType": "HTTP_GET",
      "rules": [
        {
          "first": "10",
          "second": "0.02"
        },
        {
          "first": "0",
          "second": "0.01"
        }
      ]
    },
    "HP": {
      "ruleclassType": "HTTP_PUT",
      "rules": [
        {
          "first": "0",
          "second": "0.02"
        }
      ]
    },
    "SB": {
      "ruleclassType": "STORAGE_BYTE",
      "rules": [
        {
          "first": "1",
          "second": "0.14"
        },
        {
          "first": "5",
          "second": "0.12"
        },
        {
          "first": "0",
          "second": "0.10"
        }
      ]
    }
  },
  "name": "Default Rating Plan"
}
```

10.2.4. Response Element Descriptions

For *RatingPlan* element descriptions see **"PUT /ratingPlan Create a new rating plan"** (page 136)

10.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Rating Plan does not exist
400	Missing required parameter: {ratingPlanId}

10.3. GET /ratingPlan/list

GET /ratingPlan/list Get the list of rating plans in the system

10.3.1. Syntax

```
GET /ratingPlan/list
```

There is no request payload.

10.3.2. Example Using cURL

The **example** below retrieves the list of rating plans that are in the system. Note that this method does not return the full content of the rating plans -- just the ID, name, and currency for each plan.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/ratingPlan/list | python -mjson.tool
```

The response payload in this example is as follows.

```
[
  {
    "currency": "USD",
    "encodedId": "Default-RP",
    "id": "Default-RP",
    "name": "Default Rating Plan"
  },
  {
    "currency": "USD",
    "encodedId": "Whitelist-RP",
    "id": "Whitelist-RP",
    "name": "Whitelist Rating Plan"
  }
]
```

Note The "encodedId" value is a URL-encoding of the "id" value.

10.3.3. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

10.4. POST /ratingPlan

POST /ratingPlan   Change a rating plan

10.4.1. Syntax

```
POST /ratingPlan
```

The required request payload is a JSON-formatted *RatingPlan* object.

10.4.2. Example Using cURL

The **example** below modifies the rating plan that was created in the **PUT /ratingPlan** example. Again the *RatingPlan* object is specified in a text file named *ratingStorageOnly.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @ratingStorageOnly.txt https://localhost:19443/ratingPlan
```

Note that in editing the *RatingPlan* object in the *ratingStorageOnly.txt* file you could edit any attribute except for the "id" attribute. The "id" attribute must remain the same, so that you're modifying an existing rating plan rather than creating a new one. For an example *RatingPlan* object see **PUT /ratingPlan**.

10.4.3. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Rating Plan does not exist
400	Missing required parameter : {id, name}
400	Invalid JSON Object

10.5. PUT /ratingPlan

PUT /ratingPlan Create a new rating plan

10.5.1. Syntax

```
PUT /ratingPlan
```

The required request payload is a JSON-formatted *RatingPlan* object. See example below.

10.5.2. Example Using cURL

The **example** below creates a new rating plan that charges users based only on storage level, with no charges for traffic. In this example the JSON-formatted *RatingPlan* object is specified in a text file named *ratingStorageOnly.txt* which is then referenced as the data input to the cURL command.

```
curl -X PUT -H "Content-Type: application/json" -k -u sysadmin:password \
-d @ratingStorageOnly.txt https://localhost:19443/ratingPlan
```

The *ratingStorageOnly.txt* file content in this example is as follows.

```
{
  "currency": "USD",
  "id": "Storage-Only",
  "mapRules": {
    "BI": {
      "ruleclassType": "BYTES_IN",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "BO": {
      "ruleclassType": "BYTES_OUT",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HD": {
      "ruleclassType": "HTTP_DELETE",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HG": {
      "ruleclassType": "HTTP_GET",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HP": {
      "ruleclassType": "HTTP_PUT",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "SB": {
      "ruleclassType": "STORAGE_BYTE",
      "rules": [
        {
          "first": "100",
          "second": "0.2"
        },
        {
          "first": "0",
          "second": "0.15"
        }
      ]
    }
  },
  "name": "Storage Bytes Only"
}
```

10.5.3. Request Elements

currency

(Optional, string) An international currency code (such as "USD", "JPY", or "EUR"). Defaults to "USD". Example:

```
"currency": "USD"
```

id

(Mandatory, string) Unique identifier that you assign to the rating plan. Example:

```
"id": "Default-RP"
```

mapRules

(Optional, map<string,*RuleClass*>) Map of rating rules per service dimension. The string is the service dimension acronym. For each <string,*RuleClass*> combination the string is one of "BI" (bytes in), "BO" (bytes out), "HG" (HTTP GETs), "HP" (HTTP PUTs), "HD" (HTTP DELETEs), or "SB" (storage bytes). For each service dimension there is a corresponding *RuleClass*object that expresses the rating rules for that service dimension.

Example:

```
"BI": {RuleClass}
```

Note You can omit the *mapRules* entirely in the unlikely event that you want to create a rating plan that does not charge for anything. But if you do include a *mapRules* map you must set a <string,*RuleClass*> combination for each of the six service dimensions, including dimensions for which you do not want to charge.

The *RuleClass* object consists of the following attributes and nested objects:

ruleclassType

(String) Type of rating rule: one of {STORAGE_BYTE, BYTES_IN, BYTES_OUT, HTTP_GET, HTTP_PUT, HTTP_DELETE}. These are the service usage dimensions for which pricing can be set. Example:

```
"ruleclassType": "BYTES_IN"
```

rules

(List<*Pair*>) List of rating rules to apply to the rule class type. There is one rule (one *Pair*object) per pricing tier.

Each *Pair* object consists of the following attributes:

first

(String) Rating tier size, as a number of units. In the first *Pair* within a list of *Pair* objects, the "first" attribute specifies the N in the rating rule "The first N units are to be priced at X per unit". (The specific units follow from the service usage type. See **"Service Usage Units"** (page 139) below). In the next *Pair* object in the list, the "first" attribute specifies the N in "The next N units are to be priced at X per unit"; and so on. For your final tier — for pricing additional units above and beyond the already defined tiers — use "0" as the value for the "first" attribute. For an example see the description of "second" below.

second

(String) For the rating tier specified by the "first" attribute, the rate per unit. The rate is specified as an integer or decimal. (The currency is as specified in the parent *RatingPlan*object.) For example, suppose the currency is set as dollars in the *RatingPlan*object, and you want the first 10 units to be charged at $2 per unit, the next 10 units to be priced at $1.50 per unit, and any additional units to be charged at $1 per unit. Your first *Pair* would be:

```
 {"first":"10","second":"2.00"}
```

Your next *Pair* would be:

```
{"first":"10","second":"1.50"}
```

Your third and final *Pair* would be:

```
{"first":"0","second":"1.00"}
```

Note For each service dimension that you do not want to charge for, specify just one *Pair* object with both "first" and "second" set to "0".

10.5.3.1. Service Usage Units

In a *Pair* object, the "first" attribute indicates a number of units constituting a pricing tier, and the "second" attribute indicates the price per unit within that pricing tier. What constitutes a unit depends on the service usage type that the rating rule is being applied to. For illustration suppose that in the examples below, the currency (as specified in the parent *RatingPlan* object) is dollars.

- For **storage bytes (SB), the unit is GiB-month** — the average number of GiBs of data stored for the month (which is calculated by summing the month’s hourly readings of stored bytes, converting to GiB, then dividing by the number of hours in the month). So if in a *Pair* object the "first" attribute is set to "5" and the "second" attribute is set to "2", this means that within this pricing tier which spans 5 GiB-month, the charge is $2 per GiB-month.
- For **data transfer bytes in or out (BI or BO)**, the unit is number of GiBs. So if in a *Pair* object the "first" attribute is set to "5" and the "second" attribute is set to "2", this means that within this pricing tier which spans 5 GiBs, the charge is $2 per GiB.
- For **HTTP GETs, PUTs, or DELETEs (HG, HP, or HD)**, the unit is blocks of 10,000 requests. So if in a *Pair* object the "first" attribute is set to "5" and the "second" attribute is set to "2", this means that within this pricing tier which spans 50,000 requests, the charge is $2 per 10,000 requests.

name

(Mandatory, string) Name of rating plan. Example:

```
"name": "Default Rating Plan"
```

10.5.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: {id, name}
400	Invalid JSON Object
409	Unique Constraint Violation : {id}

Chapter 11. system

The Admin API methods built around the **system** resource are for retrieving system information or performing certain system maintenance tasks.

Methods associated with the *system* resource:

- **"GET /system/audit"** (page 141)
- **"GET /system/bucketcount"** (page 144)
- **"GET /system/bucketusage"** (page 147)
- **"GET /system/bucketlist"** (page 145)
- **"GET /system/bytecount"** (page 149)
- **"GET /system/bytestiered"** (page 151)
- **"GET /system/dcnodelist"** (page 152)
- **"GET /system/groupbytecount"** (page 152)
- **"GET /system/groupobjectcount"** (page 155)
- **"GET /system/license"** (page 157)
- **"GET system/objectcount"** (page 162)
- **"GET /system/objectlockenabled"** (page 164)
- **"GET /system/token/challenge"** (page 165)
- **"GET /system/version"** (page 166)
- **"POST /system/processProtectionPolicy"** (page 167)
- **"POST /system/repairusercount"** (page 168)

11.1. GET /system/audit

GET /system/audit   Get summary counts for system

11.1.1. Syntax

```
GET /system/audit?[region=string]
```

There is no request payload.

11.1.2. Parameter Descriptions

region

(Optional, string) The service region for which to retrieve audit data. If the region is not specified in the request, then the returned audit data will be for the whole system (all regions), combined.

11.1.3. Usage Notes

Audit data is automatically updated within the system at the top of each hour. When you call the *GET /system/audit* method you are retrieving the audit data from the most recent hourly update. If you want up-to-the-minute counts -- rather than the counts as of the top of the last hour -- first call the method *POST /system/audit*, with no request body. This updates the counts. Then, you can retrieve the freshly updated counts using the *GET /system/audit* method. If you have a multi-region system and want to update each region's audit data, you would need to submit a separate *POST /system/audit* request to one Admin host in each region. Again, this is necessary only if you want audit data that is fresher than the automatic update done (in every region) at the top of each hour.

11.1.4. Example Using cURL

The **example** below retrieves the summary counts for the system.

```
curl -X GET -k -u sysadmin:password  https://localhost:19443/system/audit \
| python -mjson.tool
```

The response payload is a JSON-formatted *AuditData* object, which in this example is as follows.

```
{
  "byteCount": 647687490,
  "bytesInCount": 0,
  "bytesOutCount": 0,
  "licenseExpiration": 1590094491952,
  "nodes": [
    {
      "name": "10.50.50.201"
    },
    {
      "name": "10.50.50.202"
    },
    {
      "name": "10.50.50.203"
    }
  ],
  "objectCount": 13,
  "os": "Linux 3.10.0-957.1.3.el7.x86_64 amd64",
  "tieredBytesCount": 0,
  "timestamp": 1563724800000,
  "userCount": 3
}
```

11.1.5. Response Element Descriptions

byteCount

(number) Net bytes of object data stored in the system. This measure excludes overhead from replication and erasure coding.

Example:

```
"byteCount": 647687490
```

bytesInCount

(number) This measure is not implemented currently and its value will always be "0".

Example:

```
"bytesInCount": 0
```

bytesOutCount

(number) This measure is not implemented currently and its value will always be "0".

Example:

```
"bytesOutCount": 0
```

licenseExpiration

(number) License expiration date-time, in UTC milliseconds.

Example:

```
"licenseExpiration": 1590094491952
```

nodes

(set) The list of nodes that comprise the HyperStore system, identified by IP address.

Example:

```
  "nodes": [
    {
      "name": "10.50.50.201"
    },
    {
      "name": "10.50.50.202"
    },
    {
      "name": "10.50.50.203"
    }
  ],
```

objectCount

(number) Number of objects currently stored in the system.

Example:

```
"objectCount": 13
```

os

(string) Operating system version being used by HyperStore hosts.

Example:

```
"os": "Linux 3.10.0-957.1.3.el7.x86_64 amd64"
```

tieredBytesCount

(number) Number of bytes of object data that has been auto-tiered to a remote destination or destinations.

Example:

```
"tieredBytesCount": 0
```

timestamp

(number) Date-time when audit data was automatically updated at the top of the most recently completed hour. In UTC milliseconds. Note that this timestamp is not affected by calling the *POST /system/audit* method (this method updates the counts, but not the timestamp).

Example:

```
"timestamp": 1563724800000
```

userCount

(number) Number of active users in the system. This includes administrators as well as regular users.

Example:

```
"userCount": 3
```

11.1.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Region {region} is not valid

11.2. GET /system/bucketcount

GET /system/bucketcount Get aggregate count of buckets owned by a group's members

11.2.1. Syntax

```
GET /system/bucketcount?groupId=string[&region=string]
```

There is no request payload.

11.2.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve the count.

region

(Optional, string) The service region for which to retrieve the count. If you omit the region parameter, the call response will be a total count across all regions combined. If your HyperStore system has only one service region there is no reason to use this parameter.

11.2.3. Usage Notes

This method returns an aggregate count of all buckets owned by a group's members. It does not break down the count into individual users within the group.

11.2.4. Example Using cURL

The **example** below retrieves the count of buckets owned by users within the group "testgroup1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bucketcount?groupId=testgroup1'
```

The response payload is a text string, which in this example is as follows:

```
5
```

11.2.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or these method-specific status codes:

Status Code	Description
204	No content
400	Invalid region name

11.3. GET /system/bucketlist

GET /system/bucketlist   Get list of buckets owned by each of a group's members

11.3.1. Syntax

```
GET /system/bucketlist?groupId=string
```

There is no request payload.

11.3.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve the list.

11.3.3. Example Using cURL

The **example** below retrieves the list of buckets owned by each user within the group "testgroup1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bucketlist?groupId=testgroup1'
| python -mjson.tool
```

The response payload in this example is as follows:

```
[
  {
    "userId":"testuser1",
    "buckets":[
      {
        "bucketName":"bucket1",
        "createTime":"1554755537223",
        "region": "losangeles",
        "policyName": "rf3"
      },
      {
        "bucketName":"bucket2",
        "createTime":"1554755542554",
        "region": "losangeles",
        "policyName": "rf3"
      },
      {
        "bucketName":"bucket3",
        "createTime":"1554755548227",
        "region": "losangeles",
        "policyName": "rf3"
      }
    ]
  },
  {
    "userId":"testuser2",
    "buckets":[
      {
        "bucketName":"testbucket4",
        "createTime":"1554755580759",
        "region": "boston",
        "policyName": "ec42"
      },
      {
        "bucketName":"testbucket5",
        "createTime":"1554755587516",
        "region": "boston",
        "policyName": "ec42"
      }
    ]
  }
]
```

11.3.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
204	No content
400	Missing required parameter: {groudId}

11.4. GET /system/bucketusage

GET /system/bucketusage Get stored byte and object counts for each bucket owned by a group's members

11.4.1. Syntax

```
GET /system/bucketusage?groupId=string[&userId=string][&bucketName=string]
```

There is no request payload.

11.4.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve the current byte counts and object counts per bucket. If the optional *userId* and *bucketName* parameters are not specified, then this API call returns the byte count and object count for each bucket owned by each user in the group. The response data is arranged by user, showing for each user the buckets owned by that user and the byte and object counts for each of those buckets.

userId

(Optional, string) The user for whom to retrieve current byte counts and object counts per bucket. Use this parameter if you want to retrieve data just for a single user rather than for all users in the group.

bucketName

(Optional, string) The bucket for which to retrieve the current byte count and object count. Use this parameter if you want to retrieve data just for a single bucket. If you specify the *bucketName* you must also specify the *groupId* and *userId* to identify the bucket owner.

11.4.3. Usage Notes

By default, per-bucket byte and object counts are not tracked in the system and using the *GET /system/bucketusage* call will not work. For instructions to enable this feature see "Per-Bucket Object and Byte Counts" in the "Preparing the Usage Reporting Feature" section of the *Cloudian HyperStore Administrator's Guide*.

After you've enabled this feature, you can use this API to retrieve the current counts of stored bytes and stored objects for every bucket owned by every user in a specified group, with a single API call. Optionally you can narrow the response to bucket usage information for a single user (by specifying a *groupId***and** a *userId*) or a single bucket (by specifying a *groupId***and** a *userId***and** a *bucketName*).

Note With the *groupId* you must specify a single group. Specifying ALL as the *groupId*is not supported.

11.4.4. Example Using cURL

The **example** below retrieves the byte and object counts for all buckets owned by all users in the group named "testgroup4".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bucketusage?groupId=testgroup4'
| python -mjson.tool
```

The response payload in this example is as follows:

```
[
  {
    "userId": "user1",
    "buckets": [
      {
        "bucketName": "bucket1",
        "byteCount": 600166,
        "objectCount": 4,
        "policyName": "rf3"
      },
      {
        "bucketName": "bucket2",
        "byteCount": 1201849,
        "objectCount": 3,
        "policyName": "rf3"
      },
      {
        "bucketName": "bucket3",
        "byteCount": 1282152,
        "objectCount": 3,
        "policyName": "rf3"
      }
    ]
  },
  {
    "userId": "user2",
    "buckets": [
      {
        "bucketName": "mybucket",
        "byteCount": 2912140,
        "objectCount": 10,
        "policyName": "rf3"
      }
    ]
  }
]
```

11.4.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: {groudId}
400	bucketName is specified and groupId or userId is empty
400	Group is not found or user is not found or bucket is not found

11.5. GET /system/bytecount

GET /system/bytecount Get stored byte count for the system, a group, a user, or a bucket

11.5.1. Syntax

```
GET /system/bytecount?groupId=string&userId=string[&bucketName=string][&region=string]
```

There is no request payload.

11.5.2. Parameter Descriptions

groupId, userId

(Mandatory, string) Use the groupId and userId parameters to specify whether you want to retrieve a count for the whole system, for one whole group, or for one particular user:

- Whole system: *groupId=ALL&userId=**
- One whole group: *groupId=<groupId>&userId=** (example: *groupId=Dev&userId=**)
- One particular user : *groupId=<groupId>&userId=<userId>* (example: *groupId-d=Dev&userId=Cody*)

bucketName

(Optional, string) Use the bucketName parameter if you want to retrieve a count for one specific bucket.

Note To retrieve a count for a bucket you must also specify the group and the user who owns the bucket -- for example *groupId=Dev&userId=Cody&bucketName=bucket1*

region

(Optional, string) The service region for which to retrieve the count. If you omit the region parameter (and also omit the bucketName parameter), the call response will be a total count across all regions combined. If your HyperStore system has only one service region there is no reason to use this parameter.

11.5.3. Usage Notes

The byte count is for "net" bytes (also known as "logical" bytes). Overhead due to replication or erasure coding does not count toward this figure. For example, if a 1MiB object is replicated three times in the system (as part of a replication storage policy), this counts as 1MiB toward the total byte count -- not as 3MiB.

By default, per-bucket byte counts are not tracked in the system and using the *bucketName* parameter with the *GET /system/bytecount* call will not work. For instructions to enable this feature see the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

11.5.4. Examples Using cURL

The **example** below retrieves the byte count for the system as a whole (all users in all groups).

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bytecount?groupId=ALL&userId=*'
```

The response payload is the byte count in plain text, which in this example is as follows:

```
73836232
```

This next example retrieves the byte count for the "Pubs" group as a whole (all users in the Pubs group).

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bytecount?groupId=Pubs&userId=*'
```

The response payload is:

```
542348
```

This next example retrieves the byte count for the user "PubsUser1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bytecount?groupId=Pubs&userId=PubsUser1'
```

The response payload is:

```
66712
```

This next example retrieves the byte count for "bucket1", owned by the user "PubsUser1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bytecount?groupId=Pubs&userId=PubsUser1&bucketName=bucket1'
```

The response payload is:

```
32945
```

11.5.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}
400	Group or user not found
400	Must specify group and user if bucket if specified
400	User cannot be "*" if bucket if specified
400	Bucket not found
400	Bucket is not in specified region
400	Invalid region

11.6. GET /system/bytestiered

GET /system/bytestiered   Get tiered byte count for the system, a group, or a user

11.6.1. Syntax

```
GET /system/bytestiered?groupId=string&userId=string
```

There is no request payload.

11.6.2. Parameter Descriptions

groupId, userId

(Mandatory, string) Use the groupId and userId parameters to specify whether you want to retrieve a count for the whole system, for one whole group, or for one particular user:

- Whole system: *groupId=ALL&userId=**
- One whole group: *groupId=<groupId>&userId=** (example: *groupId=Dev&userId=**)
- One particular user : *groupId=<groupId>&userId=<userId>* (example: *groupId-d=Dev&userId=Cody*)

11.6.3. Usage Notes

The response is the total current volume of tiered storage in destination systems **other than HyperStore**. Data auto-tiered from one region to another within a HyperStore system, or from one HyperStore system to another HyperStore system, does not count toward this figure.

11.6.4. Example Using cURL

The **example** below retrieves the tiered volume for the system as a whole (all users in all groups).

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/bytestiered?groupId=ALL&userId=*'
```

The response payload is the tiered volume as a plain text string, which in this example is as follows:

```
"62G"
```

The tiered volume is expressed as "<*n*>G" (for number of GiBs) or "<*n*>T" (for number of TiBs) or "<*n*>P" (for number of PBs). If the tiered volume is currently less than 1GiB then "0" is returned.

11.6.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}

11.7. GET /system/dcnodelist

GET /system/dcnodelist   Get list of data centers and nodes

11.7.1. Syntax

```
GET /system/dcnodelist?[region=string]
```

There is no request payload.

11.7.2. Parameter Descriptions

region

(Optional, string) The service region for which to retrieve a list of data centers and nodes. If not supplied, the default service region is assumed.

11.7.3. Usage Notes

This method returns a list of the data centers in a service region and the nodes within each data center.

11.7.4. Example Using cURL

The **example** below retrieves the list of data centers and nodes within the "north" service region.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/dcnodelist?region=north'
```

The response is formatted as follows:

```
{"DC1":["centos7-vm1","centos7-vm2","centos7-vm3"]}
```

In this example there is just one data center in the region, named "DC1". Note that in the response the nodes within the data center are identified by hostname (not by IP address).

11.7.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Invalid parameter: region=*region*

11.8. GET /system/groupbytecount

GET /system/groupbytecount   Get stored byte counts for each of a group's users

11.8.1. Syntax

```
GET /system/groupbytecount?groupId=string[&limit=integer][&offset=string]
```

There is no request payload.

11.8.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve the counts.

limit

(Optional, integer) For purposes of pagination, the optional *limit* parameter specifies the maximum number of users to return in one response. In the response the users are sorted alphanumerically and if there are more than "limit" users in the group, then the number of users returned will be "limit plus 1" (for example, 101 users if the limit is 100). The last, extra returned user — the "plus 1" — is an indicator that there are more users than could be returned in the current response (given the specified "limit" value). That last user’s ID can then be used as the "offset" value in a subsequent request that retrieves additional users.

Note If the offset user happens to be the last user in the entire set of users, the subsequent query using the offset will return no users.

Defaults to 100.

offset

(Optional, string) The user ID with which to start the response list of users for the current request, sorted alphanumerically. The optional "offset" parameter can be used for purposes of pagination within a large result set that is being retrieved via multiple sequential requests. See the description of "limit" above for more information.

If "offset" is not specified, the first user in the response list will be the alphanumerically first user from the entire set of users in the group.

11.8.3. Usage Notes

The byte counts are for "net" bytes. Overhead due to replication or erasure coding does not count toward these figures. For example, if a 1MiB object is replicated three times in the system (as part of a replication storage policy), this counts as 1MiB toward the byte count -- not as 3MiB.

11.8.4. Example Using cURL

The **example** below retrieves the stored byte counts for each of the users in the "Pubs" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/groupbytecount?groupId=Pubs' | python -mjson.tool
```

The response payload is a JSON-formatted *UserUsage* object, which in this example is as follows. The group has three users in it. If a user has multiple buckets, the user's byte count value is the sum total across all of the user's buckets. For example the user "brady" has a total of 220508 stored bytes.

```
[
  {
    "canonicalUserId": "da870acdd136d60789fb5c761fef4a4a",
    "groupId": "Pubs",
    "usageVal": 220508,
    "userId": "brady"
  },
  {
    "canonicalUserId": "9bdcdd44ce1f9266adb9f22a8313feb4",
    "groupId": "Pubs",
    "usageVal": 225365,
    "userId": "gilmore"
  },
  {
    "canonicalUserId": "9a00529cdfb6496a09c5105913b486ac",
    "groupId": "Pubs",
    "usageVal": 76744,
    "userId": "gronk"
  }
]
```

11.8.5. Response Element Descriptions

canonicaUserId

(Number) The user's system-generated canonical user ID. Example:

```
"canonicalUserId": "da870acdd136d60789fb5c761fef4a4a"
```

groupId

(Number) The ID of the group to which the user belongs. Example:

```
"groupId": "Pubs"
```

usageVal

(Number) The user's current stored byte count. If the user has multiple buckets, the count is a combined total across all of the user's buckets.

Example:

```
"usageVal": 220508
```

userId

(String) The user's user ID. Example:

```
"userId": "brady"
```

11.8.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Missing required parameter: {groupId}

Status Code	Description
400	Limit should be greater than zero.

11.9. GET /system/groupobjectcount

GET /system/groupobjectcount   Get stored object counts for each of a group's users

11.9.1. Syntax

```
GET /system/groupobjectcount?groupId=string[&limit=integer][&offset=string]
```

There is no request payload.

11.9.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to retrieve the counts.

limit

(Optional, integer) For purposes of pagination, the optional *limit* parameter specifies the maximum number of users to return in one response. In the response the users are sorted alphanumerically and if there are more than "limit" users in the group, then the number of users returned will be "limit plus 1" (for example, 101 users if the limit is 100). The last, extra returned user — the "plus 1" — is an indicator that there are more users than could be returned in the current response (given the specified "limit" value). That last user's ID can then be used as the "offset" value in a subsequent request that retrieves additional users.

Note If the offset user happens to be the last user in the entire set of users, the subsequent query using the offset will return no users.

Defaults to 100.

offset

(Optional, string) The user ID with which to start the response list of users for the current request, sorted alphanumerically. The optional "offset" parameter can be used for purposes of pagination within a large result set that is being retrieved via multiple sequential requests. See the description of "limit" above for more information.

If "offset" is not specified, the first user in the response list will be the alphanumerically first user from the entire set of users in the group.

11.9.3. Example Using cURL

The **example** below retrieves the stored object counts for each of the users in the "Pubs" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/groupobjectcount?groupId=Pubs' | python -mjson.tool
```

The response payload is a JSON-formatted *UserUsage* object, which in this example is as follows. The group has three users in it. If a user has multiple buckets, the user's object count value is the sum total across all of the user's buckets. For example the user "brady" has a total of 5 stored objects.

```
[
  {
    "canonicalUserId": "da870acdd136d60789fb5c761fef4a4a",
    "groupId": "Pubs",
    "usageVal": 5,
    "userId": "brady"
  },
  {
    "canonicalUserId": "9bdcdd44ce1f9266adb9f22a8313feb4",
    "groupId": "Pubs",
    "usageVal": 5,
    "userId": "gilmore"
  },
  {
    "canonicalUserId": "9a00529cdfb6496a09c5105913b486ac",
    "groupId": "Pubs",
    "usageVal": 2,
    "userId": "gronk"
  }
]
```

11.9.4. Response Element Descriptions

canonicaUserId

(Number) The user's system-generated canonical user ID. Example:

```
"canonicalUserId": "da870acdd136d60789fb5c761fef4a4a"
```

groupId

(Number) The ID of the group to which the user belongs. Example:

```
"groupId": "Pubs"
```

usageVal

(Number) Either the user's current stored byte count (in response to a *GET /system/groupbytecount* request) or the user's current stored object count (in response to a *GET /system/groupobjectcount* request). If the user has multiple buckets, the count is a combined total across all of the user's buckets.

Example:

```
"usageVal": 220508
```

userId

(String) The user's user ID. Example:

```
"userId": "brady"
```

11.9.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter: {groupId}
400	Limit should be greater than zero.

11.10. GET /system/license

GET /system/license   Get HyperStore license terms

11.10.1. Syntax

```
GET /system/license
```

There is no request payload.

Note For background information about HyperStore licensing, see Licensing and Auditing.

11.10.2. Example Using cURL

The **example** below retrieves license data for the HyperStore system in which the call is submitted.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/system/license | python -mjson.tool
```

The response payload is a JSON-formatted *LicenseData* object, which in this example is as follows.

```
{
  "appliances": null,
  "enforcing": true,
  "environment":null
  "expiration": 1621915200000,
  "externalLicense": null,
  "gracePeriod": 0,
  "hyperIQ": "90",
  "issued": 1586751232344,
  "licenseHolder:" "Acme Enterprises",
  "licenseNumber:" "123456789012345678",
  "maxNetStorage": "100T",
  "maxRawStorage": null,
  "maxTieredStorage": "-1"
  "objectLockMode": "DISABLED",
  "phoneHomeBucket": "acmebucket",
  "storageExceeded": false,
  "storageMode": "NET",
  "tieringExceeded": false,
  "warnPeriod": 30
}
```

11.10.3. Response Element Descriptions

appliances

(JSON object) Information about each HyperStore appliance in the cluster, if any. Information for each appliance consists of:

- nodeId
- maxStorage -- This is the amount of raw storage capacity usage licensed for this individual appliance machine.
- productName

If there are no HyperStore appliances in the cluster, the *appliances* attribute is set to 0.

Example:

```
"appliances": null
```

enforcing

(Boolean) If *true*, then the system will enforce the licensed storage maximum by rejecting S3 PUTs and POSTs if the cluster stored volume reaches 110% of the licensed maximum. If this happens, then support for S3 PUTs and POSTs will resume again after the cluster stored volume is less than 100% of the licensed maximum (either because data has been deleted, or because a new license with higher maximum cluster stored volume has been installed).

Also, if this attribute is set to *true*, then the system will enforce the licensed tiering maximum by no longer auto-tiering data if the tiered volume reaches 110% of the licensed tiering maximum. If this happens, then support for auto-tiering will resume again after the tiered volume is less than 100% of the licensed tiering maximum (either because tiered data has been deleted through HyperStore interfaces, or because a new license with higher maximum tiered volume has been installed).

For more information on license enforcement see "Licensing and Auditing" in the ***Cloudian HyperStore Administrator's Guide***.

Example:

```
"enforcing": true
```

environment

(String) This will be *null* unless the system is a HyperStore Single-Node system for deployment in an AWS Outposts environment. Examples:

```
"environment":null
```

```
"environment":"aws_outposts_singlenode"
```

expiration

(Number) License expiration date-time in UTC milliseconds. Example:

```
"expiration": 1621915200000
```

externalLicense

(String) This will be "aws" if this HyperStore system was licensed through AWS Local Zones, and otherwise will be null. Example:

```
"externalLicense": null
```

gracePeriod

(Number) After the license expiration date passes, the number of days until the HyperStore system is automatically disabled. Example:

```
"gracePeriod": 0
```

hyperIQ

(String) Your HyperStore license's level of support for 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.

The *hyperIQ* attribute in the LicenseData object indicates what level of HyperIQ functionality will be available to you **if you acquire and set up HyperIQ**.

- basic -- HyperIQ dashboards for OS and service status monitoring are supported indefinitely.
- <number of days> -- HyperIQ dashboards for OS and service status monitoring are supported indefinitely, and also an S3 analytics dashboard is supported for <number of days> duration from the HyperStore license issuance. This HyperStore license type has "Enterprise HyperIQ" support, with the distinction (as versus only the "basic" support level) being the availability of the S3 analytics dashboard in HyperIQ.

Example

```
"hyperIQ": "90"
```

issued

(Number) License issuance date-time in UTC milliseconds. Example:

```
"issued": 1586751232344
```

licenseHolder

(String) Name of the organization to which this license was issued. Example:

```
"licenseHolder:" "Acme Enterprises"
```

licenseNumber

(String) Unique number identifying this license. Example:

```
"licenseNumber:" "123456789012345678"
```

maxNetStorage

(String) Applicable only if "storageMode" is "NET". If "storageMode" is "RAW" then this attribute will be null.

Maximum allowed Net storage volume for your entire HyperStore system. This is expressed as "<*n*>G" (for number of GiBs) or "<*n*>T" (for number of TiBs), or so on. For example, "100T" for one hundred TiBs. This value does not use decimals and will be expressed in GiBs unless it's exact number of TiBs (that is, an exact multiple of 1024 GiBs). For example, 1024 GiBs would be expressed as "1T" but 1030 GiBs would be expressed as "1030G".

"Net" storage volume usage **excludes overhead from replication or erasure coding**. For example a 1GiB object protected by 3X replication counts as 1GiB toward the "maxNetStorage" limit — not as 3GiB.

Example:

```
"maxNetStorage": "100T"
```

maxRawStorage

(String) Applicable only if "storageMode" is "RAW". If "storageMode" is "NET" then this attribute will be null.

If "storageMode" is "RAW", the "maxRawStorage" attribute indicates any additional raw storage licensed for your cluster above and beyond the raw storage licensed to each of your appliance nodes (as indicated by the "maxStorage" child attributes within the "appliances" attribute). Typically the "maxRawStorage" attribute would have a non-zero value only if your cluster has a mix of appliance nodes and software-only nodes. In such an environment, total licensed raw storage for the cluster is the sum of each of the individual appliance machine raw storage licenses **plus** the "maxRawStorage".

In a cluster consisting purely of appliance nodes, the "maxRawStorage" value would typically be 0. In such an environment, total licensed raw storage for the cluster is the sum of each of the individual appliance machine raw storage licenses.

When non-zero, "maxRawStorage" is expressed as "<*n*>G" (for number of GiBs) or "<*n*>T" (for number of TiBs), or so on. For example, "100T" for one hundred TiBs. This value does not use decimals and will be expressed in GiBs unless it's exact number of TiBs (that is, an exact multiple of 1024 GiBs). For example, 1024 GiBs would be expressed as "1T" but 1030 GiBs would be expressed as "1030G".

Raw storage volume usage counts **all stored data**, including overhead from replication or erasure coding. For example a 1GiB object protected by 3X replication counts as 3GiB toward a raw storage license limit. Also, stored metadata counts toward the limit as well.

Example:

```
"maxRawStorage": null
```

maxTieredStorage

(String) Maximum allowed volume of auto-tiered data stored in external systems other than HyperStore, after having been transitioned to those systems from HyperStore. This is expressed as "<*n*>G" (for number of GiBs) or "<*n*>T" (for number of TiBs), or so on. For example, "100T" for one hundred TiBs. This value does not use decimals and will be expressed in GiBs unless it's exact number of TiBs (that is, an exact multiple of 1024 GiBs). For example, 1024 GiBs would be expressed as "1T" but 1030 GiBs would be expressed as "1030G".

All auto-tiered data stored in any destination system **other than HyperStore** counts toward this limit. Data auto-tiered from one of your HyperStore regions to another region, or from your HyperStore system to an external HyperStore system, does not count toward this limit.

This attribute may have the value "-1" to indicate "unlimited" (i.e. the license places no limit on tiered data volume).

Example:

```
"maxTieredStorage": "-1"
```

objectLockMode

(String) The license's type of support for the HyperStore Object Lock (WORM) feature:

- DISABLED -- Object Lock is not supported.
- COMPATIBLE -- Compatible Object Lock is supported.
- CERTIFIED -- Certified Object Lock is supported.

For more information about the Object Lock feature including description of the two types of licensed support, see "Object Lock Feature Overview" in the ***Cloudian HyperStore Administrator's Guide***.

Example:

```
"objectLockMode": "DISABLED"
```

phoneHomeBucket

(String) Name of the bucket in which the license holder's Smart Support (Phone Home) data is stored. Null if this functionality is disabled. Example:

```
"phoneHomeBucket": "acmebucket"
```

storageExceeded

(Boolean) This flag sets to *true* if the cluster storage volume reaches 110% of licensed maximum storage. It sets back to *false* when the cluster storage volume is less than 100% of licensed maximum storage (either because data has been deleted, or because a new license with higher maximum storage volume has been installed).

Example:

```
"storageExceeded": false
```

storageMode

(String) The type of storage volume licensing applied by this license: either "NET" or "RAW". See "maxNetStorage" and "maxRawStorage" for more detail. Example:

```
"storageMode": "NET"
```

tieringExceeded

(Boolean) This flag sets to *true* if the tiered storage volume reaches 110% of the licensed maximum tiered volume. It sets back to *false* when the tiered storage volume is less than 100% of the licensed maximum tiered volume (either because tiered data has been deleted through HyperStore interfaces, or because a new license with higher maximum tiered volume has been installed).

Example:

```
"tieringExceeded": false
```

warnPeriod

(Number) Starting this many days before the license expiration date, an expiration warning message will display at the top of the Cloudian Management Console. Example:

```
"warnPeriod": 30
```

11.10.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

11.10.5. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianSystemLicense*
- Parameters: Same as for *GET /system/license* (no parameters)
- Response body: Same response data as for *GET /system/license* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can use this method
  - HyperStore group admin user cannot use this method
  - HyperStore regular user cannot use this method
  - IAM user can only use this method if granted *admin:GetCloudianSystemLicense* permission by policy, and subject to the same restriction as the parent HyperStore user
- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianSystemLicense

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianSystemLicenseResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<LicenseData>
<expiration>2024-05-21T13:54:51.952-07:00</expiration>
etc...
...
...
</LicenseData>
</GetCloudianSystemLicenseResponse>
```

11.11. GET system/objectcount

GET system/objectcount Get stored object count for the system, a group, a user, or a bucket

11.11.1. Syntax

```
GET /system/objectcount?groupId=string&userId=string[&bucketName=string][&region=string]
```

There is no request payload.

11.11.2. Parameter Descriptions

groupId, userId

(Mandatory, string) Use the groupId and userId parameters to specify whether you want to retrieve a count for the whole system, for one whole group, or for one particular user:

- Whole system: _groupId=ALL&userId=*_
- One whole group: _groupId=<groupId>&userId=*_ (example: _groupId=Dev&userId=*_)
- One particular user : *groupId=<groupId>&userId=<userId>* (example: *groupId=Dev&userId=Cody*)

bucketName

(Optional, string) Use the bucketName parameter if you want to retrieve a count for one specific bucket.

> **Note** To retrieve a count for a bucket you must also specify the group and the user who owns the bucket -- for example *groupId=Dev&userId=Cody&bucketName=bucket1*

region

(Optional, string) The service region for which to retrieve the count. If you omit the region parameter (and also omit the bucketName parameter), the call response will be a total count across all regions combined. If your HyperStore system has only one service region there is no reason to use this parameter.

11.11.3. Usage Notes

By default, per-bucket object counts are not tracked in the system and using the *bucketName* parameter with the *GET /system/objectcount* call will not work. For instructions to enable this feature see the "Usage Reporting and Billing" section in the ***Cloudian HyperStore Administrator's Guide***.

11.11.4. Examples Using cURL

The **example** below retrieves the object count for the system as a whole (all users in all groups).

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/objectcount?groupId=ALL&userId=*'
```

The response payload is the object count in plain text, which in this example is as follows:

```
1023
```

This next example retrieves the object count for the "Pubs" group as a whole (all users in the Pubs group).

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/objectcount?groupId=Pubs&userId=*'
```

The response payload is:

```
215
```

This next example retrieves the object count for the user "PubsUser1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/objectcount?groupId=Pubs&userId=PubsUser1'
```

The response payload is:

```
54
```

This next example retrieves the object count for the bucket "bucket1", which is owned by the user "PubsUser1".

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/objectcount?groupId=Pubs&userId=PubsUser1&bucketName=bucket1'
```

The response payload is:

```
31
```

11.11.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}
400	Group or user not found
400	Must specify group and user if bucket if specified
400	User cannot be "*" if bucket if specified
400	Bucket not found
400	Bucket is not in specified region
400	Invalid region

11.12. GET /system/objectlockenabled

GET /system/objectlockenabled Get Object Lock enabled/disabled status

11.12.1. Syntax

```
GET /system/objectlockenabled
```

There is no request payload.

11.12.2. Usage Notes

The response to this call will be:

- *true* if the system is in either of these conditions:
    - Your HyperStore license supports the "Compatible" Object Lock type.

      OR

    - Your HyperStore license supports the "Certified" Object Lock type and you have both enabled the HyperStore Shell and disabled root password access to HyperStore nodes (for instructions see the HyperStore Shell section of the ***Cloudian HyperStore Administrator's Guide***). (Note that if your license supports the "Certified" type of Object Lock, then Object Lock is not enabled in the system until you have enabled the HyperStore Shell and disabled root password access to HyperStore nodes.)
- *false* if the system is in either of these conditions:
    - Your HyperStore license does not support the Object Lock feature

      OR

    - Your HyperStore license supports the "Certified" Object Lock type but you have not yet enabled the HyperStore Shell and disabled root password access to HyperStore nodes.

11.12.3. Example Using cURL

The **example** below retrieves the Object Lock feature status.

```
curl -X GET -k -u sysadmin:password https://localhost:19443/system/objectlockenabled
```

The response is the Object Lock feature status in plain text, which in this example is as follows.

```
false
```

11.13. GET /system/token/challenge

GET /system/token/challenge Get token challenge to provide to Cloudian Support

11.13.1. Syntax

```
GET /system/token/challenge?action=purge&params=bucketName:string
```

There is no request payload.

11.13.2. Parameter Descriptions

action

(Mandatory, string) Action for which a token is being generated. Currently the only supported action is *purge*.

bucketName

(Mandatory, string) Name of the bucket that you want to purge of data.

11.13.3. Usage Notes

Using this Admin API call to generate a token challenge is one step in the procedure for purging data from an Object Locked bucket, if your system is licensed for the "Compatible" Object Lock type. After you generate the token challenge you provide it to Cloudian Support, by opening a support case. For the full procedure see **"Purging an Object Locked Bucket"** (page 56).

> **Note** If your system is licensed for the "Certified" Object Lock type you cannot purge data from an Object Locked bucket, and there is no reason to use the *GET /system/token/challenge* call.

11.13.4. Example Using cURL

The **example** below generates a token challenge for purging data from an Object Locked bucket named *pubs1*.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/system/token/challenge?action=purge&params=bucketName:pubs1'
```

The response is the token challenge, which in this example is as follows.

```
A.fa868eff0219b2ecdf58d2aff6fa355584b090a2a6ff4073d28a2245bbc23f09
```

11.13.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missed required parameter: action, bucketName
400	Unsupported 'action' value
403	Token challenge is supported only for licensed "Compatible" Object Lock type

11.14. GET /system/version

GET /system/version Get HyperStore system version

11.14.1. Syntax

```
GET /system/version
```

There is no request payload.

11.14.2. Example Using cURL

The **example** below retrieves the HyperStore system version.

```
curl -X GET -k -u sysadmin:password https://localhost:19443/system/version
```

The response payload is the system version information in plain text, which in this example is as follows.

```
8.1.1 Compiled: 2024-08-16 16:32
```

11.14.3. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

11.14.4. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianSystemVersion*
- Parameters: Same as for *GET /system/version* (no parameters)
- Response body: Same response data as for *GET /system/version* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user, group admin user, and regular user can all use this method
  - IAM user can only use this method if granted *admin:GetCloudianSystemVersion* permission by policy
- Sample request and response:

```
REQUEST

http://localhost:16080/?Action=GetCloudianSystemVersion

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianSystemVersionResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<String>
8.1.1 Compiled: 2024-08-16 16:32
</String>
</GetSystemVersionResponse>
```

11.15. POST /system/processProtectionPolicy

POST /system/processProtectionPolicy Process pending storage policy deletion or creation jobs

11.15.1. Syntax

```
POST /system/processProtectionPolicy
```

There is no request payload.

11.15.2. Usage Notes

This method processes any pending storage policy deletion 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. This operator action marks the policy with a "DELETED" flag and makes it immediately unavailable to service users. However, the full process of deleting the storage policy from the system is not completed until the *POST /system/processProtectionPolicy* method is run.

This method 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.

Note This method is invoked once a day by a HyperStore cron job. For more information see the "Cron Jobs and Automated System Maintenance" section of the ***Cloudian HyperStore Administrator's Guide***.

11.15.3. Example Using cURL

The **example** below triggers the processing of any pending storage policy deletion or creation jobs.

```
curl -X POST -k -u sysadmin:password \
https://localhost:19443/system/processProtectionPolicy
```

11.15.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

11.16. POST /system/repairusercount

POST /system/repairusercount   Reconcile user counts in Redis and Cassandra

11.16.1. Syntax

```
POST /system/repairusercount
```

There is no request payload.

11.16.2. Usage Notes

Use this method if you have reason to suspect that user counts in your audit data are inaccurate. This method will synchronize the user counts in Redis (which are used in audit data) to the metadata in the Cassandra

UserInfo table.

11.16.3. Example Using cURL

The **example** below syncs the user counts in Redis with the Cassandra metadata.

```
curl -X POST -k -u sysadmin:password https://localhost:19443/system/repairusercount
```

11.16.4. Response Codes

This method will return one of one of the **"Common Response Status Codes"** (page 13).

This page left intentionally blank

Chapter 12. tiering

The Admin API methods built around the **tiering** resource are for managing account credentials to use for accessing auto-tiering destination systems. You can post tiering credentials to associate with specific HyperStore source buckets, and the system will securely store the credentials and use them when implementing auto-tiering for those buckets. For S3-compliant tiering destinations you also have the option of posting a system default tiering credential, which can be made available for all bucket owners to use for auto-tiering to a system default tiering destination.

> **Note**
> * These Admin API methods are **not applicable to a HyperStore Single-Node system**.
> * Having a system default tiering credential is only supported for S3-compliant tiering destinations -- not for Azure or Spectra.

Methods associated with the *tiering* resource:

- **"DELETE /tiering/credentials"** (page 171)
- **"DELETE /tiering/azure/credentials"** (page 172)
- **"DELETE /tiering/spectra/credentials"** (page 172)
- **"GET /tiering/credentials"** (page 173)
- **"GET /tiering/credentials/src"** (page 174)
- **"GET /tiering/azure/credentials"** (page 175)
- **"GET /tiering/spectra/credentials"** (page 176)
- **"POST /tiering/credentials"** (page 176)
- **"POST /tiering/azure/credentials"** (page 178)
- **"POST /tiering/spectra/credentials"** (page 179)

12.1. DELETE /tiering/credentials

DELETE /tiering/credentials Delete a tiering credential for Amazon, Google, or other S3-compliant destination

12.1.1. Syntax

```
DELETE /tiering/credentials[?bucketName=string]
```

There is no request payload.

12.1.2. Parameter Descriptions

bucketName

(Optional, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system. If the *bucketName* parameter is omitted then the request applies to the system default auto-tiering credential.

12.1.3. Example Using cURL

The **example** below deletes the S3 auto-tiering credential currently associated with a HyperStore source bucket named "bucket1".

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/tiering/credentials?bucketName=bucket1
```

12.1.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

12.2. DELETE /tiering/azure/credentials

DELETE /tiering/azure/credentials   Delete a tiering credential for Azure

12.2.1. Syntax

```
DELETE /tiering/azure/credentials?bucketName=string
```

There is no request payload.

12.2.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

12.2.3. Example Using cURL

The **example** below deletes the Azure auto-tiering credential currently associated with a HyperStore source bucket named "bucket2".

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/tiering/azure/credentials?bucketName=bucket2
```

12.2.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

12.3. DELETE /tiering/spectra/credentials

DELETE /tiering/spectra/credentials   Delete a tiering credential for Spectra

12.3.1. Syntax

```
DELETE /tiering/spectra/credentials?bucketName=string
```

There is no request payload.

12.3.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

12.3.3. Example Using cURL

The **example** below deletes the Spectra auto-tiering credential currently associated with a HyperStore source bucket named "bucket1".

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/tiering/spectra/credentials?bucketName=bucket1
```

12.3.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

12.4. GET /tiering/credentials

GET /tiering/credentials   Get a tiering credential for Amazon, Google, or other S3-compliant destination

12.4.1. Syntax

```
GET /tiering/credentials[?bucketName=string]
```

There is no request payload.

12.4.2. Parameter Descriptions

bucketName

(Optional, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system. If the *bucketName* parameter is omitted then the request applies to the system default auto-tiering credential.

12.4.3. Example Using cURL

The **example** below retrieves the S3 auto-tiering credential currently associated with a HyperStore source bucket named "bucket1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/tiering/credentials?bucketName=bucket1
```

The response payload is the S3 access key in plain text, which in this example is as follows. The secret key is not returned.

```
00cc33c4b1ef9f50282a
```

12.4.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

12.5. GET /tiering/credentials/src

GET /tiering/credentials/src Check whether a bucket uses a bucket-specific or system default tiering credential

12.5.1. Syntax

```
GET /tiering/credentials/src[?bucketName=string]
```

There is no request payload.

12.5.2. Parameter Descriptions

bucketName

(Optional, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

12.5.3. Usage Notes

For buckets that auto-tier to Amazon, Google, or other S3-compliant destinations, you can use this method to check whether the bucket is using a bucket-specific tiering credential or the system default tiering credential (or no credential, if the bucket has not yet been configured for auto-tiering). You can omit the "bucketName" parameter if you just want to check whether or not the system default tiering credential has been created for the system. The method responds with a plain text string -- either "BUCKET" (bucket-specific credential), "SYSTEM" (system default credential), or NONE (no credential has been set for the system yet).

Note This method is not supported for buckets that tier to Azure or Spectra.

12.5.4. Example Using cURL

The **example** below checks the S3 auto-tiering credential type for a HyperStore source bucket named "bucket1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/tiering/credentials/src?bucketName=bucket1
```

In this example the response payload is BUCKET, indicating that "bucket1" uses a bucket-specific tiering credential in its S3 auto-tiering configuration.

```
BUCKET
```

12.5.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

12.6. GET /tiering/azure/credentials

GET /tiering/azure/credentials Get a tiering credential for Azure

12.6.1. Syntax

```
GET /tiering/azure/credentials?bucketName=string
```

There is no request payload.

12.6.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

12.6.3. Example Using cURL

The **example** below retrieves the Azure auto-tiering credential currently associated with a HyperStore source bucket named "bucket2".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/tiering/azure/credentials?bucketName=bucket1
```

The response payload is the Azure account name and account key in plain text with comma-separation, which in this example is as follows.

```
123456,Oy1wMUklsF81331LIGY5RlVqa8Rg+iWT6zEFt6I1
```

12.6.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

12.7. GET /tiering/spectra/credentials

GET /tiering/spectra/credentials   Get a tiering credential for Spectra

12.7.1. Syntax

```
GET /tiering/spectra/credentials?bucketName=string
```

There is no request payload.

12.7.2. Parameter Descriptions

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

12.7.3. Example Using cURL

The **example** below retrieves the Spectra auto-tiering credential currently associated with a HyperStore source bucket named "bucket1".

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/tiering/spectra/credentials?bucketName=bucket1
```

The response payload is the access key in plain text, which in this example is as follows. The secret key is not returned.

```
00d5dc27224f9d529257
```

12.7.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

12.8. POST /tiering/credentials

POST /tiering/credentials   Post a tiering credential for Amazon, Google, or other S3-compliant destination

12.8.1. Syntax

```
POST /tiering/credentials?accessKey=urlencoded-string&secretKey=urlencoded-string
[&bucketName=string]
```

There is no request payload.

12.8.2. Parameter Descriptions

accessKey

(Mandatory, string) Access key for the tiering destination account. Must be URL-encoded if the key includes non-ASCII characters.

secretKey

(Mandatory, string) Secret key for the tiering destination account. Must be URL-encoded if the key includes non-ASCII characters.

bucketName

(Optional, string) Name of the **HyperStore source bucket** that will use the credential for auto-tiering to a destination system. If the *bucketName* parameter is omitted then the request creates the system default auto-tiering credential.

12.8.3. Usage Notes

If an access key or secret key includes a non-ASCII character and you do not URL-encode the key, the API server will return a 403 error.

12.8.4. Example Using cURL

The **example** below posts S3 auto-tiering credentials for a HyperStore source bucket named "b1".

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/tiering/credentials?accessKey=00cc&secretKey=YuaO&bucketName=b1'
```

When implementing auto-tiering from this source bucket to an S3-compatible destination system (as configured by the bucket lifecycle configuration), the HyperStore system will use this credential.

Note In the example above, the access key and secret key are truncated so that the '*https://...*' segment can be shown on one line.

12.8.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accessKey, secretKey}

12.9. POST /tiering/azure/credentials

POST /tiering/azure/credentials   Post a tiering credential for Azure

12.9.1. Syntax

```
POST /tiering/azure/credentials?accountName=urlencoded-string&accountKey=urlencoded-
string&bucketName=string
```

There is no request payload.

12.9.2. Parameter Descriptions

accountName

(Mandatory, string) Name of the Azure tiering destination account. Must be URL-encoded if the name includes non-ASCII characters.

accountKey

(Mandatory, string) Account key for the Azure tiering destination account. Must be URL-encoded if the key includes non-ASCII characters.

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

Note Having a system default tiering credential is not supported not for Azure. Each bucket must have its own credentials.

12.9.3. Usage Notes

If an account name or account key includes a non-alphanumeric character and you do not URL-encode the key, the API server will return a 403 error.

12.9.4. Example Using cURL

The **example** below posts Azure auto-tiering credentials for a HyperStore source bucket named "b2".

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/tiering/azure/credentials?accountName=123&accountKey=Oy1&bucketName=b2'
```

When implementing auto-tiering from this source bucket to an Azure destination system (as configured by the bucket lifecycle configuration), the HyperStore system will use this credential.

Note In the example above, the account name and key are truncated so that the '*https://...*' segment can be shown on one line.

12.9.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accountName, accountKey}

12.10. POST /tiering/spectra/credentials

POST /tiering/spectra/credentials   Post a tiering credential for Spectra

12.10.1. Syntax

```
POST /tiering/spectra/credentials?accessKey=urlencoded-string&secretKey=urlencoded-
string&bucketName=string
```

There is no request payload.

12.10.2. Parameter Descriptions

accessKey

(Mandatory, string) Access key for the tiering destination account. Must be URL-encoded if the key includes non-ASCII characters.

secretKey

(Mandatory, string) Secret key for the tiering destination account. Must be URL-encoded if the key includes non-ASCII characters.

bucketName

(Mandatory, string) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

Note Having a system default tiering credential is not supported not for Spectra. Each bucket must have its own credentials.

12.10.3. Usage Notes

If an access key or secret key includes a non-ASCII character and you do not URL-encode the key, the API server will return a 403 error.

12.10.4. Example Using cURL

The **example** below posts Spectra auto-tiering credentials for a HyperStore source bucket named "b3".

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/tiering/spectra/credentials?accessKey=00d5&secretKey=PxvA&bucketName=b3'
```

When implementing auto-tiering from this source bucket to a Spectra destination system (as configured by the bucket lifecycle configuration), the HyperStore system will use this credential.

Note In the example above, the access key and secret key are truncated so that the '*https://...*' segment can be shown on one line.

12.10.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accessKey, secretKey}

Chapter 13. usage

The Admin API methods built around the **usage** resource are for managing HyperStore usage reporting. This includes support for retrieving service usage data for specified users, groups, or buckets. There are also methods for aggregating usage data and ensuring its accuracy — many of these methods are invoked regularly by HyperStore system cron jobs.

For an overview of the HyperStore usage reporting feature, see the "Usage Reporting and Billing" section in the ***Cloudian HyperStore Administrator's Guide***.

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.

Methods associated with the *usage* resource:

- **"DELETE /usage"** (page 181)
- **"GET /usage"** (page 183)
- **"POST /usage/bucket"** (page 199)
- **"POST /usage/repair"** (page 201)
- **"POST /usage/repair/bucket"** (page 202)
- **"POST /usage/repair/dirtyusers"** (page 203)
- **"POST /usage/repair/user"** (page 205)
- **"POST /usage/rollup"** (page 206)
- **"POST /usage/storage"** (page 207)
- **"POST /usage/storageall"** (page 208)

13.1. DELETE /usage

DELETE /usage   Delete usage data

13.1.1. Syntax

```
DELETE /usage?granularity=enum&startTime=string[&unitCount=integer]
```

There is no request payload.

13.1.2. Parameter Descriptions

granularity

(Mandatory, enum) The time period granularity of the usage data to delete. Supported values are:

- *hour* — Hourly rollup data
- *day* — Daily rollup data

- *month* — Monthly rollup data
- *raw* — Raw transactional data (not rolled up).

startTime

(Mandatory, string)

The start time **in GMT**. The format depends on the granularity of the usage data that you are generating or deleting:

- For hourly rollup data use format *yyyyMMddHH*. The start time will be the start of the hour that you specify.
- For daily rollup data use format *yyyyMMdd*. The start time will be the start of the day that you specify.
- For monthly rollup data use format *yyyyMM*. The start time will be the start of the month that you specify.
- For raw data use format *yyyyMMddHHmm*. The start time will be the start of the minute that you specify.

unitCount

(Optional, integer) The number of units of the specified "granularity" to delete. Supported range is [1,100].

For example, with "granularity" = hour and "unitCount" = 24, a *DELETE /usage* operation will delete 24 hours worth of hourly rollup data, starting from your specified "startTime". In the case of "granularity" = raw, a *DELETE /usage* operation will delete "unitCount" minutes worth of raw transactional data -- for example 10 minutes worth of raw transactional data if "unitCount" = 10.

Defaults to 1 unit if not specified.

13.1.3. Usage Notes

This method deletes service usage data from the Reports keyspace in Cassandra. Separate data exists for the raw, hourly roll-up, daily roll-up, and monthly roll-up levels. Note that when you delete usage data, usage data for **all** groups and users will be deleted for your specified granularity and time period.

Apart from using this API method, usage data deletion is also managed by configurable retention periods after which the system automatically deletes the data. See "Setting Usage Data Retention Periods" in the "Usage Reporting" section of the ***Cloudian HyperStore Administrator's Guide***.

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.

Note If you have enabled the per-bucket usage data feature, this API method does not delete per-bucket usage data. It deletes only per-group and per-user usage data. Deletion of per-bucket usage data is managed exclusively by the configuration retention periods.

13.1.4. Example Using cURL

The **example** below deletes daily roll-up usage data from the day of May 1st, 2017.

```
curl -X DELETE -k -u sysadmin:password \
'https://localhost:19443/usage?granularity=day&startTime=20170501&unitCount=1'
```

13.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

13.2. GET /usage

GET /usage   Get usage data for group, user, or bucket

13.2.1. Syntax

```
GET /usage?[id=string|canonicalUserId=string|bucket=string]&operation=enum&startTime=string
&endTime=string&granularity=enum&reversed=bool[&limit=integer][&pageSize=integer]
[&offset=string][&region=string][&regionOffset=string]
```

There is no request payload.

13.2.2. Parameter Descriptions

id

(Optional, string) The identifier of a user or group for which to retrieve usage data, in format "<groupId>|<userId>" (for example "Dev|dstone", where Dev is the group ID and dstone is the user ID). To retrieve usage data for a whole group rather than a single user, use "<groupId>|*" (for example "Dev|*").

Do not use the "id" parameter for users who have been deleted from the system. For deleted users, use the "canonicalUserId" parameter described below.

With a *GET /usage* request you must use **either** "id" **or** "canonicalUserId" **or** "bucket". Do not use more than one of these query parameters.

canonicalUserId

(Optional, string) The system-generated canonical ID of a user for which to retrieve usage data. Use this parameter if you want to retrieve usage data for a user who has been deleted from the system. If you don’t know the user’s canonical ID, you can obtain it by using the *GET /user/list* method (this method can retrieve user profile information — including canonical ID — for all deleted users within a specified group).

With a *GET /usage* request you must use **either** "id" **or** "canonicalUserId" **or** "bucket". Do not use more than one of these query parameters.

bucket

(Optional, string): The bucket name. Use this parameter if you want to retrieve usage data for a specific bucket (rather than for a user or group). With a *GET /usage* request you must use **either** "id" **or** "canonicalUserId" **or** "bucket". Do not use more than one of these query parameters. Note that bucket names are globally unique within a HyperStore system, so specifying a bucket name is sufficient to uniquely identify a bucket.

> **Note** With the exception of the *POST /usage/repair/bucket* method, bucket usage statistics are disabled by default. For information on enabling this feature see "Preparing the Usage Reporting Feature" in the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

operation

(Mandatory, enum) The type of service usage data to retrieve. Supported values are:

- *SB* — Number of stored bytes
- *SO* — Number of stored objects
- *HG* — Number of S3 HTTP GET requests (includes HEADs also). The returned usage data also includes information about bytes downloaded.
- *HP* — Number of S3 HTTP PUT requests (includes POSTs also). The returned usage data also includes information about bytes uploaded.
- *HD* — Number of S3 HTTP DELETE requests

  > **Note** Usage tracking and reporting for the HG, HP, and HD metrics is disabled by default. For information on enabling these metrics see "Preparing the Usage Reporting Feature" in the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

- *BI* — For bucket usage only, the number of data transfer bytes IN (bytes of data uploaded).
- *BO* — For bucket usage only, the number of data transfer bytes OUT (bytes of data downloaded).

  > **Note** The BI and BO operation types are supported only for *GET /usage?bucket* requests. For user and group level usage statistics, the inbound and outbound data transfer size information is included within the HP and HG operation type usage data.

- *TB* — For bucket usage only, the total bytes count for the bucket.
- *TO* — For bucket usage only, the total objects count for the bucket.

  > **Note** The TB and TO operation types are supported only for *GET /usage?bucket* requests. The TB and TO counts for a bucket for a specified time period (from startTime to endTime) will exist only if you previously executed the *POST /usage/repair/bucket* method during that time period. That method generates the TB and TO counts for the bucket which the system then stores along with a timestamp indicating when the count

was generated.

For *GET /usage?bucket* requests the SB and SO operation types are also supported, but these will return the **change** in the stored bytes and stored object counts during the specified time interval -- for example, the total increase in a bucket's stored bytes total during a specified day, rather than the total number of bytes in the bucket on that day. For the latter you would use the TB operation type.

startTime

(Mandatory, string)

The start time **in GMT**.

With a *GET /usage* request this is the start time of the interval for which to retrieve usage data. Format is *yyyyMMddHHmm*.

Note For retrieving **bucket** usage data, the start time's "*mm*" -- the minutes -- must be 00.

endTime

(Mandatory, string) The end time **in GMT** of the interval for which to retrieve usage data. Format is *yyyyMMddHHmm*.

Note For retrieving **bucket** usage data, the end time's "*mm*" -- the minutes -- must be 00.

granularity

(Mandatory, enum) The time period granularity of the usage data to retrieve. Supported values are:

- *hour* — Hourly rollup data
- *day* — Daily rollup data
- *month* — Monthly rollup data
- *raw* — Raw transactional data (not rolled up).

Note For a GET with granularity "raw", the interval between "startTime" and "endTime" must not exceed 24 hours. If the interval is larger than this, a 400 Bad Request response will be returned.

reversed

(Optional, boolean) If this is set to "false", the retrieved usage data results will be listed in chronological order. If this is set to "true", results will be listed in reverse chronological order. Defaults to "false" if not specified.

Note This parameter is not supported if you are retrieving **bucket** usage data.

limit

(Optional, integer) The maximum number of results to return — that is, the maximum number of

<*UsageData*> objects to return in the response body — if pagination is not used (if no "pageSize" value is specified).

Defaults to 10,000 if not specified.

pageSize

(Optional, integer) For pagination, the maximum number of results to return per request. If a "pageSize" is specified, this supersedes the "limit" value.

Defaults to 0 if not specified.

Note This parameter is not supported if you are retrieving **bucket** usage data.

offset

(Optional, integer) If you use the "pageSize" parameter in support of paginating a large result set, in the response the system will return one additional result beyond your specified "pageSize" value (for example, if you specify "pageSize=25", the system will return 26 results). From the extra result (listed last in the response body), use the result's timestamp as the "offset" parameter value in your next request. That result will then be the first of the results returned for that request.

For each request you submit, the last of the returned results will be an extra result from which you can use the timestamp as the "offset" value for the next request. If there is no extra result in the response, that indicates that the result set has been exhausted.

Defaults to 0 if not specified.

Note This parameter is not supported if you are retrieving **bucket** usage data.

region

(Optional, string) The region for which to retrieve usage data. To retrieve usage data for all regions, specify the string "ALL". If no "region" value is specified, the default region is assumed. This parameter is not supported if the "bucket" parameter is used (for retrieving data for a specified bucket).

Note *GET /usage* requests for user or group level statistics should be submitted only to the Admin Service in the default region. Use the "region" query parameter to specify the region for which you want to retrieve usage data.

GET /usage requests for **bucket** usage data can be submitted to the Admin Service any region, and the results will be from that region. Do not specify the "region" parameter for bucket usage data requests.

regionOffset

(Optional, string) If you use a "region" value of "ALL", use the "regionOffset" parameter to specify the region name of your local region. This helps with pagination of the result set.

Note This parameter is not supported if you are retrieving **bucket** usage data.

13.2.3. Usage Notes

The *GET /usage?bucket=string...* option is supported only if bucket usage statistics are enabled in the system. Bucket usage statistics are disabled by default. For information on enabling this feature see "Preparing the Usage Reporting Feature" in the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

13.2.4. Examples Using cURL

The first **example** below retrieves the monthly stored bytes usage data for the "QA" group, from July 2017.

```
curl -X GET -k -u sysadmin:password \
'https://
localhost
:19443/usage?id=
QA|*&operation=SB&startTime=201707010000&endTime=201708010000&granularity=month' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *UsageData* objects, which in this example is as follows. Note that in this case we are retrieving monthly roll-up data from a time interval that spans just one month, so here there is just one *UsageData* object in the list.

```
[
  {
    "averageValue": "107956",
    "bucket": null,
    "count": "744",
    "groupId": "QA",
    "ip": "",
    "maxValue": "305443",
    "operation": "SB",
    "region": "taoyuan",
    "timestamp": "1498867200000",
    "uri": "",
    "userId": "*",
    "value": "80319535",
    "whitelistAverageValue": "0",
    "whitelistCount": "0",
    "whitelistMaxValue": "0",
    "whitelistValue": "0"
  }
]
```

The next example below retrieves the total bytes count for a bucket named "bucket1" as of the specified hour interval. Note that to support retrieving the total bytes (TB) count or total objects (TO) count for a bucket as of a specified time interval, the **POST /usage/repair/bucket** method must have been executed for that bucket sometime during that time interval (since that method generates the TB and TO counts). If that method has not been executed for a bucket during a given time interval -- such as a particular hour or day -- then you cannot subsequently retrieve a TB or TO count for that bucket from that interval.

```
curl -X GET -k -u sysadmin:password \

'
https://
```

```
localhost
:19443/usage?bucket=
bucket1&operation=TB&startTime=201712201400&endTime=201712201500&granularity=raw' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *UsageData* objects, which in this example is as follows.

```
[
  {
    "averageValue": "4242572",
    "bucket": "bucket1",
    "count": "0",
    "groupId": null,
    "ip": null,
    "maxValue": "0",
    "operation": "TB",
    "policyId": "880e7d065225009b481ff24ae8d893ce",
    "region": null,
    "timestamp": "1513781460000",
    "uri": null,
    "userId": null,
    "value": "4242572",
    "whitelistAverageValue": "0",
    "whitelistCount": "0",
    "whitelistMaxValue": "0",
    "whitelistValue": "0"
  }
]
```

Note If the *POST /usage/repair/bucket* method had been called multiple times during the time period specified in the *GET /usage?bucket* request, and the requested granularity is "raw", then multiple *UsageData* objects would be returned in the response, each with a TB value and each with a timestamp indicating when the *POST /usage/repair/bucket* call had generated that TB value. By contrast, if the requested granularity is a roll-up period such as "hour", then only most recent TB value generated during that roll-up period would be returned.

For example, suppose that you have been executing the *POST /usage/repair/bucket* call on a particular bucket at 11AM and 11PM every day. Subsequently, if the start and end times in a *GET /usage?bucket* request span one week and the requested granularity is "day", the response will return one *UsageData* object for each day of the week, and the TB count shown for each day will be the one generated by the *POST /usage/repair/bucket* call executed at 11PM on each day.

13.2.5. Response Element Descriptions

averageValue

(String) Average value of the usage statistic during the granularity interval.

For user level or group level statistics, when usage report granularity = **hour, day**, or **month**, the "averageValue" will equal the "value" divided by the "count".

When usage report granularity = **raw** or for bucket usage statistics of any granularity, the "averageValue" will equal the "value".

Example:

```
"averageValue": "107956"
```

bucket

(String) Name of the bucket with which the usage data is associated. This attribute will have a value only for bucket usage data. For user level or group level usage data this attribute will have *null* value.

Example:

```
"bucket": null
```

count

(String) Data count. The specific meaning of this attribute depends on the usage reporting granularity and operation type.

When usage report granularity = **raw** or for bucket usage statistics of any granularity, "count" is not relevant and always returns a "0".

For user level or group level statistics, when usage report granularity = **hour, day**, or **month**:

- For operation type SB or SO:
  - For granularity **hour**, the "count" will always be "1".
  - For granularity **day**, the "count" will be the number of hourly data points recorded by the system within the day. For a past day, this will be "24"; for the current day, this will be the number of hours that have completed so far within the day.
  - For granularity **month**, the "count" will be the number of hourly data points recorded by the system within the month. For a past month, this will be the total number of days in that month X 24 hours-per-day; for the current month, this will be the number of hours that have completed so far in the month.

  Note For SB and SO, the "count" is relevant only insofar as it is used as the denominator in the calculation of an average storage value for the granularity interval (the numerator in the calculation is the "value" attribute).

- For operation type HG, HP, or HD, the "count" is the count of requests within the granularity interval (within the hour, day, or month). For example, if the operation type is HD and the granularity is hour, this is the count of HTTP Delete requests during the hour. Requests from whitelisted source IP addresses are excluded from HG, HP, or HD counts (unless the usage data is for a specific bucket, in which case the whitelist feature does not apply and whitelisted source addresses are not treated any differently than other source addresses in regard to usage tracking.)

Example:

```
"count": "744"
```

groupId

(String) Group ID with which the usage data is associated.

For bucket usage this attribute will have *null* value.

Example:

```
"groupId": "QA"
```

ip

(String) IP address of the client that submitted an S3 request. Applicable only if the usage reporting granularity is **raw** and the operation type is HG, HP, HD, BI, or BO. Otherwise this attribute will have *null* value.

Example:

```
"ip": ""
```

maxValue

(string) Maximum value recorded during the granularity interval. For example, for operation type SB this would be the largest storage byte level reached during the granularity interval. The "maxValue" is reported only for rollup granularities (hour, day, month). For raw granularity and for bucket usage data of any granularity it will have a value of "0".

Requests from whitelisted source addresses are excluded from HG, HP, HD "maxValue".

Example:

```
"maxValue": "305443"
```

operation

(String) Operation type for which the usage statistics are reported:

- SB = Storage Bytes
- SO = Storage Objects
- HG = S3 HTTP GETs (and HEADs)
- HP = S3 HTTP PUTs (and POSTs)
- HD = S3 HTTP DELETEs
- BI = For bucket usage only, the data transfer IN bytes
- BO = For bucket usage only, the data transfer OUT bytes
- TB = For bucket usage only, the total bytes count for the bucket.
- TO = For bucket usage only, the total objects count for the bucket.

Note The TB and TO operation types are supported only for bucket usage statistics. The TB and TO counts for a bucket for a specified time period (from startTime to endTime) will exist only if you previously executed the *POST /usage/repair/bucket* method one or more times during that time period. That method generates the TB and TO counts for the bucket, which the system then stores along with a timestamp indicating when the counts were generated. It's these saved TB and TO counts that are returned by *GET /usage* for the bucket. In the case of rolled-up usage data, the most recent TB and TO counts from within the roll-up period are used.

The SB and SO operation types are also supported for bucket usage statistics, but these will return the **change** in the stored bytes and stored object counts during the specified time interval -- for example, the total increase in a bucket's stored bytes total during each day in the interval (if you are using granularity "day"), rather than the total number of bytes in the bucket on each day. The latter is captured by the TB operation type.

Example:

```
"operation": "SB"
```

policyId

(String) System-generated unique ID of the storage policy used by the bucket.

This attribute is relevant only to bucket usage data and will be *null* for group or user-level usage data.

Example:

```
"policyId": "880e7d065225009b481ff24ae8d893ce"
```

region

(String) Service region in which the usage occurred.

For bucket usage this attribute will have *null* value.

Example:

```
"region": "taoyuan"
```

timestamp

(String) Timestamp for creation of this usage data, in UTC milliseconds. The specific meaning of the timestamp depends on the usage reporting granularity and operation type.

When usage report granularity = **raw**:

- For operation type SB or SO, the "timestamp" is the time when the storage level was recorded by the /usage/storage API call (which is run by cron job every five minutes)
- For operation type HG, HP, HD, BI, or BO, the "timestamp" is the time when the transaction occurred.
- For operation type TB or TO (supported for bucket usage only), the "timestamp" is the time when the *POST /usage/repair/bucket* call that calculated the TB and TO counts was executed.

When usage report granularity = **hour, day,** or **month**:

- The "timestamp" is the time that the granularity interval started (the start of the hour, day, or month for which data is encapsulated in the *UsageData* object).

Example:

```
"timestamp": "1498867200000"
```

uri

(String) URI of the data object. Applicable only for user and group level usage data and only if the usage reporting granularity is "raw". For user and group level usage data with granularity other than "raw", this attribute will have an empty value.

For bucket usage this attribute will have a *null* value.

Example:

```
"uri": ""
```

userId

(String) User ID with which the usage data is associated. For group level usage data the *userId* attribute will be "*".

For bucket usage this attribute will have *null* value.

Example:

```
"userId": "*"
```

value

(String) Data value. The specific meaning of this attribute depends on the usage reporting granularity and operation type.

When usage report granularity = **raw**:

- For operation type SB or SO (or TB or TO in the case of bucket usage statistics), the "value" is the current storage bytes or current number of stored objects.
- For operation type HG, HP, HD, BI, or BO, the "value" is the data transfer size for the single transaction, in bytes.

  Note With Multipart Upload operations (for large objects), each part upload counts as a separate transaction toward the HP and BI statistics.

When usage report granularity = **hour, day,** or **month**:

- For operation type SB or SO for user or group level usage data, the "value" is the sum of the storage level measures recorded by the system during the granularity interval (in bytes for SB or in number of objects for SO). For example, for granularity day, a current SB measure is recorded for each hour during the day, and the sum of those hourly measures is the SB "value" for the day. For SB and SO, this aggregate "value" is relevant only insofar as it is used as the numerator in the calculation of an average storage value for the granularity interval (the denominator in the calculation is the "count" attribute).

  Note In the case of bucket usage data the hour, day, or month “rollup” value for SB or SO is the **change** to the stored byte or stored object count in the bucket during the rollup period.

- For operation type HG, HP, HD, BI, or BO, the "value" is the sum data transfer size for the granularity interval, in bytes. For example, if operation type is HP and the granularity is hour, the "value" is the aggregated data transfer size of all HTTP PUT and POST requests during the hour.

Requests from whitelisted source IP addresses are excluded from HG, HP, and HD values (unless the usage data is for a specific bucket, in which case the whitelist feature does not apply and whitelisted source addresses are not treated any differently than other source addresses in regard to usage tracking.)

Example:

```
"value": "80319535"
```

whitelistAverageValue

(String) Same as "averageValue" above, except this is exclusively for traffic from whitelisted source addresses.

Note For bucket usage data, traffic from whitelisted sources is bundled in with the main usage statistics rather than being separated out. For bucket usage all "whitelist*" attributes will have "0" as their value.

Example:

```
"whitelistAverageValue": "0"
```

whitelistCount

(String) Same as "count", except this is exclusively for traffic from whitelisted source addresses. Example:

```
"whitelistCount": "0"
```

whitelistMaxValue

(String) Same as "maxValue" above, except this is exclusively for traffic from whitelisted source addresses. Example:

```
"whitelistMaxValue": "0"
```

whitelistValue

(String) Same as "value", except this is exclusively for traffic from whitelisted source addresses. Example:

```
"whitelistValue": "0"
```

13.2.6. Usage Data Calculation Notes

How Particular S3 Operations Impact Usage Data Counts

To support usage reporting, billing, and the implementation of Quality of Service (QoS) limits, the following counters are maintained for individual users and for groups:

- Storage bytes
- Storage objects
- Number of requests
- Data bytes IN
- Data bytes OUT

When calculating size for storage byte tracking, the size of the object metadata is included as well as the size of the object itself. If compression is used for storage of S3 objects, the uncompressed object size is counted toward storage byte tracking.

When calculating size for data transfer byte tracking (IN and OUT), the size of the HTTP headers is included as well as the size of the object itself.

The table below shows how particular S3 operations (the left-most column) impact the various service usage counters (shown in the remaining columns).

Operation	Storage Bytes	Storage Objects	Num Requests	Bytes IN	Bytes OUT
DELETE (Add delete marker)	Add Total-Size which is same as size of object path including bucketname (i.e., <bucketname>/<objectname>), unless replacing existing delete marker, then no change	Incremented by 1, unless replacing existing DM, then no change	No change	No change	No change

Operation	Storage Bytes	Storage Objects	Num Requests	Bytes IN	Bytes OUT
DELETE (No delete marker added) object, bucket	If object is successfully deleted, decremented by Total-Size of deleted object. If request is to region where bucket is not loc-ated, no change.	If object is successfully deleted, decremented by 1. If request is to region where bucket is not located, no change.	No change	No change	No change
DELETE object tag-ging	Decremented by size of old tag-ging string	No change	No change	No change	No change
DELETE policy	No change	No change	No change	No change	No change
DELETE uploadId (MP Abort)	If successfully deleted, decre-mented by Total-Size of uploaded parts and 1V value added in MP initiate	If suc-cessfully deleted, decremented by 1	No change	No change	No change
GET bucket, ser-vice, policy, loc-ation, acl, bucketlogging, ver-sioning, list uploads, list parts	No change	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
GET object	No change	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
GET object tagging	No change	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
HEAD object	No change	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
POST MP initiate	Add object name size and metadata size.	Incremented by 1	Incremented by 1	Add Transfer-Size of request	Add Transfer-Size of response
POST MP com-plete	If replacement object, decrement by Total-Size of old object. Total size of completed object metadata is set to total size of MP parts and initiate request.	If replace-ment object, decrement 1	Incremented by 1	Add Transfer-Size of request	Add Transfer-Size of response
POST object	Incremented by Total-size minus	Incremented	Incremented	Transfer-	Transfer-

Operation	Storage Bytes	Storage Objects	Num Requests	Bytes IN	Bytes OUT
	Total-size of old object, if any	by 1 if new object	by 1	Size of request	Size of response
PUT bucket	Incremented by bucketname size if bucket created in region, otherwise 0	Incremented by 1 if bucket created in region, otherwise 0	Incremented by 1	Transfer-Size of request	Transfer-Size of response
PUT bucket logging object	Incremented by Total-Size of log object	Incremented by 1	No change	No change	No change
PUT part	Add Content-Length of part body. If replacing an existing part, subtract Content-Length of old part body.	No change	Incremented by 1	Add Transfer-Size of request	Add Transfer-Size of response
PUT object	Incremented by Total-size minus Total-size of old object, if any	Incremented by 1 if new object	Incremented by 1	Transfer-Size of request	Transfer-Size of response
PUT object CRR (Cross Region Replication)	Incremented by Total-size of original object and replica object combined, plus 51 bytes of metadata associated with implementing CRR	Incremented by one if new object	Incremented by 1	Transfer-Size of request	Transfer-Size of response
PUT object copy	Two cases: (1) Metadata COPY. Increment by source total-size + difference between new and old object name. (2) Metadata REPLACE. Increment by source content-length + new objectname + new meta headers. In both cases, if replacement object, then Total-Size of replacement object is subtracted.	Incremented by one if new object	Incremented by 1	Transfer-Size of request	Transfer-Size of response
PUT policy, logging, acl, versioning	No change	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
PUT object tagging	Incremented by size of new tagging string minus size of old tagging string (if any)	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response
Upload Part Copy	Increment by source content-length. If replacement object, then Total-Size of replacement object is subtracted.	No change	Incremented by 1	Transfer-Size of request	Transfer-Size of response

How Request Processing Errors Impact Usage Counts

If an S3 request for uploading or downloading data fails to complete due to a processing error within the

HyperStore system, the request still counts towards the data transfer bytes total for usage tracking and QoS implementation. For example, if a user tries to upload a 1MiB object and the request fails to complete, the 1MiB is still added to the user’s total for Data Bytes In. It would not impact the user’s Stored Bytes or Stored Objects counts.

How Auto-Tiering Impacts Usage Counts

With the HyperStore auto-tiering feature, objects can -- on a specified scheduled -- be auto-tiered to Amazon S3 or a different S3 compliant destination system. When an object is transitioned to the destination system, its size is removed from the Storage Bytes count in the local HyperStore region. At the same time, a reference to the transitioned object is created and the size of this reference — 8KiB, regardless of the transitioned object size — is added to the local Storage Bytes count. For example, if a 100KiB object is auto-tiered to Amazon or a different HyperStore region or system, the net local effect is a 92KiB reduction in the local Storage Bytes count.

If the object is temporarily restored to local HyperStore storage (through the S3 API method POST Object restore), then while the object is locally restored the object’s size is added to the local Storage Bytes count and the 8KiB for the reference is subtracted from the count. After the restore interval ends, the object size is once again subtracted from Storage Bytes and the 8KiB for the reference is added back.

Auto-tiering does **not** impact the Storage Objects count.

Note In regard to the maximum stored bytes that your license permits you — for objects that have been auto-tiered to Amazon, the size of the tiered objects does not count toward your maximum allowed storage capacity. However, the 8KiB per tiered object (described above) **does** count toward your licensed maximum storage capacity.

How Server-Side Encryption Impacts Usage Counts

For Server-Side Encryption where the objects are encrypted in storage, "Bytes In" and "Bytes Out" reflect the original, unencrypted object size. The "Storage Bytes" value uses the encrypted object size. Headers are not encrypted, and thus not included. The increase of size of the encrypted object, i.e., the "padding size", depends on the AES block size and the amount of padding required.

The padding formula for AES/CBC/PKCS5 padding is as described below.

```
AES block size = 16

In the PKCS5 padding always a pad block is added at the end. So the padding
bytes vary from 1 to 16.

*Non-chunked objects*

Cipher size = (plain text size / 16 + 1) * 16.
Padding size = cipher size - plain text size

For example:

20 bytes object: total cipher size = (20/16 + 1) * 16 = 32 bytes
11 bytes object: total cipher size = (11/16 + 1) * 16 = 16 bytes

*Chunked objects*

Number of full chunks = plain text size / max chunk size
Last (partial) chunk size = plain text size % max chunk size
```

```
Cipher chunk size = (max chunk size / 16 + 1) * 16

- If last (partial) chunk size == 0
  last chunk padding size = 0

- If last (partial) chunk size > 0
  cipher last (partial) chunk size = (last (partial) chunk size / 16 + 1) * 16
  last chunk padding size = cipher last (partial) chunk size - last (partial) chunk size

padding size = number of full chunks * (cipher chunk size - max chunk size)
               + last chunk padding size

For example:
max chunk size=1024

1024 bytes object: total cipher size = plain text size + padding size
                                     = 1024 + 1*(1040-1024) + 0
                                     = 1024 + 16
                                     = 1040

1025 bytes object: total cipher size = plain text size + padding size
                                     = 1025 + 1*(1040-1024) + 15
                                     = 1025 + 16 + 15
                                     = 1056
```

How Compression Impacts Usage Counts

For S3 service usage tracking (for purposes of QoS enforcement and billing), the uncompressed size of objects is always used, even if you enable compression for all or some of your storage policies.

13.2.7. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters
400	Invalid parameter: region = {region}
400	Invalid parameter: regionOffset = {region}
400	Conflicting parameters: {canonicalUserId, id}

13.2.8. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUsage*
- Parameters: Same as for *GET /usage*, except all parameter names start with an upper case letter rather than lower case

- Response body: Same response data as for *GET /usage* except the data is formatted in XML rather than JSON
- Role-based restrictions:
    - HyperStore system admin user can get usage for any group, user, or bucket
    - HyperStore group admin user can only get usage for her own group, for users within her own group, or for buckets owned by users within her own group
    - HyperStore regular user can only get his own usage or usage for a bucket that he owns
    - IAM user can only use this method if granted *admin:GetCloudianUsage* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

    Note The "GetCloudianUsage" action retrieves usage data for Cloudian HyperStore user accounts, not for subsidiary IAM users. The system does not maintain usage data per IAM user. For example, if a HyperStore group administrator grants *admin:GetCloudianUsage* permission to an IAM user, the IAM user will be able to retrieve usage information for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianUsage* permission to an IAM user, the IAM user will be able to retrieve usage information for the **parent HyperStore user**.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUsage&Id=QA|*&Operation=SB&StartTime=201807010000
&EndTime=201808010000&Granularity=month

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUsageResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<UsageData>
etc...
...
...
</UsageData>
<UsageData>
etc...
...
...
</UsageData>
</ListWrapper>
</GetCloudianUsageResponse>
```

13.3. POST /usage/bucket

POST /usage/bucket Get raw usage data for multiple buckets

13.3.1. Syntax

```
POST /usage/bucket
```

The required request payload is a JSON-formatted *UsageBucketReq* object. See example below.

13.3.2. Usage Notes

This method retrieves complete **raw** usage data for one or multiple specified buckets, from during a specified time period. This method does not support retrieving rolled up hourly, daily, or monthly usage data and it does not support filtering by the service operation type.

The *POST /usage/bucket* method is supported only if bucket usage statistics are enabled in the system. Bucket usage statistics are disabled by default. For information on enabling this feature see "Preparing the Usage Reporting Feature" in the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

Note If you want to retrieve rolled up usage data for a bucket, or bucket usage data for just a particular service operation type, use the **GET /usage** method instead. Note however that with the **GET /usage** method you can only get usage data for one bucket at a time.

13.3.3. Example Using cURL

The **example** below retrieves raw usage data for two buckets named "b123" and "mybucket", for a one hour period. In this example the JSON-formatted *UsageBucketReq* object is specified in a text file named *buckets_usage.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @buckets_usage.txt https://localhost:19443/usage/bucket | python -mjson.tool
```

The *buckets_usage.txt* file content in this example is as follows.

```
{
  "buckets": [
    "b123",
    "mybucket"
  ],
  "endTime": "201611291900",
  "startTime": "201611291800"
}
```

The response payload is a JSON-formatted list of *UsageBucketRes* objects (with one such object for each bucket), which in this example is as follows. The response payload is truncated here.

```
[
  {
    "bucket": "b123",
    "data": [
      {
        "averageValue": "3222",
        "bucket": "b123",
        "count": "0",
        "groupId": null,
        "ip": "10.10.0.1",
        "maxValue": "0",
        "operation": "BO",
        "region": null,
        "timestamp": "1480442520000",
        "uri": null,
        "userId": null,
        "value": "3222",
        "whitelistAverageValue": "0",
        "whitelistCount": "0",
        "whitelistMaxValue": "0",
        "whitelistValue": "0"
      },
...
...
```

Note If during your specified start and end time interval there were no operations of a particular type in the bucket, then no data will be returned for that operation type. For example, if there were no deletes during the interval then no "HD" operation usage data will be returned.

13.3.4. Request Element Descriptions

buckets

(Mandatory, list<string>) List of the buckets for which to retrieve raw usage data. Example:

```
"buckets": ["b123","mybucket"]
```

endTime

(Mandatory, string) End time (in GMT) of the interval for which to retrieve raw usage data. Format is *yyyyMMddHHmm*. Example:

```
"endTime": "201611291900"
```

startTime

(Mandatory, string) Start time (in GMT) of the interval for which to retrieve raw usage data. Format is *yyyyMMddHHmm*. Example:

```
"startTime": "201611291800
```

13.3.5. Response Element Descriptions

bucket

(String) Bucket with which the usage data is associated. Example:

```
"bucket": "b123"
```

data

(List<*UsageData*>) List of *UsageData* objects. For descriptions of individual *UsageData* elements see **"GET /usage Get usage data for group, user, or bucket"** (page 183). Note that in the context of a *UsageBucketRes* object, the *UsageData* objects will always be for "raw" granularity.

13.3.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required attributes : {buckets, startTime, endTime}
400	Invalid JSON Object

13.4. POST /usage/repair

POST /usage/repair Repair storage usage data for group or system

13.4.1. Syntax

```
POST /usage/repair?groupId=string[&summarizeCountsOnly=bool][&region=string]
```

There is no request payload.

13.4.2. Parameter Descriptions

groupId

(Mandatory, string) The group for which to repair user-level and group-level storage usage counts. If groupId is "ALL", repair is performed for all groups.

summarizeCountsOnly

(Optional, boolean) If set to "true" while "groupId" = a specific group, then the operation will not validate or repair usage data counters for individual users within the specified group. Instead, it will presume the user-level counters to be correct, and will only sum up the user-level counters in order to update the counters for the group as a whole. This option is useful after you have been running `POST /usage/repair/user` operations (which validate and repair usage counters for individual users without updating the group-level counters for the groups that those users belong to).

If set to "true" while "groupId" = ALL, then the operation will only sum up the existing group-level usage counters to update the counters for the system as a whole.

If set to "false", then the operation runs in the normal manner, by first validating and repairing user-level usage counters within the specified group and then using that repaired data to update the group-level counters for the group.

Defaults to "false" if the "summarizeCountsOnly" parameter is omitted.

13.4.3. Usage Notes

This method checks and repairs storage usage data for specified user groups or for all groups in the system. For each repaired group the operation repairs the storage usage counts for individual users within the group as well as the aggregate counts for the group as a whole. If you have enabled per-bucket object and byte counts in your system, then the usage repair will check and repair those per-bucket counts also.

For background information on storage usage data repair, see "Validating Storage Usage Data" in the "Usage Reporting and Billing" section of the Cloudian HyperStore Administrator's Guide.

This is a resource-intensive operation if you have a large number of users in your system. Note that a more focused type of storage usage repair is run as a recurring HyperStore cron job -- see **POST /usage/repair/dirtyusers**.

Note In a multi-region HyperStore system, this method can be applied to usage data in all regions by submitting the request to the Admin Service in the default region and omitting the "region" query parameter. You cannot directly run this method against Admin Service nodes in non-default regions.

13.4.4. Example Using cURL

The **example** below checks and repairs storage usage data for the "engineering" group.

```
curl -X POST -k -u sysadmin:password \
https://localhost:19443/usage/repair?groupId=engineering
```

13.4.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

13.5. POST /usage/repair/bucket

POST /usage/repair/bucket   Retrieve total bytes and total objects for a bucket

13.5.1. Syntax

```
POST /usage/repair/bucket?bucket=string
```

There is no request payload.

13.5.2. Parameter Descriptions

bucket

(Mandatory, string): The bucket name.

13.5.3. Usage Notes

This method calculates and returns the current counts for total bytes stored and number of objects stored in a specified bucket. The calculation entails reading metadata in the Metadata DB for objects in the bucket.

This is potentially a resource-intensive operation, depending on how many objects are in the bucket.

> **Note In HyperStore 7.4 and later, the recommended methods for retrieving per bucket byte counts and object counts are the *GET /system/bytecount* and *GET /system/objectcount* calls**-- not the *POST /usage/repair/bucket* call.

13.5.4. Example Using cURL

The **example** below calculates and returns the current total bytes stored (TB) and total objects stored (TO) for a bucket named "testbucket1".

```
curl -X POST -k -u sysadmin:password \
https://localhost:19443/usage/repair/bucket?bucket=testbucket1 | python -mjson.tool
```

The response payload is the JSON-formatted TB and TO values.

```
{
  "TB": 305360,
  "TO": 9
}
```

13.5.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

13.6. POST /usage/repair/dirtyusers

POST /usage/repair/dirtyusers Repair storage usage data for users with recent activity

13.6.1. Syntax

```
POST /usage/repair/dirtyusers[?summarizeCounts=bool]
```

There is no request payload.

13.6.2. Parameter Descriptions

summarizeCounts

(Optional, boolean) If set to "true", then the *POST /usage/repair/dirtyusers* operation -- after repairing usage counters for individual users -- will update the group-level usage counters for the groups to which those repaired users belong. It will then also update system-level usage counts, based on the updated group counters.

If set to "false", then the operation will repair only user-level counters, and will not update the group or whole-system counters.

Defaults to "true".

13.6.3. Usage Notes

This method checks and repairs storage usage data for users whose storage bytes and/or storage object counts in the Redis QoS database have changed since the last time those users' counts were subjected to a usage repair. This method selects users at random from among this set of "dirty" users, and performs usage repair for a maximum number of 1000 of those users per method execution.

For background information on storage usage data repair, see "Validating Storage Usage Data" in the "Usage Reporting and Billing" section of the ***Cloudian HyperStore Administrator's Guide***.

Note This method is invoked once every 12 hours by a HyperStore usage data processing cron job. In a multi-region system, a separate cron job is run from within each region.

Note At the conclusion of this method's run, in *cloudian-admin.log* there will be an INFO level message from the CassandraUsageAccess::repairDirtyUsers component that indicates "1000 users processed. N remaining", where N is the number of remaining dirty users for whom usage repair was not performed.

Also in *cloudian-admin.log*, the CassandraUsageAccess::repairDirtyUsers component writes two INFO messages for each user processed — one message indicating the start of processing the user and one message indicating the completion of processing the user. If a correction was made to the user’s Redis QoS counts for stored bytes and/or stored objects, a third INFO message is sandwiched between the other two, indicating "Processed storage update: " and the correct counts.

13.6.4. Example Using cURL

The **example** below checks and repairs storage usage data for "dirty" users. It will also update group-level and system-level storage usage counts based on the repaired user-level counts.

```
curl -X POST -k -u sysadmin:password https://localhost:19443/usage/repair/dirtyusers
```

13.6.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

13.7. POST /usage/repair/user

POST /usage/repair/user   Repair storage usage data for a user

13.7.1. Syntax

```
POST /usage/repair/user?groupId=string&userId=string[&region=string]
```

There is no request payload.

13.7.2. Parameter Descriptions

groupId

(Mandatory, string) The ID of the group to which the target user belongs.

userId

(Mandatory, string) The ID of the user for whom usage data repair is to be performed.

region

(Optional, string) The region for which to perform the usage data repair. If the region parameter is not specified, the repair is performed for all service regions.

13.7.3. Usage Notes

This method checks and repairs storage usage data for a single specified user. If you have enabled per-bucket object and byte counts in your system, then the usage repair will also check and repair those per-bucket counts for the user's buckets.

This operation does not update the group-level usage counters for the group to which the user belongs. For information about doing the latter, see **POST /usage/repair** — particularly the "summarizeCountsOnly" option. This is relevant especially when you are repairing multiple individual users within a group, one at a time, using the *POST /usage/repair/user* method. In that case you should subsequently update the group-level usage counters for the group, using the **POST /usage/repair** method with the "summarizeCountsOnly" option.

13.7.4. Example Using cURL

The **example** below checks and repair storage usage data for the user "gladdes" in the "engineering" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/usage/repair/user?groupId=engineering&userId=gladdes'
```

13.7.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

13.8. POST /usage/rollup

POST /usage/rollup   Roll up usage data

13.8.1. Syntax

```
POST /usage/rollup?granularity=enum&startTime=string&unitCount=integer
```

There is no request payload.

13.8.2. Parameter Descriptions

granularity

(Mandatory, enum) The time period granularity of the usage data to generate. Supported values are:

- *hour* — Hourly rollup data
- *day* — Daily rollup data
- *month* — Monthly rollup data

startTime

(Mandatory, string) The start time **in GMT**. The format depends on the granularity of the usage data that you are generating:

- For hourly rollup data use format *yyyyMMddHH*. The start time will be the start of the hour that you specify.
- For daily rollup data use format *yyyyMMdd*. The start time will be the start of the day that you specify.
- For monthly rollup data use format *yyyyMM*. The start time will be the start of the month that you specify.

unitCount

(Optional, integer) The number of units of the specified "granularity" to generate. Supported range is [1,100].

Defaults to 1 unit if not specified.

13.8.3. Usage Notes

This method triggers the generation of "rollup" (aggregated across a time interval) service usage data from more granular data. Hourly rollup data is derived from "raw" transactional data. Daily rollup data and monthly rollup data are derived from hourly rollup data.

This method does not return the rolled up service usage data in the response, it only generates the rollup data and stores it in the system. To retrieve raw or rolled-up service usage data use the **GET /usage** method.

Note
*This API method is triggered by HyperStore usage data processing cron jobs. The cron job to create hourly rollup data runs each hour; the cron job to create daily rollup data runs once per day; and the

cron job to create monthly rollup data runs once per month.
* In a multi-region system the rollup operations act only on usage data in the local service region. Consequently, cron jobs that trigger these operations are configured in each region.

13.8.4. Example Using cURL

The **example** below creates hour roll-up usage data for the hour from midnight to 1AM on August 15, 2017.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/usage/rollup?granularity=hour&startTime=2017081500&unitCount=1'
```

13.8.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

13.9. POST /usage/storage

POST /usage/storage   Post raw storage usage data for users with recent activity

13.9.1. Syntax

```
POST /usage/storage
```

There is no request payload.

13.9.2. Usage Notes

The Redis QoS database maintains per-user and per-group counters for stored bytes and number of stored objects, based on transaction data that it receives from the S3 Service. This Admin API method writes these Redis-based stored bytes and stored object counts to the "Raw" column family in the Cassandra "Reports" keyspace. Subsequently the **POST /usage/rollup** method can be used to roll up this "Raw" data into hourly, daily, and monthly aggregate data in Cassandra.

This API method applies only to users who have uploaded or deleted objects since the last time this method was executed.

Note This API method is triggered every five minutes by a HyperStore usage data processing cron job. The method acts only on usage data in the local service region. Consequently, in a multi-region system, cron jobs that trigger this method are automatically configured in each region.

13.9.3. Example Using cURL

The **example** below triggers the writing of raw stored bytes and stored objects counts into Cassandra, for users who have been active since the last running of this API method.

```
curl -X POST -k -u sysadmin:password https://localhost:19443/usage/storage
```

13.9.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

13.10. POST /usage/storageall

POST /usage/storageall   Post raw storage usage data for all users

13.10.1. Syntax

```
POST /usage/storageall
```

There is no request payload.

13.10.2. Usage Notes

This method performs the same operation as described for **POST /usage/storage** except it applies to **all** users, not just recently active users.

Note This API method is triggered once each day by a HyperStore usage data processing cron job. The method acts only on usage data in the local service region. Consequently, in a multi-region system, cron jobs that trigger this method are automatically configured in each region.

13.10.3. Example Using cURL

The **example** below triggers the writing of raw stored bytes and stored objects counts into Cassandra, for all users.

```
curl -X POST -k -u sysadmin:password https://localhost:19443/usage/storageall
```

13.10.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13).

Chapter 14. user

The Admin API methods built around the **user** resource are for managing HyperStore account root users. This includes support for creating and deleting user accounts. These methods also support management of users' security credentials and the assignment of rating plans to users.

Note These API methods do not apply to IAM users that can be created by an account root user under the user's account. IAM users are created and managed via the IAM API. For information on Hyper-Store's support of the IAM API, see the IAM section of the ***Cloudian HyperStore AWS APIs Support Reference***.

Methods associated with the *user* resource:

- **"DELETE /user"** (page 210)
- **"DELETE /user/credentials"** (page 211)
- **"DELETE /user/deleted"** (page 212)
- **"DELETE /user/mfa/deleteDevice"** (page 213)
- **"GET /user"** (page 215)
- **"GET /user/credentials"** (page 217)
- **"GET /user/credentials/list"** (page 219)
- **"GET /user/credentials/list/active"** (page 222)
- **"GET /user/islocked"** (page 225)
- **"GET /user/list"** (page 226)
- **"GET /user/mfa/list"** (page 230)
- **"GET /user/password/verify"** (page 231)
- **"GET /user/ratingPlan"** (page 232)
- **"POST /user"** (page 236)
- **"POST /user/credentials"** (page 237)
- **"POST /user/credentials/status"** (page 238)
- **"POST /user/mfa/createDevice"** (page 239)
- **"POST /user/mfa/deactivateDevice"** (page 241)
- **"POST /user/mfa/enableDevice"** (page 242)
- **"POST /user/mfa/resyncDevice"** (page 244)
- **"POST /user/mfa/verify"** (page 245)
- **"POST /user/password"** (page 247)
- **"POST /user/ratingPlanId"** (page 248)
- **"POST /user/unlock"** (page 249)
- **"PUT /user"** (page 251)
- **"PUT /user/credentials"** (page 256)

14.1. DELETE /user

DELETE /user Delete a user

14.1.1. Syntax

```
DELETE /user?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.1.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

> **Note**
> * You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
> * The group ID for system admins is "0".

14.1.3. Usage Notes

Note that by default you cannot delete:

- A user who still owns buckets. To disable this restriction and allow deletion of users who own buckets, use the configuration setting *common.user.allowDeleteWithBuckets*.
- A user who still owns IAM resources (IAM groups, users, roles, or policies). To disable this restriction and allow deletion of users who own IAM resources, use the configuration setting *common.user.allowDeleteWithIAMResources*.

The operations associated with deleting a user are performed asynchronously. If you receive an OK response to a *DELETE /user* request, this indicates that the user's status has successfully transitioned to "deleting", and the associated operations are underway. You can use the **GET /user/list** method to check on which users within a group are in "deleting" status or "deleted" status ("deleted" status indicates that all associated operations have completed, including deletion of the user's stored S3 buckets and objects).

> **Note** Service usage report data for a deleted user is retained for a period of time as configured by the *s3.usage.reports.monthlyRollupTTL* setting. You can retrieve usage data for a recently deleted user via the **GET /usage** method.

Note Regarding system admin users:
* You cannot delete the **default** system administrator account (the user with user ID "admin"). This is not allowed.
* For other system admin users, deleting the user also deletes the user's HyperStore Shell account.

14.1.4. Example Using cURL

The **example** below deletes a user with ID "John" who is in the "QA" group.

```
curl -X DELETE -k -u sysadmin:password \
'https://localhost:19443/user?userId=John&groupId=QA'
```

14.1.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId}
400	User does not exist

14.2. DELETE /user/credentials

DELETE /user/credentials Delete a user's S3 security credential

14.2.1. Syntax

```
DELETE /user/credentials?accessKey=urlencoded-string
```

There is no request payload.

14.2.2. Parameter Descriptions

accessKey

(Mandatory, string) The S3 access key from the key pair to delete. Must be URL-encoded if the key includes non-ASCII characters.

Note An S3 security credential is a key pair consisting of an "access key" (public key) and a "secret key" (private key).

14.2.3. Example Using cURL

The **example** below deletes a user's S3 credential as specified by the access key. Note that since each S3 access key is unique in the system, you do not need to specify the user to whom the key is assigned.

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/user/credentials?accessKey=21289bab1738ffdc792a
```

14.2.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {accessKey}
400	Invalid Access Key

14.3. DELETE /user/deleted

DELETE /user/deleted   Purge profile data of a deleted user or users

14.3.1. Syntax

```
DELETE /user/deleted[?canonicalUserId=string|groupId=string]
```

There is no request payload.

14.3.2. Parameter Descriptions

canonicalUserId

(Optional, string) System-generated canonical user ID of the deleted user for whom to purge profile data.

If you don't know the user's canonical ID you can retrieve it via the Admin API methods *GET /user* or *GET /user/list*.

groupId

(Optional, string) Unique identifier of the group for which to purge all user profile data for deleted users.

14.3.3. Usage Notes

After deleting a user or users, you can use this Admin API method if you want to purge the deleted users' profile information from the Cassandra database. Otherwise, the deleted users' profile information is retained in Cassandra indefinitely.

Use the *canonicalUserId* parameter to specify just a single user for whom to purge profile data, **or** use the *groupId* parameter to purge profile data for all deleted users in the specified group. Do not use both *canonicalUserId* and *groupId* together.

IMPORTANT ! If you purge a deleted user’s profile information, you will no longer be able to retrieve that user’s profile information via the **GET /user/list** method. This means that you will no longer be able to retrieve the deleted user’s canonical user ID. Without a deleted user’s canonical user ID, you will not be able to retrieve usage history for the user. Consequently, you should purge a deleted user’s profile information **only if** you have some independent record of the user’s canonical user ID (outside of the Cassandra database); or if you are confident that you will no longer require access to the deleted user’s usage history.

14.3.4. Example Using cURL

The **example** below purges a single deleted user's profile data.

```
curl -X DELETE -k -u sysadmin:password \
https://localhost:19443/user/deleted?canonicalUserId=bd0796cd9746ef9cc4ef656ddaacfac4
```

14.3.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Conflicting or missing parameters : {canonicalUserId, groupId}
400	User does not exist or is not in a deleted state.

14.4. DELETE /user/mfa/deleteDevice

DELETE /user/mfa/deleteDevice   Delete an MFA device from a user's account

14.4.1. Syntax

```
DELETE /user/mfa/deleteDevice?[userId=string&groupId=string]
[canonicalUserId=string]&serialNumber=string
```

There is no request payload.

14.4.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

serialNumber

Serial number of the MFA device to delete. This takes the form of an ARN. Here is an example of the MFA serial number format used for HyperStore:

```
arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device
```

14.4.3. Usage Notes

Use this method to delete an MFA device from a HyperStore account root user.

Note You cannot delete an MFA device that is currently enabled/activated. If the device is activated, before you can delete it you must first deactivate it by calling the **POST /user/mfa/deactivateDevice** method.

14.4.4. Example Using cURL

The **example** below deletes an MFA device from the account of the user "PubsUser1" in the "Pubs" group.

```
curl -X DELETE -k -u sysadmin:password \
'https://localhost:19443/user/mfa/deleteDevice?userId=PubsUser1&groupId=Pubs&
serialNumber=arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device'
```

14.4.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	User is inactive or is not found
400	Invalid parameter or missing required parameter
404	Specified MFA Device does not exist or does not belong to specified user

Status Code	Description
409	Device is active. Must deactivate before deleting.

14.5. GET /user

GET /user   Get a user's profile

14.5.1. Syntax

```
GET /user?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.5.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

14.5.3. Usage Notes

To retrieve profile information for a user who has been deleted from the system, use the "canonicalUserId".

14.5.4. Example Using cURL

The **example** below retrieves a user with ID "John" who is in the "QA" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user?userId=John&groupId=QA' | python -mjson.tool
```

The response payload is a JSON-formatted *UserInfo* object, which in this example is as follows.

```
{
  "active": "true",
  "address1": "",
  "address2": "",
  "canonicalUserId": "bd0796cd9746ef9cc4ef656ddaacfac4",
  "city": "",
  "country": "",
  "emailAddr": "",
  "fullName": "John Thompson",
  "groupId": "QA",
  "ldapEnabled": false,
  "phone": "",
  "state": "",
  "userId": "John",
  "userType": "User",
  "website": "",
  "zip": ""
}
```

14.5.5. Response Element Descriptions

For descriptions of *UserInfo* object elements see **"PUT /user Create a new user"** (page 251).

14.5.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	User does not exist
400	Missing Required parameters : {userId, groupId}

14.5.7. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUser*
- Parameters: Same as for *GET /user*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /user* except the data is formatted in XML rather than JSON
- Role-based restrictions:
  - HyperStore system admin user can get any user's profile
  - HyperStore group admin user can only get the profiles of users within her own group
  - HyperStore regular user can only get own profile
  - IAM user can only use this method if granted *admin:GetCloudianUser* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

Note The "GetCloudianUser" action retrieves profile data for Cloudian HyperStore user accounts, not for subsidiary IAM users. For example, if a HyperStore group administrator grants *admin:GetCloudianUser* permission to an IAM user, the IAM user will be able to retrieve profile information for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianUser* permission to an IAM user, the IAM user will be able to retrieve profile information for the **parent HyperStore user**.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUser&UserId=John&GroupId=QA

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUserResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<CassandraUserInfo>
<active>Active</active>
etc...
...
...
</CassandraUserInfo>
</GetCloudianUserResponse>
```

14.6. GET /user/credentials

GET /user/credentials Get a user's S3 secret key corresponding to a supplied access key

14.6.1. Syntax

```
GET /user/credentials?accessKey=urlencoded-string
```

There is no request payload.

14.6.2. Parameter Descriptions

accessKey

(Mandatory, string) The S3 access key from the key pair to retrieve. Must be URL-encoded if the key includes non-ASCII characters.

Note An S3 security credential is a key pair consisting of an "access key" (public key) and a "secret key" (private key).

14.6.3. Example Using cURL

The **example** below retrieves the S3 credentials object corresponding to a specified S3 access key. Note that since each S3 access key is unique in the system, you do not need to specify the user to whom the key is assigned.

```
curl -X GET -k -u sysadmin:password \
https://localhost:19443/user/credentials?accessKey=009c156c79e64e0e4928 \
| python -mjson.tool
```

The response payload is a JSON-formatted *SecurityInfo* object, which in this example is as follows.

```
{
  "accessKey": "009c156c79e64e0e4928",
  "active": true,
  "createDate": 1502279336024,
  "expireDate": null,
  "secretKey": "wVHk2nA0M03RWSMIrFHFAtuhow6S1DKN0gWjPhDG"
}
```

14.6.4. Response Element Descriptions

For descriptions of *SecurityInfo* object elements see **"PUT /user/credentials Create a new S3 credential for a user"** (page 256).

14.6.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Data Found
400	Missing required parameters : {accessKey}

14.6.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUserCredentials*
- Parameters: Same as for *GET /user/credentials*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /user/credentials* except:
    - The data is formatted in XML rather than JSON
    - The *secretKey* is not included in the response

- Role-based restrictions:
  - HyperStore system admin user can get any user's credentials
  - HyperStore group admin user can only get the credentials of users within her own group
  - HyperStore regular user can only get own credentials
  - IAM user can only use this method if granted *admin:GetCloudianUserCredentials* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

    Note The "GetCloudianUserCredentials" action retrieves credentials for Cloudian HyperStore user accounts, not for subsidiary IAM users. For example, if a HyperStore group administrator grants *admin:GetCloudianUserCredentials* permission to an IAM user, the IAM user will be able to retrieve credentials for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianUserCredentials* permission to an IAM user, the IAM user will be able to retrieve the **parent HyperStore user's** credentials.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUserCredentials&AccessKey=009c156c79e64e0e4928

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUserCredentialsResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<SecurityInfo>
<accessKey>40602cdfaef3a676594d</accessKey>
etc...
...
</SecurityInfo>
</GetCloudianUserCredentialsResponse>
```

14.7. GET /user/credentials/list

GET /user/credentials/list   Get a user's list of S3 security credentials

14.7.1. Syntax

```
GET /user/credentials/list?[userId=string&groupId=string][canonicalUserId=string]
[isRootAccountOnly=boolean]
```

There is no request payload.

14.7.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

isRootAccountOnly

If this parameter is included in the call and set to *true*, then only credentials belonging to the HyperStore account root user will be return. If this parameter is omitted or set to *false*, this call will return the account root user's credentials and also the credentials of any IAM users that have been created under the account root.

14.7.3. Usage Notes

This method retrieves **all** of the user's S3 credentials -- active credentials as well as inactive (disabled) credentials.

14.7.4. Example Using cURL

The **example** below retrieves all of the S3 security credentials belonging to user "John" in the "QA" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/credentials/list?userId=John&groupId=QA' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *SecurityInfo* objects, which in this example is as follows.

```
[
  {
    "accessKey": "009c156c79e64e0e4928",
    "active": true,
    "createDate": 1502279336024,
    "expireDate": null,
    "secretKey": "wVHk2nA0M03RWSMIrFHFAtuhow6S1DKN0gWjPhDG"
  },
  {
    "accessKey": "21289bab1738ffdc792a",
    "active": false,
    "createDate": 1502283467021,
    "expireDate": null,
    "secretKey": "o5jqJtqV36+sENGLozEUg1EXEmQp9V6yfCHLFCJk"
  }
]
```

14.7.5. Response Element Descriptions

For descriptions of *SecurityInfo* object elements see **"PUT /user/credentials   Create a new S3 credential for a user"** (page 256).

14.7.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Access Key found
400	Missing Required parameters : {userId, groupId}
400	User/Group does not exist

14.7.7. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUserCredentialsList*
- Parameters: Same as for *GET /user/credentials/list*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /user/credentials/list* except:
  - The data is formatted in XML rather than JSON
  - The *secretKey* is not included in the response
- Role-based restrictions:
  - HyperStore system admin user can get any user's credentials list
  - HyperStore group admin user can only get the credentials list of users within her own group
  - HyperStore regular user can only get own credentials list
  - IAM user can only use this method if granted *admin:GetCloudianUserCredentialsList* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

    Note The "GetCloudianUserCredentialsList" action retrieves credentials for Cloudian HyperStore user accounts, not for subsidiary IAM users. For example, if a HyperStore group administrator grants *admin:GetCloudianUserCredentialsList* permission to an IAM user, the IAM user will be able to retrieve a credentials list for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianUserCredentialsList* permission to an IAM user, the IAM user will be able to retrieve the **parent HyperStore user's** credentials list.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUserCredentialsList&UserId=John&GroupId=QA

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUserCredentialsListResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<SecurityInfo>
<accessKey>40602cdfaef3a676594d</accessKey>
etc...
...
</SecurityInfo>
<SecurityInfo>
etc...
...
</SecurityInfo>
</ListWrapper>
</GetCloudianUserCredentialsListResponse>
```

14.8. GET /user/credentials/list/active

GET /user/credentials/list/active   Get a user's list of active S3 security credentials

14.8.1. Syntax

```
GET /user/credentials/list/active?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.8.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

14.8.3. Usage Notes

This retrieves the user's **active** S3 credentials. Inactive (disabled) credentials are not returned.

14.8.4. Example Using cURL

The **example** below retrieves the active S3 credentials for user "John" in the "QA" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/credentials/list/active?userId=John&groupId=QA' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *SecurityInfo* objects, which in this example is as follows (note that this user has only one active credential).

```
[
  {
    "accessKey": "009c156c79e64e0e4928",
    "active": true,
    "createDate": 1502279336024,
    "expireDate": null,
    "secretKey": "wVHk2nA0M03RWSMIrFHFAtuhow6S1DKN0gWjPhDG
  }
]
```

14.8.5. Response Element Descriptions

For descriptions of *SecurityInfo* object elements see **"PUT /user/credentials   Create a new S3 credential for a user"** (page 256).

14.8.6. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	No Access Key found
400	Missing Required parameters : {userId, groupId}
400	User/Group does not exist

14.8.7. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUserCredentialsListActive*
- Parameters: Same as for *GET /user/credentials/list/active*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /user/credentials/list/active* except:
    - The data is formatted in XML rather than JSON
    - The *secretKey* is not included in the response
- Role-based restrictions:
    - HyperStore system admin user can get any user's active credentials list
    - HyperStore group admin user can only get the active credentials list of users within her own group
    - HyperStore regular user can only get own active credentials list
    - IAM user can only use this method if granted *admin:GetCloudianUserCredentialsListActive* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

    Note The "GetCloudianUserCredentialsListActive" action retrieves credentials for Cloudian HyperStore user accounts, not for subsidiary IAM users. For example, if a HyperStore group administrator grants *admin:GetCloudianUserCredentialsListActive* permission to an IAM user, the IAM user will be able to retrieve an active credentials list for **any HyperStore user in the group administrator's group**. And if a HyperStore regular user grants *admin:GetCloudianUserCredentialsListActive* permission to an IAM user, the IAM user will be able to retrieve the **parent HyperStore user's** active credentials list.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUserCredentialsListActive&UserId=John&GroupId=QA

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUserCredentialsListActiveResponse xmlns="https://iam.amazonaws.com/doc/2010-05-
08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<SecurityInfo>
<accessKey>40602cdfaef3a676594d</accessKey>
etc...
...
</SecurityInfo>
<SecurityInfo>
etc...
...
```

```
</SecurityInfo>
</ListWrapper>
</GetCloudianUserCredentialsListActiveResponse>
```

14.9. GET /user/islocked

GET /user/islocked Get a user's lock-out status

14.9.1. Syntax

```
GET /user/islocked?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.9.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

14.9.3. Usage Notes

This method retrieves the user's CMC password lock-out status. A user may become temporarily locked out of the CMC if you have the password lock-out feature enabled in your system, and within a defined time interval the user makes too many login attempts using an incorrect password. The password-lock feature is configurable by the *common.user.password.lock.enabled* setting and the other *common.user.password.lock.** settings.

If a user is locked out, the lock-out will expire automatically after a configurable period (30 minutes by default). Alternatively, you can unlock a user immediately by using the **"POST /user/unlock Unlock a locked-out user"** (page 249) method.

14.9.4. Example Using cURL

The **example** below retrieves the lock-out status of the user *John* in the *QA* group. The response indicates that John is locked out.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/islocked?userId=John&groupId=QA'

true
```

14.9.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or conflicting parameters : {userId, groupId,canonicalUserId}
400	User/Group does not exist

14.10. GET /user/list

GET /user/list   Get a list of user profiles

14.10.1. Syntax

```
GET /user/list?groupId=string&userType=enum&userStatus=enum[&prefix=string]
[&limit=integer][&offset=string][&extended=boolean]
```

There is no request payload.

14.10.2. Parameter Descriptions

groupId

(Mandatory, string) Unique identifier of the group for which to retrieve a user list.

Note The group ID for system admins is "0".

userType

(Mandatory, enum) Retrieve users of this type. Options are:

- *admin* — Administrators. If the "groupId" parameter is set to "0" this would be system admins; for any other group this would be group admins.
- *user* — Regular users who lack administrative privileges.
- *all* — Retrieve users of all types.

userStatus

(Mandatory, enum) Retrieve users who have this status. Options are:

- *active* — Active users.
- *inactive* — Inactive users. These users have had their status set to inactive via the *POST /user*

method (with the *UserInfo* object attribute "active" set to false). These users' stored S3 objects still exist, but their S3 access credentials have been deactivated.

- *deleted* — Deleted users. These users have been deleted from the service via the *DELETE /user* method. Their S3 access credentials have been deleted, and their S3 buckets and objects have been deleted and are unrecoverable.
- *deleting* — These users are in the process of being deleted from the service via the *DELETE /user* method. The deletion process for these users has not yet completed.
- *all* — Retrieve active users and inactive users. This does **not** retrieve users who have status "deleted" or "deleting". To retrieve deleted or deleting users, specify "deleted" or "deleting" for the *userStatus* request parameter -- not "all".

> **Note** Since the CMC does not support retrieving users with status "deleted" or "deleting", the only way to retrieve a list of such users is through the Admin API's *GET /user/list* method.

prefix

(Optional, string) If specified, a user ID prefix to use for filtering. For example, if you specify "prefix=arc" then only users whose user ID starts with "arc" would be retrieved.

Defaults to empty string (meaning that no prefix-based filtering is performed).

limit

(Optional, integer) For purposes of pagination, the maximum number of users to return in one response. In the response the users are sorted alphanumerically and if more than "limit" users meet the filtering criteria, then the actual number of users returned will be "limit plus 1" (for example, 101 users if the limit is 100). The last, extra returned user — the "plus 1" — is an indicator that there are more users than could be returned in the current response (given the specified "limit" value). That last user’s ID can then be used as the "offset" value in a subsequent request that retrieves additional users.

> **Note** If the offset user happens to be the last user in the entire set of matching users, the subsequent query using the offset will return no users.

Defaults to 100.

offset

(Optional, string) The user ID with which to start the response list of users for the current request, sorted alphanumerically. The "offset" parameter can be used for purposes of pagination within a large result set that is being retrieved via multiple sequential requests. See the description of "limit" above for more information.

If "offset" is not specified, the first user in the response list will be the alphanumerically first user from the entire result set.

extended

(Optional, boolean) If *extended=true*, then in the *GET user/list* response each listed *UserInfo* object will be extended to also include the user's lock-out status, and the user's MFA enablement date and device serial number (if the user has MFA enabled).

Defaults to false.

14.10.3. Example Using cURL

The **example** below retrieves a list of all active users in the "QA" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/list?groupId=QA&userType=all&userStatus=active' \
| python -mjson.tool
```

The response payload is a JSON-formatted list of *UserInfo* objects, which in this example is as follows.

```
[
  {
    "active": "true",
    "address1": "",
    "address2": "",
    "canonicalUserId": "fd221552ff4ddc857d7a9ca316bb8344",
    "city": "",
    "country": "",
    "emailAddr": "",
    "fullName": "Glory Bee",
    "groupId": "QA",
    "ldapEnabled": false,
    "phone": "",
    "state": "",
    "userId": "Glory",
    "userType": "User",
    "website": "",
    "zip": ""
  },
  {
    "active": "true",
    "address1": "",
    "address2": "",
    "canonicalUserId": "bd0796cd9746ef9cc4ef656ddaacfac4",
    "city": "",
    "country": "",
    "emailAddr": "",
    "fullName": "John Thompson",
    "groupId": "QA",
    "ldapEnabled": false,
    "phone": "",
    "state": "",
    "userId": "John",
    "userType": "User",
    "website": "",
    "zip": ""
  },
  {
    "active": "true",
    "address1": "",
    "address2": "",
    "canonicalUserId": "4dc9cd1c20c78eb6c84bb825110fddcb",
    "city": "",
    "country": "",
    "emailAddr": "",
    "fullName": "Xiao Li",
    "groupId": "QA",
    "ldapEnabled": false,
    "phone": "",
    "state": "",
    "userId": "Xiao",
    "userType": "GroupAdmin",
    "website": "",
    "zip": ""
  }
]
```

14.10.4. Response Element Descriptions

For descriptions of *UserInfo* object elements see **"PUT /user   Create a new user"** (page 251).

14.10.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {groupId, userType, userStatus}
400	Invalid user type. Valid values {admin, user, all}
400	Invalid user status. Valid values {active, inactive, all}
400	Invalid limit

14.10.6. RBAC Version of this Method

HyperStore's IAM Service supports an RBAC version of this API method. For an overview of the RBAC feature see **"Role-Based Access to Admin API Operations"** (page 17).

- Action name: *GetCloudianUserList*
- Parameters: Same as for *GET /user/list*, except all parameter names start with an upper case letter rather than lower case
- Response body: Same response data as for *GET /user/list* except the data is formatted in XML rather than JSON
- Role-based restrictions:
    - HyperStore system admin user can get any group's user list
    - HyperStore group admin user can only get the user list for her own group
    - HyperStore regular user cannot use this method
    - IAM user can only use this method if granted *admin:GetCloudianUserList* permission by an IAM policy, and subject to the same restriction as the parent HyperStore user.

Note The "GetCloudianUserList" action retrieves user profile data for Cloudian

HyperStore user accounts, not for subsidiary IAM users. For example, if a HyperStore group administrator grants *admin:GetCloudianUserList* permission to an IAM user, the IAM user will be able to retrieve profile information for **any HyperStore user in the group administrator's group**.

- Sample request and response (abridged):

```
REQUEST

http://localhost:16080/?Action=GetCloudianUserList&GroupId=QA&UserType=all&UserStatus=active

<request headers including authorization info>

RESPONSE

200 OK

<?xml version="1.0" encoding="utf-8"?>
<GetCloudianUserListResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
<ResponseMetadata>
<RequestId>system-generated-request-id</RequestId>
</ResponseMetadata>
<ListWrapper>
<CassandraUserInfo>
<active>Active</active>
etc...
...
</CassandraUserInfo>
<CassandraUserInfo>
etc...
...
</CassandraUserInfo>
</ListWrapper>
</GetCloudianUserListResponse>
```

14.11. GET /user/mfa/list

GET /user/mfa/list Get a list of a user's MFA devices

14.11.1. Syntax

```
GET /user/mfa/list?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.11.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

14.11.3. Usage Notes

Use this method to retrieve information about the MFA device(s) currently activated for the specified HyperStore account root user, if any.

14.11.4. Example Using cURL

The **example** below retrieves the MFA device that is activated for user "PubsUser1" in the "Pubs" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/mfa/list?userId=PubsUser1&groupId=Pubs'
```

The response payload is a JSON-formatted list of *MFADevice* objects, which in this example is as follows.

```
{"MFADevices":[{"EnableDate":"2024-04-02T13:56:38Z","SerialNumber":"arn:aws:iam::
79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device"}]}
```

14.11.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Invalid parameter or missing required parameter
404	User inactive or not found

14.12. GET /user/password/verify

GET /user/password/verify   Verify a user's supplied password

14.12.1. Syntax

```
GET /user/password/verify?userId=string&groupId=string&password=string
```

There is no request payload.

14.12.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

14.12.3. Usage Notes

This method verifies that the supplied CMC password is the correct password for the user.

It also verifies that the user is not currently locked out by the CMC password lockout feature. If the user is currently locked out then password verification will fail even if the supplied password is the correct password for the user.

14.12.4. Example Using cURL

The **example** below verifies the supplied password for a user "John" in the "QA" group.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/password/verify?userId=John&groupId=QA&password=P1a2s3s4!'
```

The response payload is a plain text value: "valid" or "invalid" or "expire" (password needs to be changed). In this example the response is:

```
valid
```

The "valid" response indicates that the supplied password is the correct and still valid password for the user.

14.12.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or this method-specific status code:

Status Code	Description
400	Missing Required parameters : {userId, groupId, password}

14.13. GET /user/ratingPlan

GET /user/ratingPlan   Get a user's rating plan content

14.13.1. Syntax

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

There is no request payload.

14.13.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

> **Note**
> * You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
> * The group ID for system admins is "0".

14.13.3. Example Using cURL

The **example** below retrieves the content of the rating plan that is assigned to user "John" in group "QA" in the default service region.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/ratingPlan?userId=John&groupId=QA' \
| python -mjson.tool
```

The response payload is a JSON-formatted *RatingPlan* object, which in this example is as follows.

```
{
  "currency": "USD",
  "id": "Gold",
  "mapRules": {
    "BI": {
      "ruleclassType": "BYTES_IN",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "BO": {
      "ruleclassType": "BYTES_OUT",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HD": {
      "ruleclassType": "HTTP_DELETE",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HG": {
      "ruleclassType": "HTTP_GET",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "HP": {
      "ruleclassType": "HTTP_PUT",
      "rules": [
        {
          "first": "0",
          "second": "0"
        }
      ]
    },
    "SB": {
      "ruleclassType": "STORAGE_BYTE",
      "rules": [
        {
          "first": "100",
          "second": "0.25"
        },
        {
          "first": "0",
          "second": "0.15"
        }
      ]
    }
  },
  "name": "Gold Rating Plan"
}
```

14.13.4. Response Element Descriptions

For descriptions of *RatingPlan* object elements see **"PUT /ratingPlan Create a new rating plan"** (page 136).

14.13.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Rating Plan does not exist
400	Missing Required parameters : {userId, groupId}
400	Region {region} is not valid

14.14. GET /user/ratingPlanId

GET /user/ratingPlanId   Get a user's rating plan ID

14.14.1. Syntax

```
GET /user/ratingPlanId?userId=string&groupId=string[&region=string]
```

There is no request payload.

14.14.2. Parameter Descriptions

userId

(Mandatory, string) User identifier, unique within the group. This is the user ID that was supplied by the user (or by whoever created the user) at the time of user account creation.

groupId

(Mandatory, string) Unique identifier of the group to which the user belongs.

region

(Optional, string) If your service deployment has multiple service regions, rating plan assignment is on a per-region basis. Use the "region" parameter to indicate the service region for which to retrieve a rating plan ID. If this parameter is not supplied, the default region is assumed.

14.14.3. Example Using cURL

The **example** below retrieves the rating plan ID for user "John" in group "QA" in the default service region.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/ratingPlanId?userId=John&groupId=QA'
```

The response payload is the rating plan identifier in plain text, which in this example is as follows.

```
Gold
```

14.14.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	Rating Plan does not exist
400	Missing Required parameters : {userId, groupId}
400	Region {region} is not valid

14.15. POST /user

POST /user   Change a user's profile

14.15.1. Syntax

```
POST /user
```

The required request payload is a JSON-formatted *UserInfo* object.

14.15.2. Example Using cURL

The **example** below modifies the user profile that was created in the **PUT /user** example. Again the *UserInfo* object is specified in a text file named *user_John.txt* which is then referenced as the data input to the cURL command.

```
curl -X POST -H "Content-Type: application/json" -k -u sysadmin:password \
-d @user_John.txt https://localhost:19443/user
```

Note that in editing the *UserInfo* object in the *user_John.txt* file before doing the POST operation you could edit any attribute except for the "userId" or "canonicalUserId" attributes. For an example *UserInfo* object see **PUT /user**.

Note You cannot change the "userType" attribute of the default system administrator account. This is not allowed.

14.15.3. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	User does not exist
400	Missing Required parameters : {userId, groupId, userType}

Status Code	Description
400	Invalid JSON object
400	Invalid User Name

14.16. POST /user/credentials

POST /user/credentials   Post a user's supplied S3 credential

14.16.1. Syntax

```
POST /user/credentials?userId=string&groupId=string&accessKey=urlencoded-string
&secretKey=urlencoded-string[&iamUser=urlencoded-string]
```

There is no request payload.

14.16.2. Parameter Descriptions

userId

(Mandatory, string) User identifier, unique within the group. This is the user ID that was supplied by the user (or by whoever created the user) at the time of user account creation.

groupId

(Mandatory, string) Unique identifier of the group to which the user belongs.

Note The group ID for system admins is "0".

accessKey

(Mandatory, string) The S3 access key. Must be URL-encoded.

Note An S3 security credential is a key pair consisting of an "access key" (public key) and a "secret key" (private key).

Note If the *iamUser* parameter is used, the *accessKey* value must match against the pattern *[a-zA-Z0-9_]* and must be at least 16 characters and no more than 128 characters long.

secretKey

(Mandatory, string) The S3 secret key. Must be URL-encoded.

iamUser

(Optional, string) IAM user name, if the *POST /user/credentials* call is being used to supply security credentials for an IAM user rather than for a HyperStore account root user. Note that:

- Using this parameter is appropriate if an IAM user has an access key and secret key that were created in an external system, and you want the IAM user to be able to use those same security credentials for accessing HyperStore services.
- The IAM user must exist in the HyperStore system (the IAM user must be created through the CMC or through use of the HyperStore IAM Service by a third party IAM client application).
- If the *iamUser* parameter is used with this Admin API call, the *userId* and *groupId* parameters must be used to identify the HyperStore account root user under whom the IAM user exists.
- The iamUser value must match against the pattern *[a-zA-Z0-9_+=,.@-]+*. The minimum length is 1, maximum 64. Example valid value: *tom@company.com*
- Must be URL-encoded.

14.16.3. Usage Notes

HyperStore does not support commas in the *accessKey* value or the *secretKey* value. If a comma is present in either of those values, the POST request will fail with a 400 "Bad Request" error.

14.16.4. Example Using cURL

The **example** below posts a supplied S3 access key and secret key for user "John" in the "QA" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/credentials?userId=John&groupId=QA&accessKey=21289&secretKey=o5jqJtq'
```

Note To allow the single quote-enclosed *'https:...'* segment in the above example to be shown on one line, the access key and secret key values are truncated.

14.16.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId, accessKey, secretKey}
400	User does not exist (or IAM user does not exist, if *iamUser* parameter is used)
400	Invalid characters in accessKey or secretKey
403	Reached maximum number of credentials allowed (or maximum number for the IAM user, if *iamUser* parameter is used)
409	Access Key already exists

14.17. POST /user/credentials/status

POST /user/credentials/status	Deactivate or reactivate a user's S3 credential

14.17.1. Syntax

```
POST /user/credentials/status?accessKey=urlencoded-string[&isActive=bool]
```

There is no request payload.

14.17.2. Parameter Descriptions

accessKey

(Mandatory, string) The S3 access key from the key pair. Must be URL-encoded if the key includes non-ASCII characters.

> **Note** An S3 security credential is a key pair consisting of an "access key" (public key) and a "secret key" (private key).

isActive

(Optional, boolean) The status to apply to the credentials — *true* for active or *false* for inactive. Defaults to *false* if this parameter is not supplied in the request.

14.17.3. Example Using cURL

The **example** below deactivates a user's S3 credential. Note that since each S3 access key is unique in the system, you do not need to specify the user to whom the key is assigned.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/credentials/status?accessKey=21289bab1738ffdc792a&isActive=false'
```

14.17.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {accessKey}
400	Invalid Access Key

14.18. POST /user/mfa/createDevice

POST /user/mfa/createDevice   Create a virtual MFA device for a user

14.18.1. Syntax

```
POST /user/mfa/createDevice?[userId=string&groupId=string][canonicalUserId=string]
&virtualMFADeviceName=string[&path=string]
```

There is no request payload.

14.18.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

> **Note**
> * You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
> * The group ID for system admins is "0".

virtualMFADeviceName

(Mandatory, string) Name to give to the virtual MFA device. The name can include upper and lowercase alphanumeric characters with no spaces. You can also include any of the following characters: _+=,.@-

> **Note** MFA device names are case-insensitive (and so, you cannot have one device named *MyDevice* and another one named *mydevice* -- these would be treated as the same device).

path

(Optional, string) The path for the virtual MFA device. The path must begin and end with forward slashes. If this parameter is omitted from the request, the path defaults to a forward slash (/) by itself.

14.18.3. Usage Notes

Use this method to create a (representation of a) virtual MFA device under a HyperStore user account root. **When first created, the MFA device is not enabled** (is not activated for the user's account). After the MFA device is created you can activate the device for the user by calling the **POST /user/mfa/enableDevice** method.

14.18.4. Example Using cURL

The **example** below creates a virtual MFA device in the account of the user "PubsUser1" in the "Pubs" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/mfa/createDevice?userId=PubsUser1&groupId=Pubs&
virtualMFADeviceName=myDevice'
```

The response is as below. Note that:

- The *serialNumber* will be need when a **POST /user/mfa/enableDevice** call is submitted to activate the device for the HyperStore account root user
- The *base32StringSeed* and *qrCodePNG* are sensitive data and should be securely destroyed after the virtual MFA device is enabled.

```
{"base32StringSeed":"RVI3R1lLS1ZKMzNCSzJLT0tSNFhLUUdSWFk1WVhQNTRTTTIyUEs2REpRNk
VESlZJVTRLVFdURFhGMjVBTlpJRQ==","enableDate":null,"qrCodePNG":"iVBORw0KGgoAAAAN
SUhEUgAAAPoAAAD6AQAAAACgl2eQAAACdElEQVR4Xu2W0Y3jMAxEpUak/rvYUqRGqJtHZgE7OCzu4zL
7Y8FwEvsFGJDDkdr5eX219ydv6wFqPUCtB6j1ALX+EVit9b1G62eNqR9jbh5Zga1rx+irr7n5EZO7F1
ho7MW1HqmSn2Zgq0j9HMlb80T/FYCOqVStoxbRboAKHZVodb5sfudlBNK09/UXV38WYEWXa0MiA/r7q
Q9YUxXSi9FYatmYwSg7AflUZqVHUvnyjP5lBU6aFE62kXE1u+qWF1B4qS7ySJs5QDnAtWwAHskbzJBE
vHNtlgGgMwpwEiymOrVQSe+cgEySLqVP4CD6gxPI5CpMnZLUReEuhTIAbKSSRZr1XbtqDO5G4GRRot7
Kv2RISjQCG5N0sJ2dG0OmHfzJB0iXYmundzPKqkhWILtEenHJObWnJWQDCAtiTBNzeIlZhFoBdCFzLp
VLOyzluk/354GgOpoV9jJeKEfz0wlkdmSB2E3UNT7HpVAGgEeZIK+KZZjUsgGkd47OwShNGcL0TiuAW
RQhKlb6RN915LsWygBIJDnGU+5Tx4v9OmLYAG1lQxGOriBEZpAn2TAbUMTIQ96mZSFmTCuQgjBsfgn03
jd3AyBhmaPsbnm+mOrY1bQOINC0kAbbZR79wQugDq9w1M2bpN6n+/MAHmVmOGIEQ0uKmAEVStXJI1Yr5
4p4q+SngYSkTBsrzs0ppoFOQJJUKQrUsS9Jhn2twMYxgwjhsDNyd7sFqQMgzDU4pCm7avbqJtIDcOBVx
Qa2UYr2W7NcgEJU2rJRRFinfE4gmdW49Brj1OgaAUyLUMKUZmmOJ7qdwA/rAWo9QK0HqPUAtf4D8Ac0m
4WgtkTYDwAAAABJRU5ErkJggg==",
"serialNumber":"arn:aws:iam::9886099a0803541bc4766213e6ad716e:mfa/myDevice",
"user":null}
```

14.18.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	User is inactive or is not found
400	Invalid parameter or missing required parameter
404	Specified user not found
409	Virtual MFA device already exists

14.19. POST /user/mfa/deactivateDevice

POST /user/mfa/deactivateDevice   Deactivate a user's MFA device

14.19.1. Syntax

```
POST /user/mfa/deactivateDevice?[userId=string&groupId=string]
[canonicalUserId=string]&serialNumber=string
```

There is no request payload.

14.19.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

serialNumber

Serial number of the MFA device to deactivate. This takes the form of an ARN. Here is an example of the MFA serial number format used for HyperStore:

```
arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device
```

14.19.3. Usage Notes

Use this method to deactivate an MFA device that is currently assigned to a HyperStore account root user. If no MFA device is active for the user, the user will no longer be required to provide an MFA code when logging into the CMC.

14.19.4. Example Using cURL

The **example** below deactivates the MFA device that's assigned to user "PubsUser1" in the "Pubs" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/mfa/deactivateDevice?userId=PubsUser1&groupId=Pubs&
serialNumber=arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device'
```

14.19.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	User is inactive or is not found
400	Invalid parameter or missing required parameter
404	Specified MFA Device does not exist or does not belong to specified user

14.20. POST /user/mfa/enableDevice

POST /user/mfa/enableDevice   Activate an MFA device for a user

14.20.1. Syntax

```
POST /user/mfa/enableDevice?[userId=string&groupId=string]
[canonicalUserId=string]&serialNumber=string&
authenticationCode1=string&authenticationCode2=string
```

There is no request payload.

14.20.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

serialNumber

Serial number of the MFA device to activate for the user. This takes the form of an ARN. Here is an example of the MFA serial number format used for HyperStore:

```
arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device
```

The serial number of the device is part of the response to the **POST /user/mfa/createDevice** call that you used to create the device.

authenticationCode1

First MFA authentication code submitted by the user during the activation process.

authenticationCode2

Second successive MFA authentication code submitted by the user during the activation process.

14.20.3. Usage Notes

After using the Admin API **POST /user/mfa/createDevice** method to create a virtual MFA device for a Hyper-Store account root user, use the *POST /user/mfa/enableDevice* method to activate the device for that account root user. After the device has been activated for the user, the user will be required to enter an MFA code when logging into the CMC (after entering their user name and password).

14.20.4. Example Using cURL

The **example** below enables an MFA device for HyperStore account root user "PubsUser1" in the "Pubs" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/mfa/enableDevice?userId=PubsUser1&groupId=Pubs&
serialNumber=arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device&
authenticationCode1=817156&authenticationCode2=379924'
```

14.20.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters
403	Invalid MFA authentication code
404	User is not found or MFA device is not found
409	Exceeding max MFA devices or this MFA device is already enabled on another user

14.21. POST /user/mfa/resyncDevice

POST /user/mfa/resyncDevice   Resync a user's MFA device

14.21.1. Syntax

```
POST /user/mfa/resyncDevice?[userId=string&groupId=string]
[canonicalUserId=string]&serialNumber=string&
authenticationCode1=string&authenticationCode2=string
```

There is no request payload.

14.21.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

serialNumber

Serial number of the MFA device to resync for the user. This takes the form of an ARN. Here is an example of the MFA serial number format used for HyperStore:

```
arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device
```

authenticationCode1

First MFA authentication code submitted by the user during the resync process.

authenticationCode2

Second successive MFA authentication code submitted by the user during the resync process.

14.21.3. Usage Notes

Use this method to resync an MFA device that's assigned to a HyperStore account root user.

14.21.4. Example Using cURL

The **example** below resyncs the MFA device that's assigned to user "PubsUser1" in the "Pubs" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/mfa/resyncDevice?userId=PubsUser1&groupId=Pubs&
serialNumber=arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device&
authenticationCode1=921317&authenticationCode2=565602'
```

14.21.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters
403	Invalid MFA authentication code
404	User is not found or MFA device is not found

14.22. POST /user/mfa/verify

POST /user/mfa/verify   Verify a user's supplied MFA code

14.22.1. Syntax

```
POST /user/mfa/verify?[userId=string&groupId=string][canonicalUserId=string]&serialNumber=string&
authenticationCode=string
```

There is no request payload.

14.22.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

Note
* You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
* The group ID for system admins is "0".

serialNumber

Serial number of the MFA device that is assigned to the user. This takes the form of an ARN. Here is an example of the MFA serial number format used for HyperStore:

```
arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device
```

authenticationCode

MFA authentication code submitted by the user.

14.22.3. Usage Notes

Use this method to verify an MFA code supplied by a HyperStore account root user. The CMC uses this method when a user for whom MFA is enabled logs into the console.

14.22.4. Example Using cURL

The **example** below verifies a MFA code that's supplied by user "PubsUser1" in the "Pubs" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/mfa/verify?userId=PubsUser1&groupId=Pubs&
serialNumber=arn:aws:iam::79da61faead93bdb326419974418c93e:mfa/root-account-mfa-device&
authenticationCode=632615'
```

14.22.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters
403	Invalid MFA authentication code
404	User is not found or MFA device is not found

14.23. POST /user/password

POST /user/password Create or change a user's password

14.23.1. Syntax

```
POST /user/password?userId=string&groupId=string&password=string[&oldpassword=string]
```

There is no request payload.

14.23.2. Parameter Descriptions

userId

(Mandatory, string) User identifier, unique within the group. This is the user ID that was supplied by the user (or by whoever created the user) at the time of user account creation.

groupId

(Mandatory, string) Unique identifier of the group to which the user belongs.

Note The group ID for system admins is "0".

password

(Mandatory, string) The user’s new CMC password.

Passwords must meet the following conditions by default:

- Minimum of nine characters, maximum of 64 characters
- Must contain:
  - At least one lower case letter
  - At least one upper case letter
  - At least one number
  - At least one special character such as !, @, #, $, %, ^, etc.

Note You can optionally configure HyperStore to require a higher minimum password length. You can also optionally configure additional password restrictions such as a password expiration period, a restriction against a user's new password being too similar to their previous password, a restriction on password reuse, and a restriction against too-frequent password changes. In the ***common* configuration setting group**, these restrictions are configurable by the *common.user.password.** settings.

oldpassword

(Optional, string) The user's existing password (which will be replaced by the new password that's specified by the *password* parameter). If the *oldpassword* is supplied with the request, the system will verify that it correctly matches the user's existing password in the system. If it does not match, the *POST*

/user/password request fails with a 400 Bad Request error.

> **Note** You must supply the *oldpassword* parameter if you want the system to apply the configurable check against the new password being too similar to the existing password. This check is controlled by the configuration setting *common.user.password.dupCharRatioLimit* and is disabled by default. For the check to work you must first configure the setting, and then supply the existing password as well as the new password when making *POST /user/password* API calls.

14.23.3. Usage Notes

Use this method to create or update a user's CMC login password.

If you are updating an existing password for a user, use the "password" parameter to specify the **new** password, not the existing password.

> **Note** For "SystemAdmin" users this password also serves as the user's HyperStore Shell password. For more information see the "HyperStore Shell (HSH)" section of the ***Cloudian HyperStore Administrator's Guide***.

14.23.4. Example Using cURL

The **example** below posts a CMC password for the user "John" in the "QA" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/password?userId=John&groupId=QA&password=P1a2s3s4!'
```

14.23.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
204	User does not exist
400	Missing Required parameters : {userId, groupId, password}
400	Exceeded max password length
400	Password strength is too weak.
400	Invalid oldpassword

14.24. POST /user/ratingPlanId

POST /user/ratingPlanId   Assign a rating plan to a user

14.24.1. Syntax

```
POST /user/ratingPlanId?userId=string&groupId=string&ratingPlanId=string[&region=string]
```

There is no request payload.

14.24.2. Parameter Descriptions

userId

(Mandatory, string) User identifier, unique within the group. This is the user ID that was supplied by the user (or by whoever created the user) at the time of user account creation.

groupId

(Mandatory, string) Unique identifier of the group to which the user belongs.

ratingPlanId

(Mandatory, string) Unique identifier of the rating plan to assign to the user, for billing purposes.

region

(Optional, string) If your service deployment has multiple service regions, rating plan assignment is on a per-region basis. With the *POST /user/ratingPlanId* method, use the "region" parameter to indicate the service region in which to apply the specified rating plan. For example, if *userId=Cody&groupId=Engineering&ratingPlanId=Gold&region=East*, then the Gold rating plan will be applied to user Cody's service activity in the East region.

If this parameter is omitted the default service region is assumed.

14.24.3. Example Using cURL

The **example** below assigns the "Gold" rating plan to user "John" in the "QA" group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/ratingPlanId?userId=John&groupId=QA&ratingPlanId=Gold'
```

14.24.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId, ratingPlanId}
400	Region {region} is not valid

14.25. POST /user/unlock

POST /user/unlock   Unlock a locked-out user

14.25.1. Syntax

```
POST /user/unlock?[userId=string&groupId=string][canonicalUserId=string]
```

There is no request payload.

14.25.2. Parameter Descriptions

userId, groupId, canonicalUserId

(Mandatory [to use one approach or the other], string) Specify the user **either** by supplying the *userId* (the ID provided by the user at the time of user account creation) in combination with the *groupId* of the group to which the user belongs, **or** by supplying the *canonicalUserId* (the unique ID generated for the user by the system).

If you use the "userId" and "groupId" parameters, do not use the "canonicalUserId" parameter. If you use the "canonicalUserId" parameter, do not use the "userId" or "groupId" parameters.

> **Note**
> * You can retrieve a user's canonical user ID by using the Admin API methods *GET /user* or *GET /user/list*.
> * The group ID for system admins is "0".

14.25.3. Usage Notes

This method releases the lock on a user who is locked out from logging into the CMC. A user may become temporarily locked out of the CMC if you have the password lock-out feature enabled in your system, and within a defined time interval the user makes too many login attempts using an incorrect password. The password-lock feature is configurable by *common.user.password.lock.enabled* and the other *common.user.password.lock.** settings.

For a locked-out user, once the user stops attempting to log in with an incorrect password the lock-out releases automatically after a configurable time interval (30 minutes by default). Alternatively, the *POST /user/unlock* method lets you release the lock immediately.

14.25.4. Example Using cURL

The **example** below unlocks the user *John* in the *QA* group.

```
curl -X POST -k -u sysadmin:password \
'https://localhost:19443/user/unlock?userId=John&groupId=QA'
```

The second example uses the *GET /user/islocked* method to confirm that John is now unlocked.

```
curl -X GET -k -u sysadmin:password \
'https://localhost:19443/user/islocked?userId=John&groupId=QA'

false
```

14.25.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing or conflicting parameters : {userId, groupId,canonicalUserId}
400	User/Group does not exist

14.26. PUT /user

PUT /user Create a new user

14.26.1. Syntax

```
PUT /user
```

The required request payload is a JSON-formatted *UserInfo* object. See example below.

Note This method does not create a CMC login password for the new user. After creating a new user with the *PUT /user* method, use the **POST /user/password** method to a create a CMC password for the user.

14.26.2. Example Using cURL

The **example** below creates a new user "John" in the "QA" group. In this example the JSON-formatted *UserInfo* object is specified in a text file named *user_John.txt* which is then referenced as the data input to the cURL command.

```
curl -X PUT -H "Content-Type: application/json" -k -u sysadmin:password \
-d @user_John.txt https://localhost:19443/user | python -mjson.tool
```

The response payload is a JSON-formatted *UserInfo* object.

Immediately below is the input file for this example (the *UserInfo* object submitted in the request). Below that is the response payload for this example (the *UserInfo* object returned in the response). The difference between the two is that the *UserInfo* object submitted in the request does not include a "canonicalUserId" attribute, whereas the *UserInfo* object returned in the response body does have this attribute. The system has generated a canonical user ID for the new user.

Request payload:

```
{
  "active": "true",
  "address1": "",
  "address2": "",
  "city": "",
  "country": "",
  "emailAddr": "",
  "fileEndpoints": "",
  "fullName": "John Thompson",
  "groupId": "QA",
  "ldapEnabled": false,
  "phone": "",
  "state": "",
  "userId": "John",
  "userType": "User",
  "website": "",
  "zip": ""
}
```

Response payload:

```
{
  "active": "true",
  "address1": "",
  "address2": "",
  "canonicalUserId": "bd0796cd9746ef9cc4ef656ddaacfac4",
  "city": "",
  "country": "",
  "emailAddr": "",
  "fileEndpoints": "",
  "fullName": "John Thompson",
  "groupId": "QA",
  "ldapEnabled": false,
  "phone": "",
  "state": "",
  "userId": "John",
  "userType": "User",
  "website": "",
  "zip": ""
}
```

14.26.3. Request and Response Element Descriptions

active

(Optional, string) Whether the user is active — "true" or "false".

- "true" indicates an active user.
- "false" indicates that the user is not an active user. Non-active users are users who are in one of these statuses:
  - Inactive — These users have had their status set to inactive via the *POST /user* method (with UserInfo object attribute "active" set to false). These users' stored S3 objects still exist, but their S3 access credentials have been deactivated and they cannot log into the CMC.
  - Deleted — These users have been deleted from the S3 service via the *DELETE /user* method. Their S3 access credentials have been deleted, and their S3 buckets and objects have been deleted and are unrecoverable. (These users cannot be retrieved through the *GET /user* method — they can only be retrieved through the *GET /user/list* method.)
  - Deleting — These users are in the process of being deleted from the S3 service via the *DELETE /user* method. The deletion process for these users has not yet completed. (These users cannot be retrieved through the *GET /user* method — they can only be retrieved through the *GET /user/list* method.)

If the "active" attribute is unspecified for a *PUT /user*operation, it defaults to "true". If the "active" attribute is unspecified for a *POST /user*operation, the user will retain her existing status.

Example:

```
"active": "true"
```

Note The only way to retrieve a list of users who have been deleted or are in the process of being deleted is to specify "deleted" or "deleting" for the *userStatus* request parameter with a *GET /user/list* request. In the response body, the "active" attribute for such users will say "false".

address1

(Optional, string) User’s street address line 1. Example:

```
"address1": "123 Main St."
```

Note In the "address1", "address2", "zip", "email", "website", and "phone" fields, the Admin Service prohibits the use of any of these characters:

```
` | ; & > <
```

address2

(Optional, string) User’s street address line 2. Example:

```
"address2": ""
```

canonicalUserId

(Optional [in POST], string) Canonical user ID, globally unique within the HyperStore system. This is automatically generated by the system when a new user is created. The canonicalUserId is unique per user across the system and over time, even in the case where a user with a specific <groupId>|<userId> combination is deleted from the system and then a new user is subsequently added with the same <groupId>|<userId> combination. The new user will be assigned a different canonicalUserId than the deleted user. This allows past and present users to be uniquely identified for purposes such as usage reporting.

The client must not supply a canonicalUserId in a *PUT /user* request and does not need to supply one in a *POST /user* request.

Example:

```
"canonicalUserId": "bd0796cd9746ef9cc4ef656ddaacfac4"
```

city

(Optional, string) User’s city. Example:

```
"city": "Portsmouth"
```

country

(Optional, string) User’s country. Example:

```
"country": "US"
```

emailAddr

(Optional, string) User's email address. Example:

```
"emailAddr": "me@mail.com"
```

fileEndpoints

(Mandatory, string) Name(s) of the File Service pools that the user is allowed access to, for publishing file shares. Use comma separation if multiple pools. If not applicable to this user specify `""` as in the example below:

```
"fileEndpoints": ""
```

fullName

(Optional, string) User's full name. By default the maximum length is 64 characters. This maximum is configurable by the setting *common.user.name.maxLength*.

Example:

```
"fullName": "John Thompson"
```

groupId

(Mandatory, string) Group ID of the group to which the user belongs.

Note For all SystemAdmin type users the groupId is "0".

Example:

```
"groupId": "QA"
```

Second example (for a system administrator):

```
"groupId": "0"
```

ldapEnabled

(Optional, boolean) Whether the CMC authenticates the user by checking an LDAP system, *true* or *false*. Defaults to *false*. If the user is enabled for LDAP, when authenticating the user the CMC uses the LDAP connection information configured for the user's group. For more information see LDAP Integration. If the user is not LDAP enabled, the CMC authenticates the user by requiring a password that the CMC maintains.

Example:

```
"ldapEnabled": false
```

phone

(Optional, string) User's phone number. Example:

```
"phone": "890-123-4567"
```

state

(Optional, string) User's state. Example:

```
"state": "NH"
```

userId

(Mandatory, string) User ID.

- Only letters, numbers, dashes, underscores, dots, and ampersands (@) are allowed.
- By default the maximum length is 64 characters. This maximum is configurable by the setting *common.user.name.maxLength*.
- The following IDs are reserved for system use and are not available to individual users: "anonymous", "public", "null", "none", "admin", "0".

Example:

```
"userId": "John"
```

Note The character rules for the user IDs of system administrators are more strict:
* Maximum length = 26 characters (this is not configurable)
* Only lower case letters, numbers, and underscores are allowed
* Must start with a letter
* Cannot end with an underscore

userType

(Mandatory, string) User type. One of {"User","GroupAdmin","SystemAdmin"}. Example:

```
"userType": "User"
```

Note
* If you set the userType to "SystemAdmin", set the *groupId* to "0".
* When you create a "SystemAdmin" user the system automatically creates a corresponding HyperStore Shell user. For more information see the "HyperStore Shell (HSH)" section of the ***Cloudian HyperStore Administrator's Guide***.

website

(Optional, string) User’s website URL. Example:

```
"website": "www.me.com"
```

zip

(Optional, string) User’s postal zip code. Example:

```
"zip": "12345"
```

14.26.4. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing required attributes : {userId, groupId, userType}
400	Invalid JSON object
400	Problem accessing /user
400	User Id is not allowed : {userId}

Status Code	Description
400	Invalid User Name
409	Problem accessing /user

14.27. PUT /user/credentials

PUT /user/credentials Create a new S3 credential for a user

14.27.1. Syntax

```
PUT /user/credentials?userId=string&groupId=string
```

There is no request payload.

14.27.2. Parameter Descriptions

userId

(Mandatory, string) User identifier, unique within the group. This is the user ID that was supplied by the user (or by whoever created the user) at the time of user account creation.

groupId

(Optional, string) Unique identifier of the group to which the user belongs.

Note The group ID for system admins is "0".

14.27.3. Example Using cURL

The **example** below creates a new S3 credential for user "John" in the "QA" group.

```
curl -X PUT -k -u sysadmin:password \
'https://localhost:19443/user/credentials?userId=John&groupId=QA' \
| python -mjson.tool
```

The response payload is a JSON-formatted *SecurityInfo* object, which in this example is as follows.

```
{
  "accessKey": "28d945de2a2623fc9483",
  "active": true,
  "createDate": 1502285593100,
  "expireDate": null,
  "secretKey": "j2OrPGHF69hp3YsZHRHOCWdAQDabppsBtD7kttr9"
}
```

14.27.4. Response Element Descriptions

accessKey

(String) User’s access key (public key) for the HyperStore S3 service. Example:

```
"accessKey": "009c156c79e64e0e4928
```

active

(Boolean) Whether the credential is active, *true* or *false*. An inactive credential cannot be used to access the HyperStore S3 service. Example:

```
"active": true
```

createDate

(String) Creation timestamp for the credential in UTC milliseconds. Example:

```
"createDate": 1502279336024
```

expireDate

(String) Credential expiration date. In the current version of HyperStore this attribute's value is always null. Example:

```
"expireDate": null
```

secretKey

(String) User’s secret key (private key) for the HyperStore S3 service. Example:

```
"secretKey": "wVHk2nA0M03RWSMIrFHFAtuhow6S1DKN0gWjPhDG"
```

14.27.5. Response Codes

This method will return one of the **"Common Response Status Codes"** (page 13) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId}
400	User does not exist
403	Reached maximum number of credentials allowed