

Cloudian HyperStore Administration Guide

Version 7.2



This page left intentionally blank

Confidentiality Notice

The information contained in this document is confidential to, and is the intellectual property of, Cloudian, Inc. Neither this document nor any information contained herein may be (1) used in any manner other than to support the use of Cloudian software in accordance with a valid license obtained from Cloudian, Inc, or (2) reproduced, disclosed or otherwise provided to others under any circumstances, without the prior written permission of Cloudian, Inc. Without limiting the foregoing, use of any information contained in this document in connection with the development of a product or service that may be competitive with Cloudian software is strictly prohibited. Any permitted reproduction of this document or any portion hereof must be accompanied by this legend.

This page left intentionally blank

Contents

Chapter 1. Introduction	1
1.1. What's New in HyperStore 7.2	1
1.1.1. S3 API New Features and Enhancements	1
1.1.2. Admin API New Features and Enhancements	1
1.1.3. System Operations New Features and Enhancements	2
1.2. HyperStore Documentation	3
1.3. HyperStore Overview	5
1.4. Licensing and Auditing	6
1.4.1. License Expiration	6
1.4.2. Licensed Maximum HyperStore Storage Usage	7
1.4.3. Licensed Maximum Tiered Storage Usage	9
1.4.4. WORM License	10
1.4.5. License Updating	10
1.4.6. Auditing	10
1.5. HyperStore Services	10
1.5.1. S3 Service	12
1.5.2. HyperStore Service and the HSFS	12
1.5.3. Cassandra Service	15
1.5.4. Redis Credentials and Redis QoS Services	16
1.5.5. Redis Monitor Service	17
1.5.6. Admin Service	17
1.5.7. IAM Service	18
1.5.8. Cloudian Management Console (CMC) Service	18
1.5.9. Supporting Services	18
1.6. System Diagrams	19
1.6.1. System Levels	19
1.6.2. Service Interconnections	20
1.6.3. Services Distribution -- 3 Nodes, Single DC	21
1.6.4. Services Distribution -- Multi-DC, Single Region	22
1.6.5. Services Distribution -- Multi-Region	23
1.6.6. Specialized Services Availability	24

1.6.7. S3 PUT Processing Flow	25
1.6.8. S3 GET Processing Flow	27
1.6.9. Data Freshness for Replicated Object Reads	28
1.6.10. Dynamic Consistency Levels	29
1.6.11. How vNodes Work	32
Chapter 2. Getting Started with a New HyperStore System	41
Chapter 3. Upgrading Your HyperStore Software Version	45
3.1. Preparing to Upgrade Your System	45
3.1.1. Additional Upgrade Preparation If Your System Currently Has Failed Disks	48
3.1.2. Additional Upgrade Preparation If You Are Using Elasticsearch	48
3.2. Upgrading Your System	48
3.2.1. Upgrade Failure and Roll-Back	50
3.3. Verifying Your System Upgrade	50
Chapter 4. Working with HyperStore Major Features	53
4.1. Auto-Tiering	53
4.1.1. Auto-Tiering Feature Overview	53
4.1.2. Setting Up Auto-Tiering	56
4.1.3. Accessing Auto-Tiered Objects	61
4.2. Billing	63
4.2.1. Billing Feature Overview	63
4.2.2. Creating Rating Plans for Billing	63
4.2.3. Assigning Rating Plans to Users	65
4.2.4. Creating a "Whitelist" for Free Traffic	66
4.2.5. Generating Billing Data for a User or Group	67
4.3. Cluster Resizing	68
4.3.1. Cluster Resizing Feature Overview	68
4.4. Data Centers	71
4.4.1. Data Centers Feature Overview	71
4.4.2. Multi-DC Setup	74
4.5. Data Repair	75
4.5.1. Data Repair Feature Overview	75
4.5.2. Configuring Automatic Data Repair	78
4.5.3. Checking Data Repair Status	79
4.5.4. Disabling or Stopping Data Repairs	79

4.6. Disk Failure Handling	82
4.6.1. Disk Failure Handling Feature Overview	82
4.6.2. Configuring Disk Failure Handling	83
4.6.3. Checking Disk Status	84
4.6.4. Disk Error Alerts	84
4.6.5. Bringing a Disk Online	84
4.7. Disk Usage Balancing	85
4.7.1. Disk Usage Balancing Feature Overview	85
4.7.2. Configuring Disk Usage Balancing	87
4.7.3. Checking Disk Usage Status	88
4.7.4. Triggering an Immediate Balance Check	88
4.8. Group and User Provisioning	89
4.8.1. Group and User Provisioning Feature Overview	89
4.8.2. Provisioning Groups	89
4.8.3. Provisioning Users	90
4.9. Identity and Access Management (IAM)	91
4.9.1. Identity and Access Management (IAM) Feature Overview	91
4.9.2. Using the HyperStore IAM Service	93
4.9.3. IAM Extensions for Role-Based Access to HyperStore Admin Functions	95
4.10. LDAP Integration	101
4.10.1. LDAP Integration Feature Overview	101
4.11. Object Metadata and Tagging	103
4.11.1. Object Metadata and Tagging Feature Overview	103
4.11.2. Creating Object Metadata and Tags	105
4.11.3. Retrieving Object Metadata and Tags	106
4.11.4. Elasticsearch Integration for Object Metadata	106
4.12. Quality of Service	110
4.12.1. Quality of Service (QoS) Feature Overview	110
4.12.2. Enabling QoS Enforcement	112
4.12.3. Setting QoS Limits for Groups	112
4.12.4. Setting QoS Limits for Users	113
4.13. S3 Interface	114
4.13.1. S3 Storage Interface Feature Overview	114
4.13.2. Using the CMC as Your S3 Client	115

4.13.3. Using Third Party S3 Applications	116
4.13.4. Developing Custom S3 Applications for HyperStore	116
4.14. Server-Side Encryption	117
4.14.1. Server-Side Encryption Feature Overview	117
4.14.2. Using Regular Server-Side Encryption (SSE)	119
4.14.3. Using Server-Side Encryption with Customer-Provided Keys (SSE-C)	121
4.14.4. Using Server-Side Encryption with AWS KMS	123
4.14.5. Using Server-Side Encryption with Gemalto KeySecure KMS	125
4.14.6. Enabling AES-256	127
4.15. Service Regions	128
4.15.1. Service Regions Feature Overview	128
4.15.2. Setting Up a Multi-Region System	129
4.15.3. Creating a Bucket in a Specific Region	131
4.15.4. Cross-Region Replication	132
4.15.5. Using the CMC or Admin API in a Multi-Region System	136
4.16. Smart Support	137
4.16.1. Smart Support and Diagnostics Feature Overview	137
4.16.2. Configuring Smart Support and Node Diagnostics	138
4.16.3. Executing Node Diagnostics Collection	140
4.17. Storage Policies	140
4.17.1. Storage Policies Feature Overview	140
4.17.2. Consistency Levels	143
4.17.3. Storage of Object Metadata	144
4.17.4. Storage of System Metadata	145
4.17.5. Creating and Managing Storage Policies	147
4.17.6. Assigning a Storage Policy to a Bucket	147
4.17.7. Finding an Object's Replicas or EC Fragments	148
4.17.8. Storage Policy Resilience to Downed Nodes	148
4.18. Usage Reporting	153
4.18.1. Usage Reporting Feature Overview	153
4.18.2. Enabling Non-Default Usage Reporting Features	156
4.18.3. Validating Usage Data for Storage Levels	157
4.18.4. Setting Usage Data Retention Periods	157
4.18.5. Protecting Usage Data Through Replication	158

4.18.6. Generating a Usage Report	158
4.19. WORM (Bucket Lock)	159
4.19.1. WORM (Bucket Lock) Feature Overview	159
4.19.2. Setting Up a Bucket Lock	163
4.19.3. Setting Up a Privileged Delete User for a Bucket	164
Chapter 5. Cloudian Management Console (CMC)	170
5.1. Dashboard	171
5.1.1. Dashboard	171
5.2. Analytics	180
5.2.1. Cluster Usage	180
5.2.2. Capacity Explorer	183
5.2.3. Usage By Users & Groups	185
5.2.4. Object Locator	191
5.3. Buckets	192
5.3.1. Add a Bucket	192
5.3.2. Set Bucket Properties	194
5.3.3. Delete a Bucket	214
5.4. Objects	214
5.4.1. Create or Delete a "Folder"	215
5.4.2. Object Naming Restrictions	216
5.4.3. Upload an Object	217
5.4.4. Set Object Properties	219
5.4.5. Search for an Object	224
5.4.6. Download an Object	225
5.4.7. Restore an Auto-Tiered Object	225
5.4.8. Delete an Object	228
5.5. Users & Groups	229
5.5.1. Manage Users	229
5.5.2. Manage Groups	237
5.5.3. Rating Plan	243
5.5.4. Account Activity	247
5.5.5. Whitelist	248
5.5.6. Set Quality of Service (QoS) Controls	250
5.6. IAM	254

5.6.1. Manage IAM User	254
5.6.2. Manage IAM Group	258
5.7. Cluster	264
5.7.1. Data Centers	264
5.7.2. Node Status	268
5.7.3. Node Activity	279
5.7.4. Node Advanced	282
5.7.5. Cluster Information	286
5.7.6. Configuration Settings	293
5.7.7. Storage Policies	307
5.7.8. Repair Status	333
5.7.9. Operation Status	338
5.8. Alerts	339
5.8.1. Alerts	339
5.8.2. Alert Rules	344
5.8.3. How HyperStore Implements Alerts	351
5.9. My Account	352
5.9.1. Profile	352
5.9.2. Security Credentials	353
5.10. Customizing the CMC	355
5.10.1. Showing/Hiding CMC UI Functions	355
5.10.2. Rebranding the CMC UI	356
5.10.3. Implementing Single Sign-On for the CMC	358
Chapter 6. Node and Cluster Operations	365
6.1. Starting and Stopping Services	365
6.1.1. Start or Stop Services on All Nodes in the Cluster	365
6.1.2. Start or Stop Services on One Node	367
6.1.3. Automatic Service Start on Node Reboot	368
6.2. Adding Nodes	369
6.2.1. Special Requirements if an Existing Node is Down	369
6.2.2. Preparing to Add Nodes	369
6.2.3. Adding Nodes	372
6.3. Adding a Data Center	379
6.3.1. Special Requirements if an Existing Node is Down or Unreachable	379

6.3.2. Preparing to Add a Data Center	380
6.3.3. Adding a Data Center	381
4.4. Adding a Region	385
6.4.1. Preparing to Add a Region	386
6.4.2. Adding a Region	387
5.5. Removing a Node	390
6.5.1. Preparing to Remove a Node	391
6.5.2. Removing a Node	396
6.6. Replacing a Node	400
6.7. Restoring a Node That Has Been Offline	400
6.7.1. 6.7.1 Repairing a Node That's Been Down for Longer than the Proactive Repair Limit	402
6.8. Changing a Node's IP Address	402
6.9. Backing Up and Restoring a Cluster	402
6.9.1. Backing Up an Entire Cluster	403
6.9.2. Restoring an Entire Cluster	404
10.10. Change Node Role Assignments	405
6.10.1. Move the Redis Credentials Master or QoS Master Role	405
6.10.2. Move or Add a Redis Credentials Slave or Redis QoS Slave	408
6.10.3. Move the Cassandra Seed Role	410
6.10.4. Reduce or Change the List of CMC Hosts	411
6.10.5. Move the Redis Monitor Primary or Backup Role	412
6.10.6. Move the Cron Job Primary or Backup Role	414
6.10.7. Move the Puppet Master Primary or Backup Role	415
6.10.8. Change Internal NTP Servers or External NTP Servers	418
11.11. Cron Jobs and Automated System Maintenance	419
6.11.1. System cron Jobs	420
6.11.2. Scheduled Auto-Repair	426
6.11.3. Cassandra Data Compaction	426
Chapter 7. Disk Operations	427
7.1. Disabling a HyperStore Data Disk	427
7.1.1. The Impact of Disabling a Disk	427
7.1.2. Disabling a Disk	427
7.2. Enabling a HyperStore Data Disk	428
7.2.1. The Impact of Enabling a Disk	429

7.2.2. Enabling a Disabled Disk	429
7.3. Replacing a HyperStore Data Disk	430
7.3.1. The Impact of Replacing a Disk	431
7.3.2. Replacing a Disk	431
7.4. Replacing a Cassandra Disk	432
7.5. Responding to Data Disks Nearing Capacity	436
7.6. Responding to Cassandra Disks Nearing Capacity	437
7.7. Adding Disks is Not Supported	437
Chapter 8. System Monitoring	439
8.1. Using the CMC to Monitor Your HyperStore System	439
8.2. Additional Monitoring Tools	440
8.2.1. Using the Admin API to Monitor HyperStore	440
8.2.2. Doing an HTTP Health Check	440
8.2.3. Using JMX to Monitor Java-Based HyperStore Services	440
8.2.4. Using Native Linux Utilities for System Resource Monitoring	445
8.2.5. Using nodetool to Monitor Cassandra	445
8.2.6. Using the Redis CLI to Monitor Redis	447
Chapter 9. System Configuration	449
9.1. CMC's Configuration Settings Page	449
9.2. Installer Advanced Configuration Options	449
9.3. Puppet Overview	454
9.4. Pushing Configuration File Edits to the Cluster and Restarting Services	454
9.4.1. Using the Installer to Push Configuration Changes and Restart Services	454
9.4.2. Option for Triggering a Puppet Sync-Up from the Command Line	456
9.4.3. Excluding a Down Node from Installer-Driven Configuration Push	457
9.4.4. Automatic Puppet Sync-Up on an Interval	457
9.5. HyperStore Configuration Files	457
9.5.1. common.csv	458
9.5.2. hyperstore-server.properties.erb	486
9.5.3. mts.properties.erb	496
9.5.4. mts-ui.properties.erb	523
9.5.5. Other Configuration Files	537
9.5.6. Using JMX to Dynamically Change Configuration Settings	539
9.6. Configuration Special Topics	540
9.6.1. Anti-Virus Software	540
9.6.2. NTP Automatic Set-Up	540

9.6.3. Changing S3, Admin, or CMC Listening Ports	541
9.6.4. Changing S3, Admin, or CMC Service Endpoints	542
9.6.5. TLS/SSL for S3 Service (Self-Signed Certificate)	543
9.6.6. TLS/SSL for S3 Service (CA-Verified Certificate)	547
9.6.7. TLS/SSL for the CMC (Self-Signed Certificate)	553
9.6.8. TLS/SSL for the CMC (CA-Verified Certificate)	556
9.6.9. TLS/SSL for IAM Service (Self-Signed Certificate)	561
9.6.10. TLS/SSL for IAM Service (CA-Verified Certificate)	563
9.6.11. Tuning HyperStore Performance Parameters	568
9.6.12. Vanity Domains for S3 Buckets	569
9.6.13. Configuring the HyperStore Firewall	569
Chapter 10. Logging	575
10.1. HyperStore Logs	575
10.1.1. Admin Service Logs	575
10.1.2. Cassandra Logs	576
10.1.3. CMC Log	578
10.1.4. HyperStore Service Logs	579
10.1.5. Monitoring Service Logs	582
10.1.6. Phone Home (Smart Support) Log	583
10.1.7. Redis and Redis Monitor Logs	584
10.1.8. S3 Service Logs	586
10.1.9. HyperStore Firewall Log	592
10.2. Log Configuration Settings	593
10.3. Aggregating Logs to a Central Server	594
10.4. Setting Up Elastic Stack for S3 Request Traffic Analysis	598
10.4.1. Installing Elasticsearch, Kibana, and Logstash	598
10.4.2. Installing Filebeat	601
10.4.3. Configuring Kibana for Custom Metrics Visualizations	602
Chapter 11. Commands	609
11.1. hsstool	609
11.1.1. hsstool cleanup	610
11.1.2. hsstool cleanupec	616
11.1.3. hsstool info	623
11.1.4. hsstool ls	625

11.1.5. hsstool metadata	626
11.1.6. hsstool opctl	629
11.1.7. hsstool opstatus	630
11.1.8. hsstool proactiverepairq	637
11.1.9. hsstool rebalance	642
11.1.10. hsstool repair	647
11.1.11. hsstool repaircassandra	656
11.1.12. hsstool repairec	658
11.1.13. hsstool repairqueue	667
11.1.14. hsstool ring	672
11.1.15. hsstool status	674
11.1.16. hsstool trmap	676
11.1.17. hsstool whereis	680
11.2. Redis Monitor Commands	685
11.2.1. Redis Monitor Commands	686
Chapter 12. Admin API	695
12.1. Introduction	695
12.1.1. HyperStore Admin API Introduction	695
12.1.2. Admin API Methods List	696
12.1.3. Common Request and Response Headers	699
12.1.4. Common Response Status Codes	700
12.1.5. cURL Examples	700
12.1.6. HTTP and HTTPS for Admin API Access	701
12.1.7. HTTP/S Basic Authentication for Admin API Access	703
12.1.8. Admin API Logging	704
12.2. billing	705
12.2.1. GET /billing	705
12.2.2. POST /billing	708
12.2.3. billing Query Parameters	709
12.2.4. billing Objects	710
12.3. bppolicy	713
12.3.1. GET /bppolicy/bucketsperpolicy	713
12.3.2. GET /bppolicy/listpolicy	714
12.3.3. bppolicy Query Parameters	715

12.3.4. bppolicy Objects	716
12.4. bucketops	716
12.4.1. GET /bucketops/id	716
12.4.2. GET /bucketops/gettags	717
12.4.3. POST /bucketops/purge	718
12.4.4. bucketops Query Parameters	719
12.4.5. bucketops Objects	720
12.5. group	721
12.5.1. DELETE /group	721
12.5.2. GET /group	722
12.5.3. GET /group/list	724
12.5.4. GET /group/ratingPlanId	727
12.5.5. POST /group	727
12.5.6. POST /group/ratingPlanId	728
12.5.7. PUT /group	729
12.5.8. group Query Parameters	730
12.5.9. group Objects	731
12.6. monitor	735
12.6.1. DELETE /monitor/notificationrule	735
12.6.2. GET /monitor/events	736
12.6.3. GET /monitor/nodelist	738
12.6.4. GET /monitor/host	740
12.6.5. GET /monitor	745
12.6.6. GET /monitor/history	748
12.6.7. GET /monitor/notificationrules	749
12.6.8. POST /monitor/acknowledgeevents	750
12.6.9. POST /monitor/notificationruleenable	751
12.6.10. POST /monitor/notificationrule	752
12.6.11. PUT /monitor/notificationrule	753
12.6.12. monitor Query Parameters	754
12.6.13. monitor Objects	755
12.7. permissions	774
12.7.1. GET /permissions/publicUrl	774

12.7.2. POST /permissions/publicUrl	775
12.7.3. permissions Query Parameters	776
12.7.4. permissions Objects	776
12.8. qos	778
12.8.1. DELETE /qos/limits	778
12.8.2. GET /qos/limits	779
12.8.3. POST /qos/limits	781
12.8.4. qos Query Parameters	782
12.8.5. qos Objects	783
12.9. ratingPlan	787
12.9.1. DELETE /ratingPlan	787
12.9.2. GET /ratingPlan	788
12.9.3. GET /ratingPlan/list	790
12.9.4. POST /ratingPlan	791
12.9.5. PUT /ratingPlan	792
12.9.6. ratingPlan Query Parameters	793
12.9.7. ratingPlan Objects	794
12.10. system	796
12.10.1. GET /system/audit	796
12.10.2. GET /system/bucketcount	797
12.10.3. GET /system/bucketlist	798
12.10.4. GET /system/bytecount	799
12.10.5. GET /system/bytestiered	800
12.10.6. GET /system/groupbytecount	801
12.10.7. GET /system/groupobjectcount	802
12.10.8. GET /system/license	803
12.10.9. GET system/objectcount	805
12.10.10. GET /system/version	806
12.10.11. POST /system/processProtectionPolicy	807
12.10.12. POST /system/repairusercount	808
12.10.13. system Query Parameters	808
12.10.14. system Objects	809
12.11. tiering	814

12.11.1. DELETE /tiering/credentials	814
12.11.2. DELETE /tiering/azure/credentials	815
12.11.3. DELETE /tiering/spectra/credentials	815
12.11.4. GET /tiering/credentials	816
12.11.5. GET /tiering/credentials/src	816
12.11.6. GET /tiering/azure/credentials	817
12.11.7. GET /tiering/spectra/credentials	818
12.11.8. POST /tiering/credentials	819
12.11.9. POST /tiering/azure/credentials	819
12.11.10. POST /tiering/spectra/credentials	820
12.11.11. tiering Query Parameters	821
12.12. usage	822
12.12.1. DELETE /usage	822
12.12.2. GET /usage	823
12.12.3. POST /usage/bucket	827
12.12.4. POST /usage/repair	828
12.12.5. POST /usage/repair/bucket	829
12.12.6. POST /usage/repair/dirtyusers	830
12.12.7. POST /usage/repair/user	831
12.12.8. POST /usage/rollup	832
12.12.9. POST /usage/storage	833
12.12.10. POST /usage/storageall	833
12.12.11. usage Query Parameters	834
12.12.12. usage Objects	839
12.13. user	849
12.13.1. DELETE /user	849
12.13.2. DELETE /user/credentials	850
12.13.3. DELETE /user/deleted	851
12.13.4. GET /user	852
12.13.5. GET /user/credentials	854
12.13.6. GET /user/credentials/list	856
12.13.7. GET /user/credentials/list/active	858
12.13.8. GET /user/list	860

12.13.9. GET /user/password/verify	863
12.13.10. GET /user/ratingPlan	864
12.13.11. GET /user/ratingPlanId	866
12.13.12. POST /user	867
12.13.13. POST /user/credentials	868
12.13.14. POST /user/credentials/status	868
12.13.15. POST /user/password	869
12.13.16. POST /user/ratingPlanId	870
12.13.17. PUT /user	870
12.13.18. PUT /user/credentials	872
12.13.19. user Query Parameters	873
12.13.20. user Objects	875
12.14. whitelist	879
12.14.1. GET /whitelist	879
12.14.2. POST /whitelist	880
12.14.3. POST /whitelist/list	881
12.14.4. whitelist Query Parameters	882
12.14.5. whitelist Objects	882
Chapter 13. S3 API	885
13.1. Introduction	885
13.1.1. HyperStore Support for the Amazon S3 API	885
13.1.2. Error Responses	885
13.1.3. Authenticating Requests (AWS Signature Version 4)	887
13.1.4. Common Request Headers	888
13.1.5. Common Response Headers	888
13.1.6. Access Control List (ACL)	888
13.1.7. HyperStore Extensions to the S3 API	890
13.2. Operations on the Service	892
13.2.1. GET Service	892
13.3. Operations On Buckets	893
13.3.1. DELETE Bucket	893
13.3.2. DELETE Bucket cors	893
13.3.3. DELETE Bucket encryption	893
13.3.4. DELETE Bucket lifecycle	894

13.3.5. DELETE Bucket lock-policy	894
13.3.6. DELETE Bucket policy	895
13.3.7. DELETE Bucket replication	895
13.3.8. DELETE Bucket tagging	895
13.3.9. DELETE Bucket website	896
13.3.10. GET Bucket (List Objects) Version 1	896
13.3.11. GET Bucket (List Objects) Version 2	897
13.3.12. GET Bucket acl	899
13.3.13. GET Bucket cors	899
13.3.14. GET Bucket encryption	899
13.3.15. GET Bucket lifecycle	900
13.3.16. GET Bucket location	901
13.3.17. GET Bucket lock-policy	901
13.3.18. GET Bucket logging	903
13.3.19. GET Bucket Object versions	904
13.3.20. GET Bucket policy	905
13.3.21. GET Bucket replication	905
13.3.22. GET Bucket tagging	906
13.3.23. GET Bucket versioning	906
13.3.24. GET Bucket website	907
13.3.25. HEAD Bucket	907
13.3.26. List Multipart Uploads	907
13.3.27. POST Bucket lock-policy	908
13.3.28. POST Bucket lockId	911
13.3.29. PUT Bucket	913
13.3.30. PUT Bucket acl	914
13.3.31. PUT Bucket cors	915
13.3.32. PUT Bucket encryption	916
13.3.33. PUT Bucket lifecycle	916
13.3.34. PUT Bucket logging	922
13.3.35. PUT Bucket policy	923
13.3.36. PUT Bucket replication	927
13.3.37. PUT Bucket tagging	929

13.3.38. PUT Bucket versioning	930
13.3.39. PUT Bucket website	930
13.4. Operations On Objects	930
13.4.1. Delete Multiple Objects	931
13.4.2. DELETE Object	932
13.4.3. DELETE Object tagging	932
13.4.4. GET Object	932
13.4.5. GET Object acl	934
13.4.6. GET Object tagging	934
13.4.7. GET Object torrent	934
13.4.8. HEAD Object	935
13.4.9. OPTIONS Object	936
13.4.10. POST Object	936
13.4.11. POST Object restore	938
13.4.12. PUT Object	938
13.4.13. PUT Object - Copy	940
13.4.14. PUT Object acl	942
13.4.15. PUT Object tagging	942
13.4.16. Abort Multipart Upload	943
13.4.17. Complete Multipart Upload	943
13.4.18. Initiate Multipart Upload	944
13.4.19. List Parts	945
13.4.20. Upload Part	946
13.4.21. Upload Part - Copy	947
Chapter 14. IAM API	949
14.1. Identity and Access Management (IAM)	949
14.2. IAM Supported Actions	950
14.2.1. HyperStore Extension to the IAM API: Actions for HyperStore Admin Tasks	965
14.3. IAM Supported Policy Document Elements	966
14.3.1. Policy Document Content for Granting S3 or IAM Permissions	966
14.3.2. Policy Document Content for Granting HyperStore Administrative Permissions	967
14.4. IAM Common Parameters	969
14.5. IAM Common Errors	970
Chapter 15. Open Source License Agreements	971

Chapter 1. Introduction

1.1. What's New in HyperStore 7.2

This topic introduces the main new features and enhancements for Clodian HyperStore version 7.2. Click an item for a summary of the change and links to further information.

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

1.1.1. S3 API New Features and Enhancements

xxx

xxx

More information:

-

HyperStore's IAM Service now enabled by default

HyperStore's IAM Service is now enabled by default. Previously this service was disabled by default, and you had to modify the system configuration if you wanted to activate the service.

More information:

- "Identity and Access Management (IAM) Feature Overview" (page 91)

1.1.2. Admin API New Features and Enhancements

xxx

xxx

More information:

-

Group attributes for filtering S3 endpoint displays

The **GroupInfo** object now includes attributes that give you the option to restrict which S3 endpoints display for the group's users when they log into the CMC and go to the **Security Credentials** page. By default in that CMC page users can view all the S3 endpoints that are configured in the system.

You can specify the S3 endpoint display filtering for a group when you create or edit the group through the Admin API or through the CMC.

More information:

- "**group**" (page 721)
- "**GroupInfo Object**" (page 731)

- **"Add a Group"** (page 237) (CMC)
- **"Edit a Group"** (page 241) (CMC)

New API calls for retrieving current storage usage for each user in a specified group

The Admin API now allows you, in a single call, to retrieve current stored byte counts for each user in a specified group; or, in a single call, to retrieve current stored object counts for each user in a specified group.

More information:

- **"GET /system/groupbytecount Get stored byte counts for all of a group's users"** (page 801)
- **"GET /system/groupobjectcount Get stored object counts for all of a group's users"** (page 802)

1.1.3. System Operations New Features and Enhancements

xxx

xxx

More information:

-

HyperStore firewall

HyperStore now includes a built-in firewall on each node that protects HyperStore internal services while allowing access to HyperStore public services. For fresh installations of HyperStore 7.2, the HyperStore firewall is enabled by default. For HyperStore systems originally installed as an older version and then upgraded to 7.2, the HyperStore is available but is disabled by default. You can enable or disable the firewall using the installer's Advanced Configuration Options.

More information:

- **"Configuring the HyperStore Firewall"** (page 569)

Automatic creation of installation staging directory

Previously you needed to create an installation directory when you first install HyperStore and then create a different staging directory each time that you upgrade HyperStore to a new version. Now, when you extract the HyperStore package (.bin file) for a HyperStore version that you are installing or upgrading to, a staging directory named `/opt/cloudian-staging/<version-number>` is automatically created and the package contents are extracted into that directory.

More information:

- **"Upgrading Your HyperStore Software Version"** (page 45)

Run hsstool from any directory

In previous HyperStore versions `hsstool` had to be run from the `/opt/cloudian/bin` directory. Now HyperStore automatically adds `/opt/cloudian/bin` to each host's \$PATH variable, so you can run `hsstool` from any directory.

More information:

- **"hsstool "** (page 609)

CMC's Operation Status page now includes region and DC info

In the CMC's **Operation Status** page, the information for each operation now includes the region and data center in which the target node resides. Previously this page only displayed the hostname of the target node

without identifying the region and data center.

More information:

- **"Operation Status"** (page 338)

Puppet Master and Cronjob primary and backup roles now allocated for high availability in multi-DC install

For a new multi-DC installation of HyperStore, the Puppet Master backup role is now allocated to a node in a different DC as the node hosting the Puppet Master primary role; and the Cronjobs backup role is allocated to a node in a different DC as the node hosting the Cronjob primary role.

More information:

- **"Services Distribution -- Multi-DC, Single Region"** (page 22)

hsstool proactiveverpairq output is now clearer

The output of the `hsstool proactiveverpairq` command now makes clear that this command returns only an estimate of the number of objects in a node's proactive repair queue (whereas the more resource-intensive `hsstool proactiveverpairq -a` option returns an exact count).

More information:

- **"hsstool proactiveverpairq"** (page 637)

/etc/init.d service scripts superseded by systemctl

The `/etc/init.d` scripts are no longer supported as a way to stop or start services on individual HyperStore nodes. Instead use `systemctl`, which is the preferred method on CentOS 7.

More information:

- **"Start or Stop Services on One Node"** (page 367)

Improved disk status display in CMC

In the CMC's **Node Status** page, the scheme of color-coded icons for indicating disk status has been made simpler and more clear.

More information:

- **"View a Node's Disk Detail"** (page 272)

Improved logging of CMC user logins

All user logins to the CMC are now recorded in the CMC application log `cloudian-ui.log`. The log entries include the user ID, group ID, and source IP address.

More information:

- **"HyperStore Logs"** (page 575)

1.2. HyperStore Documentation

The HyperStore user documentation consists of:

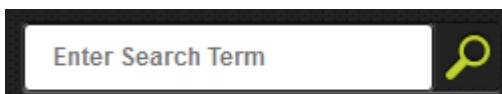
- HyperStore Help (HTML5)
- HyperStore Administrator's Guide (PDF)

- HyperStore Installation Guide (PDF)
- HyperStore Quick-Start for Software-Only Users (PDF)

The Help is available through the CMC (by clicking the Help button) and is also available in the directory `<installation-staging>/doc/HyperStoreHelp` on each of your HyperStore nodes (in that directory you can open the `HyperStoreHelp.html` file). The PDF guides can be downloaded through the links above, and are also available in the directory `<installation-staging>/doc/HyperStorePDFManuals` on each HyperStore node.

The Help has the exact same content as the Installation Guide and Administrator's guide, just in HTML rather than PDF. Further, starting with section "1. Introduction to HyperStore", the Help uses the exact same section numbering as is used in the Administrator's Guide -- so for example, section 4.1.2 in the Help is the same content as section 4.1.2 in the Administrator's Guide.

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



In the Help, in most cases screen shots are presented initially as small thumbnail images. This allows for a more compact initial view of the content on a page and makes it easier for you to skim through the text on the page. If you want to see the full size image simply hold your cursor over it.

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

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

12.5 group

[Admin API Methods]

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

- ▶ [12.5.1 DELETE /group](#) Delete a group
- ▶ [12.5.2 GET /group](#) Get a group's profile
- ▶ [12.5.3 GET /group/list](#) Get a list of group profiles
- ▶ [12.5.4 GET /group/ratingPlanId](#) Get a group's rating plan ID
- ▶ [12.5.5 POST /group](#) Change a group's profile
- ▶ [12.5.6 POST /group/ratingPlanId](#) Assign a rating plan to a group
- ▶ [12.5.7 PUT /group](#) Create a new group

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

12.5 group

[Admin API Methods]

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

▼ 12.5.1 DELETE /group Delete a group

Note Before you can delete a group you must first delete all users associated with the group, using the [DELETE /user](#) method.

The request line syntax for this method is as follows.

```
DELETE /group?groupId=xxx
```

There is no request payload.

Example Using cURL

The [example](#) below deletes the "QA" group.

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

Response Format

There is no response payload. For response status code this method will return either one of the [12.1.3 Common Response Status Codes](#) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {groupId}

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



If you have a comment or request regarding the HyperStore documentation, please send it to this email address:

cloudian-pubs@cloudian.com

You will not receive a reply, but the Cloudian Technical Publications team will review your comment and, if appropriate, redress the issue in an upcoming HyperStore release. Thank you for your feedback.

1.3. HyperStore Overview

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

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

- **Amazon S3 API compliance.** The HyperStore system is fully compatible with Amazon S3's HTTP REST API. Customers' existing HTTP S3 applications will work with the HyperStore service, and existing S3 development tools and libraries can be used for building HyperStore client applications.

- **Secure multi-tenancy.** The HyperStore system provides the capability to securely have multiple users reside on a single, shared infrastructure. Data for each user is logically separated from other users' data and cannot be accessed by any other user unless access permission is explicitly granted.
- **Group support.** An enterprise or work group can share a single HyperStore account. Each group member can have dedicated storage space, and the group can be managed by a designated group administrator.
- **Quality of service controls.** HyperStore system administrators can set storage quotas and usage rate limits on a per-group and per-user basis. Group administrators can set quotas and rate controls for individual members of the group.
- **Access control rights.** Read and write access controls are supported at per-bucket and per-object granularity. Objects can also be exposed via public URLs for regular web access, subject to configurable expiration periods.
- **Reporting and billing.** The HyperStore system supports usage reporting on a system-wide, group-wide, or individual user basis. Billing of groups or users can be based on storage quotas and usage rates (such as bytes in and bytes out).
- **Horizontal scalability.** Running on commodity off-the-shelf hardware, a HyperStore system can scale up to thousands of nodes across multiple data centers, supporting millions of users and hundreds of petabytes of data. New nodes can be added without service interruption.
- **High availability.** The HyperStore system has a fully distributed, peer-to-peer architecture, with no single point of failure. The system is resilient to network and node failures with no data loss due to the automatic replication and recovery processes inherent to the architecture. A HyperStore cluster can be deployed across multiple data centers to provide redundancy and resilience in the event of a data center scale disaster.

1.4. Licensing and Auditing

Subjects covered in this section:

- *Introduction (immediately below)*
- **"License Expiration"** (page 6)
- **"Licensed Maximum HyperStore Storage Usage"** (page 7)
- **"Licensed Maximum Tiered Storage Usage"** (page 9)
- **"WORM License"** (page 10)
- **"License Updating"** (page 10)
- **"Auditing"** (page 10)

A valid Cloudian software license is required to run HyperStore software. Evaluation licenses are available as well as production licenses. Before using HyperStore software, you must obtain a license from your Cloudian sales representative or by registering for a free trial on the Cloudian website.

1.4.1. License Expiration

In the CMC's [Cluster Information](#) page you can view your HyperStore license expiration date.

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

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

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

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

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

It's best to update your license well in advance of your license expiration date. See "[License Updating](#)" (page 10) below.

1.4.2. Licensed Maximum HyperStore Storage Usage

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

With a license based on **Net** storage, the limit is on total object storage bytes minus overhead from object replication or erasure coding. For example if a 1GB object is replicated three times in your system it counts as only 1GB toward a Net storage limit.

A Net storage license is typically used if your cluster consists entirely of software-only nodes, with no HyperStore Appliance nodes.

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

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

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

If your license is based on Raw storage, then your total licensed Raw storage limit will be automatically increased if you add a new HyperStore Appliance node to the system. The amount of Raw storage added to

your licensed system maximum depends on the particular HyperStore Appliance that you've added to your system. Conversely, if you remove an Appliance from your system this will reduce your total licensed system Raw storage maximum; and the Raw storage allowance associated with a particular Appliance machine cannot be transferred to other nodes in your system.

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

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

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

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

Note: In the case of Raw usage, your deletions will not impact your system's raw usage count until the [hourly system cron job for processing the object deletion queue](#) runs. By contrast, a Net usage count is decremented immediately when you delete objects.

- You acquire and install a new license with a larger storage maximum (see "[License Updating](#)" (page 10)). Upon new license installation, S3 PUTs and POSTs will again be allowed.

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

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

Also, if some users use the **cross-region replication** feature to replicate objects from one HyperStore bucket to another HyperStore bucket within the same HyperStore system, then the original source objects and the object replicas in the destination bucket both count toward your system stored bytes count. For more information see "[Cross-Region Replication Impact on Usage Tracking](#)" (page 135)

IMPORTANT: Regardless of whether your system has a Net storage license or a Raw storage license, **if the data disks on a node become 90% full then that node will stop accepting new S3 writes**. This

is not a license enforcement mechanism but rather a system safety feature. For more information see ["Disk Usage Balancing Feature Overview" \(page 85\)](#).

1.4.3. Licensed Maximum Tiered Storage Usage

Depending on your particular license terms, your HyperStore license may enforce a maximum for volume of tiered data stored in external systems other than HyperStore. Data that's been auto-tiered out of HyperStore and is currently stored in any third party system -- such as Amazon, Azure, Google, Spectra BlackPearl, and so on -- counts toward your licensed tiered storage 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.

Note In the case of bucket auto-tiering configurations that tier to non-HyperStore destinations and also use the "[Retain Local Copy for X Days](#)" option, during the period that a local copy of an auto-tiered object is retained the object's size will count toward both your tiered storage limit and your HyperStore storage limit. The same is true of auto-tiered objects that users temporarily [restore to local storage](#).

In many respects alerting and enforcement around the tiered storage maximum parallels that for the HyperStore storage maximum.

In the CMC's [Cluster Information](#) page you can view your tiered storage maximum and also your current tiered storage usage level. If your tiered usage level exceeds 70% of your licensed maximum the CMC displays a warning message in the [Cluster Information](#) page; and this becomes a critical message if the usage exceeds 90% of licensed maximum.

The tiered usage maximum will be **automatically enforced** by the system no longer allowing auto-tiering to any destination system other than HyperStore (tiering to HyperStore systems is not affected). This enforcement will kick in when your tiered storage byte count reaches **110%** of your licensed maximum. At this point auto-tiering to non-HyperStore destinations will no longer work until one of the following occurs:

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

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

- You acquire and install a new license with a larger tiered storage maximum (see ["License Updating" \(page 10\)](#)). Upon new license installation, auto-tiering to non-HyperStore destinations will be allowed again and automatically resume.

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

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

1.4.4. WORM License

HyperStore licenses come in different types in regard to support or lack of support for the HyperStore WORM feature (Write Once, Read Many). See "**HyperStore Licensing for WORM (Bucket Lock)**" (page 160).

1.4.5. License Updating

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

Once you've obtained a new license file you can use the CMC to dynamically apply the new license file to your HyperStore system. In the CMC, go to the [Cluster Information](#) page to install a new license file.

1.4.6. Auditing

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

1.5. HyperStore Services

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

For services distribution diagrams, see "**Services Distribution -- 3 Nodes, Single DC**" (page 21).

Note Within your installation cluster, the HyperStore installer **automatically chooses the hosts** for services that are not intended to run on every node. These host assignments are recorded to an installation configuration file that the installer generates when it runs (*CloudianInstallConfiguration.txt* in your installation staging directory). After your installation is completed, these host assignments can also be viewed on the CMC's [Cluster Information](#) page. If you want to modify these assignments after install, see "**Change Node Role Assignments**" (page 405).

If you installed HyperStore software on only one node, then all these services will run on that node.

Service Category	Service	Where It Is Installed
Common Services	S3 Service	Every node.
	HyperStore Service	Every node.
	Cassandra Service	Every node.
	Admin Service	Every node.
	IAM Service	Every node (but service is disabled by default).
	Cloudian Management Console (CMC)	Every node.
Specialized Support Services	Redis Credentials DB Master	One node per entire HyperStore system.
	Redis Credentials DB Slaves	Two per data center. If you have a large cluster (25 nodes or more in a data center), consult with Cloudian Support about whether you should add more Redis Credentials slaves. For instructions on adding slaves, see " Move or Add a Redis Credentials Slave or Redis QoS Slave " (page 408). (Note: If you upgraded from a HyperStore version older than 6.0, you will have only one Redis Credentials slave per data center.) Slaves will not be on same node as Credentials master
	Redis QoS DB Master(s)	One node per service region.
	Redis QoS DB Slave(s)	One node per data center. Slave will not be on same node as QoS Master.
	Redis Monitor	One primary node and one backup node per entire HyperStore system.
	Crontab configuration and Monitoring Data Collector	One primary node and one backup node per service region.
	Local NTP server	One local NTP server per service region.
	Puppet Master	One primary node and one backup node per entire

Service Category	Service	Where It Is Installed
		HyperStore system.

1.5.1. S3 Service

The HyperStore system provides a high-performance S3 proxy service. The S3 Service processes S3 REST requests incoming from client applications (including the Cloudian Management Console). On the back side, the S3 Service interfaces with:

- The HyperStore Service, Cassandra "UserData_<policyid>" keyspaces, and Cassandra "ECKeyspace" keyspace in order to store, retrieve, and delete users' S3 data objects.
- The Cassandra "AccountInfo" keyspace to update and retrieve user account information.
- The Cassandra "Reports" keyspace to update users' transaction history.
- The Redis "Credentials" DB to implement user authentication, S3 bucket validation, and other functions in support of S3 request processing.
- The Redis "QoS" DB to enforce group and user level quality of service restrictions.

The S3 Service is built on [Jetty](#) server technology.

1.5.2. HyperStore Service and the HSFS

As an object store, Cassandra provides a wealth of valuable built-in functionality including data partitioning, automatic replication, easy cluster expansion, quorum calculation, and so on. For storing small data items, Cassandra also provides good performance. But as the data size increases, storing data on the Linux file system becomes more efficient than storing it in Cassandra.

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

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

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

- Replicated storage
- Erasure coded storage

For more information on data storage and protection options, see "**Storage Policies Feature Overview**" (page 140).

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

- For S3 object replicas:

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

- For S3 object erasure coded fragments:

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

```
255>/<filename>
```

- The path segments are:

- The <mountpoint> is one of your HyperStore data mount points as configured by the "**hyper-store_data_directory**" (page 462) setting in *common.csv*.
- The *hsfs* or *ec* segment distinguishes replicated data (designated here as "hsfs") from erasure-coded data (designated as "ec").
- The <base62-encoded-vNode-token> is a base-62 encoding of the token belonging to the **vNode** to which the object replica or erasure coded fragment is assigned.
- The <policyid> segment indicates the storage policy used by the S3 storage bucket with which the object is associated.
- The two <000-255> segments of the path are based on a hash of the <filename>, normalized to a 255*255 number.
- The <filename> is a dot-separated concatenation of the object's system-assigned token and a timestamp based on the object's Last Modified Time. The token is an MD5 hash (in decimal format) of the bucket name and object name. The timestamp is formatted as <UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>. The last element of the timestamp is the IP address (in hexadecimal format) of the S3 Service node that processed the object upload request.

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

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

```
/hy-
perstore1/hsfs/1L1tEZZCCQwdQBdGe14yNk/c4a276180b0c99346e2285946f60e59c/109/154/
55898779481268535726200574916609372181.1487608689783689800-0A320A15
```

In the above example:

- "hyperstore1" is one of the HyperStore data mount points configured for the system (as specified by the configuration setting *common.csv*: *hyperstore_data_directory*)
- "hsfs" indicates that the object is a replicated object (not an erasure-coded object)
- "1L1tEZZCCQwdQBdGe14yNk" is the Base-62 encoding of the token belonging to the vNode to which the object instance is assigned
- "c4a276180b0c99346e2285946f60e59c" is the system-generated identifier of the storage policy used by the S3 storage bucket with which the object is associated.
- "109/154" is a hash of the file name, normalized to a 255*255 number.
- "55898779481268535726200574916609372181.1487608689783689800-0A320A15" is the file name. The "55898779481268535726200574916609372181" segment is the object's system-assigned token, in decimal format. The "1487608689783689800-0A320A15" segment is the object's Last Modified Time timestamp, in format <UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>.

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

1.5.2.1. File Digests

Each replica file and each erasure coded fragment file has a corresponding digest containing the hexadecimal MD5 hash of the file as well as a small amount of metadata including the object name (the name of the object for which the file is a replica or an erasure coded fragment) and a last modified timestamp. These digests are used by the HyperStore Service when reading and writing objects and are also used by "hsstool" (page 609) operations such as repair and cleanup. The digests are stored in high-performance [RocksDB](#) databases (persistent key-value stores) on the same mount point as the corresponding file. On each mount point, there is one RocksDB database for storing digests for replica data files, and one RocksDB database for storing digests for erasure coded data files.

For replica data files, the digest database is stored under:

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

For erasure coded data files, the digest database is stored under:

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

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

Note When an existing object is updated by an S3 client, the object's token (a decimal formatted MD5 hash of the object key) remains the same but the object's digest (including a hexadecimal formatted MD5 hash of the object data) changes.

For an erasure coded S3 object, each fragment has the same token (based on object key) but a different digest (based on fragment content).

For multipart S3 objects — uploaded to the system through the S3 API methods for Multipart Uploads — each part has a different token (since each part has a distinct object key incorporating a part number) and a different digest (based on part content).

Retrieving a Digest

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

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

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

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

```
getDigestString=bucket1/LocalInstallProcedure.docx

10/25/2016 06:27:30 -0700 org.archive.jmx.Client getDi-
gestString=bucket1/LocalInstallProcedure.docx:
68855684469431950092982403183202182439.1477401010696032189-0A0A1608
9c741e3e7bbe03e05510071055151a6e
bucket1/LocalInstallProcedure.docx
2016-10-25T13:10:10.696Z
/var/lib/cloud-
ian/hsfs/1mDFFH13tL1DIyHNKDX3d/a7a28896654319cc7af4c39748a27e3d/243/187/
68855684469431950092982403183202182439.1477401010696032189-0A0A1608
13164
```

For clarity, in the example above an empty line has been inserted between the JMX command and the response. This example is for a replicated object named *LocalInstallProcedure.docx* from the bucket named *bucket1*. In the response, *68855684469431950092982403183202182439.1477401010696032189-0A0A1608* is the Rocks DB database key for this entry (in format *<objectToken>.<timestamp>*). The subsequent lines are the digest contents. *9c741e3e7bbe03e05510071055151a6e* is the replica's MD5 hash in hexadecimal; the */var/lib/cloudian/hsfs/...* line is the replica file path and name; and *13164* is the replica's file size in bytes.

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

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

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

1.5.3. Cassandra Service

The HyperStore system uses the Apache open source storage platform [Cassandra](#) to store several types of data. The HyperStore system creates and uses several distinct "keyspaces" (approximately equivalent to databases) within Cassandra:

- The **UserData_<policyid>** keystores store:
 - User bucket information
 - Object metadata. For an overview of the HyperStore system's support for object metadata, see "[Object Metadata and Tagging Feature Overview](#)" (page 103).

Note: There is one *UserData_<policyid>* keyspace for each storage policy in the system. For information about storage policies see "[Storage Policies Feature Overview](#)" (page 140).

- The **AccountInfo** keyspace stores information about S3 user accounts and group accounts.
- The **Reports** keyspace stores system-wide, per-group, and per-user S3 usage data, in support of the HyperStore usage reporting functionality. It will also store per-bucket usage data if you [enable per-bucket usage tracking](#).
- The **Monitoring** keyspace stores system monitoring statistics in support of HyperStore's system monitoring functionality. It also stores status information for node operations such as repairs and cleanups; and stores token range maps that are used by the system when you add nodes to your cluster.
- The **ECKespace** keyspace does not actually store any erasure coded object data; rather, the HyperStore system creates this keyspace so that the HyperStore erasure coding feature can leverage Cassandra functions for token-based mapping of objects (erasure coded object fragments, in this case) to nodes within the storage cluster.

S3 client applications do not access Cassandra databases directly; all S3 client access is to the [S3 Service](#), which in turn accesses Cassandra in support of S3 operations. The [HyperStore Service](#) and [Admin Service](#) also access Cassandra.

1.5.4. Redis Credentials and Redis QoS Services

The HyperStore system uses the lightweight, open source [Redis](#) key-value data store to store a variety of data that supports HyperStore S3 service features. There are two types of Redis DBs in a HyperStore deployment:

- The **Redis Credentials DB** stores user credentials and additional S3 operation supporting data such as multi-part upload session information and public URL access counters.
- The **Redis QoS DB** stores user-level and group-level [Quality of Service](#) settings that have been established by system administrators. The DB is also used to keep count of user requests, so that Quality of Service limits can be enforced by the system.

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

Note for multi-region systems

In a multi-region HyperStore deployment there will be:

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

1.5.4.1. Redis Node Roles

Each Redis DB is implemented across two or more nodes, with the nodes playing different roles. These roles are:

- **master** — All write requests from Redis clients are implemented on the master node. There is only one master node for each Redis DB.

Note for multi-region systems

In a multi-region HyperStore deployment, the universal Redis Credentials DB has one master node and each regional Redis QoS DB has its own master node.

- **slave** — In each Redis DB, data from the Redis master node is asynchronously replicated on to one or more slave nodes (at least one slave node per data center). The slave nodes support doing reads for

Redis clients but not writes. If a master node fails, the master role is automatically failed over to a slave node. This fail-over process is managed by the "**Redis Monitor Service**" (page 17).

Among the master node and slave node(s) of the Redis QoS DB, these roles are assigned for supporting reads of the database by clients:

- **readprimary** — Within a Redis QoS DB, the "read primary" node is the first-choice Redis QoS host for doing reads. This role should be assigned to a slave node, not the master. In a multi-datacenter deployment each datacenter should have its own Redis QoS read primary node.
- **readsecondary** — Within a Redis QoS DB, the "read secondary" node is the second-choice Redis QoS host for doing reads, if the "read primary" isn't reachable. The read secondary role can be assigned to the master.

Note: The static "readprimary" and "readsecondary" roles do not apply to Redis Credentials. Instead, Redis Credentials clients when doing reads use round-robin to select from among the Redis Credentials slaves within a data center.

Redis roles are assigned to your HyperStore nodes automatically during installation.

1.5.5. Redis Monitor Service

The Redis Monitor monitors Redis Credentials DB and Redis QoS DB cluster health and implements automatic failover of the Redis master node role within each of the two Redis DBs. For redundancy, the Redis Monitor runs on two HyperStore nodes, configured as primary on one node and as backup on the other node.

If the Redis Monitor detects that a Redis master node has gone down, it promotes an available slave node to the master node role; and informs the Redis cluster's clients (the S3 Service, Admin Service, and HyperStore Service) of the identity of the new master.

Note In a multi-DC system the HyperStore installer puts the Redis Monitor backup in the same DC as the Redis Monitor primary. Keep them in the same DC -- do **not** migrate them such that the primary and backup are in different DCs.

Note for multi-region systems

In a multi-region HyperStore deployment, a single Redis Monitor instance will monitor multiple regional Redis QoS DBs as well as the one universal Redis Credentials DB. Each regional Redis QoS DB will have its own configured cluster membership list that the Redis Monitor will refer to if a slave needs to be promoted to master.

1.5.6. Admin Service

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

- Provisioning groups and users.
- Managing quality of service (QoS) controls.
- Creating and managing rating plans.

- Generating usage data reports.
- Generating bills.

For information on the Admin API see "[HyperStore Admin API Introduction](#)" (page 695).

The "**Cloudian Management Console (CMC) Service**" (page 18) is a client to the Admin Service. You also have the option of building your own Admin Service client.

The Admin Service is closely integrated with the "**S3 Service**" (page 12). Both leverage [Jetty](#) technology; both are installed together; and both are started and stopped together by the same commands.

1.5.7. IAM Service

The HyperStore IAM Service, which is disabled by default, supports Amazon-compliant API methods for Identity and Access Management. If you enable the IAM Service then this service is started and stopped by the same commands that start and stop the S3 Service.

When the IAM Service is enabled, the "**Cloudian Management Console (CMC) Service**" (page 18) is a client to the IAM Service. You also have the option of using a third party IAM client or building your own client.

The IAM Service also supports role-based access control (RBAC) for certain read-only HyperStore administration functions.

For more information about this service see "[Identity and Access Management \(IAM\) Feature Overview](#)" (page 91).

1.5.8. Cloudian Management Console (CMC) Service

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

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

- Provisioning groups and users.
- Managing quality of service (QoS) controls.
- Creating and managing rating plans.
- Generating usage data reports.
- Generating bills.
- Viewing and managing users' stored data objects.
- Setting access control rights on users' buckets and stored objects.

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

The CMC acts as a client to the "**Admin Service**" (page 17) and the "**S3 Service**" (page 12).

1.5.9. Supporting Services

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

- **Cloudian Monitoring Agent** — The Cloudian Monitoring Agent runs on each HyperStore node and monitors node health and performance statistics. The Agent also plays a role in the triggering of event notification emails to system administrators. System and node statistics are viewable through the CMC; and you can configure event notification rules through the CMC as well.
- **Cloudian Monitoring Data Collector** — The Cloudian Monitoring Data Collector runs (together with the system maintenance cron jobs) on one node in each of your service regions, and regularly collects data from the Monitoring Agents. The Monitoring Collector writes its collected node health statistics to Cassandra's "Monitoring" keyspace. The Monitoring Collector is also configured (together with the cron jobs) on a backup node, and automatic failover to the backup occurs if the primary node goes offline or if *crond* goes down on the primary.
- **Puppet** — As part of the HyperStore software installation, the HyperStore installer installs the open source version of [Puppet](#) and uses it to implement initial HyperStore system configuration. HyperStore also uses Puppet for support of ongoing configuration management. For more information see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454). Puppet agents run on every node. The Puppet Master runs on one node and is also configured on a backup node. Manual failover to the backup is supported if the primary Puppet Master instance goes down.
- **Pre-Configured *ntp*** — Accurate, synchronized time across the cluster is vital to HyperStore service. When you install your HyperStore cluster, the installation script automatically configures a robust NTP set-up using *ntp*. In each HyperStore data center four of your HyperStore nodes are automatically configured to act as internal NTP servers, which synchronize with external NTP servers (by default the servers from the *pool.ntp.org* project). Other HyperStore hosts in each data center are configured as clients of the internal NTP servers. For more information see "**NTP Automatic Set-Up**" (page 540).

To see which of your HyperStore nodes are internal NTP servers and which external NTP servers they are synchronizing with, log into the CMC and go to the [Cluster Information](#) page.

Note If a HyperStore data center has only four or fewer nodes, then all the nodes in the data center are configured as internal NTP servers.

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

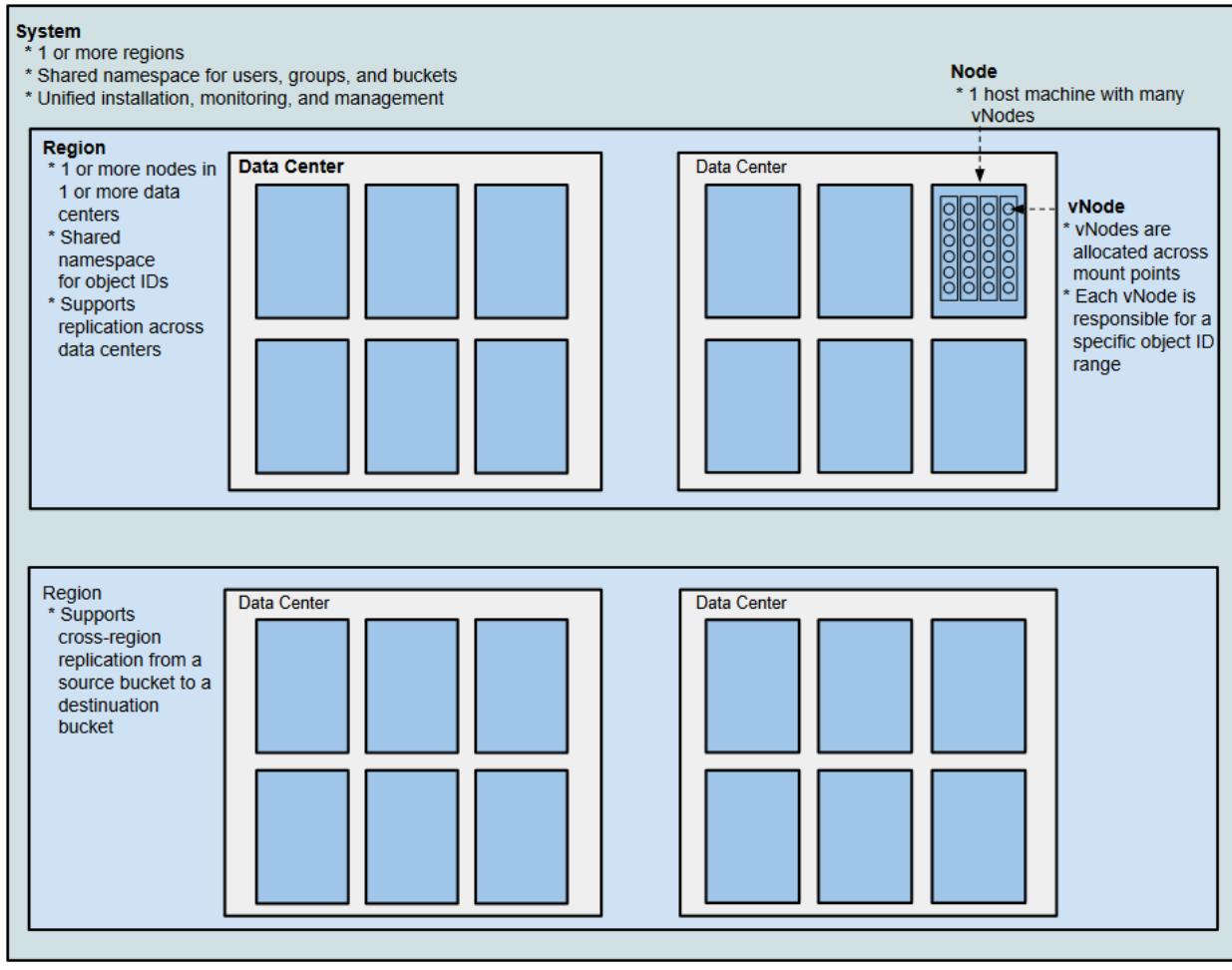
1.6. System Diagrams

1.6.1. System Levels

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

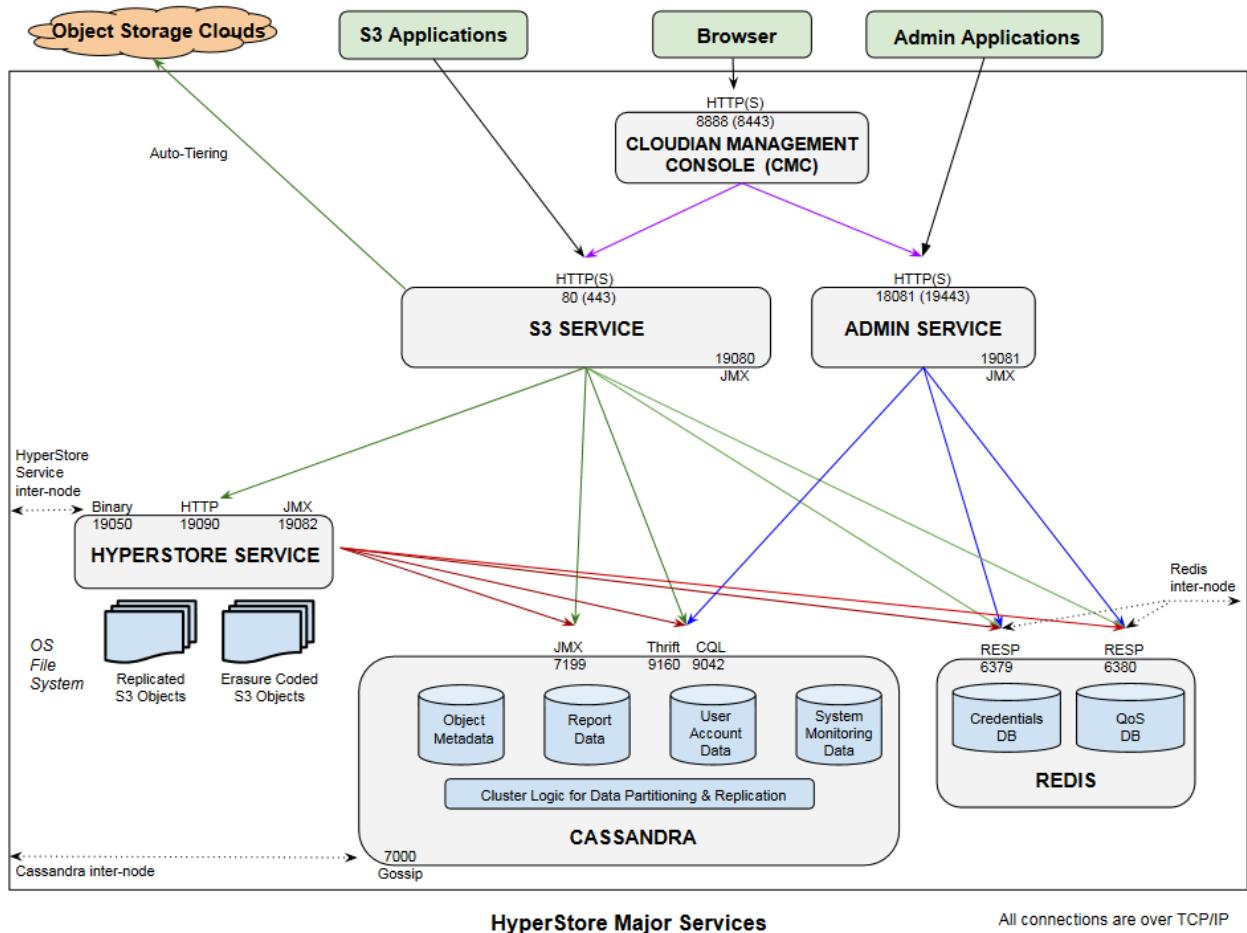
- System
- Region (also known as a "Cluster")
- Data Center

- Node
- vNode



1.6.2. Service Interconnections

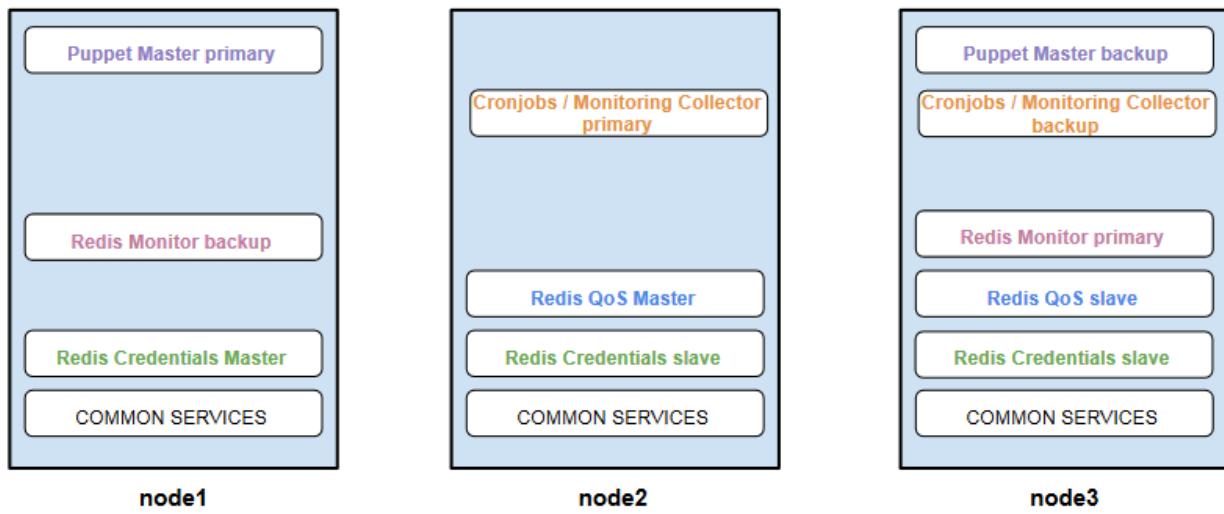
The diagram below shows the major service components that comprise a HyperStore system, the connections between those services, the direction of the connections, and the default listening ports to which connections are made. Outgoing connections from each service component are color-coded — for example, outgoing connections from the S3 Service are shown in green, outgoing connections from the Admin Service are blue, and so on.



1.6.3. Services Distribution -- 3 Nodes, Single DC

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

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



HyperStore Services Distribution -- Three Node System

Note For information about the exact location of services in your HyperStore system, log into the CMC and go to the [Cluster Information](#) page. The system allows you to move services from one host to another, if you wish to do so. For instructions see "[Change Node Role Assignments](#)" (page 405).

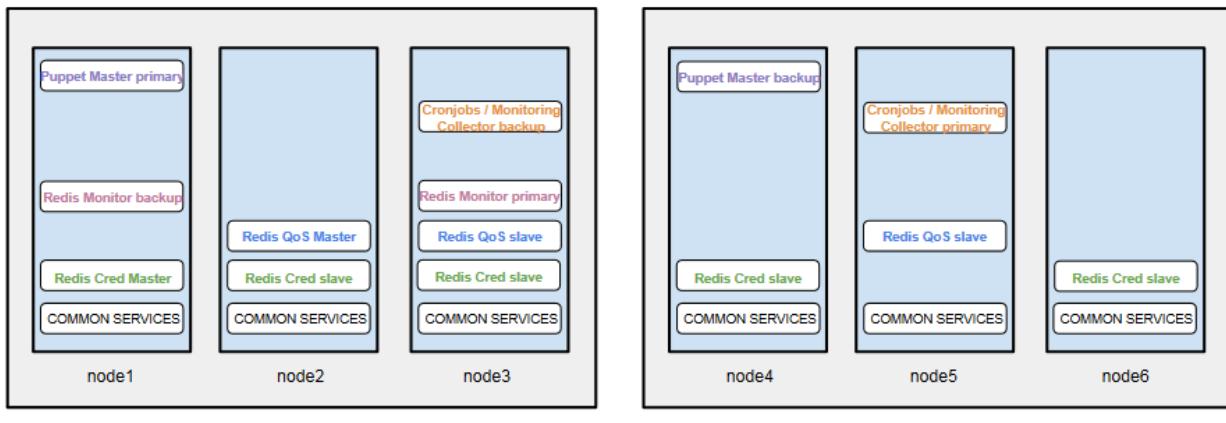
Note If you have a very large cluster (25 nodes or more in a data center), consult with Cloudian Support about whether you should add more Redis Credentials slaves. For instructions on adding Credentials slaves, see "[Move or Add a Redis Credentials Slave or Redis QoS Slave](#)" (page 408).

Note Starting with HyperStore 7.2, the Salt master is located with the Puppet master.

1.6.4. Services Distribution -- Multi-DC, Single Region

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

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



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

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

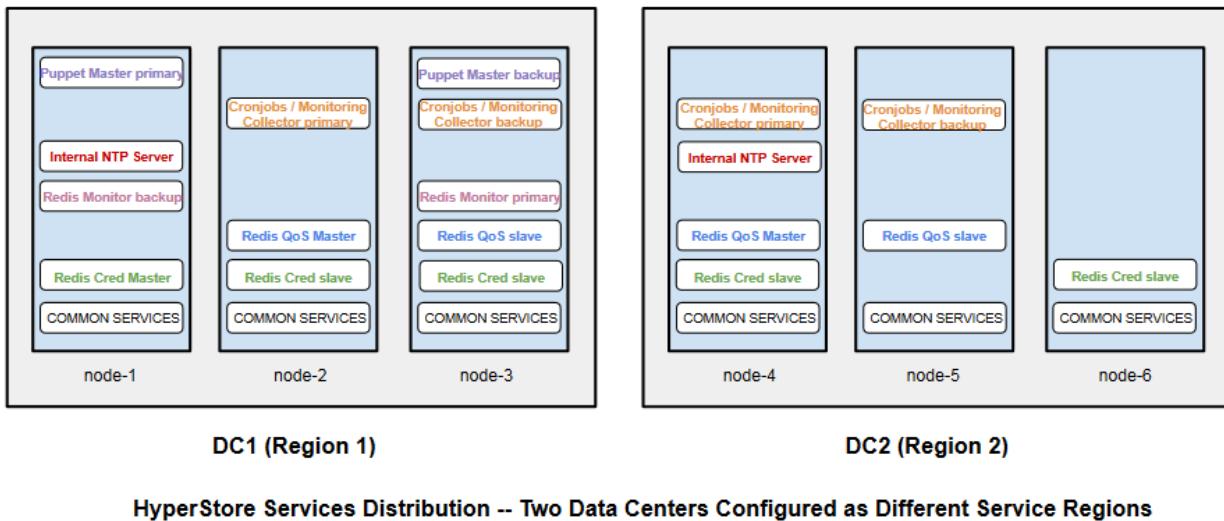
Note To check the current location of specialized services within your multi-DC HyperStore system, go to the CMC's [Cluster Information](#) page.

Note Starting with HyperStore 7.2, the Salt master is located with the Puppet master.

1.6.5. Services Distribution -- Multi-Region

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

- The "COMMON SERVICES" (S3 Service, Admin Service, HyperStore Service, Cassandra, CMC, Puppet Agent, and Monitoring Agent) run on every node in your multi-region system.
- The whole multi-region system is served by a single active Puppet master and a single Redis Credentials master.
- Each region has its own Redis QoS master and its own active Cronjob host.



Note Starting with HyperStore 7.2, the Salt master is located with the Puppet master.

1.6.6. Specialized Services Availability

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

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

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



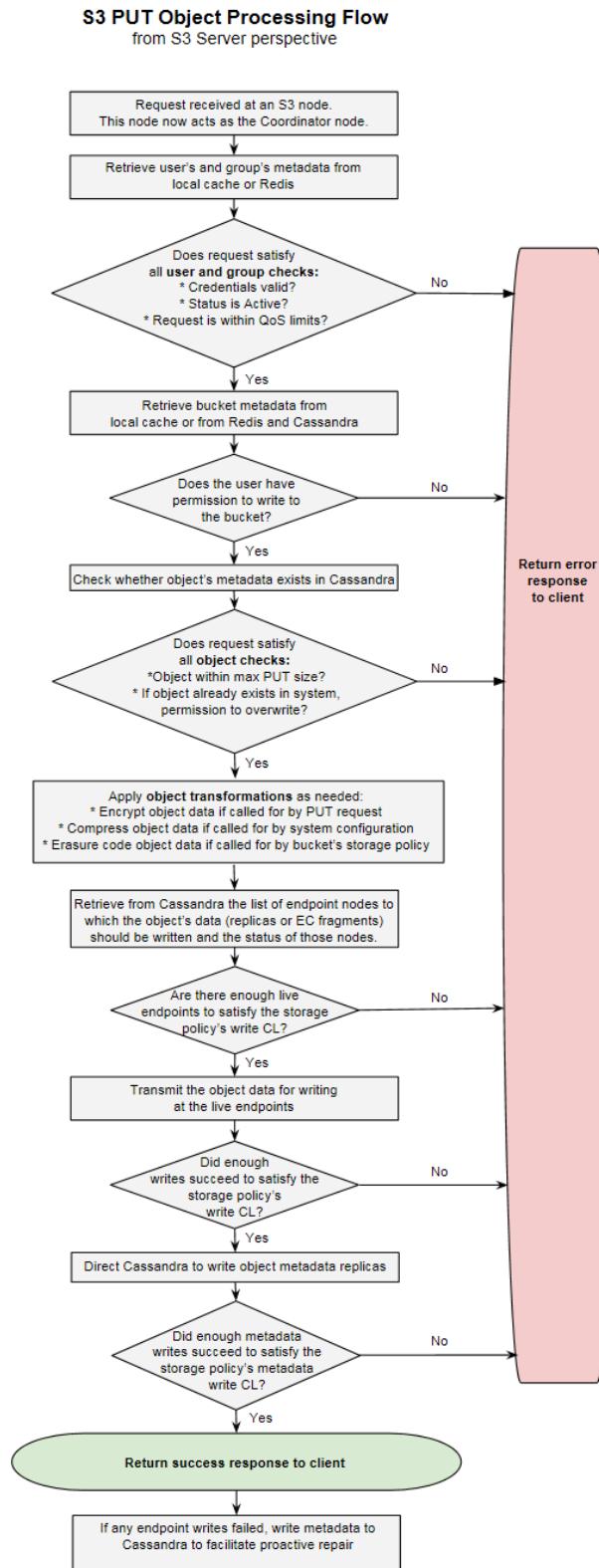
Note The automatic failover of the Cronjobs and Monitoring Data Collector roles from the primary to the backup instance invokes the `cloudianInstall.sh` script and will fail if `cloudianInstall.sh` is already running. When you occasionally use `cloudianInstall.sh` for system configuration tasks, remember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

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

Note Starting with HyperStore 7.2, the Salt master is located together with the Puppet master, and if you manually fail over the Puppet master role to the backup node, the Salt master role moves to that backup node as well.

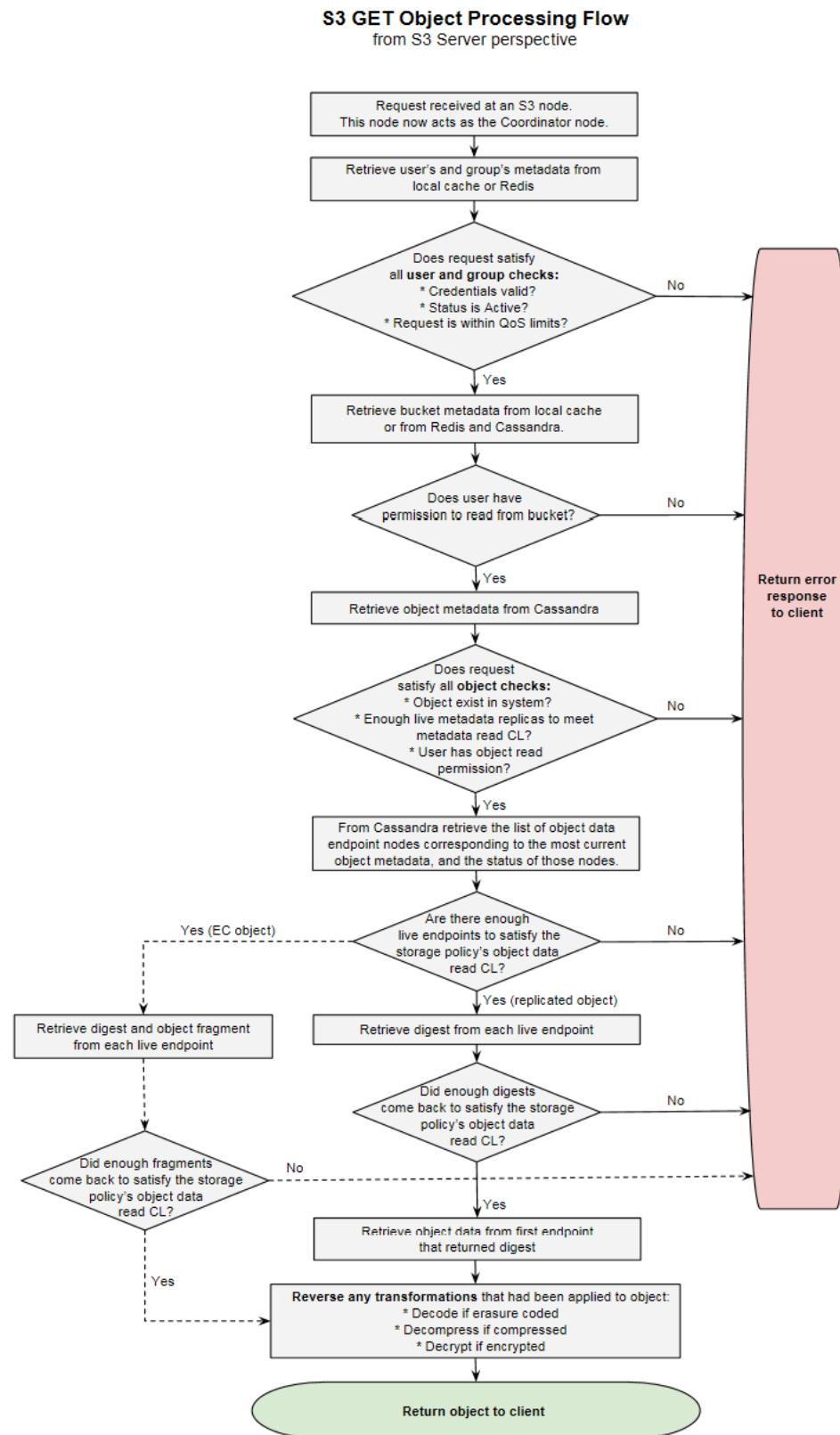
1.6.7. S3 PUT Processing Flow

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



1.6.8. S3 GET Processing Flow

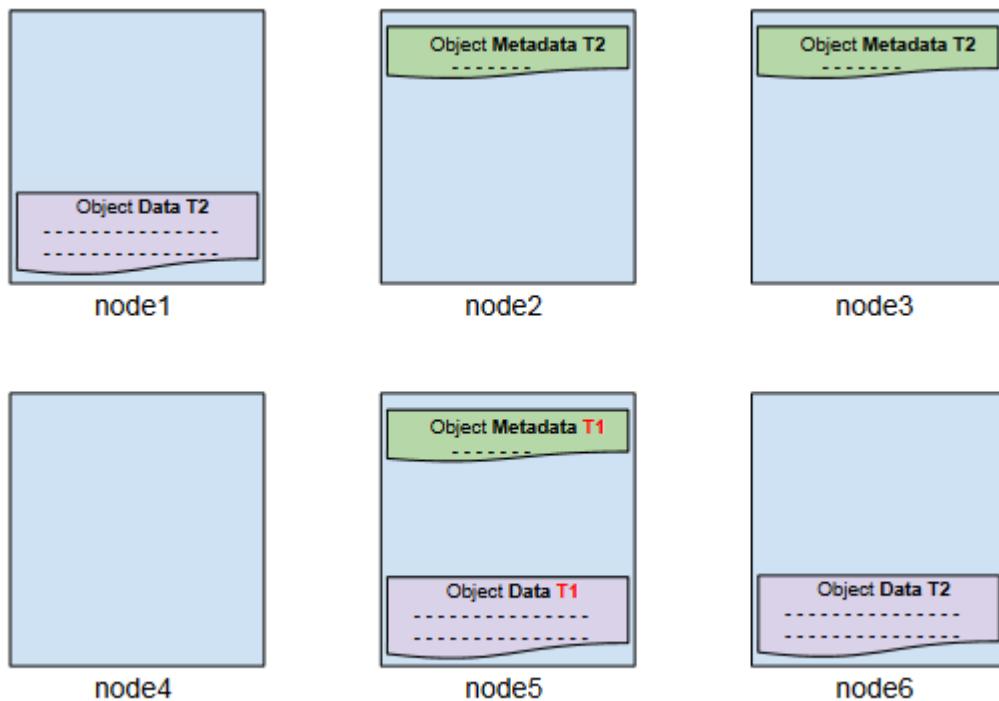
This diagram shows the main aspects of how the HyperStore system processes an S3 GET Object request, from the perspective of the S3 Service.



1.6.9. Data Freshness for Replicated Object Reads

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

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



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

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

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

For more information on S3 write and read availability under various consistency level configurations, see ["Storage Policy Resilience to Downed Nodes" \(page 148\)](#).

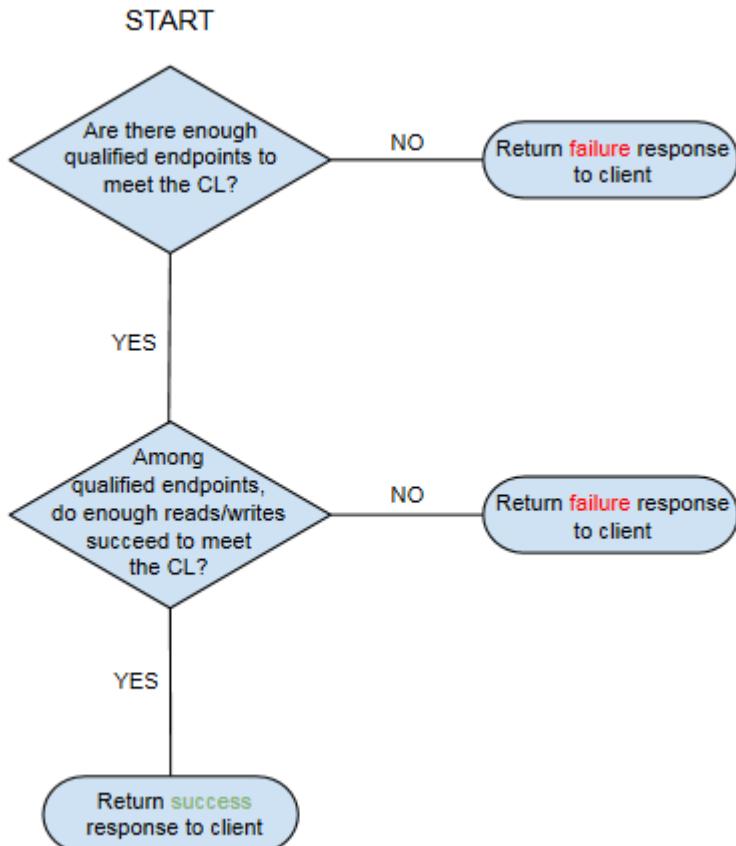
1.6.10. Dynamic Consistency Levels

When you ["Add a Storage Policy" \(page 308\)](#) to your HyperStore system one of the policy dimensions that you configure is consistency requirements for writes and reads of object data and metadata. When doing so, you have the option of configuring "dynamic" consistency level requirements. With dynamic CLs, you specify two or more consistency levels that differ in strictness. If the stricter consistency level (the "primary" consistency level) cannot be met for a given S3 request because there are not enough qualified endpoints, then the system tries to meet the less strict level (the "fallback" consistency level).

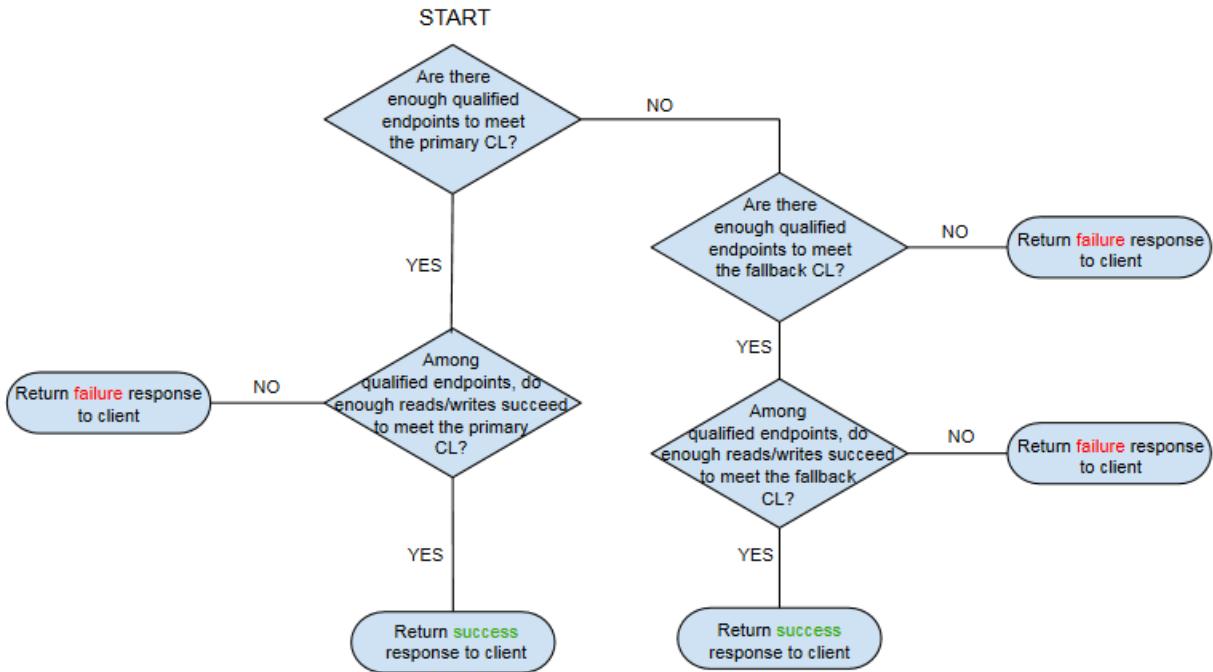
Note Although HyperStore applies dynamic consistency level logic only to the writing and reading of **object metadata** (and not object data), this still has the effect of controlling overall S3 request success or failure logic -- since for an S3 write or read operation to succeed it must succeed for the object metadata as well as the object data.

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

1.6.10.1. Standard Consistency Level Logic



1.6.10.2. Dynamic Consistency Levels Logic



1.6.10.3. Dynamic Consistency Levels Logic Described

When the S3 Service processes an S3 request such as a PUT or a GET of an object, it checks the system for a list of "endpoints" for that particular object — the nodes that the object data and object metadata should be written to or read from. The system determines the endpoint node list based on the object token and the applicable storage policy (such as 3X replication or 4+2 erasure coding). The system then checks each of the endpoint nodes for any of these disqualifying conditions:

- Cassandra Service is down or unreachable
- HyperStore Service has been marked as down by the S3 Service (see "[Node Status Configuration](#)" (page 521))
- Node has been put into Maintenance Mode by operator (see "[Start Maintenance Mode](#)" (page 285))
- Node is in a StopWrite condition due to all data disks being 90% full or more (disqualifying only for write requests, not read requests; see "[Automatic Stop of Writes to a Node at 90% Usage](#)" (page 86))
- Disk on which requested data resides is [disabled](#) (relevant only for read requests)

If the number of qualified endpoint nodes -- endpoint nodes free of any of these disqualifying conditions -- is enough to achieve the primary CL, the system proceeds with trying to achieve the primary CL. If enough endpoint writes or reads succeed to achieve the primary CL, a success response is returned to the S3 client. If not -- such as if an error is encountered at one of the endpoints during the attempted write or read of object metadata -- then the S3 request fails and an error is returned to the S3 client. The system will **not** try to achieve the fallback CL in this scenario.

If the number of qualified endpoint nodes is too few to achieve the primary CL but enough to achieve the fallback CL, the system proceeds with trying to achieve the fallback CL. If enough endpoint writes or reads succeed to achieve the fallback CL, a success response is returned to the S3 client. If not then the S3 request fails and an error is returned to the S3 client.

If the number of qualified endpoint nodes is too few to achieve the fallback CL, then the S3 request fails and an error is returned to the S3 client.

As an example, with a Replication Within Single Data Center storage policy, for the object Write CL configuration you could select the "ALL" CL and also the (less stringent) "QUORUM" CL. With this configuration, for a given request to write an S3 object, if the number of qualified endpoints is enough to achieve the ALL level, the system will try to achieve the ALL level for the operation. If the number of qualified endpoints is not enough to achieve ALL but is enough to achieve QUORUM, the system will try to achieve QUORUM for the operation. If the number of qualified endpoints is not enough to achieve even the QUORUM level, the operation will fail.

It's important to note that once the system determines that enough qualified endpoints are available to try to meet the stricter of the configured CLs, the system is then committed to that path in terms of the object metadata writes or reads. If there is a subsequent failure on that path — such as a metadata write failure on one or more of the endpoints — then the request fails and an error is returned to the client. The system does not go back and try to achieve the less-strict CL in this scenario.

1.6.11. How vNodes Work

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

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

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

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

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

For example:

Number of Data Disks on Host	Size of Data Disks	Total TBs of Data Disk on Host	Number of Tokens Assigned To Host
2	500GB	1TB	2

Number of Data Disks on Host	Size of Data Disks	Total TBs of Data Disk on Host	Number of Tokens Assigned To Host
			(1TB X .7 = .7, but minimum is 1 token per data disk)
4	8TB	32TB	22 (32TB X .7 = 22.4, rounded down = 22)
8	4 X 8TB 4 X 10TB	72TB	50 (72TB X .7 = 50.4, rounded down = 50)
12	10TB	120TB	84 (120TB X .7 = 84)
24	16TB	384TB	268 (384TB X 0.7 = 268.8, rounded down = 268)

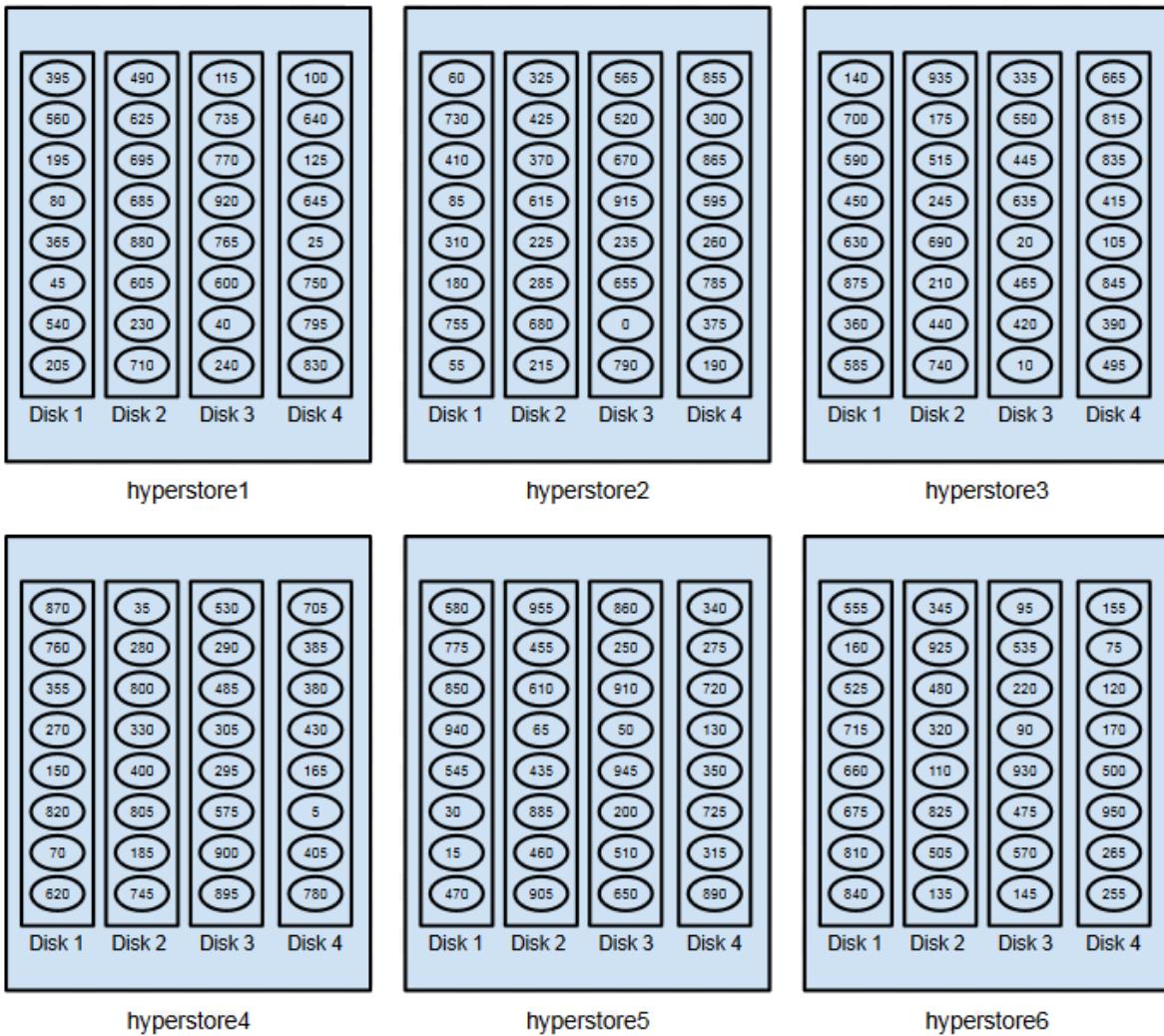
Note The data disk sizes in the table above are only for simple illustration of how the algorithm works. In calculations for actual hosts, the system uses not the disk raw size (the decimal-based size stated by the disk manufacturer) but rather the disk usable size after the disks are formatted and the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

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

In the [HyperStore File System](#) mounted to each HyperStore data disk there are sub-directories that demarcate each vNode's data.

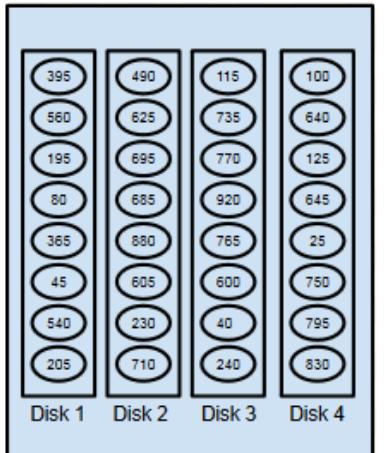
For illustration of how vNodes work to guide the distribution of data across a cluster, consider a cluster of six HyperStore hosts each of which has four disks designated for S3 object storage. Suppose that each physical host is assigned 32 tokens. And suppose for illustration that there is a simplified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

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

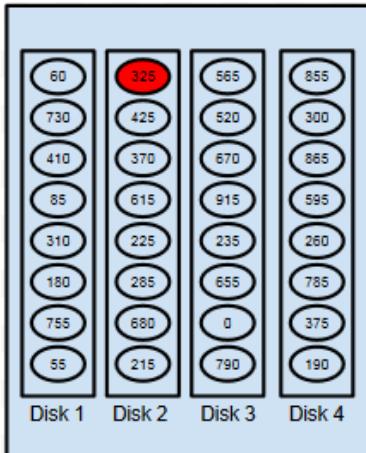


Now further suppose that you've configured your HyperStore system for 3X replication of S3 objects. And say that an S3 object is uploaded to the system and the hashing algorithm applied to the unique <buck-ename>/<objectname> combination gives us a hash value of 322 (for this simplified example; in reality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

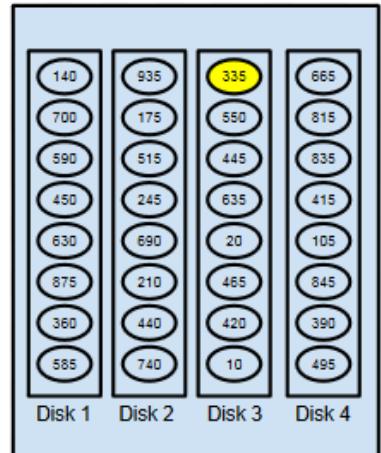
- With its object name hash value of 322, the "primary replica" of the object is stored on the vNode responsible for the token range that includes the value 322. This is the vNode assigned token 325 (highlighted in red in the diagram below) -- this vNode has responsibility for a token range spanning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



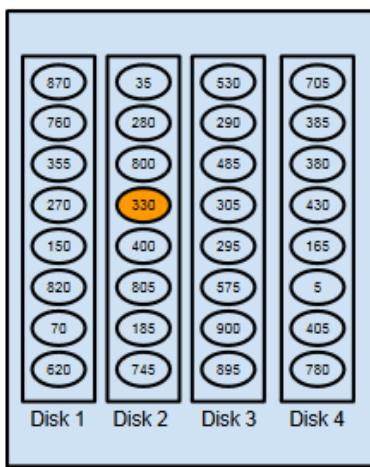
hyperstore1



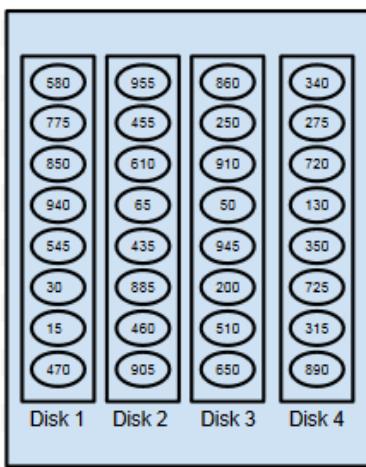
hyperstore2



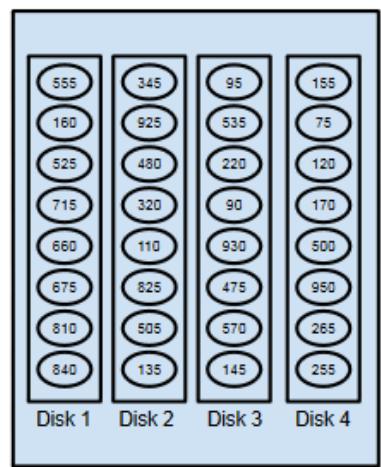
hyperstore3



hyperstore4



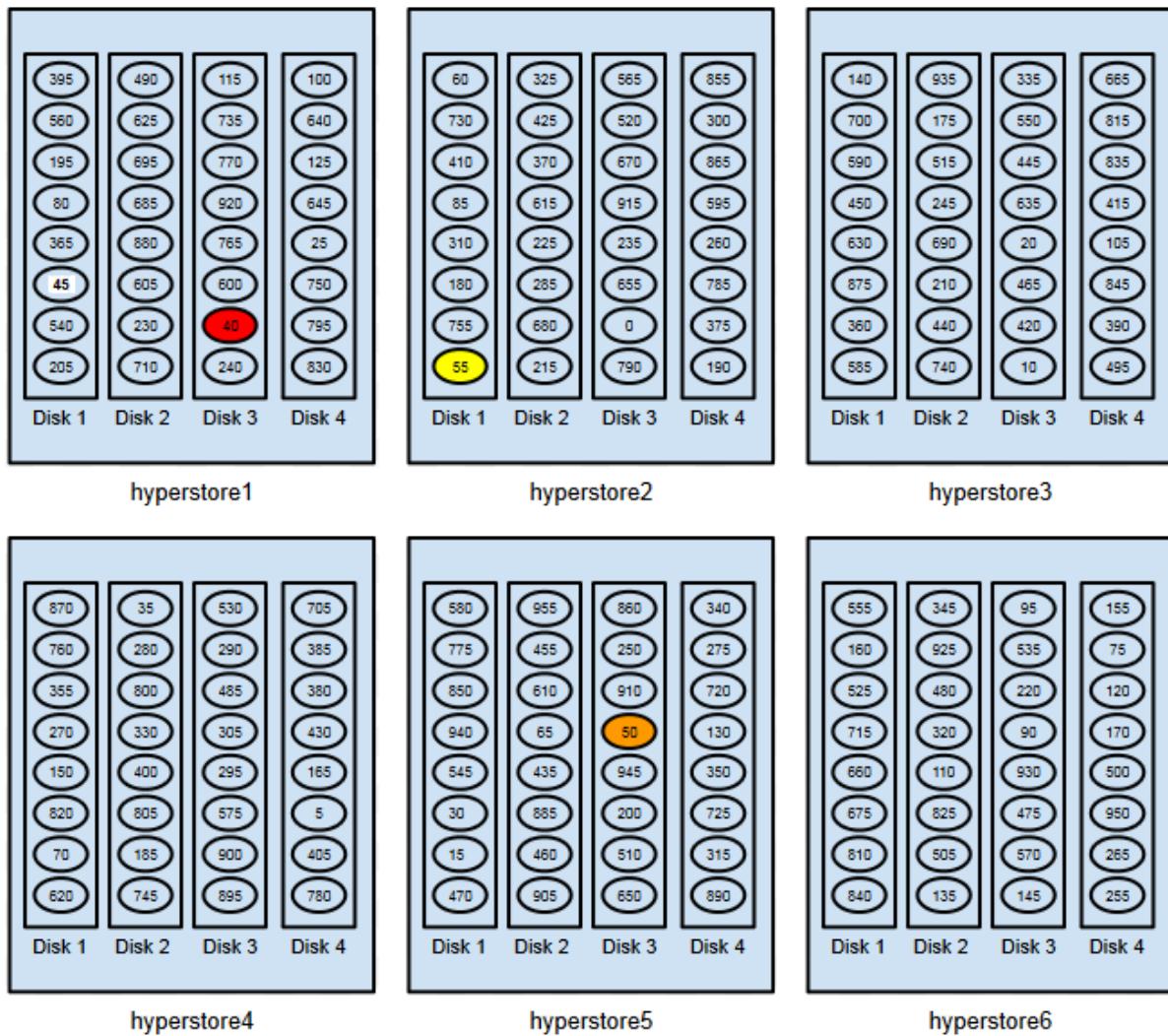
hyperstore5



hyperstore6

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

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



The Disk Perspective

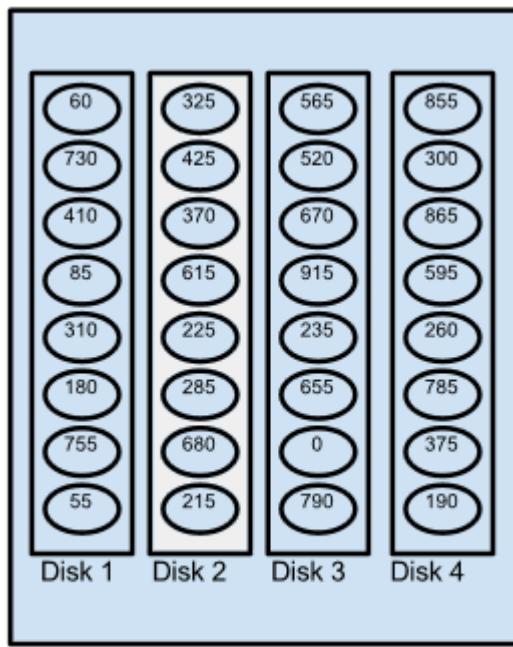
Now let's change perspective and see how things look for a particular disk from within the cluster. Recall that we've supposed a simplified token space with a total of 192 tokens (0, 5, 10, 15, and so on up to 955), randomly distributed across the cluster so that each host has 32 tokens and each host's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

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

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

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

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



hyperstore2

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

Multi-Data Center Deployments

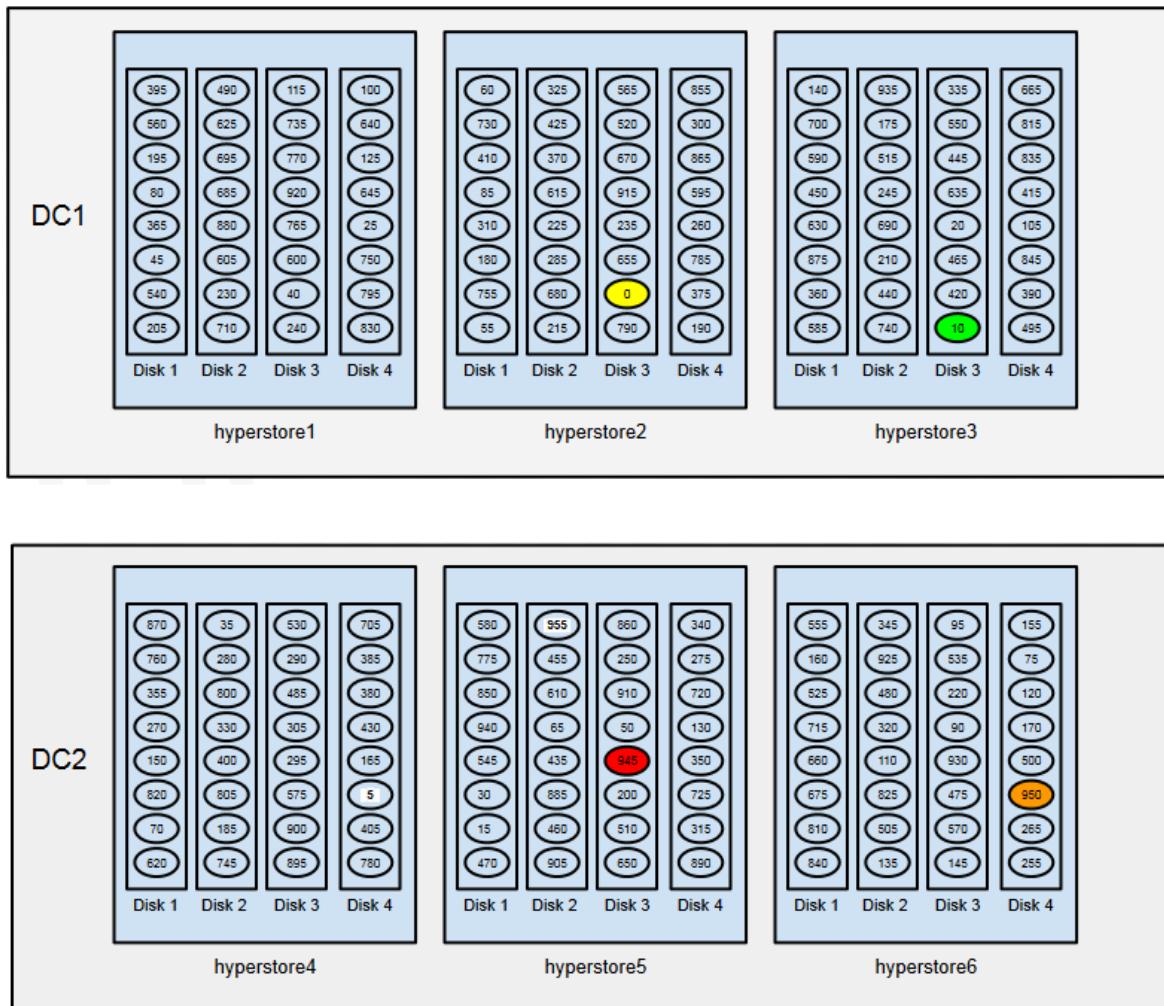
If you deploy your HyperStore cluster across multiple data centers within the same service region, your multiple data centers will be integrated in one unified token space.

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

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

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

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



Multi-Region Deployments

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

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

Erasure Coded Data

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

Cassandra Data

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

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

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

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

vNode Benefits

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

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

This page left intentionally blank

Chapter 2. Getting Started with a New HyperStore System

Once a newly installed HyperStore system is up and running, you can use the Cloudian Management Console (CMC) to take the system for a test drive. Follow these steps:

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

Since the CMC runs on all of your HyperStore nodes by default, you can use any node's IP address.

2. The connection will automatically switch over to SSL and you will get a certificate warning. Follow the prompts to add an exception for the certificate and accept it. You should then see the CMC's login screen.

The screenshot shows the Cloudian Management Console (CMC) login interface. On the left, there is a large Cloudian logo with the word "CLOUDIAN" below it. On the right, there is a "LOGIN" form with the following fields:

- Group Name: System Admin
- User ID: (empty input field)
- Password: (empty input field)
- A green "LOGIN" button at the bottom right of the form.

3. Log in with the system administrator user ID **admin** and default password **public**.
4. In the upper right corner of the CMC UI, hold your cursor over your user ID ("admin"). From the drop-down menu that displays, select **Security Credentials** to open the **Sign-In Credentials** interface. For security, change the system administrator password.

The screenshot shows the Cloudian Management Console (CMC) Sign-In Credentials interface. At the top, there is a navigation bar with links for Analytics, Buckets & Objects, Users & Groups, Cluster, Alerts, and a user profile dropdown for "Admin". The "Security Credentials" link in the dropdown is highlighted with a red circle. The main content area has two sections:

- SIGN-IN CREDENTIALS**: Fields for "USER ID" (set to "admin"), "NEW PASSWORD" (empty input field), "CURRENT PASSWORD" (empty input field), and "CONFIRM PASSWORD" (empty input field). A green "CHANGE PASSWORD" button is at the bottom right.
- S3 ACCESS CREDENTIALS**: A table with columns "CREATED", "ACCESS KEY ID", and "ACTIONS". There is one row in the table with the note "* Active Access Key" and a green "CREATE NEW KEY" button.

5. Select the **Cluster** tab, then **Storage Policies**, then **Create Storage Policy**. This opens the **Create New Policy** interface. Create and **Save** a default storage policy for your system. A storage policy is a method of storing and protecting S3 object data and object metadata. Leave the "Group Visibility" unspecified so that this policy is visible to all groups. (At a later time you can edit this policy; create additional policies; and assign a different policy to be the system default policy if you wish).

STORAGE POLICIES
[+ CREATE STORAGE POLICY](#)

CREATE NEW POLICY

Policy Name	<input style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;" type="text"/> Policy Description														
<input style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;" type="text"/>															
NUMBER OF DATACENTERS															
<input 1"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; value=" type="text"/>															
DATA DISTRIBUTION SCHEME															
<input checked="" type="radio"/> Replicas Within Single Datacenter	<input type="radio"/> EC Within Single Datacenter														
NUMBER OF REPLICAS															
<input 3"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; value=" type="text"/>															
DATACENTER ASSIGNMENT															
REGION	DATACENTER	REPLICA	LOCAL EC												
<input sfo-del-region1"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; value=" type="text"/>	<input dc1"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; value=" type="text"/>	1 of 3													
		2 of 3	disable												
		3 of 3													
CONSISTENCY SETTING		GROUP VISIBILITY													
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 30%;">CONSISTENCY LEVEL</th> <th style="width: 30%;">READ</th> <th style="width: 30%;">WRITE</th> </tr> </thead> <tbody> <tr> <td>ALL</td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td>QUORUM</td> <td style="text-align: center;"><input checked="" type="checkbox"/></td> <td style="text-align: center;"><input checked="" type="checkbox"/></td> </tr> <tr> <td>ONE</td> <td style="text-align: center;"><input type="checkbox"/></td> <td></td> </tr> </tbody> </table>		CONSISTENCY LEVEL	READ	WRITE	ALL	<input type="checkbox"/>	<input type="checkbox"/>	QUORUM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ONE	<input type="checkbox"/>		<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Please select a Group </div> <div style="display: flex; justify-content: space-between; align-items: center;"> ADD </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Compression Type: <input none"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; value=" type="text"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Server-side Encryption: <input none"="" style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 5px; value=" type="text"/> </div>	
CONSISTENCY LEVEL	READ	WRITE													
ALL	<input type="checkbox"/>	<input type="checkbox"/>													
QUORUM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>													
ONE	<input type="checkbox"/>														
		SAVE CANCEL													

For detail about the available options when you configure a new storage policy see "**Add a Storage Policy**" (page 308).

6. Create a regular S3 service user account that will enable you to test the system's S3 object storage services.

42

Note The system administrator role cannot have its own S3 storage service account.

- a. Select **Users & Groups** → **Manage Groups** → **New Group** to open the **Add New Group** interface. Create a new user group.

The screenshot shows the 'ADD NEW GROUP' form. It includes fields for 'Group Name' (mandatory), 'Group Description', 'Rating Plan' (a dropdown menu with 'Please select a Rating Plan'), and two checkboxes: 'Enable LDAP Authentication' and 'Enable S3 endpoints display filter'. At the bottom are 'CANCEL' and 'SAVE' buttons. A checked checkbox labeled 'Active Group' is located at the top right of the form area.

- b. Select **Users & Groups** → **Manage Users** → **New User** to open the **Add New User** interface. Create a new regular user, assigned to the user group that you created in the previous step. Make a note of the group name, user ID, and the password that you assign to the user so that you will be able to log in as that user.

The screenshot shows the 'ADD NEW USER' form. It includes fields for 'User ID' (mandatory), 'User Type' (set to 'User'), 'Group Name' (a dropdown menu with 'Please select a Group'), 'Password' (mandatory), 'Confirm Password' (mandatory), and a 'More' link. At the bottom are 'CANCEL' and 'SAVE' buttons. A checked checkbox labeled 'Active User' is located at the top right of the form area.

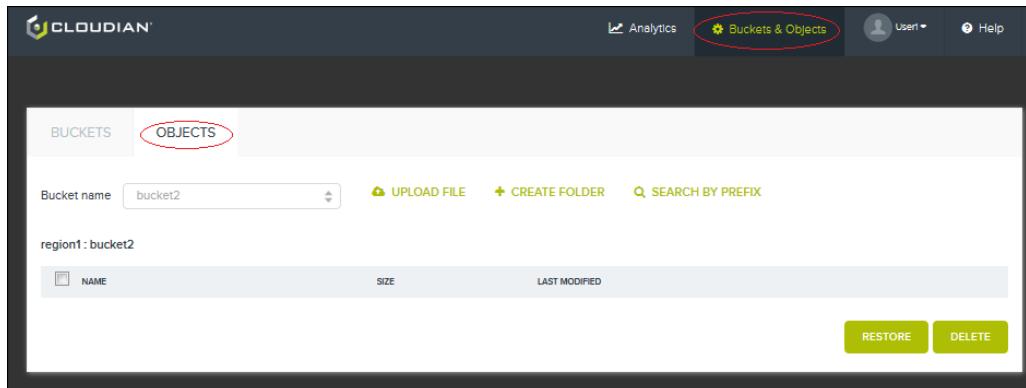
7. Log out of the CMC, and then log back in as the new user that you created. The CMC now displays only the functions that are available to regular service users, including the **Buckets & Objects** interface.

The screenshot shows the 'BUCKETS' interface with an 'ADD NEW BUCKET' form. It includes fields for 'Bucket Name' (mandatory), 'Region' (set to 'region1'), 'Storage Policy' (set to '*HSFSPolicy_region1'), and a 'Storage Policy Description' field containing 'HSFSPolicy_region1'. At the bottom are 'CANCEL' and 'CREATE' buttons. A plus sign icon labeled '+ ADD NEW BUCKET' is located at the top right of the form area.

Note As an alternative to logging out of the CMC, you can remain logged in as an administrator and use the **Manage Users** page to retrieve the user that you just created. Then click the "View User Data" link for that user. The **Buckets & Objects** interface will display for you as if you were that user.

8. Experiment with the CMC **Buckets & Objects** interface. For example:

- a. Add a new storage bucket by entering a bucket name and clicking **Create**. (You must create a bucket before you can upload any objects.)
- b. Use the **Objects** tab to create a folder in the bucket by entering a folder name and clicking **Create Folder**.



- c. Open the folder then upload a file into it by using the **Upload File** function.
- d. Verify that the uploaded file appears in the folder contents list, then verify that you can download the file by clicking on the file name.

Chapter 3. Upgrading Your HyperStore Software Version

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Preparing to Upgrade Your System" (page 45)**
- **"Upgrading Your System" (page 48)**
- **"Verifying Your System Upgrade" (page 50)**

The instructions that follow are for upgrading to HyperStore version 7.2 from **HyperStore version 7.1 or newer**. This upgrade procedure does not require S3 service interruption.

IMPORTANT:

- * To upgrade to HyperStore version 7.2, **all of your HyperStore host machines must be running RHEL 7.x or CentOS 7.x**. Upgrading to HyperStore version 7.2 is not supported if any of your hosts are running RHEL 6.x or CentOS 6.x.
- * If you are currently on a HyperStore version older than 7.1, do not use the procedure described here. Instead contact Cloudian Support to request instructions for your particular upgrade path.
- * If you are using Xen, Amazon EC2, or Logical Volume Manager (LVM), contact Cloudian Support before upgrading your system.

3.1. Preparing to Upgrade Your System

Before upgrading your HyperStore system, log into the CMC and under the **Cluster** tab use the system status display pages to make sure that your system is in a fully healthy, unrestricted state and that no major operations are in progress.

- In the **Operation Status** page, check whether there are any operations currently in progress in any of your service regions.

The screenshot shows the Cloudian CMC interface with the 'Operation Status' tab selected. The 'OPERATION LIST' table displays two entries:

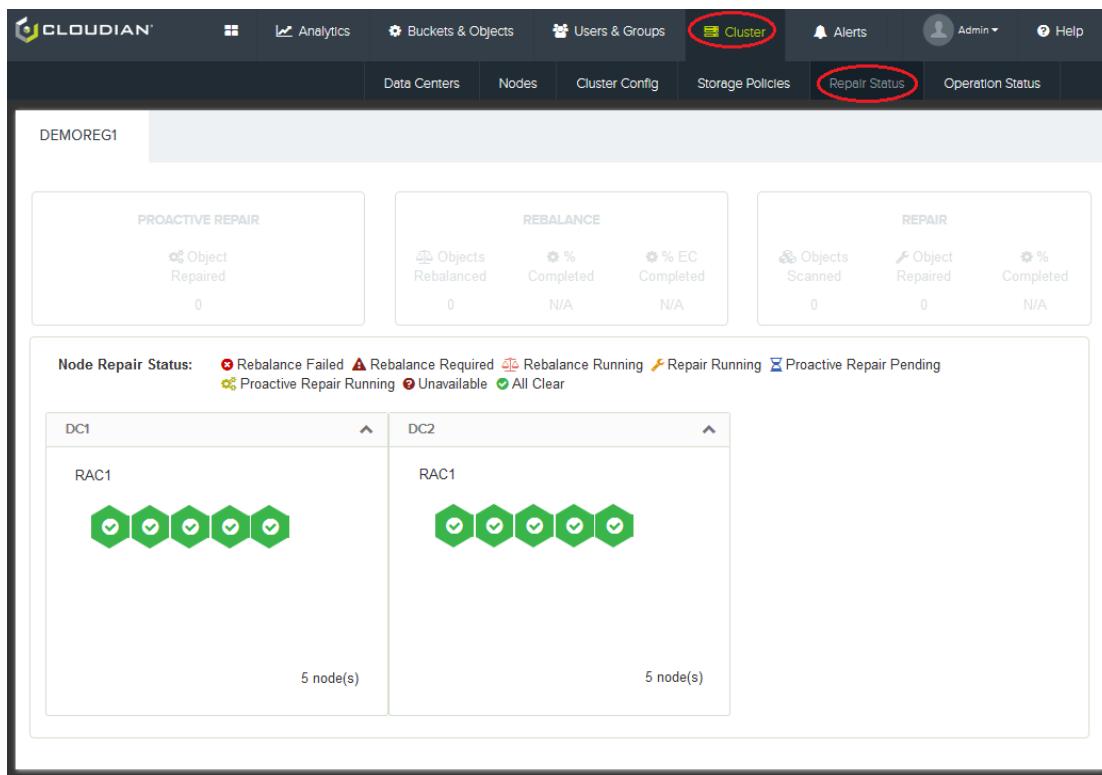
OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE
cleanup	hyperstore-demo1	✓ completed	100%	Apr-09-2018 16:18	Apr-09-2018 17:19
repair	hyperstore-demo2	✓ completed	100%	Apr-08-2018 05:05	Apr-08-2018 05:14

Below the table, it says "Showing 1 to 2 of 2 entries".

- If any **repair** is in progress when you launch the upgrade script, the script will fail in the pre-check stage and will not make any changes to your system. So before launching the upgrade

you must wait for any in-progress repair to complete (or alternatively you can stop the repair by using the "stop" option that is supported by the "**hsstool repair**" (page 647) and "**hsstool repairec**" (page 658) commands).

- If any **cleanup**, **decommission**, or **rebalance** operation is in progress, Cloudian recommends that you wait until the operation completes before you perform the upgrade (or in the case of cleanup you can stop the cleanup by using the "stop" option that is supported by the "**hsstool cleanup**" (page 610) and "**hsstool cleanuppec**" (page 616) commands).
- In the **Repair Status** page, **make sure there are no proactive repairs currently in progress** in any of your service regions. Note that in-progress proactive repairs will not display in the **Operation Status** page but will display in the **Repair Status** page.



If a proactive repair is in progress, either wait until it completes or alternatively you can stop the proactive repair by using the Maintenance command [**proactiverepair**](#) with the "stop" option selected. If a proactive repair is in progress, the upgrade script will fail in the pre-check stage and will not make any changes to your system.

- In the **Data Centers** page, **make sure that all services are up and that no node is in a restricted status**. If a service is down or a node is in one of the restricted statuses noted below, the upgrade script will fail in the pre-check stage and will not make any changes to your system..

HOST	ADMIN	CASSANDRA	HYPERSTORE	REDIS MON	REDIS CRED	REDIS QOS	S3
hyperstore-demo1	✓	✓	✓		✓ •	✓	✓
hyperstore-demo3	✓	✓	✓		✓	✓ •	✓
hyperstore-demo1b	✓	✓	✓	✓	✓ •		✓
hyperstore-demo3b	✓	✓	✓				✓
hyperstore-demo5b	✓	✓	✓				✓
hyperstore-demo2	✓	✓	✓				✓
hyperstore-demo4	✓	✓	✓				✓
hyperstore-demo6	✓	✓	✓			✓	✓
hyperstore-demo2b	✓	✓	✓	✓	✓		✓
hyperstore-demo4b	✓	✓	✓				✓

- If any **services are stopped**, start them. For instructions see "[Starting and Stopping Services](#)" (page 365).
- If any node is in **Maintenance Mode**, take it out of Maintenance Mode. For instructions see "[Stopping Maintenance Mode](#)" (page 286).
- If any node is in "**stop-write**" condition, contact Cloudian Support.

Note If the **Data Centers** page indicates that **any node has alerts**, go to the "[Node Status](#)" (page 268) page and then select that node and review its alerts. Resolve any serious issues before proceeding with the upgrade.

Note The upgrade script will automatically disable the auto-repair and proactive repair features -- so that no new repairs kick off during the upgrade -- and then after the upgrade completes the script will automatically re-enable the auto-repair and proactive repair features.

3.1.1. Additional Upgrade Preparation If Your System Currently Has Failed Disks

By default the HyperStore upgrade script will abort if it detects that any disks on your HyperStore nodes are failed or disabled. If you want to perform a HyperStore version upgrade while there are failed or disabled disks in your current system, take the following preparation steps before doing the upgrade:

1. On your Puppet master node, in the staging directory for your **current** HyperStore system, open this file in a text editor:

```
CloudianInstallConfiguration.txt
```

2. In that file, change `INSTALL_SKIP_DRIVES_CHECK=false` to `INSTALL_SKIP_DRIVES_CHECK=true`. Then save and close the file.

3.1.2. Additional Upgrade Preparation If You Are Using Elasticsearch

If you have been using an Elasticsearch cluster together with HyperStore (for search or analysis of object metadata or logging data), then **before upgrading to HyperStore 7.2 you must upgrade your Elasticsearch cluster to version 6.6 or newer**. The Elasticsearch client included in HyperStore 7.2 is compatible only with Elasticsearch server version 6.6 or newer. To avoid integration errors between HyperStore and your Elasticsearch cluster, perform the Elasticsearch upgrade **before** the HyperStore upgrade.

3.2. Upgrading Your System

To perform the upgrade to HyperStore version 7.2:

1. Download or copy the HyperStore product package (`CloudianHyperStore-7.2.bin` file) into any directory on your Puppet Master node. Also copy your current Cloudian license file into that same directory.

Note: Your current license file is located under `/etc/cloudian-<your-current-version>-puppet/modules/base/layout/files` and ends with suffix `.lic`. Copy this file to the directory in which you've placed the new HyperStore product package.

2. In that directory run the commands below to unpack the HyperStore package:

```
[any-directory]# chmod +x CloudianHyperStore-7.2.bin  
[any-directory]# ./CloudianHyperStore-7.2.bin <license-file-name>
```

This creates a **new** installation staging directory named `/opt/cloudian-staging/7.2`, and extracts the HyperStore package contents into the staging directory.

Note The new installation staging directory must persist for the life of your HyperStore system. Do not delete the new staging directory after completing the upgrade.

3. Change into the new installation staging directory:

```
[any-directory]# cd /opt/cloudian-staging/7.2
```

4. In the new staging directory, launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

```

Cloudian HyperStore(R) 7.2 Installation/Configuration
-----
0 ) Run Pre-Installation checks
1 ) Install Cloudian HyperStore
2 ) Cluster Management
3 ) Upgrade From a Previous Version
4 ) Advanced Configuration Options
5 ) Uninstall Cloudian HyperStore
6 ) Help
x ) Exit

Choice: [ ]

```

- From the installer main menu enter "3" for "Upgrade from <your version number> to 7.2". Then at the prompt, confirm that you wish to continue with the automated upgrade.

The upgrade script will first check your Puppet configuration template files (*.erb files) from your existing HyperStore system to determine whether you made any customizations to those settings (changes from the default values). If you have made such configuration changes, the installer creates a text file that lists those changes -- a "diff" file -- and prompts you to review the file. Open a separate terminal instance in which you can open and review the "diff" file that the installer created. Then, to carry forward your *.erb file customization(s) you need to manually edit the setting(s) in the new HyperStore version's configuration templates (under /etc/cloudian-<new-version-#>-puppet/modules) before continuing. For example, if in your existing HyperStore system you had set a custom value for a *hyperstore-server.properties.erb* setting, edit that same setting in /etc/cloudian-7.2-puppet/modules/cloudians3/templates/hyperstore-server.properties.erb. After saving your change (and doing likewise for any of your other customizations to *.erb setting), return to the terminal instance in which you are running the upgrade, and at the installer prompt continue with the upgrade.

Note Customizations that you may have made to the configuration file *common.csv* are handled differently. The installer detects such customizations and automatically applies the same customizations to the new version's *common.csv* file, without you having to do anything.

If you do make manual edits to the new HyperStore version's *.erb template settings during the upgrade, you do not need to do a Puppet push -- just edit the configuration templates and then resume the automated upgrade process.

The automation upgrades one node at a time -- by shutting down the node, updating the packages on the node, and then restarting the node and the services on the node -- **until all nodes are upgraded**. Messages in the terminal will indicate the progress. If you have a multi-region HyperStore system, the region that hosts the Redis Credentials master node will be upgraded last.

After the upgrade successfully completes, proceed to "**Verifying Your System Upgrade**" (page 50).

Note

* Once you've started the upgrade, you cannot <ctrl>-c out of it.

* If you have initiated the upgrade through a remote terminal, and the connection between the terminal and the Puppet master node is subsequently lost, the upgrade will continue.

3.2.1. Upgrade Failure and Roll-Back

If the upgrade fails for certain nodes, those nodes are automatically rolled back to your previously existing HyperStore software version. The terminal will display basic information about the failure, and you can get more details from the *cloudian-installation.log* and the *cloudian-upgrade.log*, both of which are generated in the installation staging directory. Try to resolve the problems on the node(s) that failed to upgrade, and then run the upgrade process again (action number 3 from the installer's main menu).

The upgrade process also generates an *upgrade-logNconfig*.tgz* "S.O.S" tar file (which packages together multiple upgrade-related files) that you can provide to Cloudian Support in the event that you need assistance in resolving any upgrade problems.

3.3. Verifying Your System Upgrade

After all HyperStore nodes have been upgraded, verify that all services are running and that the HyperStore version is now 7.2:

1. After the automated upgrade completes, you should be taken back to the main menu of the HyperStore installer. The first post-upgrade step is to confirm that all your HyperStore services are up and running:
 - a. From the installer's main menu select "Cluster Management".
 - b. From the Service Management sub-menu that displays select "Manage Services".
 - c. At the "Select a service to manage:" prompt, select All Services.
 - d. At the "Enter command" prompt, type *status*.

All services on all nodes should then indicate that they are running.

2. Next, confirm that the HyperStore software version is correct:
 - a. Still on the Service Management menu, at the "Select a service to manage:" prompt select the S3 Service.
 - b. At the "Enter command" prompt, type *version*.

On all nodes the S3 version should indicate version 7.2.

After confirming the version you can exit the installer.

3. Use the CMC to check on your upgraded cluster:
 - On the **Node Advanced** page, select command type Info then execute the "repairqueue" command to **verify that auto-repair is enabled** for replica, EC, and Cassandra data. (Although you disabled auto-repair prior to doing the upgrade, the system automatically re-enables auto-repair at the end of the upgrade process).
 - On the **Manage Users** page, confirm that you can retrieve users.
 - Log out of the CMC as system admin and log back in as a regular user, and then confirm that you can successfully download and upload objects.

4. If prior to the upgrade you had made any customizations to the branding of the CMC interface, only your customized logos and customized application name will be retained after the upgrade. You will need to re-implement any changes that you had made to the browser tab title and/or the color scheme, by again following the instructions for "**Rebranding the CMC UI**" (page 356).
5. If you have been using ElasticSearch for search of HyperStore object metadata, you should have upgraded your ES cluster to version 6.6 or new before upgrading HyperStore to version 7.2 (as noted in "**Preparing to Upgrade Your System**" (page 45)). Now, after having upgraded HyperStore, run this command from any HyperStore node to verify that a sync-up of the object metadata in your ES cluster against the object metadata in HyperStore can still be performed without error:

```
/opt/cloudian/bin/elasticsearchSync all
```

You are now done with upgrading to HyperStore 7.2.

Note If you disabled the failed disk check before performing your upgrade (as described in "**Additional Upgrade Preparation If Your System Currently Has Failed Disks**" (page 48)), note that after you've completed the upgrade, in the new instance of *CloudianInstallConfiguration.txt* in the staging directory for your new HyperStore version the *INSTALL_SKIP_DRIVES_CHECK* setting is set back to its default of *false*. So the next time you upgrade your HyperStore version the check for failed drives will be executed, unless you once again disable the check by changing that setting to *true*.

This page left intentionally blank

Chapter 4. Working with HyperStore Major Features

4.1. Auto-Tiering

4.1.1. Auto-Tiering Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Tiering Destination Accounts, Credentials, and Buckets" (page 54)**
- **"Bridge Mode" (page 55)**
- **"Option to Retain a Local Copy" (page 55)**
- **"Auto-Tiering Logging" (page 55)**
- **"Auto-Tiering Licensing" (page 55)**
- **"Auto-Tiering of Encrypted Objects" (page 56)**
- **"Auto-Tiering and User-Defined Object Metadata" (page 56)**
- **"How Auto-Tiering Impacts Usage Tracking, QoS, and Billing" (page 56)**

The HyperStore system supports an "auto-tiering" feature whereby objects can be automatically moved from local HyperStore storage to a remote storage system on a defined schedule. HyperStore supports auto-tiering from a local HyperStore bucket to any of several types of destination systems:

- S3-compliant systems: Amazon S3, Amazon Glacier, Google Storage Cloud, a HyperStore region or system, or a different S3-compliant system of your choosing
- Microsoft Azure
- Spectra Logic BlackPearl

Auto-tiering is configurable on a per-bucket basis. A bucket owner activating auto-tiering on a bucket can specify:

- The auto-tiering destination system
- Whether auto-tiering applies to all objects in the bucket, or only to objects for which the full object name starts with a particular prefix
- The tiering schedule, which can be implemented in any of these ways:
 - Move objects X number of days after they're created
 - Move objects if they go X number of days without being accessed
 - Move objects on a fixed date — such as December 31, 2018
 - Move objects immediately after they're created ("Bridge Mode"). Note that with Bridge Mode, filtering by prefix is not supported.

Although there can only be one auto-tiering destination for a given HyperStore bucket, bucket owners have the option of configuring different auto-tiering schedules for different sets of objects within the bucket, based on the object name prefix.

For more information about configuring auto-tiering in the system and on individual buckets, see "["Setting Up Auto-Tiering"](#) (page 56).

Note **Auto-tiering restrictions based on destination type:**

- * By default the largest object size that can be auto-tiered to Amazon, Google, or other S3-compliant destinations is 100GB. If you want to tier objects larger than this, consult with Cloudian Support. This 100GB limit does not apply to tiering to Azure or Spectra BlackPearl.
- * Tiering to Azure or Spectra BlackPearl is not supported for source buckets that have versioning enabled or that have had versioning enabled in the past.
- * When auto-tiering to Spectra BlackPearl is used for a bucket, objects in the bucket will not be auto-tiered unless they are larger than 5MB. Objects 5MB or smaller will remain in HyperStore. To change this limit, consult with Cloudian Support.

4.1.1.1. Tiering Destination Accounts, Credentials, and Buckets

Auto-tiering to Amazon or another destination system requires that there be an existing account that the HyperStore system can access at the destination system. There are two options for account access:

- Bucket owners can supply their own destination account credentials, on a per-bucket basis. In this way each bucket owner tiers to his or her own account at the destination system. This is the default method for auto-tiering.
- You can supply system default tiering credentials. This is the appropriate approach if all users will be tiering to the same account at the same tiering destination system. For more information see "["Setting Up Auto-Tiering"](#) (page 56).

HyperStore encrypts supplied tiering account credentials and stores them in the Redis Credentials database, where they can be accessed by the system in order to implement auto-tiering operations.

Auto-tiering moves objects from a local HyperStore source bucket into a **tiering bucket** at the destination system. The source bucket owner when configuring auto-tiering can specify as the tiering bucket a bucket that already exists in the destination system, or the source bucket owner can have HyperStore create a tiering bucket in the destination system. If having HyperStore create a tiering bucket, the source bucket owner can choose a tiering bucket name or have HyperStore automatically name the tiering bucket. When HyperStore automatically names the tiering bucket it uses this format:

<origin-bucket-name-truncated-to-34-characters>-<random-string>

The HyperStore system appends a 28-character random string to the origin bucket name to ensure that the resulting destination bucket name is unique within the destination system. If the origin bucket name exceeds 34 characters, in the destination bucket name the origin bucket name segment will be truncated to 34 characters.

After objects have been auto-tiered to the destination system they can be accessed directly through that system's interfaces (such as the Amazon S3 Console), by persons having the applicable credentials. Auto-tiered objects can also be accessed indirectly through the local HyperStore system interfaces. Tiered object access is described in more detail in "["Accessing Auto-Tiered Objects"](#) (page 61).

Note In the case of auto-tiering to Amazon Glacier, the HyperStore system creates a bucket in Amazon S3 and configures that remote bucket for immediate transitioning to Glacier. Objects are then auto-tiered from HyperStore to Amazon S3, where they are immediately subject to Amazon's automated mechanism for transitioning objects to Glacier.

4.1.1.2. Bridge Mode

The HyperStore auto-tiering feature supports a "Bridge Mode" whereby objects can be transitioned to a destination system immediately after they are uploaded to the HyperStore source bucket. As soon as such objects are successfully transitioned to the destination system they are flagged for deletion from the local HyperStore system (the actual deletion will be executed afterwards, by a cron job that runs hourly). After deletion of the local copy of the objects, only object metadata remains (see "**How Auto-Tiering Impacts Usage Tracking, QoS, and Billing**" (page 56) for detail).

If the initial attempt to move an object to the destination system fails with a temporary error, the system will retry once every six hours until either the object is successfully moved or a permanent error is encountered (an example permanent error would be if, outside of HyperStore, someone had deleted the tiering destination bucket). The retries are triggered by a [system cron job](#). The local copy will not be deleted until the object is successfully moved to the destination system, as indicated by the destination system returning a success status.

Users have the option of choosing Bridge Mode when they configure auto-tiering rules for a bucket.

Note Bridge mode is not supported for tiering to Amazon Glacier or Spectra BlackPearl.

4.1.1.3. Option to Retain a Local Copy

By default the system deletes the local copy of an object as soon as the object is successfully transitioned to the tiering destination. However, HyperStore supports an option to retain a local copy of objects that have been auto-tiered, for a specified period of time. Locally retained objects will continue to be protected by the storage policy associated with the bucket in which the objects reside (whether replication or erasure coding). After the specified retention period the system will automatically delete the local copy.

Users have the option of specifying a local retention period when they configure auto-tiering rules for a bucket.

4.1.1.4. Auto-Tiering Logging

As the HyperStore system auto-tiers objects from local storage to the tiering destination, it records information about the tiering transactions (including source bucket name, object name, and destination bucket name) in the tiering request log [cloudian-tiering-request-info.log](#). For regular auto-tiering that occurs on a defined schedule -- such as tiering objects after they're a certain number of days old -- this request logging occurs on the same node that is running the HyperStore system cron jobs. To see which of your nodes is running the cron jobs, go to the CMC's [Cluster Information](#) page.

In the special case of "Bridge Mode" auto-tiering, whichever S3 node processes the upload of a given object into the source bucket also initiates the immediate auto-tiering of the object to the destination system, and the tiering request log entry for that is written locally on that node.

4.1.1.5. Auto-Tiering Licensing

Your HyperStore license may impose a limit on how much tiered data you can have stored in external systems other than HyperStore. For details see "**Licensed Maximum Tiered Storage Usage**" (page 9).

4.1.1.6. Auto-Tiering of Encrypted Objects

For information about how auto-tiering works together with the server-side encryption feature, see "[Encryption and Auto-Tiering](#)" (page 118). As detailed in that section, encrypted objects may or may not be eligible for auto-tiering depending on the encryption method and the particular tiering destination.

4.1.1.7. Auto-Tiering and User-Defined Object Metadata

For information about how auto-tiering impacts user-defined object metadata, see "[Object Metadata, Cross-Region Replication, and Auto-Tiering](#)" (page 105).

4.1.1.8. How Auto-Tiering Impacts Usage Tracking, QoS, and Billing

When an object is auto-tiered to a destination system and deleted from local storage the size of the tiered object is subtracted from the bucket owner's HyperStore [usage count](#) for Storage Bytes. At the same time, a reference to the auto-tiered object is created and the size of this reference — 8KB, regardless of the transitioned object size — is added to the Storage Bytes count. Meanwhile, auto-tiering does not impact the Storage Objects count. An object counts toward the number of Storage Objects regardless of whether or not it is auto-tiered.

For example, if a 1MB object has been auto-tiered to Amazon then that object would count as:

- 8KB toward the bucket owner's HyperStore count for Storage Bytes.
- 1 toward the bucket owner's HyperStore count for Storage Objects.

If an auto-tiered object is temporarily restored to HyperStore storage, then while the object is restored the object's size is added back to the Storage Bytes count and the 8KB for the reference is subtracted from the count. After the restore interval ends and the restored object instance is automatically deleted, the object size is once again subtracted from Storage Bytes and the 8KB for the reference is added back. For more on temporary restoration of tiered objects see "[Accessing Auto-Tiered Objects](#)" (page 61).

Auto-tiering does not impact HyperStore usage counts for Bytes-IN or Bytes-OUT.

Note If users select the [Retain Local Copy](#) option when configuring their buckets for auto-tiering, objects that are temporarily retained in HyperStore after they've been auto-tiered will continue to count toward the local Storage Bytes count until the local copy is deleted.

HyperStore's Quality of Service (QoS) and billing features make use of Storage Bytes and Storage Object counts. So the impact of auto-tiering on QoS and billing is as described above. For example in the case of the QoS restrictions applied to users, if a bucket owner's 1MB object has been auto-tiered to Amazon, then that object counts as 8KB toward that user's QoS limit for Storage Bytes; and as 1 toward that user's QoS limit for Storage Objects.

For more information on the QoS and billing features, see "[Quality of Service \(QoS\) Feature Overview](#)" (page 110) and "[Billing Feature Overview](#)" (page 63).

4.1.2. Setting Up Auto-Tiering

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Enable and Configure the Auto-Tiering Feature in the CMC" (page 57)**
- **"Configure Auto-Tiering Rules for Individual Buckets (CMC)" (page 60)**
- **"Configure Auto-Tiering Rules for Individual Buckets (S3 API and Admin API)" (page 60)**
- **"Tiering Statistics for Spectra BlackPearl" (page 61)**

To use the CMC to set up buckets for auto-tiering, follow the high-level instructions below. First you will enable and configure the auto-tiering feature in the CMC, and then you or your users can use the CMC to configure individual buckets for auto-tiering. (If you want to use an S3 client application other than the CMC, jump down to **"Configure Auto-Tiering Rules for Individual Buckets (S3 API and Admin API)" (page 60)**).

4.1.2.1. Enable and Configure the Auto-Tiering Feature in the CMC

Note None of the settings described below is applicable to using a third party S3 client application to invoke the HyperStore S3 Service's auto-tiering feature. These settings are applicable only to using the auto-tiering feature through the CMC.

By default auto-tiering functionality is **disabled** in the CMC, such that CMC users when configuring bucket life-cycle properties will not see an option for auto-tiering. Do the following to enable and configure the auto-tiering feature so that CMC users can apply auto-tiering to their buckets:

1. In the CMC's [Configuration Settings](#) page, open the **Auto-Tiering** panel.
2. Set "Enable Auto-Tiering" to Enabled.
3. Configure how you want auto-tiering options to display to CMC users as they configure their buckets for auto-tiering. The system supports three different approaches:
 - If you want all CMC users to auto-tier to a **single system-default tiering destination account for which you are providing the account security credentials**, set "Enable Per Bucket Credentials" to Disabled and then enter the "Default Tiering URL" and the account security credentials.
 - If you want CMC users to be able to choose from a **pre-configured list of tiering destinations** (AWS S3, AWS Glacier, Google, and Azure by default) and no other destinations, leave "Enable Per Bucket Credentials" at Enabled (the default) and leave "Enable Custom Endpoint" at Disabled (the default). With this approach users provide their own account security credentials for the tiering destination. You can edit the list of tiering destinations that will display for users, and the endpoints for those destinations, as described in **"Configure Tiering Destinations" (page 58)** below.
 - If you want CMC users to be able to choose from a **pre-configured list of tiering destinations and also give users the option to specify a custom S3 tiering endpoint**, leave "Enable Per Bucket Credentials" at Enabled (the default) and set "Enable Custom Endpoint" to Enabled. With this approach users provide their own account security credentials for the tiering destination. If users specify a custom tiering endpoint, it must be an S3-compliant system and it cannot be a Glacier, Azure, or Spectra BlackPearl endpoint.
4. After finishing your edits in the **Configuration Settings** page, click **Save** at the bottom of the page to save your changes. These changes are applied dynamically and no service restart is required.

Configure Tiering Destinations

Unless you configure the system so that all users tier to the same one default tiering endpoint (as described above in Step 3's first bullet point), users when they configure auto-tiering for their buckets in the CMC will be able to choose from a list of several common tiering destinations. By default the destinations are AWS S3, AWS Glacier, Google Cloud Storage, and Azure; and by default the endpoints for those destinations are as follows:

Destination	Default Endpoint
AWS S3	https://s3.amazonaws.com
AWS Glacier	https://s3.amazonaws.com
Google Cloud Storage	https://storage.googleapis.com
Azure	https://blob.core.windows.net

Note If your original HyperStore version was older than 7.1.4, then after upgrade to 7.1.4 or later the default list of tiering destinations will also include Spectra BlackPearl with endpoint <https://b-plab.spectralogic.com>.

If you want to change this list -- by changing the endpoint for any of the destinations above, or by adding or removing destinations -- you can do so by editing the "**"cmc_bucket_tiering_default_destination_list"** (page 485) setting in [common.csv](#). The setting is formatted as a quote-enclosed list, with comma-separation between destination attributes and vertical bar separation between destinations, like this:

```
"<name>,<endpoint>,<protocol>|<name>,<endpoint>,<protocol>|..."
```

This can be multiple destinations (as it is by default), or you can edit the setting to have just one destination in the "list" if you want your users to only use that one destination.

The **<name>** will display in the CMC interface that bucket owners use to configure auto-tiering, as the auto-tiering destination name. The **<protocol>** must be one of the following:

- s3
- glacier
- azure
- spectra

If you wish you can include multiple destinations of the same type, if those destinations have different endpoints. For example, "Spectra 1,<endpoint1>,spectra|Spectra 2,<endpoint2>,spectra". Each such destination will then appear in the CMC interface for users configuring their buckets for auto-tiering.

If you make any changes to this setting, push your changes to the cluster and restart the CMC. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

Note About Tiering to a Different HyperStore Region or System

In the CMC configuration set-up described in "**Enable and Configure the Auto-Tiering Feature in the CMC**" (page 57) (above), there are two ways in which your users can use a HyperStore service region as the tiering destination:

- You configure the system so that all users tier to a system default tiering destination, and you define that default tiering URL to be a HyperStore service region endpoint.
- You configure the system so that users can enter a user-defined endpoint, and then individual users can specify a HyperStore service region as their user-defined tiering endpoint.

The HyperStore region that's being used as the tiering destination could be a different region in the same HyperStore system (for example users are tiering from Region1 to Region2 within the same HyperStore system); or it could be a service region within a different HyperStore system altogether (users are tiering from HyperStore system A to a region in HyperStore system B). For background on the distinction between a HyperStore region and a HyperStore system, see "**System Levels**" (page 19).

If your users will be tiering to a region in a **different HyperStore system**, then for tiering to that region to work "out of the box" the endpoint for that region must be specified in this format:

`s3-<region>.<domain>`

For example:

`s3-boston.company.com`

where "boston" is the actual region name in the destination HyperStore system's own system configuration.

If the endpoint for the external HyperStore system is in any format other than the above -- if, for example, the endpoint is simply `boston.company.com` rather than `s3-boston.company.com` -- then for tiering to work you must first make the following configuration change in your own HyperStore system (the source system from which the tiering will originate):

1. On the Puppet Master node, open the following file in a text editor:

```
/etc/cloudian-<version>-puppet/modules/cloudians3/templates/
tiering-regions.xml.erb
```

2. Copy the sample "Region" block at the top of the `tiering-regions.xml.erb` file and paste it toward the end of the file after the existing "Region" blocks (but before the closing "</XML>" tag that's at the very end of the file).
3. Edit the block as follows:

- Use the "Name" element to specify the region name of destination system region that you will tier to. Enter the region name exactly as it is defined in the destination HyperStore system.
- Leave the "ServiceName", "Http", and "Https" elements at their default values.
- Use the "Hostname" element to specify the service endpoint that you will tier to.

For example, if the region name is "boston" and the service endpoint is "boston.company.com", then your edited Region block would look like this:

```
<Region>
  <Name>boston</Name>
  <Endpoint>
    <ServiceName>s3</ServiceName>
    <Http>true</Http>
    <Https>true</Https>
    <Hostname>boston.company.com</Hostname>
  </Endpoint>
</Region>
```

Note Make sure that the service endpoint that you will tier to is resolvable in your DNS system.

Note Do not have two instances of "s3" in the endpoint, like `s3-tokyo.s3.enterprise.com`. This may cause auto-tiering errors (and cross-region replication errors, if you use the cross-region replication feature).

4. Save your change and close the `tiering-regions.xml.erb` file.
5. Push your changes to the cluster and restart the S3 Service. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.1.2.2. Configure Auto-Tiering Rules for Individual Buckets (CMC)

HyperStore service users can configure auto-tiering for their own buckets through the CMC's [Bucket Properties](#) page. Here users can select the tiering destination (within the limits of the system configuration as described in the preceding section), the tiering schedule (including, if desired, having different schedules for different sets of objects based on object name prefix), and whether or not to temporarily retain a local copy of tiered objects.

Alternatively, as a system administrator you can set auto-tiering for a user's bucket by retrieving the user in the [Manage Users](#) page and then clicking "View User Data" to open a [Bucket Properties](#) page for that user's data.

With either approach, the bucket first must be created in the usual way, and then the bucket can be configured for auto-tiering.

For details, see "**Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration**" (page 197)

Note Auto-tiering cannot be enabled for a bucket that has an underscore in its name. For this reason it's best not to use underscores when naming buckets in HyperStore.

4.1.2.3. Configure Auto-Tiering Rules for Individual Buckets (S3 API and Admin API)

To configure auto-tiering rules on a bucket by using the S3 API, your S3 client application will need to use HyperStore extensions to the S3 API method `PUT Bucket lifecycle`. The extensions take the form of request headers to specify the bucket's auto-tiering destination and mode (`x-gmt-tieringinfo`), whether to base auto-tiering timing on object creation time or last access time (`x-gmt-compare`), and whether to retain a local copy of the object after auto-tiering occurs (`x-gmt-post-tier-copy`). For details about these API extensions see [PUT Bucket lifecycle](#).

If you plan to configure auto-tiering on a bucket by calling the `PUT Bucket lifecycle`S3 API method, you will **first need to use the HyperStore Admin API to store into the system the tiering destination account security credentials** that HyperStore should use when auto-tiering objects from that bucket. For example, if you want to use the S3 API method `PUT Bucket lifecycle` to configure HyperStore source bucket "my-bucket" to auto-tier to AWS S3, you (or your application) must first use the HyperStore Admin API to post the security credentials that HyperStore should use when accessing AWS S3 on behalf of source bucket "my-bucket". The same requirement applies to other destination types such as Spectra BlackPearl. For details about the relevant Admin API methods, see "**tiering**" (page 814).

Also, if you want to support auto-tiering to an external HyperStore system -- not a different region within the same HyperStore system but rather a different HyperStore system altogether -- see "**Note About Tiering to a Different HyperStore Region or System**" (page 58), in regard to the format requirements for the tiering endpoint.

4.1.2.4. Tiering Statistics for Spectra BlackPearl

If your HyperStore system is using auto-tiering to Spectra BlackPearl, you can run the following command on the HyperStore cron job master node to retrieve tiering statistics.

```
curl http://localhost:80/.system/stats/tiering/spectra
```

For example:

```
[root@cloudian-node1 ~]# curl http://localhost:80/.system/stats/tiering/spectra
{"totalInQueue":0,"totalTiered":51,"tieringFail":0,"restored":1,"restoreFail":0,
"bytesTiered":629145600,"bytesRestored":1073741824}
```

The returned statistics are:

- Number of objects in queue waiting to be tiered to Spectra BlackPearl
- Number of objects tiered to Spectra BlackPearl
- Number of objects for which tiering to Spectra BlackPearl failed
- Number of tiered objects restored to HyperStore from Spectra BlackPearl
- Number of objects for which restoring to HyperStore from Spectra BlackPearl failed
- Total number of bytes tiered to Spectra BlackPearl
- Total number of bytes restored to HyperStore from Spectra BlackPearl

Note If you're not sure which node is your cron job node, you can check this in the CMC's [Cluster Information](#) page.

4.1.3. Accessing Auto-Tiered Objects

After objects have been auto-tiered to a destination system, there are two ways to access those objects:

- Indirect access through HyperStore
- Direct access through the destination system

Indirect Access Through HyperStore

When you use the CMC's [Objects](#) interface to view the contents list for a HyperStore storage bucket, objects that have been auto-tiered appear in the list and are marked with a [special icon](#). Your options for retrieving such objects through the CMC depend on how auto-tiering was configured for the bucket.

First, for any auto-tiered object (regardless of bucket configuration or tiering destination), the object can be retrieved by temporarily restoring a copy of the object into the local bucket. The CMC **Buckets & Objects** interface supports [temporarily restoring auto-tiered objects](#), for a length of time that you can specify.

Restoration of auto-tiered objects does not happen instantly. For example, for an object in Amazon S3, it can take up to six hours before a copy of the object is restored to HyperStore storage. For an object in Glacier it can be up nine hours, factoring in the time it takes for an object to be restored from Glacier to Amazon S3, before being restored to HyperStore. In the interim, the object is marked with an icon that indicates that the object is in the process of being restored. During this stage you cannot download the object.

After a copy of an object has been restored, the icon next to the object name changes again and you can then download the object through the **Buckets & Objects** interface in the usual way.

As a second option for retrieving auto-tiered objects, some objects may be directly downloadable through the **Buckets & Objects** interface without any need for first restoring the objects. This is supported only if both of the following are true:

- The tiered objects are in Amazon S3, Google Storage Cloud, or Azure (not Glacier or Spectra Black-Pearl)
- The bucket's auto-tiering is configured to support Streaming of auto-tiered objects.

If you're uncertain regarding whether an auto-tiered object meets these requirements, you can try directly downloading the auto-tiered object by clicking on its name. If direct download is not supported for the object, a response message will indicate that you need to temporarily restore a local copy of the object rather than directly downloading it.

If you want to **delete** an object that has been auto-tiered, you can do so by deleting the object through the **Buckets & Objects** interface. You do not need to restore the object first. When the HyperStore system is deleting an auto-tiered object, it first triggers the deletion of the object from the destination system, and then after that succeeds it deletes the [local reference to the object](#).

Note As an alternative to accessing auto-tiered objects through the CMC, you can use a third party S3 client application to submit [POST Object restore](#) requests (for any auto-tiered objects) or [GET Object](#) requests (for auto-tiered objects that support streaming) to the HyperStore system's S3 Service.

Direct Access Through the Destination System

Objects auto-tiered to a destination system can be accessed directly through the destination system's standard interfaces, such as the AWS Management Console for objects that have been tiered to AWS S3.

For example, if a bucket owner supplied her own AWS credentials when configuring her HyperStore bucket for auto-tiering to AWS S3, she can log into her AWS account and see the HyperStore auto-tiering destination bucket (either named as she had specified or automatically named by HyperStore -- see "["Tiering Destination Accounts, Credentials, and Buckets"](#)" (page 54) for detail). After objects have been auto-tiered from HyperStore to her AWS destination bucket, she can view the bucket content list and retrieve individual tiered objects directly through AWS.

In the case of auto-tiering from one HyperStore region to another region in the same HyperStore system, the tiered objects are accessible through the CMC's **Buckets & Objects** interface, by selecting the destination region.

IMPORTANT: Users should **not overwrite or delete tiered objects directly through the destination system's interfaces**. Doing so will cause a discrepancy between the local metadata in HyperStore and the actual data in the destination bucket. If users want to overwrite or delete tiered objects they should do so through HyperStore interfaces (such as the CMC or an S3 application accessing the HyperStore S3 Service). In the case of auto-tiering from one HyperStore region to another HyperStore region, any overwriting or deleting of objects should be done through the source bucket not the destination bucket.

4.2. Billing

4.2.1. Billing Feature Overview

The HyperStore system maintains comprehensive [service usage data](#) for each group and each user in the system. This usage data, which is [protected by replication](#), serves as the foundation for HyperStore service billing functionality.

The system provides you the ability to create rating plans that specify charges for the various types of service usage activity, and to assign each group and each user a rating plan. You can then generate bills for a user or for a whole user group, for a selected service period. The CMC has a function for displaying a single user's bill report in a browser, but in the more typical use case you will use the HyperStore Admin API to generate user or group billing data that can be ingested a third party billing application.

Cloudian HyperStore also allows for special treatment of designated source IP addresses, so that the billing mechanism does not apply any data transfer charges for data coming from or going to these "whitelisted" domains.

Note For information on how auto-tiering impacts billing calculations, see "[How Auto-Tiering Impacts Usage Tracking, QoS, and Billing](#)" (page 56).

4.2.1.1. Retention of Usage Data Used for Billing

Billing calculation is derived from hourly rollup usage data. The retention period for hourly rollup usage data is configured by *mts.properties.erb*: "[reports.rolluphour.ttl](#)" (page 511). **The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.**

4.2.1.2. Retrieving Bucket Tags

If your billing scheme makes use of bucket tags (as created by the S3 API method [PUT Bucket tagging](#)): The HyperStore Admin API supports a method for retrieving all the bucket tags for all users in a specified group. Because it is implemented through the Admin API, that method does not require the users' S3 access credentials. For more information see [GET /buckettags/gettags](#).

4.2.2. Creating Rating Plans for Billing

The HyperStore system supports the creation of billing rating plans in which charges can be specified for each of several different service activity types. The system supports rating plans that charge:

- Per GB of data in storage (based on a calculated average storage volume for the billing period)
- Per GB of data uploaded
- Per GB of data downloaded
- Per 10,000 HTTP GET or HEAD requests
- Per 10,000 HTTP PUT or POST requests
- Per 10,000 HTTP DELETE requests

A user's bill can then be calculated by applying the user's assigned rating plan to the user's activity levels for each of these activity types, and adding together the charges for each activity type to get a total charge for the billing period.

You can create multiple, named rating plans, each of which applies different charges to the various service activity types. Once you've created rating plans, those plans are then available for you to [assign to users](#).

For example, you can create higher-priced and lower-priced rating plans and then assign different plans to different users based on the users' quality of service terms.

IMPORTANT: If you want to bill for data upload or download volume, or for HTTP request volume, you must enable the "Track/Report Usage for Request Rates and Data Transfer Rates" setting in the CMC's Configuration Settings page, Usage Tracking section. By default this setting is disabled and the system does not maintain per-user HTTP request counts and data transfer byte counts.

You can create rating plans either through the CMC or through the HyperStore Admin API. The system also comes equipped with an editable default rating plan.

Creating Rating Plans for Billing (CMC)

To create rating plans through the CMC, use the [Rating Plan](#) page. For each plan you create, you can select a currency and then specify the charges to apply per each of the chargeable service activity types (per GB of stored data, per GB of data uploaded, and so on). For each rating plan, the interface supports the creation of single-tier or multi-tier pricing schemes for each chargeable activity.

The [Rating Plan](#) page also supports viewing and editing existing plans, including the [default rating plan](#) that comes with the HyperStore system.

Creating Rating Plans for Billing (Admin API)

To create a rating plan through the Admin API, use the [PUT /ratingPlan](#) method.

The Admin API also supports:

- Retrieving a list of existing rating plans: [GET /ratingPlan/list](#)
- Retrieving a rating plan: [GET /ratingPlan](#)
- Editing a rating plan: [POST /ratingPlan](#)
- Deleting a rating plan: [DELETE /ratingPlan](#)

Example of a Rating Plan Applied to Calculate a User's Monthly Bill

This example walks through a hypothetical rating plan and how it would apply to a user's service usage for a month.

Storage Charges

- Types: One type only. The average number of GBs of data stored for the month.
- Example:
 - Unit: Dollars per GB-months.
 - Pricing: From 0-1 TB at \$0.14 per GB-month, 1-10 TB at \$0.12 per GB-month, 10+ TB at \$0.10 per GB-month.
 - Usage: Store 0.1 TB for first 10 days of month, then 20 TB for remaining 21 days of month.

- Sum up usage over month: $0.1\text{TB} \times 240\text{ hours} + 20\text{TB} \times 504\text{ hours} = 10,104\text{ TB-hours}$.
- Convert to GB-months: $10,104\text{ TB-hours} \times (1024\text{ GB/TB}) \times (1\text{ month}/744\text{ hours}) = 13,906.58\text{ GB-months}$
- Apply tiered pricing: $(\$0.14 \times 1 \times 1024\text{GB}) + (\$0.12 \times 9 \times 1024\text{GB}) + (\$0.10 \times 3666.58\text{GB}) = \1615.94 .

Data Transfer Charges

- Types: 2 types. Data Transfer IN and Data Transfer OUT.
 - Each type (IN, OUT) has own cost.
- Example:
 - Unit: Dollars per GB.
 - Pricing: IN: 0GB+ at \$0.10. OUT: 0-10GB at \$0.20, 10GB+ at \$0.10
 - Usage: Transfer-IN 300GB, Transfer-OUT 100GB.
 - $(300 \times \$0.10) + (10 \times \$0.20 + 90 \times \$0.10) = \41.00 .

Request Charges

- Types: 3 types of requests are HTTP PUT/POST, HTTP GET/HEAD, and HTTP DELETE.
 - Each request type has own cost.
- Example:
 - Unit: Dollars per 10,000 requests.
 - Pricing: PUT/POST: \$0.20 per 10,000 requests, GET/HEAD: \$0.01 per 10,000 requests, DELETE: \$0.00 per 10,000 requests.
 - Usage: For the month, 25,000 PUTs/POSTs, 300,000 GETs/HEADs, 1000 Deletes.
 - $(\$0.20 \times 25,000 / 10,000) + (\$0.01 \times 300,000 / 10,000) + (\$0.00 \times 1000 / 10,000) = \0.53 .

Total Bill for Month

- Storage charges: \$1615.94
- Data transfer charges: \$41.00
- Requests charges: \$0.53
- TOTAL: \$1657.47

4.2.3. Assigning Rating Plans to Users

When you create a new user group you can assign to the group a rating plan that by default will apply to each user in the group. Optionally you can assign a rating plan to specific users within the group, overriding the group's default rating plan.

In a multi-region HyperStore system, you have the option of assigning groups or individual users different rating plans for activities in different regions. For example, you might charge users more for stored data in buckets that they've created in your North region than for stored data in buckets created in your South region.

If you do not explicitly assign a rating plan to a user, the user is automatically assigned the rating plan that's assigned to the user's group. If you do not explicitly assign a rating plan to a group, then the system [default rating plan](#) is automatically used for that group.

You can assign rating plans to users through either the CMC or the Admin API.

Assigning Rating Plans to Users (CMC)

To use the CMC to assign a rating plan to a group, use the [Manage Groups](#) page. From here you can create a new group, and while doing so assign the group a rating plan from a drop-down list of rating plans that exist in the system (the system default plan plus any plans you've created). From here you can also retrieve an existing group and change the group's rating plan assignment. Whichever rating plan you associate with a group will be applied to each user in the group, except for any users to whom you explicitly assign a rating plan.

To use the CMC to assign a rating plan to an individual user, use the [Manage Users](#) page. From here you can create a new user, and while doing so assign the user a rating plan from a drop-down list. By default the new user is assigned whichever plan is assigned to the user's group. From the [Manage Users](#) page you can also retrieve an existing user and change her rating plan assignment.

Assigning Rating Plans to Users (Admin API)

To use the HyperStore Admin API to assign a rating plan to a group, you must have first created the group (with the [PUT /group](#) method). Once a group exists, you can assign the group a rating plan by using the [POST /group/ratingPlanId](#) method. You could subsequently use this same API method to change a group's rating plan assignment. The Admin API also supports a [GET /group/ratingPlanId](#) method, for retrieving a group's current rating plan identifier.

The approach is similar if you want to assign a rating plan to an individual user. The user must have already been created (with [PUT /user](#)), and then you can use [POST /user/ratingPlanId](#) to assign the user a rating plan (and to subsequently update that assignment). The method [GET /user/ratingPlanId](#) lets you retrieve the identifier of the rating plan currently assigned to a specified user, and there's also a [GET /user/ratingPlan](#) method for retrieving the user's rating plan in full.

4.2.4. Creating a "Whitelist" for Free Traffic

There may be certain source domains from which you want users to be able to submit S3 requests to the HyperStore system without incurring any data transfer charges. The HyperStore system allows you to create such a "whitelist", consisting of IPv4 addresses and/or subnets. For traffic originating from these addresses or subnets, there is no charge for data IN or data OUT, nor is there any charge for HTTP requests — regardless of users' assigned rating plans.

Note that the whitelist does not have any impact on what users are charged for data **storage**. It allows only for free **traffic** from the specified origin domains. For data storage billing, a user's regular assigned rating plan pricing is used, even if all of the user's S3 requests originate from a whitelisted IP address.

You can creating a billing whitelist through either the CMC or the Admin API. But before doing so, you must enable the whitelist feature. It is disabled by default.

IMPORTANT: If your S3 Servers are behind a load balancer, the load balancer must be configured to pass through request source IP addresses in order for the whitelist feature to work. Also, note that when S3 requests are submitted via the CMC, the S3 Servers consider the CMC itself to be the source of the request. The CMC does not pass to the S3 Servers the IP addresses of CMC clients.

Enabling the Whitelist Feature

To enable the HyperStore billing whitelist feature:

1. On your Puppet master node, open the following configuration file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

2. Set `admin_whitelist_enabled` to true, then save your change.

3. Still on your Puppet master node, launch the HyperStore installer:

```
[root]# <your-installation-staging-directory>/cloudianInstall.sh
```

4. Select "Cluster Management" and then from the sub-menu that displays select "Push Configuration Settings to Cluster". This propagates your configuration file change to all of your HyperStore nodes.
5. From the Cluster Management sub-menu select "Manage Services", then restart your S3 Service. This applies your configuration change to the S3 Service.
6. Still on the Service Management sub-menu, restart the CMC. This applies the configuration change to the CMC.

Creating a Source IP "Whitelist" (CMC)

To create a whitelist through the CMC, use the [Whitelist](#) page. That page shows the ID and name of the default whitelist, which initially has no IP addresses in it. To create a functioning whitelist, you edit the default whitelist by adding IP addresses or subnets to it.

Once created, the whitelist takes effect. From that time forward, the HyperStore billing system will no longer apply traffic charges to users' traffic originating from the whitelisted IP addresses and subnets.

Note If you want to see a particular user's whitelisted traffic volume during a given billing period, you can do so as part of the HyperStore system's functionality for "[Generating Billing Data for a User or Group](#)" (page 67).

Creating a Source IP "Whitelist" (Admin API)

The HyperStore Admin API provides two different methods for creating a whitelist:

- Use the [POST /whitelist](#) method to post your whitelist as a JSON-encoded request payload.
- Use the [POST /whitelist/list](#) method to post your whitelist as a URI parameter.

The Admin API also supports a [GET /whitelist](#) method for retrieving the contents of your current whitelist.

4.2.5. Generating Billing Data for a User or Group

The HyperStore system supports generating a bill for a specified user or for a whole user group. The system applies the user's or group's assigned rating plan to their service usage during the billing period. Bills can be generated for any completed calendar month of service usage.

With the CMC you can generate a billing report for an individual user (but not for a whole group). With the Admin API you can generate billing data for a specified user or for a whole user group.

IMPORTANT: Billing calculation is derived from hourly rollup usage data. The retention period for hourly rollup usage data is configured by *mts.properties.erb*: "**reports.rolluphour.ttl**" (page 511). The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.

Generating Billing Data for a User (CMC)

To generate billing data through the CMC, use the [Account Activity](#) page. In this page you can specify a user and select a billing period. The most recent period you can select is the most recently completed calendar month. You cannot generate a billing report for an in-progress month.

The billing data that you can generate from this page displays in the form of a printable billing document that includes the user's name and user-ID, their user group, the billing period, and the bill generation date. The document shows a summary of the user's rating plan, the user's service activity for the billing period, and the associated charges.

The **Account Activity** page also provides you an option to view a user's service traffic originating from [whitelisted domains](#), if any.

Generating Billing Data for a User or Group (Admin API)

To generate user or group billing data through the HyperStore Admin API, use the [POST /billing](#) method. This triggers the calculation of the billing data for the specified calendar month, and returns the billing data as a JSON-encoded response payload.

The Admin API method also supports a [GET /billing](#) method, which simply retrieves billing data that you've previously generated with the *POST /billing* method. Like the *POST /billing* method, the *GET /billing* method returns billing data as a JSON-encoded response payload.

4.3. Cluster Resizing

4.3.1. Cluster Resizing Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Monitoring Cluster Storage Capacity Utilization"** (page 69)
- **"Adding Nodes to Your System"** (page 71)
- **"Removing a Node from Your System"** (page 71)

HyperStore is **horizontally scalable**, allowing you to gain additional storage and request processing capacity by adding more nodes to your cluster. When you add new nodes to your cluster, the storage capacity associated with the new nodes becomes immediately available to the system. However, the automated processes that re-distribute data from existing nodes to newly added ones -- thereby reducing storage capacity usage on the existing nodes -- may take up to several days or more to complete, depending on factors such as data volume and network bandwidth. Therefore it's important to closely monitor your current and projected system capacity usage, plan ahead for needed cluster expansions, and implement such expansions well before you've filled your current capacity.

Use the CMC Dashboard to monitor your system's current and projected storage utilization level (for more information see "**Monitoring Cluster Storage Capacity Utilization**" (page 69), further below). As best practices for cluster expansion timing, Cloudian recommends the following:

- **Start expansion planning and preparation when either of the following occur (whichever occurs first):**
 - The Dashboard shows your **utilization of system storage capacity has reached 70%**.
 - The Dashboard shows your **utilization of system storage capacity is projected to reach 90% within 120 days**. If your system has a high rate of ingest relative to capacity, this projection may occur even if your current usage has not yet reached 70%.

When planning your expansion keep in mind that:

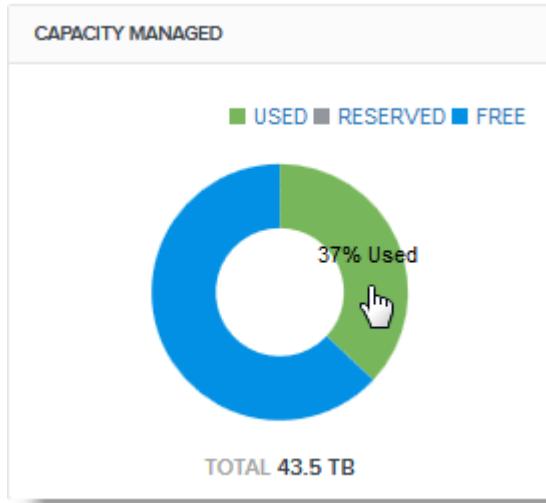
- The minimum unit of expansion is a node. HyperStore does not support adding disks to existing nodes.
- You need to allow time to acquire host machines and prepare them for being added to your cluster.
- Preferably, cluster expansions should be substantial enough that the expanded cluster will allow you to meet your projected storage needs for at least an additional six months after the expansion. In this way you can avoid having to frequently add nodes to your system.
- Cloudian Support is available to review and provide feedback on your expansion plan.
- **Execute your expansion when either of the following occur (whichever occurs first):**
 - The Dashboard shows your **utilization of system storage capacity has reached 80%**.
 - The Dashboard displays a Warning that your **utilization of system storage capacity is projected to reach 90% within 90 days**. If your system has a high rate of ingest relative to capacity, this projection may occur even if your current usage has not yet reached 80%.

IMPORTANT: Each HyperStore node is designed to [reject new writes if it reaches 90% storage capacity utilization](#). Allowing your system to surpass 80% capacity utilization poses the risk of having to rush into an urgent cluster expansion operation.

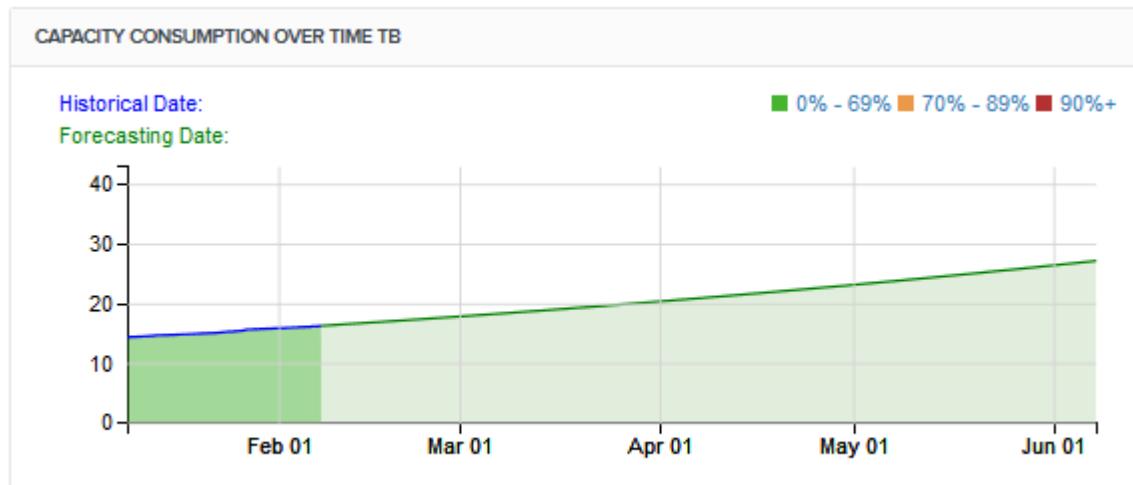
4.3.1.1. Monitoring Cluster Storage Capacity Utilization

HyperStore makes it easy to regularly monitor your system storage capacity utilization. In the CMC Dashboard you can view:

- Current storage capacity utilization.



- Projected storage capacity utilization over the next 120 days.



In both the current capacity utilization graphic and the projected utilization graphic, color-coding is used to highlight utilization levels of 70% or higher and 90% or higher.

The Dashboard also displays:

- A Warning message if the cluster is projected to reach 90% storage utilization in 90 days or less
- A Critical message if the cluster has reached 90% storage utilization

For more detail on Dashboard functionality and metrics see [Dashboard](#).

In the CMC's [Capacity Explorer](#) you can view your system's remaining free storage capacity broken down by service region (cluster), data center, and node. If less than 30% storage space remains at any one of these levels -- that is, if more than 70% of capacity is utilized in a given node, data center, or region -- this is highlighted in the interface.

In the CMC's [Node Status](#) page you view each node's storage capacity utilization as well as the utilization level for each disk on each node.

4.3.1.2. Adding Nodes to Your System

To add nodes to an existing HyperStore data center, follow the documented procedure "[Adding Nodes](#)" (page 369).

To add a new data center (with new nodes) to an existing HyperStore service region, follow the documented procedure "[Adding a Data Center](#)" (page 379).

To add a new service region (with new nodes) to an existing HyperStore system, follow the documented procedure "[Adding a Region](#)" (page 385).

IMPORTANT: With the addition of a new DC or service region you will need to create new [storage policies](#) that utilize the new DC or region. **Adding a new DC or region does not create additional storage capacity for existing buckets that use existing storage policies.** Only new buckets that utilize the new storage policies will make use of the additional storage capacity created by adding a new DC or region. In the current HyperStore version, you cannot revise an existing storage policy or reassign a new storage policy to an existing bucket.

To create additional storage capacity for your existing storage policies you must add nodes to your existing data center(s).

4.3.1.3. Removing a Node from Your System

To remove a node from your HyperStore system, follow the documented procedure "[Removing a Node](#)" (page 390).

4.4. Data Centers

4.4.1. Data Centers Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Storage Policies in a Multi-DC Service Region"** (page 72)
- **"System Monitoring and Operations in a Multi-DC Service Region"** (page 73)
- **"Specialized Services Allocation in a Multi-DC Service Region"** (page 73)

The HyperStore system can be deployed across multiple data centers (DCs). Multi-DC operation is very much part of the system's design. Deploying HyperStore across multiple DCs allows for an even greater degree of service availability and data protection than can be achieved in a single DC.

It's important to understand the distinction between having multiple data centers and having multiple service regions in a HyperStore system. In a HyperStore system with multiple **service regions**, each region has its own separate S3 service endpoint (to which S3 clients submit requests), its own independent storage cluster, and its own separate inventory of stored objects. In a multi-region HyperStore system, the regions are in most respects separate S3-compatible object storage systems, with the significant exceptions that the same population of authorized end users has access to all the service regions, and that HyperStore affords a substantial

degree of unified administration across the multiple regions. End users when they create an S3 storage bucket can choose the service region in which the bucket will reside. One of the main reasons for having multiple service regions is to allow users to choose one geographic region or another for storing their data, for reasons of proximity or regulatory requirements.

Running HyperStore in multiple **data centers** -- however far removed they are geographically -- does not necessarily constitute having multiple service regions. You can have each data center be its own separate service region, but you also can have multiple data centers be part of the same service region. Within a single service region, the nodes in separate data centers are part of the **same integrated storage cluster**, servicing a common inventory of stored objects. Objects in one data center are (by typical storage policy configurations) replicated in the other data center(s) within the same service region. Consequently, if your aim in having multiple data centers is to enhance data durability and availability, you should have those data centers be part of the same service region.

For a diagram showing the relation between data centers and service regions see "**System Levels**" (page 19).

Because a data center may or may not constitute a service region -- depending on how you configure your HyperStore deployment -- there are a variety of possible HyperStore deployment topologies:

- Just one service region, consisting of just one DC
- Just one service region, consisting of multiple DCs
- Multiple service regions, with each region consisting of just one DC
- Multiple service regions, with each region consisting of multiple DCs
- Multiple service regions, with some regions consisting of just one DC while other regions consist of multiple DCs

The rest of this overview focuses on having multiple DCs within a service region. (For more information on having multiple service regions, see "**Service Regions Feature Overview**" (page 128).)

Note HyperStore supports having different DCs operating in different time zones, within a service region.

4.4.1.1. Storage Policies in a Multi-DC Service Region

HyperStore supports **replication and erasure coding** as means of protecting stored data. If you have multiple data centers within a service region, you can create storage policies that utilize those multiple data centers in order to further enhance data availability and durability.

For example, suppose that within a given service region you have two data centers named DC-East and DC-West. When you create a replication storage policy, you can configure the policy so that, for example, for every object stored under that policy 3 replicas are stored in DC-East and 2 replicas are stored in DC-West.

In the case of erasure coding, you could for example configure a 4+2 erasure coding storage policy such that for every object stored under that policy, the object's 6 encoded fragments would be stored in DC-East and those same 6 fragments would also be replicated in DC-West. (If you have three or more DCs in a service region, you also have the option of using "distributed" erasure coding whereby each object's $k+m$ fragments are evenly distributed across DCs -- so that for each object there are only $k+m$ fragments in total -- rather than being replicated across DCs.)

Having multiple DCs in a service region also comes into play when you're configuring the **consistency requirements** associated with a particular storage policy. To continue the prior example of a storage policy that puts 3

replicas in DC-East and 2 replicas in DC-West, you can configure the policy such that for an S3 client's PUT request to be considered successful (and for a success response to be returned to the client), the writes of all the replicas in DC-East and in DC-West must succeed. Alternatively you can configure the policy so that a PUT request is considered successful as soon as a quorum of the total number of replicas is written; or a quorum within each data center; or just a quorum within the "local" data center (the data center that's received the S3 request from the client).

Just as having multiple DCs in a region gives you a variety of DC-aware options for write consistency requirements, so too it gives you DC-aware options for read consistency requirements (which determine what is necessary in order for S3 GET requests to succeed).

Note that just because you have multiple DCs within a service region, that doesn't mean that all of your storage policies have to utilize all of the DCs. For example, if in a given service region you have DC1, DC2, and DC3, you can create some storage policies that utilize all three data centers, and also a storage policy that uses only DC1 (for example, a replication policy that creates 3 replicas of each object and puts all the replicas in DC1). For buckets that use this last storage policy, all object data will be stored in DC1.

For more information on storage policies see "[Storage Policies Feature Overview](#)" (page 140).

4.4.1.2. System Monitoring and Operations in a Multi-DC Service Region

HyperStore facilitates seamless administration of a multi-DC deployment.

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

- The [Capacity Explorer](#) page lets you see how much free storage capacity remains in each DC (as well as in each node, and also in the service region as a whole).
- The [Data Centers](#) page shows you your node inventory in each DC and provides summary status information for each node. From this page you can also add nodes to your cluster on a per-DC basis.
- The [Object Locator](#) feature lets you see exactly where all of a specified object's replicas or erasure coded fragments are located (on which nodes, in which DC)

The HyperStore **installer**, together with HyperStore's use of Puppet, lets you easily manage system configuration across DCs. From the Puppet master node you can edit configuration templates and then use the installer to [disseminate and apply your changes](#) across all nodes in all DCs within the cluster.

HyperStore **hsstool** operations (which can be invoked via the CMC or on the command line; see "[hsstool](#)" (page 609)) automatically apply across the whole multi-DC cluster, when appropriate. For example, if you add one or more nodes to a data center and then use the "rebalance" operation, data load is rebalanced among all the nodes in all the data centers in the cluster. From the *hsstool* perspective, there is just one seamless storage cluster -- not separate data centers.

4.4.1.3. Specialized Services Allocation in a Multi-DC Service Region

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

slave nodes and each DC will have one node acting as a Redis QoS slave node. Some other specialized roles act at the service region level and are not affected by having multiple DCs within a region.

For a summary of services and how they are allocated, see "["HyperStore Services"](#)" (page 10).

For a diagram showing typical services distribution in a multi-DC region, see "["Services Distribution -- Multi-DC, Single Region"](#)" (page 22).

In a multi-DC region each DC has its own sub-set of HyperStore nodes that are automatically configured to act as internal NTP servers. For more information see "["NTP Automatic Set-Up"](#)" (page 540).

4.4.2. Multi-DC Setup

The HyperStore interactive installer makes it easy to install HyperStore as a multi-DC system. From a single computer on which you are running the installer, you can install an entire multi-DC HyperStore system.

Note This topic provides overview information on how to set up a multi-DC system when you first install HyperStore. For complete installation instructions, see the *HyperStore Installation Guide*.

For information on **adding a data center to an existing HyperStore system**, see "["Adding a Data Center"](#)" (page 379).

The mechanism that you use to configure your HyperStore system installation as a multi-DC system is the survey file that you create and provide as input to the HyperStore installer. The survey file is a simple inventory of all the nodes in your HyperStore system, with some basic information about the location of each node — including the service region and the DC in which each node resides.

Below is an example of an installation survey file for a HyperStore system that will be configured as just a single service region with just a single data center. For each node, the survey file specifies the region, host-name, IP address, data center name ("DC1" in this example), and rack name (use the same rack name for all nodes in a given data center). Note that you must provide a region name even if you will have only one region.

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

Here is a second example, this time for a system that will be installed as a single-region system with two data centers (named "DC1" and "DC2"):

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

Below is a third example, this time for a system that will be installed as a two-region system. Note that in this example, the "tokyo" region encompasses two data centers (named "DC1" and "DC2"), while the "osaka" region consists of just one data center (named "DC3").

```
tokyo,cloudian1,66.1.1.11,DC1,RAC1
tokyo,cloudian2,66.1.1.12,DC1,RAC1
tokyo,cloudian3,66.1.1.13,DC1,RAC1
```

```

tokyo,cloudian4,67.2.2.17,DC2,RAC1
tokyo,cloudian5,67.2.2.18,DC2,RAC1
tokyo,cloudian6,67.2.2.19,DC2,RAC1
osaka,cloudian7,68.10.3.24,DC3,RAC1
osaka,cloudian8,68.10.3.25,DC3,RAC1
osaka,cloudian9,68.10.3.26,DC3,RAC1

```

Using your survey file as input, the HyperStore interactive installer will install HyperStore software on each node in the survey file, in all your regions. During the interactive installation process you will be prompted for information about your desired system configuration. This will include the number of system metadata replicas that you want to maintain in each DC. This is for protection of reporting data and other system metadata.

4.4.2.1. Port Availability for Inter-DC Communications

On each HyperStore node, on all interfaces, all ports must be open to traffic originating from any other HyperStore node -- including HyperStore nodes in other data centers and other service regions.

4.5. Data Repair

4.5.1. Data Repair Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Repair-On-Read"** (page 76)
- **"Proactive Repair"** (page 76)
- **"Scheduled Auto-Repair"** (page 76)
- **"Operator-Initiated Data Repair"** (page 77)

Through its **storage policies** feature, HyperStore provides you the option of using eventual consistency for writes of S3 object data and metadata. For example, in the context of a 3X replication storage policy you can configure a policy such that the system returns a success response to an S3 client's PUT Object request so long as two of the three replicas can be written at the time of the request. As a second example, in the context of a 4+2 erasure coding storage policy you can configure a policy to return a success response to a PUT Object request so long as five of the six erasure coded fragments can be written at the time of the request.

Eventual consistency can reduce S3 write request latency and increase S3 write availability while still providing a high degree of data durability assurance. Eventual consistency also means that for a given object, there may be times when not all of the object's intended replicas or EC fragments exist in the system. For example, in a 3X replication context there may be times when only two replicas of an object exist in the system, rather than the intended three replicas.

HyperStore **automatically** implements several mechanisms to detect and replace missing replicas or EC fragments: repair-on-read, proactive repair, and scheduled auto-repair.

4.5.1.1. Repair-On-Read

Whenever a read request is processed for a particular replicated object, all replicas of the object are checked and any missing or out-of-date replicas are automatically replaced or updated. Repair-on-read is also performed for erasure coded object reads, in the event that there are enough fragments to decode the object but one or more of the parity fragments are missing.

4.5.1.2. Proactive Repair

When a node has been down or unreachable, then when the node comes back online it is automatically brought up to date by proactive repair. Proactive repair works by reading from Cassandra a list of objects for which a write succeeded in the system as a whole but failed on that node. Working from this object list, proactive repair streams in any locally missing replicas by copying them from nodes where they do exist; or, in the case of erasure coded objects, the proactive repair process re-generates the locally missing fragment. This all happens automatically without need for operator action.

Proactive repair covers several circumstances wherein writes may succeed in the system as a whole but fail to be written to a particular endpoint node:

- The endpoint node was momentarily unavailable
- The endpoint node had been unavailable for long enough for the [S3 layer to mark it as temporarily down](#) and stop sending write requests to it (by default this occurs after a few minutes of a high rate of timeouts or other errors returned by the node)
- The endpoint node was in a [stop-write condition](#) and so the S3 layer stopped sending write requests to it (by default this occurs when all data disks on the node are 90% full or more)
- The endpoint node was [marked down for maintenance](#) by an operator and so the S3 layer stopped sending write requests to it

The maximum time for which proactive repair jobs can be queued for a node that is unavailable is 4 hours by default, and is configurable. Also configurable is the regular interval at which each node in the cluster checks whether there are any queued proactive repair jobs for itself, and executes those jobs if there are any (default = 1 hour). For more information see "[Configuring Automatic Data Repair](#)" (page 78).

Note The proactive repair feature also automatically handles repairs for a node that you've newly added to your cluster, in the event that some data failed to be streamed to the node during the rebalance operation.

4.5.1.3. Scheduled Auto-Repair

To ensure that your HyperStore data is protected to the degree that you intend, the system automatically runs node repairs on scheduled intervals. By default:

- To repair replicated object data, each node is scheduled to have [hsstool repair](#) automatically run on it once every 30 days.
- To repair erasure coded object data, each node is scheduled to have [hsstool repairec](#) automatically run on it once every 29 days.
- To repair metadata in Cassandra, each node is scheduled to have [hsstool reparcassandra](#) automatically run on it once every 7 days.

The repairs are scheduled and launched in such a way that **within a service region only one repair of each type is running at a time**. For each repair type the system maintains a queue of nodes scheduled for auto-repair -- a replicated data repair queue, an erasure coded data repair queue, and a Cassandra metadata repair queue. For each repair type, repair of the node that is next in queue will not start until the repair operation completes on the node on which a repair is currently running. Consequently if you have a lot of nodes and/or a high volume of data in your system, the actual time between repairs of a given node may be larger than the scheduled interval. This effect is most pronounced with erasure coded data repair (which can be very long-running) and least pronounced with Cassandra repair (which is relatively fast).

Note that when a repair operation is running on a target node, **the scope of repair activity will extend to other nodes** as well. In the case of repair of replicated object data, repair of a target node will also make sure that for objects that fall within the target node's primary token range, the objects' replicas also are present on the other nodes where they are supposed to be. That same repair dynamic is true also for repair of replicated Cassandra metadata.

In the case of erasure coded object data, for single data center storage policies and for multi- data center replicated EC storage policies, repair of a target node has the effect of assessing and repairing all erasure coded data on all nodes within the data center where the target node resides. And for multi- data center distributed EC storage policies, repair of a target node has the effect of assessing and repairing all erasure coded data in all of the participating data centers. In a multi- data center HyperStore service region, the erasure coded data auto-repair queue is ordered in such a way that the target nodes alternate among the data centers -- for example after a repair completes on a target node in DC1, then the next target node will be from DC2, and then after that completes the next target node will be from DC3, and then after that completes the next target node will be from DC1 again, and so on.

Note The intervals for scheduled auto-repair are configurable as described in "[Configuring Automatic Data Repair](#)" (page 78).

Note The system allows repair operations of different types -- such as an *hsstool repair* operation and an *hsstool repairec* operation -- to run concurrently within the same service region.

4.5.1.4. Operator-Initiated Data Repair

In most circumstances HyperStore's automatic data repair mechanisms -- read-on-repair, proactive repair, and scheduled auto-repair -- are sufficient to keep the data in your cluster complete and consistent. However, there are occasions when you will need to manually trigger a repair operation:

- After removing a dead node from your cluster. See "[Removing a Node](#)" (page 390).
- When a node is brought back online after being unavailable for longer than the configurable maximum time that proactive repair can handle. See "[Restoring a Node That Has Been Offline](#)" (page 400).

For repair command syntax see [hsstool repair](#) and [hsstool repairec](#) and [hsstool repaircassandra](#).

4.5.2. Configuring Automatic Data Repair

By default, HyperStore's automatic data repair functions are configured in a way that's suitable for typical HyperStore deployments. However, there are a few settings that you can modify if you wish.

The [Scheduled Auto-Repair](#) feature is configurable by these settings in the CMC's [Configuration Settings](#) page:

- Replicas Repair Interval (default = every 30 days)
- EC Repair Interval (default = every 29 days)
- Cassandra Full Repair Interval (default = every 7 days)

See "[Auto-Repair Schedule Settings](#)" (page 305) for important details about how these interval settings are applied.

Note: If you wish, you can have some or all of the auto-repairs of replica data and erasure coded data use the "computedigest" option to combat bit rot. This feature is controlled by the "["auto_repair_computedigest_run_number"](#)" (page 468) setting in *common.csv*. By default "computedigest" is not used in auto-repair runs.

The [Proactive Repair](#) feature is configurable by these properties:

- In [hyperstore-server.properties.erb](#) the setting "["hyperstore.proactiverepair.poll_time"](#)" (page 493) controls how frequently each HyperStore node checks to see whether any proactive repair jobs are queued for itself, and executes those jobs if there are any. HyperStore nodes automatically make this check upon start-up, and then again at this configurable interval (default is 1 hour). This recurring check is necessary because even if a node hasn't been down, there may be need to execute proactive repair — for example, if network issues had made the node temporarily unreachable as other nodes were processing S3 write requests. In the case of a newly added node, this recurring check will also take care of repairing any data that failed to be streamed into the node during the rebalance operation.

Note: If for some reason you want to trigger proactive repair on a particular node immediately, you can do so by running the [hsstool proactiverepair](#) command with the "-start" option.

- In [mts.properties.erb](#) the setting "["hyperstore.proactiverepair.queue.max.time"](#)" (page 522) sets the maximum time for which proactive repair jobs can be queued for a node that is unavailable (default is 4 hours). This time limit prevents Cassandra from being over-loaded with metadata relating to proactive repair, and ensures that proactive repair is used only for its designed purpose, which is to repair object data from a relatively brief time period. If a node is down for longer than this, then when you bring the node back online you will need to let proactive repair complete and then run a manual repair also. For detail see "[Restoring a Node That Has Been Offline](#)" (page 400).

If you edit properties file settings, be sure to push your changes to your cluster and to restart the HyperStore Service (for a *hyperstore-properties.erb* edit) and/or the S3 Service (for an *mts.properties.erb* edit). For instructions see "[Pushing Configuration File Edits to the Cluster and Restarting Services](#)" (page 454).

4.5.3. Checking Data Repair Status

Cluster-wide summary information about current data repair activity is available in the CMC's [Repair Status](#) page. This includes information about any in-progress proactive repair, scheduled auto-repair, or operator-initiated repair. This CMC page also indicates whether proactive repair is pending for a node.

Repairs that you have initiated via the CMC can also be tracked in the [Operation Status](#) page. If you initiate a repair via the command line, you can track its progress by executing [hsstool opstatus](#) (either on the command line or via the CMC's [Node Advanced](#) page).

To view the cluster-wide schedule for auto-repairs, execute the [hsstool repairqueue](#) command on any node (either on the command line or via the CMC's [Node Advanced](#) page).

4.5.3.1. Repair Completion Alerts

Whenever a proactive repair, an auto-repair, or an operator-initiated repair finishes its run, HyperStore sends an alert email to the system administrator(s). An alert also appears in the CMC, in both the [Alerts](#) page and the [Node Status](#) page. The alert indicates the final status of the repair run: either COMPLETED, FAILED, or TERMINATED (if interrupted by an operator).

You can customize these alerts — including an option for having SNMP traps sent — in the CMC's [Alert Rules](#) page.

For detailed repair status information for a recently finished repair run — for example, to get more information about a FAILED repair run that you've been alerted to — in the CMC go to the [Node Advanced](#) page and from the "Info" command group execute the [hsstool opstatus](#) command for the node on which the repair ran.

4.5.4. Disabling or Stopping Data Repairs

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Temporarily Disabling Automatic Data Repairs" (page 79)**
- **"Stopping In-Progress Data Repairs" (page 82)**

HyperStore allows you to temporarily disable its automatic data repair features, and also to stop data repairs that are in progress.

IMPORTANT: The scheduled auto-repair feature and the proactive repair feature are both important for maintaining data integrity in your system. Do not permanently disable either of these features.

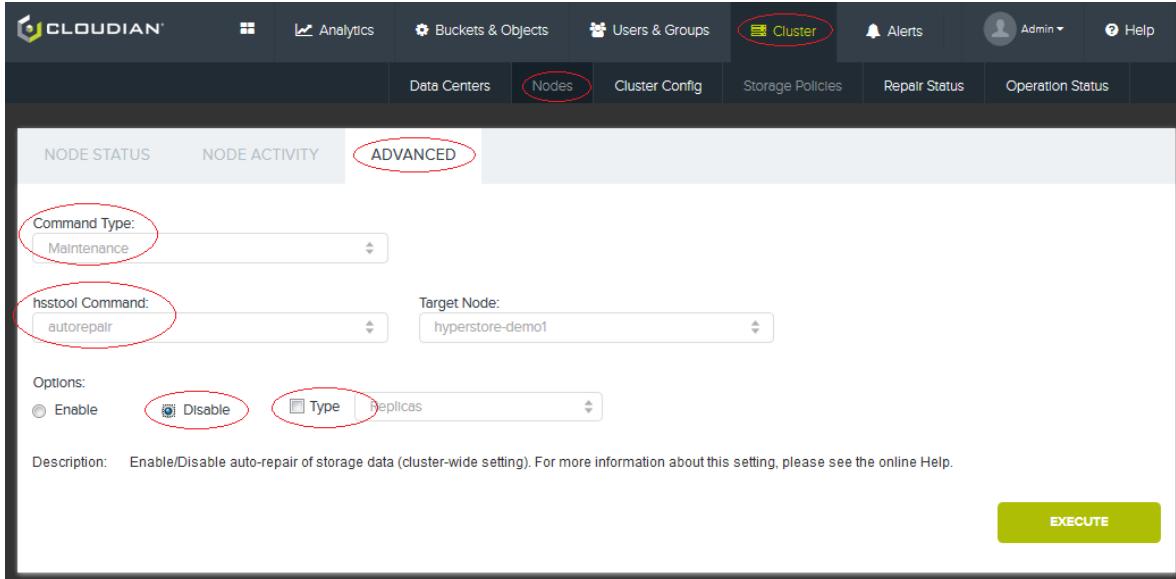
4.5.4.1. Temporarily Disabling Automatic Data Repairs

If you wish you can temporarily disable HyperStore's scheduled auto-repair feature and its proactive repair feature, if for some reason you do not want either type of repair to automatically start up for some period of time.

Note The system **automatically** disables the auto-repair and proactive repair features when you perform a HyperStore version upgrade and when you add nodes to your cluster; and the system

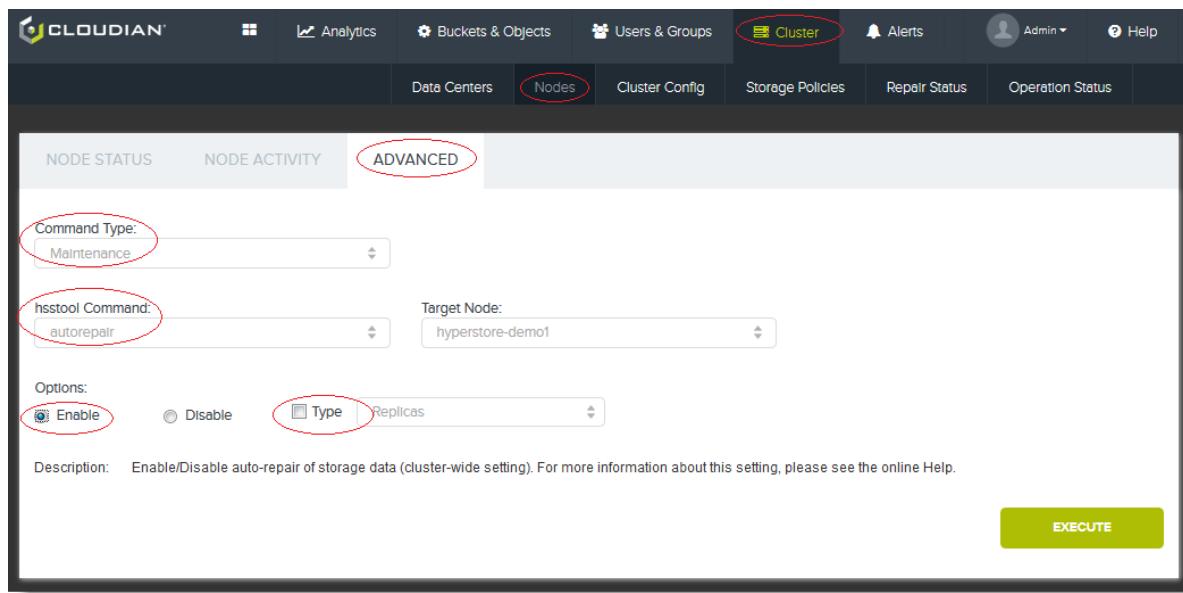
automatically re-enables the auto-repair and proactive repair features at the conclusion of those operations. So the steps below are only needed if you want to temporarily disable these features in circumstances other than system upgrade or system expansion.

To disable all automatic data repairs in a service region, go to the CMC's **Node Advanced** page and execute the maintenance command `autorepair` with the Disable option. This will prevent any new schedule-based auto-repairs or proactive repairs from launching. The target node can be any node in the service region; automatic data repairs will be disabled throughout the service region regardless of which node receives the command. Leave the "Type" option unselected.



Note Disabling automatic data repairs prevents new scheduled auto-repairs and new proactive repairs from launching, but it **does not stop repairs that are currently in-progress**. For information about doing the latter see "**Stopping In-Progress Data Repairs**" (page 82).

After completing the system operation that you are undertaking, be sure to **re-enable automatic data repairs**.

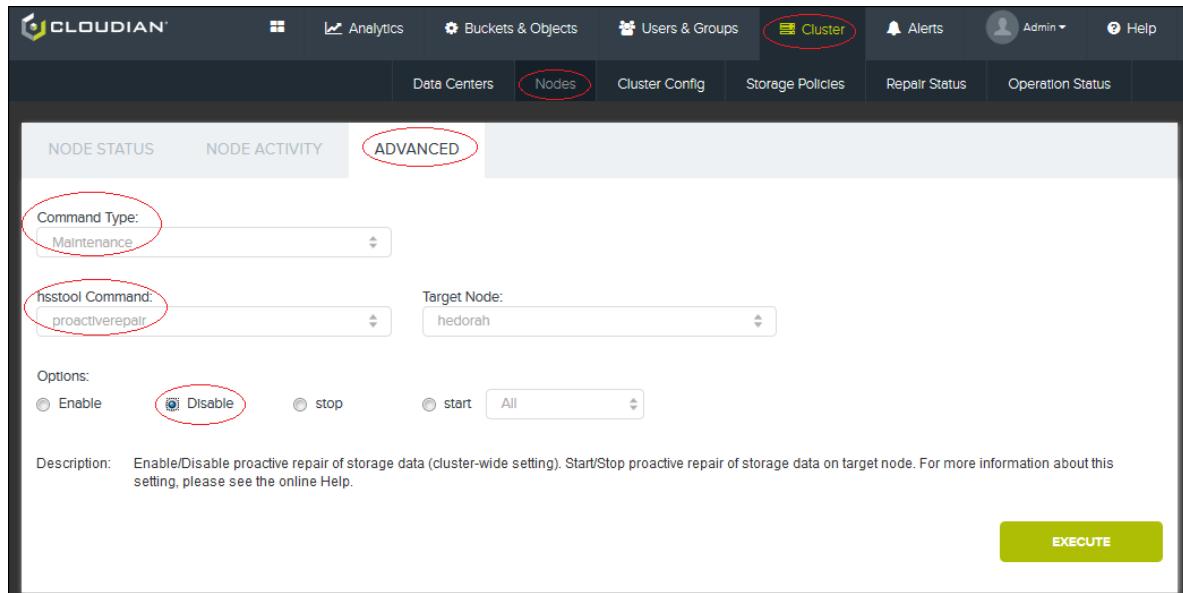


Temporarily Disabling Just Certain Types of Automatic Data Repairs

If you wish you can disable just a certain type of automatic data repair, rather than disabling all automatic data repairs.

To disable just scheduled auto-repair for replicated object data, or just scheduled auto-repair for erasure coded object data, or just scheduled auto-repair for Cassandra metadata, go to the CMC's **Node Advanced** page and execute the maintenance command *autorepair* with the Disable option and with a "Type" specified (*Replicas* or *EC* or *Cassandra*).

To disable just proactive repair go to the CMC's **Node Advanced** page and execute the maintenance command *proactiverepair* with the Disable option.



Be sure later to **re-enable whichever type of automatic data repair you disabled**, by using the same interface.

4.5.4.2. Stopping In-Progress Data Repairs

On rare occasions, you may want to stop a data repair that's in progress on a particular node.

- To stop an in-progress repair of replicated object data, use [**hsstool repair**](#) with the "-stop" option.
- To stop an in-progress repair of erasure coded object data, use [**hsstool repairec**](#) with the "-stop" option.
- To stop an in-progress repair of Cassandra metadata, use [**hsstool repaircassandra**](#) with the "-stop" option.
- To stop an in-progress proactive repair, use [**hsstool proactiverepair**](#) with the "-stop" option.

It does not matter whether the repair in progress was initiated automatically by the system or initiated by an operator -- in either case you can stop it with the "-stop" option.

Note A typical Cassandra metadata repair would take minutes, and a typical proactive repair would take minutes or hours, but a replicated object data repair or erasure coded object data repair for a node may take days (depending on the amount of data involved and on network bandwidth). If a replica repair or EC data repair is in progress you can check either the CMC's [**Operation Status**](#) page or the CMC's [**Repair Status**](#) page to see how far along the repair is and approximately how much longer it will take, before deciding whether to stop it. The [**Repair Status**](#) page also has progress information for proactive repairs.

4.6. Disk Failure Handling

4.6.1. Disk Failure Handling Feature Overview

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

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

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

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

on the other disks, and so that data is unreadable on the host. For more detail on the configuration options and the disk disabling behaviors see "**HyperStore Disk Failure Action**" (page 297).

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

4.6.1.1. Restrictions

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

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

4.6.2. Configuring Disk Failure Handling

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

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

In the CMC's [Configuration Settings](#) page, in the "System Settings" section, you can configure a "**HyperStore Disk Failure Action**" (page 297). This is the automated action for the system to take in the event of a detected disk failure, and the options are "Disable Disk + Move Its Tokens" (the default) or "None". If you change this setting in the **Configuration Settings** page, your change is dynamically applied to the cluster — no service restart is necessary.

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

- **"disk.fail.error.count.threshold"** (page 494)
- **"disk.fail.error.time.threshold"** (page 495)

By default the threshold is 10 errors in the space of 5 minutes, for a particular disk.

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

- **"disk.audit.interval"** (page 495)

By default the proactive audit occurs hourly.

After editing any of these *hyperstore-server.properties.erb* settings, push the changes to the cluster and restart the HyperStore Service. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.6.3. Checking Disk Status

You can use the CMC to check on the current status of any disk in your HyperStore system. Go to **Cluster → Nodes → Node Status**, and select a node to check on. Then review the [Disk Detail Info](#) section to see the current status of each disk on the selected node. Each data disk's status is communicated by a color-coded icon, with the status being one of OK, Error, or Disabled.

4.6.4. Disk Error Alerts

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

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

4.6.5. Bringing a Disk Online

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

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

HyperStore provides a highly automated method for bringing the same disk back online. For instructions see "**Enabling a HyperStore Data Disk**" (page 428).

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

HyperStore provides a highly automated method for replacing a disk and restoring data to the new disk. For instructions see "**Replacing a HyperStore Data Disk**" (page 430).

With either of these methods, **any tokens that were migrated away from the disabled disk will be automatically migrated back to it (or its replacement)**. Object data that was written in association with the affected tokens while the disk was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks (utilizing HyperStore "**Dynamic Object Routing**" (page 87)).

4.7. Disk Usage Balancing

4.7.1. Disk Usage Balancing Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Automatic Correction of Substantial Disk Usage Imbalance" (page 85)**
- **"Automatic Stop of Writes to a Disk at 90% Usage" (page 85)**
- **"Automatic Stop of Writes to a Node at 90% Usage" (page 86)**
- **"Dynamic Object Routing" (page 87)**

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

4.7.1.1. Automatic Correction of Substantial Disk Usage Imbalance

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

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

You do not need to run a repair operation in connection with this disk usage balancing feature. The disk usage rebalancing mechanism is not intended to move existing data between disks. It is designed to impact only objects uploaded after the time that the automated token migration was executed. If you were to perform a repair, existing objects on the over-used disk will not follow a migrated token to the token's new home on a less-used disk. For more about how the token migration works see "**Dynamic Object Routing**" (page 87).

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

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

IMPORTANT: As a best practice, closely monitor your system's current and projected disk space usage and expand your cluster well in advance of disks and nodes becoming nearly full. See "**Cluster Resizing Feature Overview**" (page 68).

Every 30 minutes each HyperStore data disk in the system is checked for capacity usage level and if 90% or more of a disk's capacity has been filled the system automatically moves all of the disk's tokens to other disks that are at less than 90% usage, on the same node. The object data on the 90% full disk remains readable and the disk will still support S3 get requests and delete requests, but any S3 writes associated with the token ranges that used to be on the disk are now directed to the new locations of those token ranges, on other disks on the same node. This disk-level "stop-write" condition triggers a warning message in the HyperStore application log, which in turn results in an alert being generated in the CMC's [Alerts](#) page.

If subsequently another disk on the node hits the 90% usage level, then that disk's tokens are also transferred to the remaining disks that are still at lower than 90% usage. This process will continue if more disks on the node hit 90% usage, with the node's storage tokens -- and consequently, the node's support for new S3 writes -- being consolidated on to fewer and fewer disks.

Note If a disk is in stop-write condition, then [hsstool repair](#), [hsstool repairec](#), [hsstool rebalance](#), and [hsstool decommission](#) operations will not write to that disk. Within the operation the tasks that entail writing object data to that disk will fail, and in the operation's status results these task failures will increment the "Failed count".

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

IMPORTANT: As a best practice, closely monitor your system's current and projected disk space usage and expand your cluster well in advance of disks and nodes becoming nearly full. See "[Cluster Resizing Feature Overview](#)" (page 68).

If **all** of a node's data disks have reached a stop-write condition -- because each disk has hit 90% disk usage -- then the **whole node goes into a "stop-write" condition** and the HyperStore system's S3 layer stops sending S3 writes to that node. The node continues to support reads and deletes.

When a node is in the "stop-write" condition:

- A critical message appears in the CMC's [Dashboard](#); an alert is generated in the [Alerts](#) page; and the node is marked by a red disk-stack icon in the [Data Centers](#) page.
- If you use [dynamic consistency levels](#) for S3 writes, the S3 layer will consider the node to be unavailable for writes and will utilize fallback consistency levels.

Note If a node is in stop-write condition, then [hsstool repair](#), [hsstool repairec](#), [hsstool rebalance](#), and [hsstool decommission](#) operations will not write to that node. Within the operation the tasks that entail writing object data to that node will fail, and in the operation's status results these task failures will increment the "Failed count".

Getting a Node Out of Stop-Write Condition

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

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

- You can **delete objects from your HyperStore system**. Note that replicated and erasure coded object data is dispersed across all of your nodes and there is not a way to target data on a specific node for deletes. Rather, you must delete objects from the system as a whole so that the overall disk space utilization in the system is reduced. This will have the effect of also reducing disk space utilization on the node that's in stop-write condition. You can delete objects either through the S3 interface or through the special Admin API call that lets you efficiently delete all the objects in a specified bucket (see "**buck-etops**" (page 716)). You can use the CMC's [Node Status](#) page to periodically check the node's disk space usage.

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

- You can **add nodes to your cluster**, and execute the associated rebalancing and cleanup operations (for instructions see "**Adding Nodes**" (page 369)). This will have the effect of reducing data utilization on your existing nodes, including the node that's in the stop-write condition. Note that rebalance and cleanup are long-running operations and so this approach to getting a node out of stop-write condition may take several days or more.

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

4.7.1.4. Dynamic Object Routing

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

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

4.7.2. Configuring Disk Usage Balancing

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

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

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

- "**hyperstore_disk_check_interval**" (page 467) in *common.csv* (default = 72 hours)
- "**disk.balance.delta**" (page 495) in *hyperstore-server.properties.erb* (default = 10 percent)

After editing either of these settings, be sure to push your changes to the cluster and restart the HyperStore Service. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

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

4.7.3. Checking Disk Usage Status

You can use the CMC to check disk usage status for each node and each disk in your HyperStore system. Go to the CMC's [Node Status](#) page and select a node to check on. Then review the [Disk Detail Info](#) section to see the current space utilization information for each disk on the selected node.

4.7.4. Triggering an Immediate Balance Check

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

To trigger an immediate disk usage balance check:

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

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

4.8. Group and User Provisioning

4.8.1. Group and User Provisioning Feature Overview

Through the HyperStore Admin API or through the CMC, you can provision the user groups and individual users who you want to authorize to use the HyperStore S3 service. You will provision groups first, and then once one or more groups exist you can add individual users to each group. All users must belong to a group.

As a system administrator you can act on groups in a variety of ways:

- Each group can be assigned quality of service (QoS) limits that will enforce upper bounds on the service usage levels of the group as a whole. Each group can also be assigned default user-level QoS controls that will limit the service usage of individual users within the group. (Optionally, you can also assign per-user QoS limits that will supersede this default.)
- You can generate service usage reports for groups (and also for individual users).
- Each group can be assigned a default rating plan which will determine how users in that group will be charged for HyperStore service usage. (Optionally, you can also assign per-user rating plans that will supersede this default.)
- You can create one or more users who have group administrator privileges. Group administrators are able to perform the following operations through the CMC:
 - Create a user within the group
 - Edit a user's profile
 - Retrieve a list of users in the group
 - Assign user-specific QoS limits
 - Provide user support by accessing a user's data in the S3 object store
 - Delete a user
 - Generate a usage report for the group
 - Generate a usage report for an individual user in the group

Note: The set of privileges that you make available to group administrators is configurable in a granular way (in the [mts-ui.properties.erb](#) file, see the `admin.manage_users.enabled` property and those that follow it). Individual CMC UI functions and sub-functions can be displayed to or hidden from group administrators depending on your configuration settings.

4.8.1.1. IAM Support

HyperStore provides limited support for the Amazon Identity and Access Management (IAM) API. For an overview of this HyperStore feature, including how to enable the feature, limitations on the scope of HyperStore's IAM support, and how to access the HyperStore IAM Service, see "**Identity and Access Management (IAM) Feature Overview**" (page 91).

4.8.2. Provisioning Groups

You can provision user groups through either the CMC or the Admin API.

Note Optionally, when creating a group you can enable LDAP-based authentication of group members. For more information see "**LDAP Integration Feature Overview**" (page 101).

Provisioning Groups (CMC)

In the CMC's **Manage Groups** page you can perform group operations including:

- "**Add a Group**" (page 237)
- "**Set Quality of Service (QoS) Controls**" (page 250)
- "**Retrieve a Group or a List of Groups**" (page 241)
- "**Edit a Group**" (page 241) (including suspending a group)
- "**Delete a Group**" (page 243)

Provisioning Groups (Admin API)

Through the HyperStore Admin API you can perform group operations including:

- Create a new group: [**PUT /group**](#)
- Assign a rating plan to a group: [**POST /group/ratingPlanId**](#)
- Assign QoS limits to a group: [**POST /qos/limits**](#)
- Retrieve a list of groups: [**GET /group/list**](#)
- Edit a group (including suspending a group): [**POST /group**](#)
- Delete a group: [**DELETE /group**](#)

4.8.3. Provisioning Users

You can provision individual users through the CMC, or the Admin API, or by enabling LDAP integration.

Note The HyperStore system does not currently support bulk provisioning of users. Users must be added one at a time.

Provisioning Users (CMC)

In the CMC's **Manage Users** page you can perform user operations including:

- "**Add a User**" (page 230)
- "**Set Quality of Service (QoS) Controls**" (page 250)
- "**Retrieve a User or List of Users**" (page 231)
- "**Edit or Suspend a User**" (page 232) (including suspending a user)
- "**Delete a User**" (page 236)

Note The CMC does not support retrieving a list of users who have been deleted from the system. If you need to retrieve a list of deleted users, you can do so through the Admin API -- see [**GET /user/list**](#).

Provisioning Users (Admin API)

Through the HyperStore Admin API you can perform user operations including:

- Create a new user: [**PUT /user**](#)
- Assign a rating plan to a user: [**POST /user/ratingPlanId**](#)
- Assign QoS limits to a user: [**POST /qos/limits**](#)
- Retrieve a list of users: [**GET /user/list**](#)
- Update a user's profile (including suspending a user): [**POST /user**](#)
- Delete a user: [**DELETE /user**](#)

Provisioning Users (LDAP / Active Directory)

As an alternative to provisioning users through the CMC or the Admin API, on a per-group basis you can enable integration between the CMC and your Active Directory or other LDAP system, such that users will be automatically provisioned within HyperStore when they log into the CMC with their LDAP credentials. For more information see "**LDAP Integration Feature Overview**" (page 101).

4.9. Identity and Access Management (IAM)

4.9.1. Identity and Access Management (IAM) Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Restrictions and Limitations in HyperStore's IAM Support"** (page 92)
- **"HyperStore IAM Extensions to Support RBAC for Admin Functions"** (page 92)
- **"Deleting or Suspending HyperStore Users Who Have Created IAM Users"** (page 92)
- **"Disabling the HyperStore IAM Service"** (page 93)

HyperStore provides **limited support** for the Amazon Web Services Identity and Access Management (IAM) API. This support enables each HyperStore user, under his or her HyperStore user account, to create IAM groups and IAM users. The HyperStore user can then grant those IAM users permissions to perform certain actions (such as reading or writing objects in a particular bucket or buckets). As with Amazon, the means by which a HyperStore user grants such permissions to IAM groups and users is by attaching "managed" IAM policies to groups or users, and/or by embedding "inline" IAM policies for groups or users. By default **newly created IAM users have no permissions**; they gain permissions only when their parent HyperStore user attaches or embeds policies for them.

In the HyperStore system all S3 object data created by IAM users belongs to the parent HyperStore user account (otherwise known as the "root" account). Consequently, if an IAM user is deleted by their HyperStore parent user, the IAM user's data is not deleted from the system.

4.9.1.1. Restrictions and Limitations in HyperStore's IAM Support

- HyperStore supports **many but not all of the Amazon IAM API "Actions"**. For the list of supported actions see "**IAM Supported Actions**" (page 950).
- HyperStore **has limited support for IAM policies**. Some of the limitations are specific to the CMC as an IAM client application:
 - The CMC does not support creating or attaching "managed" IAM policies.
 - The CMC does not support embedding inline policies for specific IAM users.
 - The CMC **does** support embedding inline policies for IAM groups. Therefore, for a HyperStore user to grant permissions to her IAM users, the only supported method in the CMC is to embed an inline policy for an IAM group, and then assign IAM users to the group so that they gain the permissions associated with the group's inline policies.

The HyperStore IAM Service, if accessed by a third party client application rather than by the CMC, offers broader support for applying IAM policies. The IAM Service supports creating "managed" IAM policies and attaching them to groups or users; and it supports embedding inline policies for groups or for users.

However, in the current release HyperStore support for IAM policy document components is not fully comprehensive. HyperStore supports **most but not all** of the policy elements, actions, resources, and/or condition keys from the AWS documentation for IAM policy formation.

- **IAM users cannot login to the CMC**, and cannot use the CMC as their S3 client application. To access the HyperStore S3 Service, IAM users must use a third party S3 client application.

4.9.1.2. HyperStore IAM Extensions to Support RBAC for Admin Functions

The HyperStore implementation of the IAM API includes extensions that:

- Allow HyperStore system admins, group admins, or regular users to execute certain read-only HyperStore administrative functions by submitting a request to the IAM Service.
- Allow HyperStore system admins, group admins, or regular users to grant their IAM users permission to execute those same read-only HyperStore administrative functions.

For more information, including information about the client tool that HyperStore provides to help you use this feature, see "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

4.9.1.3. Deleting or Suspending HyperStore Users Who Have Created IAM Users

If a HyperStore user creates IAM groups and users, and then subsequently you **delete** that HyperStore user from the system, all IAM resources associated with that HyperStore user will also be deleted from the system. That includes IAM groups, users, and policies that the HyperStore user created, the security credentials of those IAM users, and any object data that those IAM users have stored in the system.

If rather than deleting the HyperStore user you **suspend** the HyperStore user (make the user inactive), then any IAM users that the HyperStore user created will be unable to access any HyperStore services (just like the suspended HyperStore user will be unable to access HyperStore services). If you subsequently make the HyperStore user active again, then IAM users under that HyperStore user will again be able to access HyperStore services.

4.9.1.4. Disabling the HyperStore IAM Service

HyperStore's IAM Service is enabled by default, such that IAM functions display in the CMC and your third party client applications can access the IAM Service (see **"Using the HyperStore IAM Service"** (page 93) for more information). If you want to disable the IAM Service do the following:

1. On your Puppet master node open this configuration file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

1. Change the setting *iam_service_enabled,true* to *iam_service_enabled,false* and save your change.
2. Push your change to the cluster and restart the S3 Service. If you need instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 454).

If you disable the HyperStore IAM Service, then IAM functions will no longer display in the CMC and the IAM Service will no longer accept requests from IAM client applications.

4.9.2. Using the HyperStore IAM Service

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Creating S3 Access Credentials for the Default System Admin User"** (page 93)
- **"CMC Support for IAM Functions"** (page 94)
- **"Accessing the IAM Service with a Third Party Client Application"** (page 94)

Users can access and use the HyperStore IAM Service either through the CMC or through a third party client application that supports IAM calls. Whether using the CMC or a third party client application, users must have S3 access credentials (access key ID and secret key) in order to use the HyperStore IAM Service.

4.9.2.1. Creating S3 Access Credentials for the Default System Admin User

If you want the default HyperStore system admin user -- the user whose user ID is "admin" in the CMC -- to be able to use the IAM Service, do the following:

1. Log into the CMC as the "admin" user. (You will see that an **IAM** tab now displays in the CMC interface, but clicking that tab will return an authorization error until after you've completed Steps 2 and 3 below.)
2. On the right side of the CMC's top navigation bar, hold your cursor over your login name ("admin") and then in the drop-down menu select **Security Credentials**.
3. In the security credentials page's **S3 Access Credentials** section, click **Create New Key**.

This creates S3 access credentials (access key ID and secret key) for the "admin" user. S3 access credentials are required in order to access HyperStore's IAM Service. The CMC will use these credentials automatically when the "admin" user uses the CMC to access IAM functions (on the **IAM** tab). Or if you are using a third party application to access the HyperStore IAM Service, you will need to provide the credentials to that application.

Note If you created any additional system admin users prior to the HyperStore 7.1 release, and if you want those system admin users to be able to use the IAM Service, those system admin users will need to complete the steps described above to create S3 access credentials for themselves.

Regular users and group admins created in the CMC are given S3 credentials automatically as part of the user creation process, so such users already have the credentials that they need to access the IAM Service. Also, additional system admins that you create in HyperStore 7.1 or later are automatically given S3 credentials.

4.9.2.2. CMC Support for IAM Functions

HyperStore system admins, group admins, and regular users will see IAM functions when they log into the CMC. Through the CMC all types of users can:

- **"Add an IAM Group"** (page 259)
- **"Edit an IAM Group's Attributes or Membership"** (page 260)
- **"Create and Manage Inline Policies for an IAM Group"** (page 261)
- **"Delete an IAM Group"** (page 263)
- **"Add an IAM User"** (page 255)
- **"Edit an IAM User's Attributes, Credentials, or Memberships"** (page 256)
- **"Delete an IAM User"** (page 258)

Note For restrictions on the CMC's IAM capabilities see **"Restrictions and Limitations in HyperStore's IAM Support"** (page 92).

4.9.2.3. Accessing the IAM Service with a Third Party Client Application

Third party client applications can access the HyperStore IAM service at this URI:

```
http(s)://<host>:<port>/?<parameters>
```

where **<host>** is the hostname or IP address of any HyperStore node in your **default service region**, and the **<port>** is 16080 if using regular HTTP or 16443 if using HTTPS.

For example:

```
https://10.10.10.2:16443/?Action/CreateGroup&GroupName=Sales&Version=2010-05-08&AUTHPARAMS
```

Note The HyperStore IAM Service uses a self-signed certificate for its HTTPS listener, so if you are using HTTPS to access the service your client application must be configured to allow self-signed certificates.

Note The IAM Service listens on the Admin Service endpoint address (so the IAM endpoint is **s3-admin.<your-domain>:16080** or **s3-admin.<your-domain>:16443**).

The HyperStore IAM Service is compliant with the Amazon IAM API in its major respects, including that:

- The "Action" request parameter is used to indicate the IAM API method that is being invoked.
- All requests can be submitted as either a GET or POST

- All requests must be signed with an access key (Signature v2 and v4 are both supported)

For details of HyperStore's support for the IAM API see:

- "**IAM Supported Actions**" (page 950)
- "**IAM Supported Policy Document Elements**" (page 966)
- "**IAM Common Parameters**" (page 969)
- "**IAM Common Errors**" (page 970)

Note As an extension to the IAM API, the HyperStore IAM Service also supports Actions for performing HyperStore administrative tasks. See "**IAM Supported Actions**" (page 950).

4.9.3. IAM Extensions for Role-Based Access to HyperStore Admin Functions

Subjects covered in this section:

- *Introduction (immediately below)*
- "**Comparing the Admin API to the IAM API with RBAC Extensions**" (page 95)
- "**Administrative Actions Supported by the IAM API**" (page 96)
- "**Giving Administrative Action Privileges to IAM Users**" (page 98)
- "**Using admin_client.py to Call the IAM Service Extensions for Administrative Actions**" (page 99)

The HyperStore IAM Service supports extensions to the IAM API that allow for role-based access control (RBAC) for read-only HyperStore administrative functions. The 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 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.

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

In HyperStore 7.0.x and older, the Admin API was the only administrative API for the system. Now in HyperStore 7.1, certain administrative functions can also be called through HyperStore's implementation of the IAM API. 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 <ul style="list-style-type: none"> • Enabled by default • 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 <ul style="list-style-type: none"> • Disabled by default • If enabled, runs on each node in your default region only • Listens on ports 16443 (HTTPS) and 16080 (HTTP)

Admin API	IAM API with RBAC Extensions for Admin Actions
<ul style="list-style-type: none"> Makes no distinctions based on role of the requester -- all access is system administrator level access 	<ul style="list-style-type: none"> Includes a bundled self-signed certificate for HTTPS Request authentication is by 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 <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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)

4.9.3.2. 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 (see Important note below table)	Get any user's bill	Get bill of any user in own group	Get own bill	GET /billing
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

IAM Action	Permission Scope Based On Requester's Role			Corresponding Admin API Method
	System Admin	Group Admin	Regular User	
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
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 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.

4.9.3.3. 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.

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 HyperStore 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 HyperStore 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.

For more information on using IAM polices to grant admin permissions to IAM users, see "**IAM Supported Policy Document Elements**" (page 966).

4.9.3.4. 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.

Note To use the *admin_client.py* tool you will need to supply the tool with your S3 access credentials. To view your S3 credentials you can log into the CMC, hold your cursor over your login name in the upper right of the interface, and then in the drop-down menu select **Security Credentials**. Note that the default HyperStore system administrator account does not have S3 credentials by default. If you are logged in as "admin" and haven't yet created S3 credentials, in the **Security Credentials** page click **Create new key**.

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:

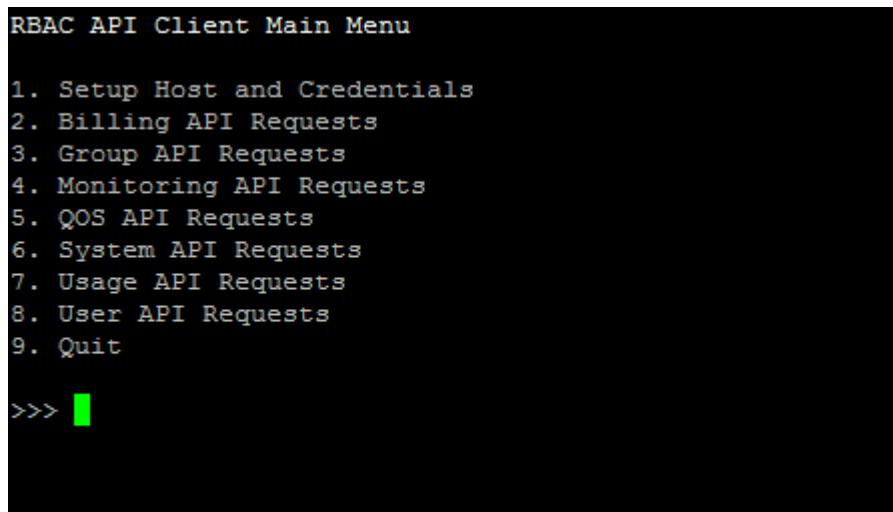
```
/opt/cloudian/tools
```

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 host information (you can connect to any HyperStore host) and the 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 96) 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 would need to use the Python tool, or a third party IAM client that's been customized to support the IAM extensions.

4.10. LDAP Integration

4.10.1. LDAP Integration Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Creating LDAP-Enabled Groups" (page 101)**
- **"Provisioning of Users within LDAP-Enabled Groups" (page 101)**
- **"Deleting Users from the CMC and/or LDAP" (page 102)**
- **"Disabling Active Directory or LDAP Integration After Having Used It" (page 103)**

HyperStore supports integrating with Active Directory or other types of LDAP systems so that users can log into the CMC with their LDAP-based login credentials. This feature is implemented on a per-group basis, so you have the option of creating some groups that are LDAP-enabled and others that are not. The system also supports having different groups use different Active Directory or LDAP servers for authentication, or having all LDAP-enabled groups use the same Active Directory or LDAP server.

Within an LDAP-enabled group, along with users who the CMC will authenticate against an Active Directory or other LDAP system you can optionally also have local users who the CMC will authenticate by use of a CMC-based password rather than LDAP.

Note Under no circumstances does the CMC try to write to your Active Directory or LDAP server — it only reads from it, for the purpose of authenticating users.

Note Only system administrators can enable Active Directory or LDAP authentication for a group. Group administrators cannot enable Active Directory or LDAP authentication for their groups.

4.10.1.1. Creating LDAP-Enabled Groups

To create an Active Directory or LDAP-enabled group through the CMC, use the CMC's **"Add a Group"** (page 237) function. When creating the group select the "Enable LDAP Authentication" option and provide the required Active Directory or LDAP information. (You can also enable LDAP authentication for an already existing group, by using the CMC's **"Edit a Group"** (page 241) function.)

To create an Active Directory or LDAP-enabled group through the HyperStore Admin API, use the [**PUT /group**](#) method and in the [**GroupInfo**](#) object in the request body set the *ldapEnabled* attribute to true and also set the other LDAP-related attributes . (You can also enable LDAP authentication for an already existing group, by using the [**POST /group**](#) method.)

4.10.1.2. Provisioning of Users within LDAP-Enabled Groups

Within a HyperStore group that has LDAP authentication enabled you can have both LDAP-authenticated users and users who are authenticated by a CMC-based password rather than LDAP:

- **For users who you want to be authenticated by Active Directory or LDAP, do not manually create those users** through the CMC (or the Admin API). Instead, simply have those users log into the CMC

using their LDAP credentials. If a user tries to log into the CMC as a member of an LDAP-authenticated group and the user is not already registered in HyperStore as a member of the group, the CMC will attempt to authenticate the user against the LDAP system. If the authentication succeeds, the CMC will **automatically provision** the user into HyperStore. This includes automatic creation of security keys for accessing the HyperStore S3 data store. Going forward whenever the user logs in the CMC will recognize the user as a registered HyperStore user, but will continue to authenticate the user against the LDAP system each time rather than by reference to a CMC-based password.

Note that for such users to be successfully provisioned, the user names that they use when logging into the CMC must satisfy HyperStore restrictions for user names:

- Must be unique within the group.
- Only letters, numbers, dashes, and underscores are allowed. No spaces or special characters.
- Maximum allowed length is 64 characters.
- Must not be any of the following: "anonymous", "public", "null", "none", "admin", "0". These names are reserved for system use.

Note If you want the group administrator to be authenticated by LDAP, have the user log into the CMC using their LDAP credentials. Once this occurs and the CMC automatically provisions the user, you can subsequently edit the user's profile (using the CMC's **Edit User** function or the Admin API's *POST/user* method) to promote them to the group admin role.

- For users who you want to be authenticated by a CMC-based password rather than by the LDAP system, create those users through the CMC's **Add New User** interface (or the Admin API's *PUT/user* method). The CMC will not use LDAP-based authentication for users created through the **Add New User** interface or the *PUT/user* method.

Note If you enable LDAP authentication for an existing group to which users have already been added (via the CMC or the Admin API), those existing users will continue to be authenticated by reference to their CMC-based passwords -- not by reference to an LDAP server.

4.10.1.3. Deleting Users from the CMC and/or LDAP

After you've enabled LDAP integration and have been using this feature, the HyperStore system behaves in the following way in respect to users being deleted from the CMC and/or LDAP:

- If you delete a user from the CMC but that user still exists in LDAP, the user will be able to log in to the CMC as if they were a first-time user and the CMC will auto-provision the user once again. If you want to prevent a user from accessing the CMC and HyperStore, but the user still exists in LDAP, the thing to do is to **deactivate** the user in the CMC (through the CMC's **Edit User** function), rather than deleting them. This will prevent the user from logging into the CMC or accessing HyperStore storage, even though they still exist in LDAP.
- If you delete a user from LDAP but do not delete them from the CMC, the user will not be able to log into the CMC. However, they still have valid S3 credentials and can access the HyperStore storage layer through a different S3 client. If you want a user who you've deleted from LDAP to not have access to the HyperStore S3 system, you should delete them from CMC also (which prevents access and also deletes the user's stored data) or else deactivate them in the CMC (which prevents access but leaves their stored data in place).

4.10.1.4. Disabling Active Directory or LDAP Integration After Having Used It

If you have LDAP enabled for a particular group for some period of time, and during that time LDAP-based users from the group logged into and were auto-provisioned into the HyperStore system, and then for some reason you subsequently disable LDAP for that group — those auto-provisioned users will no longer be able to log into the CMC. This is because the login passwords that the CMC stores in association with those auto-provisioned accounts are system-generated random strings that are simply used as placeholders in the CMC's user data schema. While LDAP integration is enabled, the users' true, operative passwords are those stored in your LDAP system.

In the "use LDAP for a while, then disable it" scenario, the auto-provisioned accounts still exist in the CMC, and the random string passwords become the operative passwords (since the CMC is now using its standard authentication mechanism rather than authenticating against your LDAP system). Because users do not know these passwords, they cannot log into the CMC.

As a system administrator you can remedy this situation by using the CMC's **Edit User** function to create new passwords for such users, and then informing the affected users of their new passwords. They will then be able to log into the CMC once again. This task could also be performed by the group administrator, using the CMC's **Edit User** function. For more information see "**Edit or Suspend a User**" (page 232).

4.11. Object Metadata and Tagging

4.11.1. Object Metadata and Tagging Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"System-Defined Object Metadata"** (page 103)
- **"User-Defined Object Metadata Based on x-amz-meta-* headers"** (page 104)
- **"User-Defined Object Tagging"** (page 104)
- **"Object Metadata and Storage Policies"** (page 104)
- **"Object Metadata, Cross-Region Replication, and Auto-Tiering"** (page 105)

Like Amazon S3, HyperStore allows for rich metadata to be associated with each stored object. There are three categories of object metadata:

- The system itself assigns certain metadata items to objects, having to do with object attributes and status
- Optionally, users can assign metadata to objects by using the S3 `x-amz-meta-*` request headers
- Optionally, users can assign key-value "tags" to objects by using the S3 API methods and headers that implement object tagging

4.11.1.1. System-Defined Object Metadata

For each object, HyperStore maintains a variety of system-defined object metadata including (but not limited to) the following:

- Creation time
- Last modified time

- Last accessed time
- Size
- ACL information
- Version, if applicable
- Public URL, if applicable
- Compression type, if applicable
- Encryption key, if applicable
- Auto-tiering state, if applicable

4.11.1.2. User-Defined Object Metadata Based on `x-amz-meta-*` headers

S3 client applications can create user-defined object metadata by the use of `x-amz-meta-*` request headers that accompany the upload of an object. The client application sets the specific header names and corresponding values (such as `x-amz-meta-project: apollo` or `x-amz-meta-author: ozowa`). This is a standard S3 API feature that is supported by HyperStore's S3 Service. For more information see "**Creating Object Metadata and Tags**" (page 105).

By default the maximum size limit for `x-amz-meta-*` based metadata is 2KB per object. It is possible to increase this size limit by using hidden configuration settings, but to do so could have significant implications for the storage space requirements on the SSDs on which you store Cassandra data. If you are interested in increasing the size limit for `x-amz-meta-*` based object metadata, consult with Cloudian Support.

4.11.1.3. User-Defined Object Tagging

S3 client applications can create user-defined object tags -- key-value pairs such as `status=complete` or `confidential=true` -- either as they upload new objects or for objects that are already stored in the system. This is implemented by the use of object tagging request headers (in the case of assigning tags to an object as it's being uploaded) or object tagging API methods (in the case of assigning tags to an object that's already in storage). This is a standard S3 API feature that is supported by HyperStore's S3 Service. For more information see "**Creating Object Metadata and Tags**" (page 105).

Object tags share some common use cases with `x-amz-meta-*` based object metadata -- for example, you could use either a `x-amz-meta-*` header or an object tag to identify an object as belonging to a particular project. But object tags support a wider range of capabilities, including being able to assign tags to an object that's already in storage, and being able to use object tags as criteria in access control policies. (A future release of HyperStore will also support using object tags as criteria for bucket lifecycle policies for auto-tiering or auto-expiration).

An object may have a maximum of 10 tags associated with it, and each tag must have a unique key (for example you can't assign `status:in-progress` and `status:complete` to the same object). Each key can be a maximum of 128 unicode characters in length, and each value can be up to 256 unicode characters.

4.11.1.4. Object Metadata and Storage Policies

HyperStore object metadata -- including object tags -- is stored in Cassandra, and is protected by replication. The degree of replication depends on the type of [storage policy](#) being used. For more information see "**Storage of Object Metadata**" (page 144).

4.11.1.5. Object Metadata, Cross-Region Replication, and Auto-Tiering

When objects are replicated from a local bucket to a remote destination as part of the [cross-region replication](#) feature, the object metadata (including object tags as well as `x-amz-meta-*` based object metadata) is replicated along with the object data.

When objects are transitioned from a local bucket to a remote destination as part of the [auto-tiering](#) feature, the object metadata (including object tags as well as `x-amz-meta-*` based object metadata) is sent along with the object data if the tiering destination is an S3-compliant system such as Amazon Web Services or Google Cloud Storage. Also, copies of the object metadata are retained in the local HyperStore system (specifically in Cassandra).

If the tiering destination is not S3-compliant, such as Microsoft Azure or Spectra Pearl Logic, then the object metadata is **not** sent along with the object data. The object metadata will exist only in the HyperStore system, even after the corresponding object data is transitioned to the destination system.

4.11.2. Creating Object Metadata and Tags

S3 client applications can create user-defined object metadata -- `x-amz-meta-*` based object metadata and/or object tags -- by using standard S3 API methods and headers that the HyperStore S3 Service supports.

Note In the current version of HyperStore, the CMC's built-in S3 client does not support creating user-defined object metadata. To create user-defined object metadata you will need to use an S3 client application other than the CMC.

4.11.2.1. Creating `x-amz-meta-*` Based Object Metadata

When uploading a new object, S3 clients can create user-defined object metadata by including one or more `x-amz-meta-*` request headers along with the *PUT Object* request. For example, `x-amz-meta-topic: merger` or `x-amz-meta-status: draft`. For HyperStore support of the *PUT Object* method, see [PUT Object](#).

The S3 API method *POST Object* — for uploading objects via HTML forms — also allows for the specification of user-defined metadata (through `x-amz-meta-*` form fields), and HyperStore supports this method as well. For HyperStore support of the *POST Object* method, see [POST Object](#).

4.11.2.2. Creating Object Tags

When uploading a new object, S3 clients can create one or more user-defined object tags by including a single `x-amz-tagging` request header along with the *PUT Object* request. For example, `x-amz-tagging: team-m=Marketing&project=SEO-2018&status=Needs-Review`. For HyperStore support of the *PUT Object* method, see [PUT Object](#).

S3 clients can also create object tags when calling the *POST Object* method, by using the `tagging` form field. For HyperStore support of the *POST Object* method, see [POST Object](#).

The *PUT Object - Copy* method supports copying or replacing an object's tags as it is copied, by making use of the `x-amz-tagging-directive` and `x-amz-tagging` request headers. For HyperStore support of the *PUT Object -- Copy* method, see [PUT Object - copy](#).

S3 clients can also create object tags for an object that is already stored in the HyperStore system, by using the S3 API method *PUT Object tagging*. For HyperStore support of this API method, see [PUT Object tagging](#).

For a high-level view of object tagging usage and methods, in the Amazon S3 online documentation see [Object Tagging](#).

4.11.3. Retrieving Object Metadata and Tags

S3 client applications can retrieve user-defined object metadata -- *x-amz-meta-** based object metadata and/or object tags -- by using standard S3 API methods and headers that the HyperStore S3 Service supports.

Note In the current version of HyperStore, the CMC's built-in S3 client does not support retrieving user-defined object metadata. To retrieve user-defined object metadata you will need to use an S3 client application other than the CMC.

4.11.3.1. Retrieving *x-amz-meta-** Based Object Metadata

S3 client applications can retrieve user-defined *x-amz-meta-** based object metadata -- as well as system-defined object metadata -- by using either of two standard S3 API methods, both of which the HyperStore system supports:

- *GET Object* returns this type of object metadata as response headers, as well as returning the object itself. For HyperStore support of this API method see [GET Object](#).
- *HEAD Object* returns this type of object metadata as response headers, without returning the object itself. For HyperStore support of this API method see [HEAD Object](#).

4.11.3.2. Retrieving Object Tags

The S3 methods *GET Object* and *Head Object* will not return object tags. They will however return a count of the object tags associated with the object (if any), in an *x-amz-tagging-count* response header. For HyperStore support of these API methods see [GET Object](#) and [HEAD Object](#).

To retrieve the object tags associated with an object, use the S3 API method *GET Object tagging*. For HyperStore support of this API method see [GET Object tagging](#).

For a high-level view of object tagging usage and methods, in the Amazon S3 online documentation see [Object Tagging](#).

4.11.4. Elasticsearch Integration for Object Metadata

Subjects covered in this section:

- **"Overview of Elasticsearch Integration for Object Metadata" (page 107)**
- **"Enabling Elasticsearch Integration for Object Metadata" (page 108)**
- **"Using the elasticsearchSync Tool" (page 109)**
- **"Option to Send Metadata to an HTTP Server Rather than to Elasticsearch" (page 110)**

4.11.4.1. Overview of Elasticsearch Integration for Object Metadata

HyperStore supports integrating with an [Elasticsearch](#) cluster, so that you can use the Elasticsearch cluster to search through the object metadata associated with the objects stored in your HyperStore system. This could be an Elasticsearch cluster that you already have running in your environment, or an Elasticsearch cluster that you install specifically for the purpose of integrating with HyperStore.

Note To work with HyperStore your **Elasticsearch version must be a 6.x version, 6.6 or newer**. For availability and performance Cloudian recommends that you have at least three nodes in your Elasticsearch cluster. If you have not yet installed Elasticsearch, Cloudian can provide you a script that simplifies this task. To obtain the script and instructions for using it contact Cloudian Sales Engineering or Support.

When you configure the HyperStore system to integrate with an Elasticsearch cluster, you can then enable object metadata search on a per storage policy basis (as described in ["Enabling Elasticsearch Integration for Object Metadata" \(page 108\)](#)). For a storage policy on which you've enabled object metadata search, the following will happen:

- For each object that subsequently gets uploaded into a HyperStore bucket that uses that storage policy, HyperStore will retain the object metadata locally (as it always does) and also transmit a copy of the object metadata into your Elasticsearch cluster (using HyperStore's built-in Elasticsearch REST client).
 - This includes HyperStore system-defined object metadata and user-defined metadata (for more information on these types of metadata see ["Object Metadata and Tagging Feature Overview" \(page 103\)](#)). It does **not** include object tags.
 - HyperStore will create in your Elasticsearch cluster an index for each bucket that uses a metadata search enabled storage policy. The indexes created in Elasticsearch are named as follows (using lower case only):

`cloudian-<cloudianclustername>-<regionname>-<datacentername>-<bucketname>-<buck-
etcreationdatetime>`

For example:

`cloudian-cloudianregion1-region1-dc1-bucket2-2017-10-19t18:25:56.955z`

Each index will be created in Elasticsearch with three shards and two replicas.

- For each object uploaded to HyperStore a "document" containing the object metadata will be created in Elasticsearch. Each such document will have a name (ID) in this format:

`<bucketname>/<objectname>`

In the case of versioned objects there will be a separate Elasticsearch document for each object version, named as:

`<bucketname>/<objectname>\u0001u-<versionId>`

The document names will be URL-encoded before transmission to the ES cluster and then URL-decoded upon retrieval from the ES cluster.

Note: Enabling object metadata search on a storage policy results in HyperStore copying into Elasticsearch any object metadata associated with objects uploaded **from that time forward**. If

you also want to load into Elasticsearch the object metadata associated with objects that are already in your HyperStore system, you can use the Elasticsearch synchronization tool that comes with HyperStore -- as described in "**Using the `elasticsearchSync` Tool**" (page 109).

- When an object is updated (overwritten by a new S3 upload operation) in HyperStore, its metadata will be updated in Elasticsearch; and when a object or object version is deleted in HyperStore (by an S3 delete operation or by execution of a bucket lifecycle policy), its metadata will be deleted in Elasticsearch.
- If objects that have metadata in Elasticsearch get auto-tiered to a remote destination, their metadata will remain in Elasticsearch.
- If a bucket is deleted from HyperStore, its corresponding index will be deleted in Elasticsearch.
- All insertions, updates, and deletes of HyperStore object metadata in your Elasticsearch cluster are implemented by an [hourly cron job](#) in HyperStore. Note that this means that object metadata associated with a new S3 upload will not immediately appear in Elasticsearch.
- If you enable metadata search for a storage policy, and then at a later time disable metadata search on that storage policy, any object metadata that had been copied to Elasticsearch during the period when metadata search was enabled will remain there (it will not be deleted by HyperStore).

Note that when you enable Elasticsearch integration, HyperStore only **writes** to your Elasticsearch cluster. It does not read from the cluster, and you cannot use HyperStore to search through or retrieve object metadata in Elasticsearch. For that you must use Elastic Stack applications, such as Kibana.

Note If you also want to use the Elastic Stack for HyperStore S3 request traffic analysis (as described in "**Setting Up Elastic Stack for S3 Request Traffic Analysis**" (page 598)), contact Cloudian Sales Engineering or Support to discuss using the same Elastic Stack cluster for both object metadata search and S3 request log analysis.

4.11.4.2. Enabling Elasticsearch Integration for Object Metadata

To enable Elasticsearch integration in your HyperStore system:

1. On your Puppet master node, make these configuration file edits:
 - In [common.csv](#), edit the `cloudian_elasticsearch_hosts` setting to equal a comma-separated list of the IP addresses of your Elasticsearch hosts. In a production environment Cloudian recommends that you use at least three Elasticsearch hosts.
 - In [mts-ui.properties.erb](#), set the `elasticsearch.enabled` property to `true`. By default it is `false`.
 - In [mts.properties.erb](#):
 - If your Elasticsearch cluster **does not use the X-Pack extension**, set `cloudian.elasticsearch.xpack.enabled` to `false`. By default this is set to `true`.
 - If your Elasticsearch cluster **does use the X-Pack extension**:
 - Set `cloudian.elasticsearch.xpack.username` and `cloudian.elasticsearch.xpack.password` to the user name and password that you have already established in your Elasticsearch cluster set-up.
 - Optionally, change the `cloudian.elasticsearch.ssl.*` settings to suit your environment.

SSL can be enabled only when Xpack is enabled. To use SSL, in your ESDB you will first need to create a new keystore: `./keytool -import -alias elasticCA -file /etc/elasticsearch/certs/client/client-ca.cer -keystore truststore.jks`.

Then copy the `truststore.jks` file to each HyperStore node, in the directory path specified by `mts.properties.erb:cloudian.elasticsearch.ssl.truststore.path` (the default setting for the path is `/opt/cloudian/conf/certs/truststore.jks`).

2. On the Puppet master node, use the installer to push your configuration changes out to the cluster, restart the S3 Service, and restart the CMC. For more detail see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

Once you have enabled Elasticsearch integration in the system as described above, log into the CMC and go to the [Storage Policies](#) page. Now when you either create a new storage policy or edit an existing storage policy, the interface will display an "Enable metadata search" option that you can select to enable Elasticsearch integration for the policy.

Note Recall that when you enable Elasticsearch for a storage policy, that means that **from that point forward** objects that get uploaded into buckets that use the storage policy will have their object metadata copied into Elasticsearch. Enabling this option does **not** copy into Elasticsearch metadata from objects that are already stored in buckets that use the storage policy.

4.11.4.3. Using the elasticsearchSync Tool

HyperStore includes a tool that you can use to synchronize object metadata in your Elasticsearch cluster to object metadata currently in your HyperStore cluster. To use the tool you must first enable Elasticsearch integration for your HyperStore system and for one or more of your storage policies, as described in "**Enabling Elasticsearch Integration for Object Metadata**" (page 108). Then, using different script options you can bring object metadata in Elasticsearch up to date for any of the following:

- All buckets for which Elasticsearch integration is enabled (that is, all buckets that use Elasticsearch-enabled storage policies).
- A single bucket for which Elasticsearch integration is enabled (a bucket that uses an Elasticsearch-enabled storage policy).
- A single object in a bucket for which Elasticsearch integration is enabled.

The primary use case for the synchronization tool is to populate Elasticsearch with object metadata that was already in your HyperStore system before you enabled Elasticsearch integration. For example, suppose you enable Elasticsearch integration for one or more of your existing storage policies that are already being used by existing buckets. HyperStore will automatically copy into Elasticsearch the metadata associated with objects that get uploaded into those buckets from that point forward. But if you want to copy into Elasticsearch **all** the metadata associated with objects that are currently in those buckets -- including objects that were already in the buckets before Elasticsearch integration was enabled -- you need to use the sync tool.

Another use case for the sync tool is if your Elasticsearch cluster has been down or unreachable for more than a few hours. HyperStore uses an [hourly cron job](#) to apply needed updates to object metadata in Elasticsearch. Elasticsearch update requests that fail during one run of the cron job are retried during the next run of the cron job. But if your Elasticsearch cluster is unreachable for more than a few hours, the Cassandra-based queue for unprocessed Elasticsearch update requests can get filled up. Consequently if Elasticsearch has been unreachable for more than a few hours you should use the sync tool when Elasticsearch comes back online (for all

Elasticsearch-enabled buckets). This will bring Elasticsearch up to date with HyperStore object uploads and deletions that occurred while Elasticsearch was unavailable.

The sync tool is located on each of your HyperStore nodes, under the `/opt/cloudian/bin` directory. Once you change into that directory, the tool syntax is as follows:

Sync all Elasticsearch-enabled buckets:

```
./elasticsearchSync all
```

Sync one Elasticsearch-enabled bucket:

```
./elasticsearchSync bucket <bucketname>
```

Sync one object in an Elasticsearch-enabled bucket:

```
./elasticsearchSync object <bucketname>/<objectname>
```

Sync one version of one object in an Elasticsearch-enabled bucket:

```
./elasticsearchSync object <bucketname>/<objectname> <versionid>
```

Note Along with performing your specified action, the running of the sync tool always triggers the processing of any Elasticsearch update requests that are currently in the retry queue.

4.11.4.4. Option to Send Metadata to an HTTP Server Rather than to Elasticsearch

HyperStore supports a configurable option to send object metadata to an HTTP server of your choosing. If you use this option, HyperStore will send object metadata to your specified HTTP URI **instead of** sending it to Elasticsearch. For more information see the description of the "**cloudian.elasticsearch.process.type**" (page 520) setting in *mts.properties.erb*.

4.12. Quality of Service

4.12.1. Quality of Service (QoS) Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Service Usage Types Subject to QoS Controls"** (page 111)
- **"QoS Assignment Granularity"** (page 111)

The Cloudian HyperStore system supports user-level and group-level Quality of Service (QoS) settings:

- **User QoS settings** place upper limits on service usage by individual users.
- **Group QoS settings** place upper limits on aggregate service usage by entire user groups.

The HyperStore system enforces QoS settings by rejecting S3 requests that would result in a user (or a user's group) exceeding the allowed service usage level.

4.12.1.1. Service Usage Types Subject to QoS Controls

Several types of service usage metrics can be configured for QoS controls:

- Storage quota, by number of KBs.
- Storage quota, by number of objects.
- Peak HTTP request rate, in requests per minute. The user is not allowed more than this many requests in a 60 second interval.
- Peak data upload rate, in KBs per minute.
- Peak data download rate, in KBs per minute.

When configuring QoS controls, you have the option of limiting some of the usage types above while leaving others unrestricted. For example, you could limit per-user and/or per-group storage volume (by KBs), while placing no restrictions on number of stored objects. Similarly, you could cap data upload rate while placing no cap on data download rate.

When the system rejects a user request because of a storage quota, it returns an HTTP 403 response to the client application. When the system rejects a user request due to rate controls, it returns an HTTP 503 response to the client application.

For HTTP request rate and for upload and download rates, the system also supports a **configurable warning level** -- which you can set to a lower threshold of usage than the threshold at which requests will be rejected. If a user's request results in the warning threshold being exceeded, the request will succeed but the system will log an INFO level message to the S3 Service application log. (Note that the system does not inform the user that the warning threshold has been exceeded -- it only writes the aforementioned log message.)

Note The storage overhead associated with replication or erasure coding does not count toward a user's storage quota. For example, a 1MB object that is protected by 3X replication or by 4+2 erasure coding counts as only 1MB toward the storage quota.

Note For information on how auto-tiering impacts the implementation QoS controls, see "**How Auto-Tiering Impacts Usage Tracking, QoS, and Billing**" (page 56).

4.12.1.2. QoS Assignment Granularity

The system also provides you the flexibility to assign different QoS settings to different users and groups. In the case of user QoS settings, the system provides you three levels of granularity:

- For the system as a whole, you can configure a **system default for user QoS settings**, applicable to all users of your S3 service.
- For particular groups, you can configure a **group-specific default for user QoS settings**. If you do, then for users in that group, the group-specific user QoS defaults will override the system-wide user QoS defaults.
- For particular users, you can configure **user-specific QoS settings**. If you do, these settings will override any group-wide or system-wide defaults.

For group QoS settings you have two levels of granularity:

- For the system as a whole, you can configure a **system default for group QoS settings**, applicable to all groups in your S3 service.
- For particular groups, you can configure **group-specific group QoS settings**. If you do, these settings will override the system-wide defaults.

Note for multi-region systems

If your HyperStore service has multiple service regions, the system also provides you the ability to configure different QoS settings for different regions. For example, you might configure default user QoS settings that allow users 20GB of storage in your "North" service region and 30GB in your "South" service region.

4.12.2. Enabling QoS Enforcement

By default, the HyperStore system's QoS functionality is **disabled**. Before enabling the functionality, think first about whether you want to implement QoS controls based just on storage quotas, or if you also want to apply QoS controls for request rates (HTTP requests and upload/download bytes). This choice impacts the configuration changes that you will make. In general, for optimal system performance you should enable only the QoS functionality that you actually intend to apply to service users.

Note that enabling QoS functionality as described below, you are merely "turning on" the S3 Service mechanisms that enforce whatever QoS restrictions you establish for users and groups. The creation of specific QoS restrictions is a separate administrative task as described in "**Setting QoS Limits for Users**" (page 113) and "**Setting QoS Limits for Groups**" (page 112).

To enable HyperStore QoS enforcement, in the CMC go to the [Configuration Settings](#) page and open the **Quality of Service** panel. Then:

- To enable enforcement of **storage quotas only** (number of bytes and/or number of objects), set just the "QoS Limits" setting to "enabled".
- To enable enforcement of **storage quotas and also traffic rates** (number of HTTP requests per minute, or bytes uploaded or downloaded per minute), set both the "QoS Limits" setting and the "QoS Rate Limits" setting to "enabled".

After you **Save**, your configuration changes are applied to the system dynamically — no service restart is required.

Note Enforcing QoS for traffic rates but not for stored bytes and objects is not supported at the system configuration level. If you want to use QoS in this way, set both "QoS Limits" and "QoS Rate Limits" to enabled, then when you're configuring QoS limits for groups and users set the stored bytes and objects controls to unlimited and the rate controls to your desired levels.

4.12.3. Setting QoS Limits for Groups

As with user QoS settings, you can set group QoS settings through either the CMC or the Admin API. Group QoS settings limit the **aggregate activity** of all users in a group.

Setting QoS Limits for Groups (CMC)

First you should decide whether you want to set default QoS limits for all groups in your HyperStore system. If

so, you can do this by going to the CMC's [Manage Groups](#) page and clicking **Group QoS Default** to open the **Group QoS Limits: Defaults** panel. Here you can configure default group QoS limits for your whole system.

Next, you can set group QoS limits for a specific group. If you do so, these limits will override any system defaults for group QoS. To do this, first retrieve the group in the [Manage Groups](#) page. Then click **Group QoS** for the group. This opens the **Group QoS Limits: Overrides** panel, where you can configure group QoS limits for that specific group.

Note For details about working with the CMC's QoS configuration panels see "[Set Quality of Service \(QoS\) Controls](#)" (page 250).

Setting QoS Limits for Groups (Admin API)

To establish group QoS settings through the Admin API, you use the same method that you do for user QoS settings: [POST /qos/limits](#). URI parameters enable you to specify that you're creating system defaults for group QoS settings, or settings for a particular group.

The [GET /qos/limits](#) and [DELETE /qos/limits](#) methods can be used to retrieve or delete group QoS settings.

4.12.4. Setting QoS Limits for Users

By system default, QoS controls for all service usage types — storage quota by bytes, storage quota by number of objects, HTTP requests per minute, upload bytes per minute, and download bytes per minute — are set to "unlimited". So when you [enable QoS enforcement by the system](#), service users still are not subject to QoS controls until you establish specific QoS limits for the various service usage types.

You can set QoS limits for users through either the CMC or the Admin API.

Setting QoS Limits for Users (CMC)

First you should decide whether you want to set default QoS limits for all users of your HyperStore system. If so, you can do this by going to the CMC's [Manage Users](#) page and clicking **User QoS Default** to open the **User QoS Limits: Defaults** panel, where you can configure default user QoS limits for your system.

Next, decide whether you want to set default QoS limits for all users who belong to a particular user group. If you do so, this will override the system-wide default, for users within that group. To do this, first retrieve the group in the [Manage Groups](#) page. Then click **User QoS Group Default** for the group. This opens **User QoS Limits: Group Defaults** panel, where you can configure default user QoS limits for the group.

Finally, you also have the option of setting QoS limits for a specific individual user. If you do so, these limits will override any system or group default limits, for that user. To do this, first retrieve the user in the [Manage Users](#) page, then click **Set QoS** for the user. This opens the **User QoS Limits: Overrides** panel, where you can configure QoS limits for that specific user.

Note For details about working with the CMC's QoS configuration panels see "[Set Quality of Service \(QoS\) Controls](#)" (page 250).

Setting QoS Limits for Users (Admin API)

With the Admin API method [POST /qos/limits](#) you can set QoS limits for HyperStore service users. With the

method's URI parameters you can specify whether you're setting default limits for all users in the system; or default limits for all users within a specified group; or limits for a specific user. URI parameters also enable you to set the numerical limits for each service usage type — for example, a Storage Bytes limit of 10GB.

The Admin API also supports:

- A [GET /qos/limits](#) method, for retrieving system-default, group-default, or user-specific QoS limits
- A [DELETE /qos/limits](#) method, for deleting system-default, group-default, or user-specific QoS limits

4.13. S3 Interface

4.13.1. S3 Storage Interface Feature Overview

Broadly you have three options for using the S3 API to put objects into the HyperStore object store, retrieve objects, and so on:

- [Use the Cloudian Management Console \(CMC\)](#). The CMC has a built-in S3 client that allows you — and your users, if you give them access to the CMC — to interact with the HyperStore object store through a graphical interface.
- [Use an off-the-shelf third party S3 client](#) to directly invoke HyperStore's implementation of the S3 API.
- [Develop and use a custom S3 client](#) to directly invoke HyperStore's implementation of the S3 API.

Note Along with providing comprehensive support for the Amazon S3 storage API, HyperStore provides limited support for the Amazon Identity and Access Management (IAM) API. For more information see "[Identity and Access Management \(IAM\) Feature Overview](#)" (page 91).

4.13.1.1. Limitations and Cautions

Maximum Objects Per S3 Bucket

Note The description below about maximum objects per bucket **applies only to buckets created in HyperStore 7.0.x or older**. Buckets created in HyperStore 7.1 and later use a redesigned metadata scheme that eliminates the two billion objects limitation and allows for very many billions of objects per bucket.

In the Cassandra column family CLOUDIAN_METADATA, the HyperStore system stores S3 object metadata on a per-bucket basis. In this column family there is one row for each S3 bucket. In each such row there is one column for each object in the bucket. This object metadata storage architecture has implications for S3 bucket capacity and client application design:

- Because Cassandra has a maximum of two billion columns per row, the HyperStore system has a maximum of two billion S3 objects per bucket.
- Rather than pushing the limits of capacity for a single bucket, which may negatively impact system performance, it's best to create multiple buckets and spread S3 object uploads across the multiple buckets.

Mass Deletes

Do not attempt to delete more than 100,000 objects from a single bucket in less than an hour (using the S3 API method [DELETE Multiple Objects](#)). Doing so will result in TombstoneOverwhelmingException errors in the Cassandra logs and an inability to successfully execute an [S3 GET Bucket \(List Objects\) Version 1](#) or [GET Bucket \(List Objects\) Version 2](#) operation on the bucket. If the system is in this error condition, you can trigger a tombstone purge as described in "[Dealing with Excessive Tombstone Build-Up](#)" (page 423).

4.13.1.2. Using TLS/SSL

By default the HyperStore S3 Service does not accept HTTPS (HTTP over TLS/SSL) connections. It listens only for regular HTTP connections, on port 80. However you can set up HTTPS support for the S3 Service as described in "[TLS/SSL for S3 Service \(Self-Signed Certificate\)](#)" (page 543) or "[TLS/SSL for S3 Service \(CA-Verified Certificate\)](#)" (page 547). If you do so then the S3 Service will listen for HTTPS connections on port 443, as well as listening for regular HTTP connections on port 80.

4.13.2. Using the CMC as Your S3 Client

The CMC's **Buckets & Objects** section serves as a graphical S3 client for interacting with the HyperStore object store. With the CMC, users can do the following:

4.13.2.1. Work with Buckets

- "[Add a Bucket](#)" (page 192)
- Set bucket properties
 - "[Set Custom S3 Permissions for a Bucket](#)" (page 194)
 - "[Set "Canned" S3 Permissions for a Bucket](#)" (page 196)
 - "[View a Bucket's Storage Policy Information](#)" (page 197)
 - "[Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration](#)" (page 197)
 - "[Configure a Bucket as a Static Website](#)" (page 205)
 - "[Configure Cross-Region Replication for a Bucket](#)" (page 207)
 - "[Set Versioning for a Bucket](#)" (page 209)
 - "[Set Logging for a Bucket](#)" (page 211)
- "[Delete a Bucket](#)" (page 214)

4.13.2.2. Work with Objects

- "[Create or Delete a "Folder"](#)" (page 215)
- "[Upload an Object](#)" (page 217)
- Set file properties
 - "[Set Custom S3 Permissions on an Object](#)" (page 220)
 - "[Set "Canned" S3 Permissions on an Object](#)" (page 221)
 - "[Set Public URL Permissions on an Object](#)" (page 223)
- "[Search for an Object](#)" (page 224)

- "**Download an Object**" (page 225)
- "**Delete an Object**" (page 228)
- "**Restore an Auto-Tiered Object**" (page 225)

Note The CMC system administrator role does not and cannot have its own S3 storage user account. However you can [create a regular user account](#) for yourself, and use that to access the data store. You can also [manage other regular users' data](#) on their behalf.

Note When the CMC or other S3 client applications delete S3 objects, the HyperStore system deletes the object metadata immediately but does not delete the actual objects immediately. Instead the objects are [batched for deletion by a cron job](#). When S3 clients overwrite S3 objects, the HyperStore system writes the new version of the object immediately, and updates the object metadata immediately, but does not delete the outdated version of the object immediately. Instead the outdated object versions are batched for deletion by the same cron job. (Note that in a bucket that has [versioning](#) enabled, the old object versions would be retained rather than deleted.)

4.13.3. Using Third Party S3 Applications

HyperStore supports nearly all of the Amazon S3 REST API. HyperStore's compliance with Amazon's S3 API is covered in detail in the "S3 API Support" section of the HyperStore documentation. That section also covers HyperStore extensions to the Amazon S3 API.

Because of HyperStore's comprehensive compliance with the Amazon S3 API, **you can use most off-the-shelf third party S3 client applications with HyperStore**. For feedback on particular S3 applications that you are considering using with HyperStore, consult with Cloudian Sales Engineering or Cloudian Support.

To check to see what is your HyperStore S3 Service endpoint -- the URI to which you will submit S3 requests with your third party application -- go to the CMC's [Cluster Information](#) page.

4.13.4. Developing Custom S3 Applications for HyperStore

In nearly every way, developing a client application for the Cloudian HyperStore storage service is the same as developing a client application for Amazon S3. Consequently, when building S3 applications for the HyperStore service you can leverage the wealth of resources available to Amazon S3 developers.

Good online resources for S3 application developers include:

- [Amazon Simple Storage Service Developer Guide](#)
- [Amazon S3 resources](#)

4.13.4.1. What's Distinct About Developing for the HyperStore S3 Service

In practice, the main differences between developing for the HyperStore S3 service and developing for Amazon S3 are:

- As detailed in the "S3 API" section of this documentation, the HyperStore S3 service supports the great majority of but not the entire Amazon S3 API.
- Also as detailed in the "S3 API" section of this documentation, the HyperStore S3 service supports a small number of extensions to the Amazon S3 API. (For an overview of the extensions see "**Hyper-Store Extensions to the S3 API**" (page 890)).
- HyperStore S3 client applications must use the HyperStore S3 service endpoint (viewable on the CMC's **Cluster Information** page) rather than the Amazon S3 service endpoint.
- Either the CMC or the HyperStore Admin API must be used to [create and manage HyperStore service user accounts](#).

4.14. Server-Side Encryption

4.14.1. Server-Side Encryption Feature Overview

Subjects covered in this section:

- Introduction (immediately below)*
- "Encryption Configuration Changes Do Not Apply Retroactively"** (page 118)
- "Encryption and Auto-Tiering"** (page 118)
- "Encryption and Cross-Region Replication"** (page 119)

The HyperStore system supports server-side encryption (SSE) to protect the confidentiality of data at rest. Several different methods of server-side encryption are supported:

- Encryption using a HyperStore **system-generated encryption key** (regular SSE)
- Encryption using a **customer-provided encryption key** (SSE-C)
- Encryption using encryption keys managed by the **Amazon Web Services Key Management Service** (AWS KMS)
- Encryption using encryption keys managed by a **Gemalto KeySecure KMS**

The selection of whether to use server-side encryption, and which method to use, can be made at the object level (as specified by headers in the object upload request), or the bucket level (so that a default encryption method is applied to all objects uploaded to the bucket), or the storage policy level (so that a default encryption method is applied to all objects uploaded to any bucket that uses the storage policy). When the system is processing a given object upload, the precedence ordering among these different configuration levels is as follows:

- If a server-side encryption method is specified in the object upload request, the system uses that method. If not, then...
- If a default server-side encryption method is specified in the configuration of the bucket to which the object is being uploaded, the system uses that method. If not, then...
- If a default server-side encryption method is specified in the configuration of the storage policy used by the bucket to which the object is being uploaded, the system uses that method.

Put differently, any encryption configuration specified by object upload request headers takes precedence over the bucket default configuration; and the bucket default configuration takes precedence over the storage policy configuration. If no encryption method is specified in either the object upload request, the configuration of the

bucket into which the object is being uploaded, or the configuration of the storage policy used by the bucket, then no server-side encryption is applied to that object.

For more information about using the supported server-side encryption methods, see:

- "[Using Regular Server-Side Encryption \(SSE\)](#)" (page 119)
- "[Using Server-Side Encryption with Customer-Provided Keys \(SSE-C\)](#)" (page 121)
- "[Using Server-Side Encryption with AWS KMS](#)" (page 123)
- "[Using Server-Side Encryption with Gemalto KeySecure KMS](#)" (page 125)

4.14.1.1. Encryption Configuration Changes Do Not Apply Retroactively

You cannot apply server-side encryption retroactively to objects that have already been uploaded to the system. For example, if you modify a bucket's configuration so that it includes server-side encryption, this will apply only to objects uploaded from that time forward -- not to objects that had been uploaded to the bucket previously. The same is true of adding server-side encryption to a storage policy's configuration: from that time forward, objects that get uploaded into buckets that use that storage policy will be encrypted, but objects that had already been uploaded previously will not be encrypted.

Conversely, if a bucket configuration or storage policy configuration uses server-side encryption and then you subsequently disable encryption for that bucket or storage policy, then from that time forward newly uploaded objects will not be encrypted -- but objects that had already been uploaded and encrypted prior to the configuration change will remain encrypted.

4.14.1.2. Encryption and Auto-Tiering

How the HyperStore system handles auto-tiering of server-side encrypted objects depends on the encryption method used and the tiering destination.

Encryption in HyperStore	Tiering Destination	Object Handling
Regular SSE	Amazon or Google	HyperStore decrypts the object, then tiers it to the destination system and specifies server-side encryption by the AES-256 method in the upload request. The destination system encrypts the tiered object using AES-256.
	Azure or Spectra BlackPearl	HyperStore decrypts the object, then tiers it to the destination system in decrypted form. These destinations do not support a server-side encryption option in object upload requests. However, Azure does support an account-level option to encrypt all incoming objects. See your Azure documentation for more information.
SSE-C	Any	HyperStore tiers the encrypted object to the destination system, where it remains encrypted. To retrieve such tiered objects, client applications must first restore the objects into HyperStore, then get the objects (with the customer-provided encryption keys included).

Encryption in HyperStore	Tiering Destination	Object Handling
		in the get request). Streaming gets of such tiered objects are not supported.
AWS KMS	Amazon	HyperStore decrypts the object, then tiers it to the destination system and specifies encryption by the AWS KMS method in the upload request. Amazon encrypts the tiered object using the AWS KMS method.
	Any other than Amazon	For objects in HyperStore that are encrypted by the AWS KMS method, tiering to destinations other than Amazon is not supported. Such objects will remain in their HyperStore bucket and not be tiered, even if they have reached the configured tiering age.
Gemalto KeySecure	Amazon or Google	HyperStore decrypts the object, then tiers it to the destination system and specifies server-side encryption by the AES-256 method in the upload request. The destination system encrypts the tiered object using AES-256.
	Azure or Spectra BlackPearl	HyperStore decrypts the object, then tiers it to the destination system in decrypted form. These destinations do not support a server-side encryption option in object upload requests. However, Azure does support an account-level option to encrypt all incoming objects. See your Azure documentation for more information.

4.14.1.3. Encryption and Cross-Region Replication

When cross-region replication is configured for a source bucket:

- Objects encrypted by the regular SSE method or the Gemalto KeySecure KMS method **are** replicated to the target replication bucket.
- Objects encrypted by the SSE-C method or the AWS KMS method **are not** replicated to the target replication bucket.

4.14.2. Using Regular Server-Side Encryption (SSE)

Subjects covered in this section:

- Introduction (immediately below)*
- "Recommended System Set-Up Before Using SSE"** (page 120)
- "Requesting SSE for Specific Objects (CMC or S3 API)"** (page 120)

- "**Configuring SSE as the Default for a Bucket (S3 API Only)**" (page 121)
- "**Configuring SSE as the Default for a Storage Policy (CMC Only)**" (page 121)

Regular server-side encryption (SSE) can be configured at the object level, the bucket level, or the storage policy level. For information about the precedence ordering among these levels see "**Server-Side Encryption Feature Overview**" (page 117).

When regular SSE is used the HyperStore system generates a unique encryption key for each object that the system encrypts, and the encryption key is stored as part of the object's metadata.

4.14.2.1. Recommended System Set-Up Before Using SSE

With regular SSE, the HyperStore system generates the encryption keys using AES-128 by default. While AES-128 will work for regular SSE, and may be acceptable in a testing or evaluation environment, for greater security Cloudian, Inc. recommends using AES-256. You can enable AES-256 in your HyperStore system as described in "**Enabling AES-256**" (page 127).

Note Amazon uses AES-256 for its regular SSE implementation, and AES-256 is called for in Amazon's SSE specification.

4.14.2.2. Requesting SSE for Specific Objects (CMC or S3 API)

Regular SSE can be set for specific objects as those objects are uploaded to the HyperStore system. This can be done either through the CMC or through a third party S3 client application that invokes the HyperStore implementation of the S3 API.

Object-Level SSE Through the CMC

In the CMC's **Buckets & Objects** page, where a user can upload objects into the HyperStore system, the interface displays a "Store Encrypted" checkbox. If the user selects this checkbox, the HyperStore system applies regular server-side encryption to the uploaded object(s). For more information see "**Upload an Object**" (page 217).

Object-Level SSE Through the S3 API

In compliance with the Amazon S3 REST API, the HyperStore system's S3 API implementation supports regular server-side encryption by the inclusion of the *x-amz-server-side-encryption: AES256* request header in any of these operations on objects:

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [POST Object](#)
- [PUT Object - copy](#)

Note If you have not [enabled AES-256](#) in your HyperStore system, the system will use AES-128 encryption even though the *x-amz-server-side-encryption* request header specifies AES256.

4.14.2.3. Configuring SSE as the Default for a Bucket (S3 API Only)

In compliance with the Amazon S3 REST API, the HyperStore system's S3 API supports setting and managing a bucket default server-side encryption method by the use of these operations:

- [**PUT Bucket encryption**](#)
- [**GET Bucket encryption**](#)
- [**DELETE Bucket encryption**](#)

To configure a bucket for regular SSE, in the *PUT Bucket encryption* request body set the *SSEAlgorithm* element to *AES256*.

Note If you have not [enabled AES-256](#) in your HyperStore system, the system will use AES-128 encryption even though the *SSEAlgorithm* element specifies *AES256*.

Note The CMC does not support setting a bucket default server-side encryption method.

4.14.2.4. Configuring SSE as the Default for a Storage Policy (CMC Only)

When you use the CMC to create storage policies, one of the configurable policy attributes is server-side encryption. To configure a storage policy to use regular SSE, in the "Server-Side Encryption" field of the storage policy configuration interface, select "SSE".

For more information on storage policy configuration see:

- **"Add a Storage Policy"** (page 308)
- **"Edit a Storage Policy"** (page 331)

4.14.3. Using Server-Side Encryption with Customer-Provided Keys (SSE-C)

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Required System Set-Up Before Using SSE-C" (page 122)**
- **"Requesting SSE-C for Specific Objects (S3 API Only)" (page 122)**

Server-side encryption with customer-provided encryption keys (SSE-C) can only be set at the per-object level, and only if you use a third party S3 client that supports requesting this type of encryption. Setting SSE-C as the default encryption method for a bucket or a storage policy is not supported.

When SSE-C is used, the HyperStore system does not store the customer-provided encryption key itself but rather stores a hash of the key (for purposes of verifying the key if it's subsequently submitted in a GET Object request). The key hash is stored with the object metadata.

IMPORTANT: When SSE-C is used, the user is responsible for managing the encryption key. If an object is uploaded to HyperStore system and encrypted with a user-provided key, the user will need to

provide that same key when later requesting to download the object. **If the user loses the key, the encrypted object will not be downloadable.** This is consistent with Amazon's implementation of the SSE-C feature. For more information on Amazon's SSE-C feature see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\)](#).

4.14.3.1. Required System Set-Up Before Using SSE-C

Before using SSE-C you must first:

- Enable AES-256 in your HyperStore system. Like Amazon S3, HyperStore's implementation of SSE-C requires AES-256 encryption. For instructions see "[Enabling AES-256](#)" (page 127).
- Set up HTTPS for your S3 Service. Like Amazon S3, HyperStore's implementation of SSE-C requires that the relevant S3 API requests be transmitted over HTTPS rather than regular HTTP. For instructions see "[TLS/SSL for S3 Service \(Self-Signed Certificate\)](#)" (page 543) or "[TLS/SSL for S3 Service \(CA-Verified Certificate\)](#)" (page 547)).

Note: HyperStore supports a configuration for allowing a regular HTTP connection between a load balancer and your S3 servers for transmission of SSE-C requests over your internal network, while client applications use HTTPS in the requests that come into the load balancer. See the configuration parameter *mts.properties.erb:"cloudian.s3.ssec.usessl"* (page 519).

4.14.3.2. Requesting SSE-C for Specific Objects (S3 API Only)

In compliance with the Amazon S3 REST API, the HyperStore system's S3 API supports server-side encryption with user-provided keys by the inclusion of the *x-amz-server-side-encryption-customer-** request headers in any of these operations on objects:

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [POST Object](#)
- [PUT Object - copy](#) (supporting also the *x-amz-copy-source-server-side-encryption-customer-** request headers)
- [GET Object](#)
- [HEAD Object](#)

For the full list of supported *x-amz-server-side-encryption-customer-** request headers for each of these operations, follow the links above.

Note The CMC does not support requesting SSE-C for objects as they are uploaded, and it does not support downloading objects that have been encrypted by the SSE-C method.

4.14.4. Using Server-Side Encryption with AWS KMS

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Required System Set-Up Before Using AWS KMS Encryption" (page 123)**
- **"Requesting AWS KMS Encryption for Specific Objects (S3 API Only)" (page 124)**
- **"Configuring AWS KMS Encryption as the Default for a Bucket (S3 API Only)" (page 125)**
- **"Deleting a Bucket That Uses AWS KMS Encryption" (page 125)**

Server-side encryption using encryption keys managed by the Amazon Web Services Key Management Service (AWS KMS) can be configured at the object level or the bucket default level, only if you use a third party S3 client that supports requesting this type of encryption. Setting the AWS KMS method as the default for a storage policy is not supported. For information about the precedence ordering between these levels see "**Server-Side Encryption Feature Overview**" (page 117).

When AWS KMS based encryption is used the HyperStore system triggers in the remote AWS KMS the creation of one "customer master key" (CMK) per bucket. The CMK is stored exclusively in the AWS KMS. For each such CMK, HyperStore stores (in Redis) a CMK ID that allows HyperStore to tell AWS KMS which CMK to use for creating an encrypted "data key" for a given object (that is, the CMK for the bucket into which the object is being uploaded). AWS KMS returns to HyperStore both the encrypted data key and plain text version of the data key. After using the plain text data key to encrypt the object, HyperStore deletes the plain text data key, while the encrypted version of the data key is stored within the encrypted object data.

Subsequently when a client downloads the object, HyperStore submits the encrypted data key to the AWS KMS, which uses the CMK to decrypt the data key. AWS KMS returns the decrypted data key to HyperStore, which uses it to decrypt the object and then deletes the decrypted data key from memory.

Note In compliance with Amazon's implementation, all S3 requests involving AWS KMS encryption must use SSL and Signature Version 4. For example HyperStore will reject object upload requests that specify AWS KMS encryption, or download requests for AWS KMS encrypted objects, if such requests use Signature Version 2.

Note For more information on the AWS KMS, see [AWS Key Management Service \(KMS\)](#).

4.14.4.1. Required System Set-Up Before Using AWS KMS Encryption

To use the AWS KMS for HyperStore server-side encryption, you must have either or a combination of:

- **AWS account access credentials for each HyperStore user group** that will use the AWS KMS encryption feature. These group account credentials will be used by HyperStore to access the AWS KMS whenever a user within the group requests AWS KMS encryption for their bucket or for specific objects, or downloads AWS KMS encrypted objects.
- **Default AWS account access credentials** for your HyperStore service as a whole. These default AWS credentials will be used by HyperStore to access the AWS KMS on behalf of users who are in groups that do not have group account credentials for AWS.

To enable AWS KMS usage in your HyperStore system, complete the following system configuration steps:

1. On your Puppet master node, open the following file in a text editor:

```
/etc/cloudian-<version>-puppet/modules/cloudians3/files/awscredentials.properties
```

2. Edit the file to specify the system default AWS access credentials, and (optionally) any group-specific AWS access credentials. Create a separate block for each group that has its own AWS access credentials, using the formatting shown in the example below. In the example there are credentials for just one group, "CloudianTest1". HyperStore will use the CloudianTest1 group's AWS credentials when accessing the AWS KMS on behalf of users in that group. For users in any other group, HyperStore will use the system default AWS credentials when accessing the AWS KMS.

```
[default]
aws_access_key_id = AKIAJKVELYABCCEIXXMA
aws_secret_access_key = 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

[CloudianTest1]
aws_access_key_id = ABCAJKVELY6YXCEIMAXX
aws_secret_access_key = abceikWvRR/7A8916x9vUDEhV+C+LIDmFCOE8MgC
```

Save your change and close the file.

3. Still on your Puppet master node, open the following file in a text editor:

```
/etc/cloudian-<version>-puppet/modules/cloudians3/templates/mts.properties.erb
```

4. Find the property `util.awskmsutil.region` and set it to the AWS service region of the AWS KMS that you want HyperStore to use. The default is "us-east-1". Save your change and close the file.
5. Push your changes to the cluster and restart the S3 Service. If you need instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.14.4.2. Requesting AWS KMS Encryption for Specific Objects (S3 API Only)

In compliance with the Amazon S3 REST API, the HyperStore system's S3 API supports AWS KMS based server-side encryption by the inclusion of the `x-amz-server-side-encryption: aws:kms` request header in any of these operations on objects:

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [POST Object](#)
- [PUT Object - copy](#)

Note The HyperStore S3 Service does not support the `x-amz-server-side-encryption-aws-kms-key-id` or `x-amz-server-side-encryption-context` request headers.

Note The CMC does not support requesting AWS KMS based encryption for objects as they are uploaded. It **does** support downloading objects that have been encrypted by the AWS KMS method.

4.14.4.3. Configuring AWS KMS Encryption as the Default for a Bucket (S3 API Only)

In compliance with the Amazon S3 REST API, the HyperStore system's S3 API supports setting and managing a bucket default server-side encryption method by the use of these operations:

- [**PUT Bucket encryption**](#)
- [**GET Bucket encryption**](#)
- [**DELETE Bucket encryption**](#)

To configure a bucket for server-side encryption with AWS KMS, in the *PUT Bucket encryption* request body set the *SSEAlgorithm* element to *aws:kms*.

Note The CMC does not support setting a bucket default server-side encryption method.

4.14.4.4. Deleting a Bucket That Uses AWS KMS Encryption

When you delete a HyperStore bucket that has used AWS KMS encryption -- either because AWS KMS encryption was the default encryption method for the bucket, or because certain objects within the bucket used AWS KMS encryption -- the bucket's "customer master key" (CMK) is scheduled for deletion from the remote AWS KMS system. The CMK deletion occurs 30 days after the deletion of the HyperStore bucket. During this 30 day period, if you do not wish the CMK to be deleted from the remote AWS KMS system you can execute a *CancelKeyDeletion* operation using the AWS Console or the AWS CLI.

4.14.5. Using Server-Side Encryption with Gemalto KeySecure KMS

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Required System Set-Up Before Using Gemalto KeySecure Encryption" (page 125)**
- **"Requesting Gemalto KeySecure Encryption for Specific Objects (CMC or S3 API)" (page 126)**
- **"Configuring Gemalto KeySecure Encryption as the Default for a Storage Policy (CMC Only)" (page 127)**

Server-side encryption utilizing a Gemalto KeySecure KMS can be set at either the per-object level or the storage policy level. Setting Gemalto as the default encryption method for a specific bucket is not supported. For information about the precedence ordering between these levels see "**Server-Side Encryption Feature Overview**" (page 117).

With this type of server-side encryption, the encryption keys are created, stored, and managed by a Gemalto KeySecure KMS.

4.14.5.1. Required System Set-Up Before Using Gemalto KeySecure Encryption

To use server-side encryption with a Gemalto KeySecure KMS you must first configure HyperStore to support this feature. It does not support it by default.

There are three parts to the configuration set-up:

1. To use Gemalto KeySecure encryption you must configure a connection from HyperStore to the Gemalto KeySecure KMS. This involves using hidden, undocumented configuration properties. For instructions on configuring the connection, **contact Cloudian Support**.
2. If you want users to be able to request Gemalto KeySecure encryption for specific objects (as part of the object upload request), add the following setting to [**mts.properties.erb**](#) on your Puppet master node:

```
sse.logic.custom=GE
```

3. If you want to be able to set Gemalto KeySecure as the default server-side encryption method for a storage policy (when you're configuring storage policies in the CMC), make the following edit to [**mts-ui.properties.erb**](#) on your Puppet master node:

```
#DEFAULT:  
keysecure.encryption.enabled=false  
  
#CHANGE TO:  
keysecure.encryption.enabled=true
```

After making configuration file changes, push the changes out to the cluster and then restart the S3 Service (to apply the changes from Step 1 and Step 2) and restart the CMC (to apply the change from Step 3). If you need instructions for doing this see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.14.5.2. Requesting Gemalto KeySecure Encryption for Specific Objects (CMC or S3 API)

Gemalto KeySecure encryption can be set for specific objects as those objects are uploaded to the HyperStore system. This can be done either through the CMC or through a third party S3 client application that invokes the HyperStore implementation of the S3 API.

Note For these methods to work, you must first set `sse.logic.custom=GE` in `mts.properties.erb` as described in "**Required System Set-Up Before Using Gemalto KeySecure Encryption**" (page 125). Without that configuration change, the methods below will invoke regular SSE -- not Gemalto KeySecure encryption.

Object-Level SSE Through the CMC

In the CMC's **Buckets & Objects** page, where a user can upload objects into the HyperStore system, the interface displays a "Store Encrypted" checkbox. If the user selects this checkbox, the HyperStore system applies Gemalto KeySecure encryption to the uploaded object(s). For more information see "**Upload an Object**" (page 217).

Object-Level SSE Through the S3 API

The HyperStore system's S3 API implementation supports Gemalto KeySecure encryption by the inclusion of the `x-amz-server-side-encryption: AES256` request header in any of these operations on objects:

- [**PUT Object**](#)
- [**Initiate Multipart Upload**](#)
- [**Upload Part**](#)
- [**POST Object**](#)
- [**PUT Object - copy**](#)

4.14.5.3. Configuring Gemalto KeySecure Encryption as the Default for a Storage Policy (CMC Only)

When you use the CMC to create storage policies, one of the configurable policy attributes is server-side encryption. To configure a storage policy to use Gemalto KeySecure encryption, in the "Server-Side Encryption" field of the storage policy configuration interface, select "KeySecure".

For more information on storage policy configuration see:

- **"Add a Storage Policy"** (page 308)
- **"Edit a Storage Policy"** (page 331)

Note For this method to work, you must first set `keysecure.encryption.enabled=true` in `mts-ui.-properties.erb` as described in **"Required System Set-Up Before Using Gemalto KeySecure Encryption"** (page 125). Without that change, the KeySecure option will not appear in the CMC's interface for storage policy configuration.

4.14.6. Enabling AES-256

By default the HyperStore system does not use AES-256, the most secure form of the Advanced Encryption Standard. Instead it uses AES-128.

You must enable AES-256 in your HyperStore system if you want to do either of the following:

- Use regular SSE in a manner compliant with the Amazon SSE specification
- Use SSE-C

Note Enabling AES-256 is not necessary for -- and not relevant to -- server-side encryption using a Gemalto KMS or AWS KMS.

To enable AES-256 in your HyperStore system, do the following:

1. On your Puppet master node, open this file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

2. In `common.csv`, set `cloudian_s3_aes256encryption_enabled` to `true`:

```
cloudian_s3_aes256encryption_enabled,true
```

Then save and close the file.

3. Push your changes out to the cluster and restart the S3 Service. For instructions see **"Pushing Configuration File Edits to the Cluster and Restarting Services"** (page 454).

AES-256 is now enabled in your HyperStore system.

4.15. Service Regions

4.15.1. Service Regions Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Default Service Region"** (page 128)
- **"Single-Region System"** (page 128)
- **"Multi-Region System Architecture"** (page 129)

Like Amazon S3, the HyperStore system supports the implementation of multiple "service regions". Setting up the HyperStore system to use multiple service regions is **optional**.

The main benefits of deploying multiple service regions are:

- Each region has its own independent HyperStore cluster for S3 object storage. Consequently, deploying multiple regions is another means of scaling out your overall HyperStore service offering (beyond using multiple nodes and multiple datacenters to scale out a single cluster). Note that in a multi-region deployment, entirely different S3 datasets are stored in each region. Each region has its own token space and by default there is no data replication across regions (although there is an option for [cross-region replication](#) on a bucket-to-bucket basis).
- With a multi-region deployment, your service users can choose the service region in which their storage buckets will be created. Users may, for example, choose to store their S3 objects in the region that's geographically closest to them; or they may choose one region rather than another for reasons of regulatory compliance or corporate policy.

Note that **deploying the HyperStore system in multiple data centers does not in itself constitute having multiple service regions**. You can have multiple data centers within a single service region. (For information on having multiple DCs within a region, see "**Data Centers Feature Overview**" (page 71).)

4.15.1.1. Default Service Region

In a HyperStore deployment with multiple service regions, one region serves as the "default region". The default region plays several roles in the context of a multi-region HyperStore system. For example:

- If service users do not specify a region when they create a new S3 storage bucket, the system will create the bucket in the default region.
- Only the Admin Service instances in the default service region support the full Admin API.

During installation of a multi-region HyperStore system, the install script prompts you to select one of your regions to serve as the default region.

4.15.1.2. Single-Region System

You may choose not to have multiple service regions. However, even if you have just one region, you still will need to give that region a name when you install your HyperStore system, and the region name will become part of your HyperStore system configuration.

4.15.1.3. Multi-Region System Architecture

A multi-region HyperStore system has the following characteristics:

- **Each region has its own S3 service endpoint** (URI used by client applications for HTTP access).
- **Each region has its own independent object storage cluster** and by default there is no object replication across regions (although there is an option for [cross-region replication](#) on a bucket-to-bucket basis).
- **Quality of service (QoS) controls are implemented separately in each region.** The QoS limits that you establish for a service region will be applied only to user activity in that particular region. In support of QoS implementation, each region has its own independent Redis QoS database. Each regional Redis QoS database has its own master node. In each region there is also one Redis QoS slave node per data center.
- **User access credentials are valid across the system as a whole.** In support of user authentication, a single, uniform Redis Credentials database serves the entire multi-region system. There is just one Redis Credentials master node for the whole system, and that node is located in your default region. Within each region, there is one Redis Credentials slave node per data center.
- The **Redis Monitor application will monitor all the Redis databases in all the regions** (and if necessary trigger failover of the Redis master role within each database). One primary Redis Monitor application instance serves the whole multi-region system, and if the primary Redis Monitor instance goes down the backup instance takes over. The primary Redis Monitor instance and backup Redis Monitor instance are on separate nodes in your default region.
- **Group and user profile information is stored only in the default region**, and is accessed there by services in the other regions. Group and user information is stored only in the Cassandra database in the default region. HyperStore services in non-default regions access Cassandra in the default region to retrieve this group and user information as needed.
- **Each CMC instance has access to and control over the entire multi-region system.** For more information see "[Using the CMC or Admin API in a Multi-Region System](#)" (page 136).
- **Just one Puppet master node is used** to propagate system configuration settings throughout the whole multi-region system, during system installation and for ongoing system configuration management.

4.15.2. Setting Up a Multi-Region System

The HyperStore interactive installer makes it easy to install your HyperStore system as a multi-region system. From a single node on which you are running the installer, you can install an entire multi-region system.

Note This topic provides overview information on how to set up a multi-region system when you first install HyperStore. For complete installation instructions, see the *HyperStore Installation Guide*. For information on **adding a service region to an existing HyperStore system**, see "[Adding a Region](#)" (page 385).

The mechanism that you use to configure your HyperStore system installation as a multi-region system is the survey file that you create and provide as input to the HyperStore installer. The survey file is a simple inventory of all the nodes in your HyperStore system, with some basic information about the location of each node — including the region in which each node resides.

Below is an example of an installation survey file for a HyperStore system that will be configured as just a single service region. For each node, the survey file specifies the region, hostname, IP address, data center name, and rack name (use the same rack name for all nodes in a given data center). Note that you must provide a region name even if you will have only one region. For each region the region name will become part of the region's S3 endpoint URL.

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

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

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

Using your survey file as input, the HyperStore interactive installer will install HyperStore software on each node in the survey file, in all your regions. During the interactive installation process you will be prompted for information about your desired system configuration. If your survey file includes multiple regions, then the installer will prompt you for which region you want to serve as the [default region](#).

Based on your survey file and your top-level domain (which the installer prompts you for), the installer will automatically generate:

- The S3 service endpoint for each of your regions, in format `s3-<region-name>.<top-level-domain>`.
Example: `s3-tokyo.my-enterprise.com`
- The S3 website endpoint (used for S3 buckets configured as static websites) for each of your regions, in format `s3-website-<region-name>.<top-level-domain>`.
Example: `s3-website-tokyo.my-enterprise.com`,

Note The installer also allows you to customize the endpoints if you wish. Also, although during the initial installation process you are only allowed to specify one S3 service endpoint per service region, after installation you can optionally configure multiple S3 service endpoints per region (for example if you want to use different S3 endpoints for different data centers within a region) by using the ["Installer Advanced Configuration Options"](#) (page 449).

The installer will also automatically configure your HyperStore system to operate as a multi-region system. For example:

- In the system configuration file `common.csv`, the ["regions"](#) (page 459) setting is automatically configured to a list of all your service regions, and the ["default_region"](#) (page 458) setting is automatically set to your default region.
- The installer automatically creates and configures region-specific versions of the configuration files that require region-specific settings:

- Region-specific instances of *region.csv* are created and configured. These are named as *<region-name>_region.csv*. For example, if in your installation survey file you indicated a "tokyo" region and a "osaka" region, the installer will automatically create and configure *tokyo_region.csv* and *osaka_region.csv*. These are system-managed files that you do not need to work with.
- Region-specific instances of *topology.csv* are created and configured — for example *tokyo_topology.csv* and *osaka_topology.csv*. These are system-managed files that you do not need to work with.
- If after the initial installation you use the "**Installer Advanced Configuration Options**" (page 449) to configure SSL support for your S3 Service in each region, then region-specific instances of *s3ssl/configs.csv* are created and configured — for example *tokyo_s3ssl/configs.csv* and *osaka_s3ssl/configs.csv*. These are system-managed files that you do not need to work with.

4.15.2.1. Port Availability for Inter-Region Communications

On each HyperStore node, on all interfaces, all ports must be open to traffic originating from any other HyperStore node -- including HyperStore nodes in other data centers and other service regions.

4.15.2.2. DNS and Load Balancing Set-Up in a Multi-Region System

In a multi-region system, each region's S3 endpoint should resolve to load balancers in that region, which distribute traffic across nodes within that region. By contrast the system-wide Admin and CMC service endpoints should resolve to load balancers in the **default service region** which distribute traffic only to nodes in the default service region. Note that only Admin Service nodes in the default region support the full Admin API functionality. For more information on DNS and load balancer set-up see "DNS Set-Up" and "Loading Balancing" in the HyperStore Installation Guide. For more information on Admin API service in a multi-region system see "**Admin API Behavior in Multi-Region Systems**" (page 695).

4.15.3. Creating a Bucket in a Specific Region

The standard Amazon S3 API enables S3 clients to specify a service region when creating a storage bucket. The CMC's built-in S3 client supports this feature.

Creating an S3 Bucket in a Specific Region (CMC)

Service users who use the CMC as their S3 client can choose a region from a drop-down list when they use the **Bucket & Objects** page to "**Add a Bucket**" (page 192). The drop-down list is automatically populated with the names of your service regions.

Creating an S3 Bucket in a Specific Region (S3 API)

When creating an S3 storage bucket, S3 client applications can specify a region in which to create the bucket. To do so, the client application includes the standard S3 request elements CreateBucketConfiguration and LocationConstraint in the body of the **PUT Bucket** request that it submits to the HyperStore S3 service. The LocationConstraint request element indicates the region in which to create the new bucket. For example:

```
<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<LocationConstraint>tokyo</LocationConstraint>
</CreateBucketConfiguration>
```

If a PUT Bucket request does not include a LocationConstraint element, the bucket will be created in the [default region](#).

4.15.4. Cross-Region Replication

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Bi-Directional Replication"** (page 134)
- **"Replication Timing and Error Handling"** (page 134)
- **"Cross-Region Replication Versus Region-to-Region Auto-Tiering"** (page 135)
- **"Cross-Region Replication Impact on Usage Tracking"** (page 135)
- **"Cross-Region Replication of Encrypted Objects"** (page 135)
- **"Cross-Region Replication and WORM"** (page 135)

Like Amazon S3, HyperStore supports cross-region replication (CRR). A bucket owner can configure a bucket so that any newly uploaded objects (objects uploaded **after** this feature is enabled on the bucket) are automatically replicated to a chosen destination bucket in a different service region. This feature enables a bucket owner to enhance the protection of data by having it stored in two geographically dispersed service regions. The feature is also useful in cases where a bucket owner wants to have the same set of data stored in two different regions in order to minimize read latency for users in those regions.

Along with allowing replication from a source bucket to a destination bucket in a different service region in the same HyperStore system, HyperStore also supports:

- Replicating from a HyperStore source bucket to a destination bucket in an external S3 system such as Amazon S3
- Replicating from a HyperStore source bucket to a destination bucket that's in the same HyperStore service region as the source bucket

As is the case with Amazon S3's implementation of this feature, with HyperStore **both the source bucket and the destination bucket must have "versioning" enabled** in order to activate cross-region replication.

Object metadata — including any access permissions assigned to an object, and any [user-defined object metadata or object tags](#) — is replicated to the destination bucket as well as the object data itself.

As with Amazon S3, HyperStore's implementation of the cross-region replication feature **does not replicate**:

- Objects that were already in the source bucket before the bucket was configured for cross-region replication.
- Objects that are encrypted with user-managed encryption keys (SSE-C) or AWS KMS managed encryption keys
- Objects that are themselves replicas from other source buckets. If you configure "bucket1" to replicate to "bucket2", and you also configure "bucket2" to replicate to "bucket3", then an object that you upload to "bucket1" will get replicated to "bucket2" but will not get replicated from there on to "bucket3". Only objects that you directly upload into "bucket2" will get replicated to "bucket3".
- Deletions of specific object versions.
 - In the case of an object deletion request that does not specify the object version, the deletion marker that gets added to the source bucket is replicated to the destination bucket.

- In the case of an object deletion request that does specify the object version, the object version is deleted from the source bucket but is **not** deleted from the destination bucket.

Note: HyperStore currently supports only Version 1 of the Amazon S3 specification for replication configuration XML, not Version 2. Those two versions differ in regard to whether or not deletion markers are replicated. The behavior described above is the Version 1 behavior, which is implemented by HyperStore.

4.15.4.1. Configuring Cross-Region Replication for a Bucket

Bucket owners can configure replication from a source bucket to destination bucket. This can be done either through the CMC or through a different S3 client application that invokes the HyperStore implementation of the S3 API.

Configure Cross-Region Replication for a Bucket (CMC)

Note Before configuring cross-region replication, be sure that both the source bucket and the destination bucket have **versioning enabled**. This is required in order to use cross-region replication. For information about enabling versioning on a bucket in HyperStore, see "**Set Versioning for a Bucket**" (page 209).

In the CMC, bucket owners can enable and configure cross-region replication through the **Buckets & Objects** page. For a given source bucket, cross-region replication is among the options that can be configured as bucket properties. For detail, see "**To Configure a Source Bucket for Cross-Region Replication**" (page 208). Along with specifying a destination bucket to which objects will be replicated from the source bucket, bucket owners can indicate whether they want **all** newly uploaded objects to be replicated or only objects for which the full object name starts with a particular prefix (such as `/profile/images`). The UI also allows a bucket owner to configuring replication from a HyperStore source bucket to an Amazon S3 destination bucket.

Also in the CMC, you can disable CRR on a source bucket for which it is currently enabled.

Configure Cross-Region Replication for a Bucket (S3 API)

S3 applications can invoke the HyperStore implementation of the S3 API to configure a source bucket for cross-region replication. For detail see [PUT Bucket replication](#). As described in that documentation, in addition to the standard Amazon S3 API for this operation HyperStore also supports extensions that give bucket owners the ability to replicate from a HyperStore source bucket to a destination bucket in an external S3 system such as Amazon S3.

As with Amazon S3, the HyperStore implementation of the S3 API requires that **versioning be enabled** on both the source and the destination bucket before cross-region replication can be applied. HyperStore supports the standard S3 [PUT Bucket versioning](#) API.

HyperStore also supports the [GET Bucket replication](#) method for retrieving a bucket's current CRR configuration, and the [DELETE Bucket replication](#) method for disabling CRR on a source bucket for which it is currently enabled.

4.15.4.2. Bi-Directional Replication

HyperStore supports bi-directional replication, whereby you configure two buckets to replicate to each other. For example, objects directly uploaded into "bucket1" can be replicated to "bucket2", while objects directly uploaded into "bucket2" are replicated to "bucket1".

As with the HyperStore cross-region replication generally, objects are not replicated if they are themselves replicas from another source bucket. Only objects directly uploaded into a bucket by a client application will be replicated. In the context of bi-directional replication this means that objects that are uploaded directly into one bucket will be replicated to the other bucket, but they will not then be replicated back into the original bucket and so on in an endless loop.

4.15.4.3. Replication Timing and Error Handling

Objects are replicated to the destination bucket immediately after they are uploaded to the source bucket. The replication request is initiated by whichever S3 Service node processes the upload request for the original object. Errors in replicating an object to the destination bucket are handled as follows:

- If the destination system returns an HTTP 403 or 404 error when HyperStore tries to replicate an object to the destination, this is treated as a permanent error. In the "**Cross-Region Replication request log (cloudian-crr-request-info.log)**" (page 590) on the node that initiates the replication request, an entry for the object replication attempt is written with status FAILED. The system does not retry replicating the object. Examples of scenarios that could result in permanent errors like this are if the destination bucket has been deleted, or if versioning has been disabled on the destination bucket, or if the source bucket owner no longer has write permissions on the destination bucket.
- Any other type of error -- such as the destination system being unreachable due to a network partition, or the destination system returning an HTTP 5xx error -- is treated as temporary. In the Cross-Region Replication request log on the node that initiates the replication request, an entry for the object replication attempt is written with status PENDING. The object replication job will be queued and retried. Retries of object replication jobs with PENDING status are executed by a [system cron job](#) that runs once every four hours. The retries for an object will recur once every four hours, until either the object is successfully replicated to the destination bucket or a permanent error is encountered. Each retry attempt results in a new entry in the Cross-Region Replication request log on the [cron job node](#), with a status of either COMPLETED, PENDING, or FAILED.

A permanent failure for replication of an object applies **only to that object** and does not impact the processing of other objects subsequently uploaded into the same source bucket. The system will continue to replicate -- or attempt to replicate -- other objects that subsequently get uploaded into the bucket. Those objects may also encounter permanent errors, but the system will continue to try to replicate newly uploaded objects unless you disable cross-region replication on the source bucket.

There is no limit on the number of replication retries for a given object or on the number of objects that are queued for retry.

Note If an attempt to replicate an object to the destination region -- either the original attempt or a retry attempt -- results in a FAILED status in the Cross-Region Replication request log, so that there will be no further retries, this triggers an Alert in the CMC's [Alerts](#) page. This type of alert falls within the alert rules for S3 service errors (there is not a separate alert rule category for CRR replication failures).

4.15.4.4. Cross-Region Replication Versus Region-to-Region Auto-Tiering

The key difference between cross-region replication and [auto-tiering](#) from one region to another is that with cross-region replication each replicated object is stored in both the source region **and** the destination region. By contrast, after an object is auto-tiered from a source region to a destination region the object data is stored **only** in the destination region.

4.15.4.5. Cross-Region Replication Impact on Usage Tracking

If a user configures cross-region replication and consequently objects are replicated from a source bucket in one HyperStore region to a **destination bucket in a different HyperStore region**, the replica data in the destination region will count toward the user's Stored Bytes count in that region. For instance a 100MB object that gets replicated in this way would count as 100MB toward the user's Stored Bytes count in the source region and also as 100MB toward the user's Stored Bytes count in the destination region, for a total of 200MB across the system. (Also, for each replicated object there are about 50 bytes of object metadata associated with implementing this feature).

Note also that to enable cross-region replication the source bucket and destination bucket must both have "versioning" enabled. Once versioning is enabled, when objects are modified by users the system continues to store the older version(s) of the object as well as storing the new version. In a cross-region replication context, this means that over time multiple versions of an object may come to be stored in both the source bucket and the destination bucket. This results in higher Stored Bytes counts for the user than would be the case if versioning were not enabled.

If a user configures cross-region replication and consequently objects are replicated from a source bucket in one HyperStore region to a **destination bucket in Amazon**, the replica data in the Amazon destination region does not count toward the user's Stored Bytes count for HyperStore. Rather, only the data in the source bucket counts toward the user's Stored Bytes count for HyperStore (plus, for each replicated object, about 50 bytes of object metadata associated with implementing this feature).

Impact on System-Wide Stored Byte Count and Licensed Max Storage Limit

When users use cross-region replication to replicate objects from one HyperStore bucket to another HyperStore bucket, the objects in the source bucket and the object replicas in the destination bucket **both** count toward your system's stored byte count -- the count that is used to determine whether you are in compliance with your licensed maximum storage limit. In this respect bucket-to-bucket cross-region replication is different than storage policy based replication or erasure coding of objects within a region, which is treated as overhead and not counted toward your stored byte count.

When users use cross-region replication to replicate objects from a HyperStore bucket to destination bucket in Amazon S3, only the data in the HyperStore bucket counts toward your system's byte count (plus, for each replicated object, about 50 bytes of object metadata associated with implementing this feature).

4.15.4.6. Cross-Region Replication of Encrypted Objects

Whether or not objects encrypted by server-side encryption are replicated from a source bucket to the destination bucket depends on the server-side encryption method used. For details see "[Encryption and Cross-Region Replication](#)" (page 119).

4.15.4.7. Cross-Region Replication and WORM

For information on this topic see "[Locked Buckets and Cross-Region Replication](#)" (page 162).

4.15.5. Using the CMC or Admin API in a Multi-Region System

4.15.5.1. Using the CMC in a Multi-Region System

Through the CMC you can monitor and manage an entire multi-region HyperStore system. If your HyperStore deployment is set up as a multi-region system, that will be reflected in these aspects of the CMC interface:

- **User Provisioning and Administration**
 - When you "**Add a User**" (page 230), you will assign the user a rating plan (for billing purposes) for each region. You can assign the same rating plan in each region, or different rating plans in each region. The same is true when you "**Add a Group**" (page 237): when you set a group-level rating plan (which serves as the default rating plan for users within the group), you choose a rating plan for each region.
 - When you "**Set Quality of Service (QoS) Controls**" (page 250), you will assign a different set of limits for each region. As described in "**Multi-Region System Architecture**" (page 129), QoS limits are applied on a per-region basis.
- **S3 Data Storage and Access**
 - When service users "**Add a Bucket**" (page 192), they can choose which region to create the bucket in.
- **Usage Reporting and Billing**
 - When you "**Create a Usage Report**" (page 185) for a user, a user group, or the system, you can choose the region for which to report usage. You can also generate reports that includes usage from all regions.
 - When you use the [Account Activity](#) page to generate a statement for billing a user, you generate a separate bill for each region.
- **HyperStore System Monitoring**
 - When you use the [Data Centers](#) page to get a high level view of each data center's status, you choose which region you want to check on. Status information is available for every region.
 - When you use the [Node Status](#) page to check the status detail for individual nodes, you start by selecting a region, and then select a node within that region.
- **HyperStore System Management**
 - When you [perform management operations on individual nodes](#) (such as cleanup or repair), you start by selecting a region, then select a node, then select the operation to perform.
 - When you are "**Adding Nodes**" (page 369) to the system, you specify the region in which to add it.

4.15.5.2. Using the Admin API in a Multi-Region System

If your HyperStore system has multiple regions, then:

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

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

For such API calls, if you do not specify a region then the [default region](#) is presumed.

- Certain Admin API calls are only supported by the Admin Service in the default region. If you submit these calls to the Admin Service in a non-default region, you will receive a 403: Forbidden response. For more information see "**Admin API Behavior in Multi-Region Systems**" (page 695).

4.16. Smart Support

4.16.1. Smart Support and Diagnostics Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Smart Support"** (page 137)
- **"On-Demand Node Diagnostics"** (page 138)

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

4.16.1.1. Smart Support

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

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

- Cluster topology information
- HyperStore license information
- Packages and dependencies version information
- Node hardware information, including information specific to HyperStore appliance models (if applicable)
- Network interface and routing table information for each node
- Storage capacity utilization information for each node
- Disk diagnostics data for each node
- Service status (service UP/DOWN) history for each node
- S3 transaction performance data for each node
- System performance data such as disk I/O and per-disk capacity usage for each node
- Current map of tokens to disks on each node
- Redis information to confirm sync between master and slave nodes
- Data repair operation status and history for each node
- Proactive repair queue information for the cluster
- Configuration files for each node

- Alerts for each node (the same as you would see in the CMC's [Alerts](#) page)
- System log files and HyperStore log files

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

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

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

4.16.1.2. On-Demand Node Diagnostics

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

The Node Diagnostics package includes:

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

On the target node(s), under the directory `/var/log/cloudian/cloudian_sysinfo`, the Node Diagnostics mechanism packages all this data into a file named `<hostname>_<YYYYMMDDhhmm>.tar.gz`. Optionally you can have the system also automatically send a copy of the package(s) to Cloudian Support.

Note The system creates the `/var/log/cloudian/cloudian_sysinfo` directory on a node the first time you use the Collect Diagnostics feature for that node.

4.16.2. Configuring Smart Support and Node Diagnostics

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

- Have the daily Smart Support upload go to an S3 destination other than Cloudian Support, by setting the `phonehome_uri`, `phonehome_bucket`, `phonehome_access_key`, and `phonehome_secret_key`

settings in *common.csv*. For more information see "**phonehome_uri**" (page 470) and the subsequent setting descriptions.

- Have the daily Smart Support upload use a local forward proxy by setting the *phonehome_proxy_host*, *phonehome_proxy_port*, *phonehome_proxy_username*, and *phonehome_proxy_password* settings in *common.csv*. For more information see "**phonehome_proxy_host**" (page 469) and the subsequent setting descriptions.
- Disable the daily Smart Support upload by setting *phonehome.enabled* in *mts.properties.erb* to false. **This is not recommended.**
- Have on-demand Node Diagnostics uploads (triggered by your using the CMC's "**Collect Diagnostics**" (page 284) function) go to an S3 destination other than Cloudian Support, by setting the *sysinfo.uri*, *sysinfo.bucket*, *sysinfo.accessKey*, and *sysinfo.secretKey* properties in *mts.properties.erb*. For more information see "**sysinfo.uri**" (page 517) and the subsequent property descriptions.

Note Be sure to do a Puppet push and restart the S3 Service if you edit any of these configuration settings. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.16.2.1. Other Configuration Considerations

Changing the timing of the daily diagnostics upload

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

Deletion of old daily diagnostics and node diagnostics packages

The deletion of old diagnostics packages is managed by Puppet, as configured in *common.csv* by the "**cleanup_directories_byage_withmatch_timelimit**" (page 460) setting (for daily system diagnostics packages associated with the Smart Support feature) and the "**cleanup_sysinfo_logs_timelimit**" (page 461) setting (for on-demand Node Diagnostics packages that you've generated). By default these settings have Puppet delete the diagnostics packages after they are 15 days old. This presumes that you have left the Puppet daemons running in your HyperStore cluster, which is the default behavior. If you do not leave the Puppet daemons running the diagnostics logs will not be automatically deleted. In that case you should delete the old packages manually, since otherwise they will eventually consume a good deal of storage space.

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

Compliance with the European Union's General Data Protection Regulation (GDPR)

As part of a GDPR compliance program, you can have HyperStore remove personally identifiable user information -- such as user IDs and client IP addresses -- from the S3 request log copies and S3 application log copies that get uploaded to Cloudian Support as part of the Smart Support and Node Diagnostics features. To do so, set "**phonehome_gdpr**" (page 470) to *true* in *common.csv* (by default it's set to *false*). Setting this parameter to *true* will not remove the user IDs and client IP addresses from the original S3 request logs on your HyperStore nodes -- just from the log file copies that get sent to Cloudian. In the log file copies that get sent to Cloudian the user ID and IP address fields will say "Not Available".

4.16.3. Executing Node Diagnostics Collection

To generate deep-dive Node Diagnostics in support of troubleshooting one or more problem nodes, use the CMC's "**Collect Diagnostics**" (page 284) function. When using that function you will be able to select one or more nodes for which to collect diagnostics. You will also be able to choose whether to have the diagnostics package(s) automatically uploaded to Cloudian Support (or an alternative S3 destination, if you have [configured the system to support this option](#)).

Each node's diagnostics package will be created on the node itself, at path `/var/log/cloudian/cloudian_sys-info/<hostname>_<YYYYMMDDhhmm>.tar.gz`.

4.17. Storage Policies

4.17.1. Storage Policies Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Supported Erasure Coding Configurations for a Single DC"** (page 141)
- **"Multi- Data Center Storage Policies"** (page 142)

Storage policies are ways of protecting data so that it's durable and highly available to users. The HyperStore system lets you pre-configure one or more storage policies. Users when they create a new storage bucket can then choose which pre-configured storage policy to use to protect data in that bucket. **Users cannot create buckets until you have created at least one storage policy.**

For each storage policy that you create you can choose from either of two data protection methods:

- **Replication** — With replication, a configurable number of copies of each data object are maintained in the system, and each copy is stored on a different node. For example, with 3X replication 3 copies of each object are stored, with each copy on a different node.
- **Erasure coding** — With erasure coding, each object is encoded into a configurable number (known as the "k" value) of data fragments plus a configurable number (the "m" value) of redundant parity fragments. Each of an object's "k" plus "m" fragments is unique, and each fragment is stored on a different node. The object can be decoded from any "k" number of fragments. To put it another way: the object remains readable even if "m" number of nodes are unavailable. For example, in a 4+2 erasure coding configuration (4 data fragments plus 2 parity fragments), each object is encoded into a total of 6 unique fragments which are stored on 6 different nodes, and the object can be decoded and read so long as any 4 of those 6 fragments are available.

In general, erasure coding requires less storage overhead -- the amount of storage consumption above and beyond the original size of the stored objects, in order to ensure data persistence and availability -- than replication. Put differently, erasure coding is more efficient in utilizing raw storage capacity than is replication.

For example, while 3X replication incurs a 200% storage overhead, 4+2 erasure coding incurs only a 50% storage overhead. Or stated in terms of storage capacity utilization efficiency, 3X replication is 33% efficient (for instance with 12TB of available storage capacity you can store 4TB of net object data) whereas 4+2 erasure coding is 67% efficient (with 12TB of available storage capacity you can store 8TB of net object data). On the other hand, erasure coding results in somewhat longer request processing latency than replication, due to the need for encoding/decoding.

In light of its benefits and drawbacks, erasure coding is best suited to **long-term storage of large objects that are infrequently accessed**.

Regardless of whether you use replication or erasure coding, if your HyperStore system spans multiple data centers, for each storage policy you can also choose how data is allocated across your data centers — for example, you could have a storage policy that for each S3 object stores 3 replicas of the object in each of your data centers; and a second storage policy that erasure codes objects and stores them in just one particular data center (for more information see "**Multi- Data Center Storage Policies**" (page 142)).

Also as part of the configuration options for each storage policy, you can choose whether to compress and/or encrypt stored objects.

Individual storage policies are **not confined to dedicated nodes or disks**. Instead, all policies utilize all the resources of your cluster, and data stored in association with a particular policy will tend to be spread fairly evenly across the cluster (with the exception that you can limit a policy to a particular data center as noted above). This helps to ensure that regardless of how many or what types of storage policies you configure, and regardless of how much data is stored in association with particular policies, the physical resources of your entire cluster — disks, CPU, RAM — will be used in an approximately even manner.

4.17.1.1. Supported Erasure Coding Configurations for a Single DC

HyperStore supports several erasure coding configurations, in terms of "k" value (number of data fragments) and "m" value (number of parity fragments):

- 4+2
- 6+2
- 8+2
- 9+3
- 12+4

The choice among these supported EC configurations is largely a matter of how many HyperStore nodes you have in the data center. For example, compared to a 4+2 configuration, 6+2 EC provides the same degree of data availability assurance (objects can be read even if 2 of the involved nodes are unavailable), while delivering a higher level of storage efficiency (4+2 is 67% efficient whereas 6+2 is 75% efficient). So 6+2 may be preferable to 4+2 if you have at least 8 HyperStore nodes in the data center.

Likewise, 9+3 EC provides a higher degree of protection and availability than 6+2 EC (since with 9+3 EC, objects can be read even if 3 of the involved nodes are unavailable) while delivering the same level of storage efficiency (both 6+2 and 9+3 are 75% efficient). So 9+3 may be preferable to 6+2 if you have at least 12 HyperStore nodes in the data center.

Note If you want to use a $k+m$ configuration other than those mentioned above, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

Note For detailed information on S3 write and read availability under various combinations of cluster size and storage policy configuration, see "**Storage Policy Resilience to Downed Nodes**" (page 148).

4.17.1.2. Multi- Data Center Storage Policies

If your HyperStore system is deployed across multiple data centers, for each storage policy that you create you can configure a data center assignment scheme for the policy. This determines which of your data centers to use for storing data, for each storage policy.

For storage policies that use replication only, in a multiple data center environment you can choose how many replicas to store in each data center -- for example, for each object store 3 replicas in DC1 and 2 replicas in DC2.

For erasure coding storage policies, you have the option of replicating the $k+m$ fragments in each of the participating DCs (so that each participating DC stores $k+m$ fragments), or distributing the $k+m$ fragments across the participating DCs (so that there are a combined total of $k+m$ fragments across the participating DCs).

For replicated erasure coding, by default your $k+m$ options are:

- 4+2
- 6+2
- 8+2
- 9+3
- 12+4

In each of the above configurations the $k+m$ fragments can be replicated across multiple DCs.

For distributed erasure coding, the supported options depend on how many data centers you are using in the storage policy. You must use at least 3 DCs for this type of policy, and by default your $k+m$ options are as indicated in the table below:

# of Participating DCs	Supported "k"+"m"	How Fragments Will Be Distributed
3	5+4	3 fragments per DC
	7+5	4 fragments per DC
4	8+4	3 fragments per DC
5	6+4	2 fragments per DC
6	8+4	2 fragments per DC
	7+5	2 fragments per DC
7	10+4	2 fragments per DC
8	10+6	2 fragments per DC
9	10+8	2 fragments per DC

Note If you want to use a $k+m$ configuration other than those mentioned above, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

Note For any type of storage policy in a multiple data center environment, you have the option of configuring the policy such that data is stored in some of your data centers and not others — for example,

you can create a policy that stores data in DC1 and DC2 but not in DC3. Note, however, that DC3 may be involved in processing S3 requests associated with buckets that use this policy. By default there is only one S3 service endpoint per region, and incoming S3 requests may resolve to any DC within the region. If the S3 Service in DC3 receives an S3 PUT request in association with a policy that stores data only in DC1 and DC2, it will transmit the uploaded object on to DC1 and D2 (it will not be stored in DC3). Likewise, if DC3 receives an S3 GET request in association with a policy that stores data only in DC1 and DC2, then DC3's S3 Service will get the object from DC1 or DC2 and pass it on to the client. If you want more absolute barriers so that for example DC3 never touches DC2's data and vice-versa, you need to set up your system so those DCs are in different [service regions](#).

4.17.2. Consistency Levels

To boost data durability and availability, HyperStore implements replication or erasure coding for object data and replication for object metadata. This entails distributing each object's data and metadata to multiple endpoint nodes across the cluster. When you create storage policies, along with configuring a replication or erasure coding scheme you will also configure consistency levels for writes and reads. Consistency levels impose requirements as to **what portion of the data and metadata writes or reads associated with each S3 request must be successfully completed before the system can return a success response** to the S3 client. If the consistency requirements cannot be met for a given S3 request at a given time -- for example, due to one or more endpoint nodes being inaccessible -- an HTTP 503 error response is returned to the client.

Below is the list of consistency levels supported by the HyperStore system. Your consistency level options when configuring a storage policy will be limited by the data distribution scheme (replication or erasure coding, single DC or multi-DC) that you have selected for that policy.

- "**Consistency Level "ALL"**" (page 323)
- "**Consistency Level "QUORUM"**" (page 328)
- "**Consistency Level "EACH QUORUM"**" (page 325)
- "**Consistency Level "LOCAL QUORUM"**" (page 326)
- "**Consistency Level "ANY QUORUM"**" (page 324)
- "**Consistency Level "ONE"**" (page 327)

For detailed information on S3 write and read availability under various combinations of cluster size, data distribution scheme, and consistency level settings, see "**Storage Policy Resilience to Downed Nodes**" (page 148).

Note In the case of writes, if the consistency requirement is met by something less than completing writes of all replicas (or all erasure coded fragments), then after returning a success response to the client the system continues to try to complete the remaining writes. If any of these writes fail they will later be recreated by [automatic data repair](#).

Note As an advanced option you can also configure "dynamic" consistency levels, whereby the system will try to achieve a "fallback" consistency level if the primary consistency level cannot be achieved. For more information see "**Dynamic Consistency Levels**" (page 29).

4.17.2.1. Note About Object Data Replica Reads

For replication based storage policies, the descriptions and examples in this documentation sometimes state that part of the read consistency requirement is being able to read X number of object data replicas. This is a simplification. Technically, what needs to be readable is X number of object data replica **digests** (which are stored in RocksDB on the same disks as the actual replica data). If the read consistency requirements are met for an S3 GET operation -- for reading object data replica digests (in RocksDB) and for reading object metadata replicas (in Cassandra) -- then the system retrieves just one object data replica in order to return the object data to the S3 client. Because each object data replica's digest is on the same disk as the replica data itself, saying that X number of object data replicas must be readable is a convenient approximation of the actual technical requirement that X number of object data replica digests must be readable.

4.17.2.2. Note About Bucket Content List Reads

In the documentation of the supported consistency levels such as "ALL", "QUORUM", and so on (see the cross references above), when read consistency requirements are discussed the focus is on reads of individual objects -- that is, the consistency requirements for successfully implementing S3 *GET Object* requests. It's worth noting however that your configured read consistency requirements also apply to bucket content list reads -- that is, implementing S3 *GET Bucket (List Objects)* requests.

Metadata for objects is stored in two different types of record in Cassandra: object-level records (with one such record for each object) and bucket-level records that identify the objects in a bucket (along with some metadata for each of those objects). Both types of object metadata are replicated to the same degree. So for example, in a 3X replication storage policy, for each object the object-level metadata record is replicated three times in the cluster and for each bucket the bucket-level object metadata records are replicated three times in the cluster.

A *GET Object* request requires reading the object's object-level metadata record and a *GET Bucket (List Objects)* request requires reading the bucket's bucket-level object metadata records. Whatever read consistency requirements you set for a storage policy apply not only to reads of individual objects but also to reads of buckets content lists. So for example if you use a QUORUM read consistency requirement, then in order to successfully execute a *GET Bucket (List Objects)* request the system must be able to read a QUORUM of the bucket-level object metadata records for the bucket.

For more on the meaning of QUORUM and the other supported consistency levels, see the cross references above.

4.17.3. Storage of Object Metadata

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Object Metadata in Replication Storage Policies"** (page 145)
- **"Object Metadata in Erasure Coding Storage Policies"** (page 145)
- **"Two Types of Object Metadata Record"** (page 145)

HyperStore [object metadata](#) is stored in Cassandra, and is protected by replication. The degree to which object metadata is replicated depends on the type of storage policy being used.

4.17.3.1. Object Metadata in Replication Storage Policies

In the case of storage policies that protect object data by replication, the corresponding object metadata is replicated to the same degree as the object data. For example with a 3X replication storage policy, the system stores three replicas of each object (with each replica stored on a different node, in the HyperStore file system) and three replicas of each object's metadata (with each replica stored on a different node, in Cassandra). In a multi-DC replication storage policy that retains two replicas of each object in DC1 and one replica of each object in DC2, the object metadata will also be replicated in this same manner -- two replicas in DC1 and one replica in DC2.

4.17.3.2. Object Metadata in Erasure Coding Storage Policies

In the case of storage policies that protect object data by erasure coding, the object metadata is protected by replication -- not erasure coding. This is because the object metadata associated with a given object is very small in size, and therefore not appropriate for erasure coding.

Specifically, for erasure coding storage policies the system will store **2m-1 replicas of object metadata**. For example, with a 4+2 erasure coding storage policy, the object metadata is protected by 3X replication.

Additional examples of 2m-1 object metadata replication:

- Single-DC, $k+m = 6+2 \rightarrow 3$ replicas of object metadata
- Single-DC, $k+m = 9+3 \rightarrow 5$ replicas of object metadata
- Replicated multi-DC, $k+m = 4+2 \rightarrow 3$ replicas of object metadata in each DC
- Distributed multi-DC, $k+m = 5+4 \rightarrow 7$ replicas of object metadata distributed across the DCs

4.17.3.3. Two Types of Object Metadata Record

Metadata for an object is stored in two different types of record in Cassandra: one record specific to the object (called "skinny row" object metadata) and one bucket-level record that gets updated with metadata for the object (called "wide row" object metadata). The latter is used for listing the contents of a bucket, among other purposes.

Both types of object metadata are replicated to the same degree. So for example, in a 3X replication storage policy, for each object the "skinny row" object metadata is replicated three times in the cluster and the "wide row" object metadata is replicated three times in the cluster.

The skinny row metadata has the same key format as the object data (*bucketname/objectname*) and so has the same hash token and will be written to the same nodes as the object data -- or a subset of those nodes in the case of an erasure coding storage policy. The wide row metadata has a different key format and hash token and so may be written to different nodes than the object data if the cluster exceeds the minimize size required for the storage policy.

Within Cassandra, both types of object metadata record are part of the *UserData_<policyid>* keyspaces.

4.17.4. Storage of System Metadata

HyperStore stores system metadata in several Cassandra keyspaces: the *AccountInfo* keyspace (for user account information), the *Reports* keyspace (for usage reporting data), and the *Monitoring* keyspace (for system monitoring data). The initial replication level for this system metadata is set by the operator during

HyperStore system installation (with a default of 3 replicas per service region). Subsequently, the system automatically adjusts the system metadata replication level in response to your creation of storage policies. The system uses whichever of these criteria yields the **highest** replication level:

- The system metadata replication level set during installation
- Matching the replication factor of any replication storage policies created in the system
- Having $2m-1$ replicas in each DC for any regular erasure coding or replicated erasure coding policies created in the system
- Having $2m+1$ replicas distributed across DCs for any distributed erasure coding policies created in the system

The table below shows examples of what the automatic system metadata replication level would be in different system configuration scenarios.

System Configuration	System Metadata Replication Level	Comment
<ul style="list-style-type: none"> • Single data center • System metadata replication level configured during install = 2 • Just one storage policy created in system: a 3X replication policy 	3	The highest replication level is yielded by matching the level of the 3X replication storage policy
<ul style="list-style-type: none"> • Single data center • System metadata replication level configured during install = 5 • Just one storage policy created in system: a 3X replication policy 	5	The highest replication level is yielded by using the level configured by the operator during system installation
<ul style="list-style-type: none"> • Single data center • System metadata replication level configured during install = 3 (the default) • Two storage policies created in system: a 3X replication policy and a 9+3 erasure coding policy 	5	The highest replication level is yielded by using $2m-1$ from the 9+3 erasure coding policy
<ul style="list-style-type: none"> • Two data centers in a single service region • System metadata replication level configured during install = 3 (the default; implemented as 1 replica in one DC and 2 in the other) • One storage policy created in system: a replicated 4+2 erasure coding policy 	3 in each DC	The highest replication level is yielded by using $2m-1$ per DC from the replicated 4+2 erasure coding policy
<ul style="list-style-type: none"> • Three data centers in a single service region • System metadata replication level configured during install = 3 (the default; implemented as 1 in each DC) • Two storage policies created in system: a replication policy at 2X per DC; and a 5+4 distributed erasure coding policy 	3 in each DC	The highest replication level is yielded by using $2m+1$ (distributed across DCs) from the distributed 5+4 erasure coding policy

The general logic behind this automated adjustment of system metadata replication level is that the **greater the resilience of your configured storage policies** -- in terms of ability to read and write object data and object metadata when a node or nodes are unavailable -- **the greater will be the resilience built into the system metadata storage configuration**.

4.17.4.1. Consistency Requirements for System Metadata Reads and Writes

Consistency requirements for object data and object metadata reads and writes are set per storage policy, when you create storage policies. By contrast consistency requirements for reads and writes of system metadata are set at the system level, by these configuration properties in [mts.properties.erb](#):

- "**cloudian.cassandra.default.ConsistencyLevel.Read**" (page 501) (default = LOCAL_QUORUM,ONE)
- "**cloudian.cassandra.default.ConsistencyLevel.Write**" (page 502) (default = QUORUM,LOCAL_QUORUM)

4.17.5. Creating and Managing Storage Policies

In the CMC's **Storage Policies** page you can do everything you need to do in regard to creating and managing storage policies:

- "**Add a Storage Policy**" (page 308)
- "**Edit a Storage Policy**" (page 331)
- "**Designate a Default Storage Policy**" (page 331)
- "**Disable a Storage Policy**" (page 332)
- "**Delete a Storage Policy**" (page 332)

At all times you must have one and only one **default storage policy** defined in each of your HyperStore service regions. The default policy is the one that will be applied when users create new buckets without specifying a policy.

Note The system supports a configurable maximum number of storage policies (*mts.properties*: "**cloudian.protection.policy.max**" (page 519), default = 25). After you have created this many storage policies, you cannot create additional new policies until you either delete unused policies or increase the configurable maximum.

Retrieving Storage Policy Usage Through the Admin API

You cannot create or modify storage policies through the Admin API. However, you can use the API to retrieve a list of buckets that use each storage policy, with the [GET /bppolicy/bucketsperpolicy](#) method.

4.17.6. Assigning a Storage Policy to a Bucket

When users create a new bucket they can select a storage policy to apply to the data that they will store in that bucket. This can be done either through the CMC or through other S3 applications that invoke HyperStore extensions to the standard S3 API.

IMPORTANT: After a bucket is created, it cannot be assigned a different storage policy. The storage policy assigned to the bucket at bucket creation time will continue to be bucket's storage policy for the life of the bucket.

Assigning a Storage Policy to a Bucket (CMC)

CMC users can select a storage policy when they create a new bucket in the CMC's **Buckets** page:

- ["Add a Bucket"](#) (page 192)

If a user does not explicitly select a policy when creating a new bucket, the system's current default storage policy is automatically applied to the bucket.

Assigning a Storage Policy to a Bucket (S3 API)

To select a storage policy for a new bucket, S3 client applications use an "x-gmt-policyid" request header when submitting a *PUT Bucket* request:

- [PUT Bucket](#)

4.17.7. Finding an Object's Replicas or EC Fragments

HyperStore lets you quickly determine which nodes are storing the replicas or erasure coded fragments of a specified S3 object. You can do this through either the CMC or the command line.

Finding an Object's Replicas or EC Fragments (CMC)

To find an object's replicas or fragments locations using the CMC:

- [Object Locator](#)

Finding an Object's Replicas or EC Fragments (Command Line)

To find an object's replicas or erasure coded fragments locations using *hsstool* on the command line:

- [hsstool whereis](#)

4.17.8. Storage Policy Resilience to Downed Nodes

When nodes are down in your cluster, HyperStore S3 service availability for object writes and reads is a function of several factors including:

- The storage policy applied to the objects -- particularly the data distribution scheme (such as 3X replication or 4+2 erasure coding) and the configured consistency level requirements.
- The number of nodes in the cluster.
- The number of nodes that are down.

The tables that follow below indicate HyperStore S3 write and read availability for common single-DC storage policy configurations, **in scenarios where either one or two nodes are down**. For simplicity the tables refer to

nodes as being "down", but the same logic applies if nodes are unavailable for other reasons such as being inaccessible on the network, or in a [stop-write condition](#), or in [maintenance mode](#).

Single DC, 3X Replication

With a **3X replication storage policy**, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have 3, 4, or 5 or more nodes in the cluster.

S3 Operation Type	Configured Consistency Level (CL)	Number of Nodes Down	3 Nodes in Cluster	4 Nodes in Cluster	5 or More Nodes In Cluster
Writes	ALL	1	All writes fail	Writes succeed for some objects and fail for others.	Writes succeed for some objects and fail for others.
		2	All writes fail	All writes fail	Writes succeed for some objects and fail for others
	QUORUM (default)	1	All writes succeed	All writes succeed	All writes succeed
		2	All writes fail	Writes succeed for some objects and fail for others	Writes succeed for some objects and fail for others
	Reads	ALL	All reads fail	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others
		2	All reads fail	All reads fail	Reads succeed for some objects and fail for others
		ONE	1 or 2	All reads succeed	All reads succeed

Single DC, $k+2$ Erasure Coding

By default HyperStore supports several $k+2$ erasure coding schemes: 4+2, 6+2, and 8+2. With a storage policy based on $k+2$ erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have $k+2$, $k+3$, or $k+4$ or more nodes in the cluster. For example with a 4+2 policy ($k = 4$), write and read resilience depends in part on whether you have 6, 7, or 8 or more nodes in the cluster; and with a 6+2 policy ($k = 6$), resilience depends in part on whether you have 8, 9, or 10 or more nodes in the cluster.

S3 Operation Type	Configured Consistency Level (CL)	Number of Nodes Down	$k+2$ Nodes in Cluster	$k+3$ Nodes in Cluster	$k+4$ or More Nodes in Cluster
Writes	ALL	1	All writes fail	Writes succeed for some objects and fail for others	Writes succeed for some objects and fail for others
		2	All writes fail	All writes fail	Writes succeed for some objects and fail for others
	QUORUM (default)	1	All writes succeed	All writes succeed	All writes succeed
		2	All writes fail	Writes succeed for some objects and fail for others	Writes succeed for some objects and fail for others
	ALL	1 or 2	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others
	QUORUM (default)	1	All reads succeed	All reads succeed	All reads succeed
		2	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others

Single DC, $k+3$ Erasure Coding

By default HyperStore supports one $k+3$ erasure coding scheme: 9+3. With a storage policy based on $k+3$ erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have $k+3$, $k+4$, or $k+5$ or more nodes in the cluster. For example with a 9+3 policy ($k = 9$), write and read resilience depends in part on whether you have 12, 13, or 14 or more nodes in the cluster.

S3 Operation Type	Configured Consistency Level (CL)	Number of Nodes Down	$k+3$ Nodes in Cluster	$k+4$ Nodes in Cluster	$k+5$ or More Nodes in Cluster
Writes	ALL	1	All writes fail	Writes succeed for some objects and fail for others	Writes succeed for some objects and fail for others
		2	All writes fail	All writes fail	Writes succeed for some objects and fail for others
	QUORUM (default)	1 or 2	All writes succeed	All writes succeed	All writes succeed
			Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others
	ALL	1 or 2	All reads succeed	All reads succeed	All reads succeed
			Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others

S3 Operation Type	Configured Consistency Level (CL)	Number of Nodes Down	$k+3$ Nodes in Cluster	$k+4$ Nodes in Cluster	$k+5$ or More Nodes in Cluster
QUORUM (default)			fail for others	fail for others	fail for others
	QUORUM (default)	1 or 2	All reads succeed	All reads succeed	All reads succeed

Single DC, $k+4$ Erasure Coding

By default HyperStore supports one $k+4$ erasure coding scheme: 12+4. With a storage policy based on $k+4$ erasure coding, the system's ability to support writes and reads when 1 or 2 nodes are down depends on your consistency level configuration and on whether you have $k+4$, $k+5$, or $k+6$ or more nodes in the cluster. For example with a 12+4 policy ($k = 12$), write and read resilience depends in part on whether you have 16, 17, or 18 or more nodes in the cluster.

S3 Operation Type	Configured Consistency Level (CL)	Number of Nodes Down	$k+4$ Nodes in Cluster	$k+5$ Nodes in Cluster	$k+6$ or More Nodes in Cluster
Writes	ALL	1	All writes fail	Writes succeed for some objects and fail for others	Writes succeed for some objects and fail for others
		2	All writes fail	All writes fail	Writes succeed for some objects and fail for others
	QUORUM (default)	1 or 2	All writes succeed	All writes succeed	All writes succeed
Reads	ALL	1 or 2	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others	Reads succeed for some objects and fail for others
	QUORUM (default)	1 or 2	All reads succeed	All reads succeed	All reads succeed

Additional Considerations for S3 Availability

S3 Availability for Large Objects

For uploading larger objects, S3 client applications typically use Multipart Upload -- which breaks the object data into contiguous parts and uploads each part separately. Amazon recommends that client applications use Multipart Upload for objects 100MB and larger. Within HyperStore, each part is assigned its own hash value and is separately replicated or erasure coded within the cluster.

Also, for each object larger than 10MB -- or for Multipart Uploads, for each object part larger than 10MB -- the HyperStore system breaks the object or part into multiple "chunks" of 10MB or smaller (for more detail on this configurable feature see "**System Settings**" (page 297)). Each chunk is assigned its own hash value and is separately replicated or erasure coded within the cluster.

For S3 write and read availability for large objects, within HyperStore the write or read of **each part and/or chunk** must satisfy the object data consistency level requirement in order for the S3 object upload or object read operation as a whole to succeed. (The metadata requirements are not impacted by object size since the metadata records are for the object as a whole and not for each part or chunk.)

In terms of the Result categories in the tables above, the consequences of an object being large are as follows:

- "**All writes/reads succeed**" -- No effect. Just as writes or reads of all individual small objects succeed, so too do writes or reads of all large object parts and chunks.
- "**All writes/reads fail**" -- No effect. Large objects fail just as small ones do.
- "**Writes/Reads succeed for some objects and fail for others.**" -- Here, the object data consistency criteria that determine which objects succeed (as displayed when you click the Result text in the tables) must be met by **each part and/or chunk** in order for the S3 object upload or object read operation as a whole to succeed. Consequently, in these scenarios, the write or read of a large object with multiple parts and/or chunks has a greater chance of failing than the write or read of a small object.

Writes of Object Metadata Per Bucket

For an S3 write operation to succeed, your configured write consistency requirement must be met for the object data and also for the object metadata. The system actually writes two types of object metadata record -- a record specific to the object and a bucket-level record that gets updated with metadata for each object in the bucket (the bucket-level record is used for listing the contents of a bucket, among other things).

For a given object, the per-object metadata record has the same key format as the object data (*buck-ename/objectname*) and therefore is assigned the same hash token as the object data and will be written to the same endpoint nodes as the object data -- or a subset of those nodes, in the case of erasure coding storage policies. By contrast, the per-bucket object metadata record has a different key format and therefore a different hash token, and so may be written to different endpoint nodes than the object data. The per-bucket object metadata record is replicated to the same degree as the per-object metadata record -- for example, three times in a 3X replication storage policy -- and must meet the same configured write consistency requirements.

To limit the complexity of the tables above, in the Result descriptions the references to object metadata are referring only to the per-object metadata record. In terms of the Result categories in the tables, the consequences of the system's need to also write per-bucket object metadata -- potentially to different endpoint nodes than the object data and per-object metadata -- are as follows:

- "**All writes succeed**" -- No effect. In these scenarios writes succeed for the per-bucket object metadata also.
- "**All writes fail**" -- No effect. In these scenarios S3 writes fail regardless of the per-bucket object metadata considerations.
- "**Writes succeed for some objects and fail for others.**" -- In these scenarios, writes succeed for objects for which the consistency requirement can be met for object data, per-object metadata, **and** per-bucket object metadata. Writes fail for objects for which the consistency requirement cannot be met for either the object data, or the per-object metadata, or the per-bucket metadata. In such down node scenarios where writes succeed for some objects and fail for others, whether the write of a given object succeeds or fails is determined not only by the hash token assigned to the object's data and per-object metadata record but also by the hash token assigned to the per-bucket object metadata record.

Note Per-bucket object metadata is not used for S3 object reads and has **no impact on any of the read availability scenarios** in the tables above. Only the per-object metadata record is relevant to object reads.

Redis DB Access

The HyperStore system's S3 Service needs information from the Redis Credentials database and the Redis QoS database in order to process S3 write and read requests. In most cases -- particularly in larger clusters -- 1 or 2 nodes being down in your system will not impact the availability of these databases (which are implemented across multiple nodes in master-slave relationships). If problems within your system were to lead to either of these databases being completely offline -- such as if all the nodes running Redis Credentials are down, or all the nodes running Redis QoS are down -- the S3 layer can use cached Redis data for a while. But if the cached data expires and a Redis database is still completely offline, then S3 requests will start to fail.

4.18. Usage Reporting

4.18.1. Usage Reporting Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Bucket Usage Statistics" (page 154)**
- **"How Usage Data is Calculated, Tracked, and Processed" (page 154)**
- **"S3 Request Traffic Analysis and Visualization" (page 156)**

By default the HyperStore system keeps track of the following service usage metrics for each user group and each individual user:

- Number of Storage Bytes
- Number of Storage Objects

Optionally, you can configure the system to also track the following metrics for each group and user:

- Number of HTTP Requests
- Number of Bytes IN (bytes uploaded into the system)
- Number of Bytes OUT (bytes downloaded from the system)

Like Amazon S3, the HyperStore system attributes all service usage to **bucket owners**. If a bucket owner grants permission (via ACL policies) for other users to use the bucket, the system attributes the service activity of the grantees to the bucket owner. For example, if grantees upload objects into the bucket, the associated Bytes IN activity and Storage Bytes impact is attributed to the bucket owner — not to the grantees.

The HyperStore system's tracking of service usage by groups and users serves two main purposes:

- **Usage reporting.** Based on service usage tracking data, you can generate service usage reports for individual users, for user groups, for a whole service region, or for your entire HyperStore system.
- **Billing.** Usage tracking provides the foundation for [billing users or groups](#) on the basis of their service usage level.

4.18.1.1. Bucket Usage Statistics

Optionally, you can also configure the system to track usage statistics on a per-bucket basis. This may be useful if for example your HyperStore service is set up such that there is just single "user" for service access purposes, with that one user having multiple buckets each for a different purpose.

In the current release, bucket statistics are not available through the CMC. Bucket statistics are supported only by Admin API calls.

The bucket statistics feature is **disabled by default**. For information on enabling this feature see "[Enabling Non-Default Usage Reporting Features](#)" (page 156).

Note The Admin API call that returns the current total number of bytes and total number of objects in a bucket -- `POST /usage/repair/bucket` -- works even if bucket statistics are disabled in the system. The other bucket statistics related API calls only work if bucket statistics are enabled. For more information on the API calls see "[usage](#)" (page 822).

4.18.1.2. How Usage Data is Calculated, Tracked, and Processed

The table below provides a high level view of how the system generates and handles usage data for users and groups. Usage tracking for Storage Bytes and Storage Objects is handled somewhat differently than tracking for HTTP Requests and Bytes IN/OUT, including that the latter is disabled by default.

Usage Data Types	System Handling
<ul style="list-style-type: none"> • Storage Bytes • Storage Objects 	<p>For each S3 request that the S3 Service processes, the S3 Service updates per-user and per-group Storage Bytes and Storage Objects counters that are maintained in the Redis QoS database. In calculating the increment to Storage Bytes associated with a given S3 request, the system includes the object name size and object metadata size as well as the size of the object itself.</p> <p>Note that storage overhead associated with replication or erasure coding does not count toward a user's Storage Bytes count. For example, a 1MB object that is protected by 3X replication or by 4+2 erasure coding counts as only 1MB toward the Storage Bytes count.</p> <p>These per-user and per-group Storage Bytes and Storage Objects counters in Redis are used by the system to enforce storage quotas, if such quotas are part of your Quality of Service configurations.</p> <p>Every five minutes, freshly updated Redis QoS counts for Storage Bytes and Storage Objects are written to the Raw column family in Cassandra's Reports keyspace, where the data is subjected to additional processing in support of reporting and billing functionality. Each hour the Raw data is automatically processed to derive hourly roll-up data which is written to the RollupHour column family. The hourly roll-up data includes, for each user and each group, the hour's maximum value and weighted average value for Storage Bytes and for Storage Objects.</p> <p>For example, if during a given hour User1 has 10MB of Storage Bytes for the first 20 minutes of the hour and then 15MB for the next 40 minutes of the hour, her weighted average Storage Bytes for the hour is:</p>

Usage Data Types	System Handling
	<p>10MB X 20/60 = 3.33MB + 15MB X 40/60 = 10 MB = 13.33MB weighted average for hour</p> <p>This hourly data is in turn rolled up once each day to derive daily roll-up values (including, for each user and group, the day's maximum and day's average for Storage Bytes and Storage Objects); and the daily roll-up values are rolled up once each month to derive monthly roll-up values (including monthly maximums and averages for each user and group). This data is stored in the RollupDay column family and RollupMonth column family, respectively.</p> <p>Note The writing of Storage Bytes and Storage Objects counters from Redis over to Cassandra, and the usage data roll-ups that take place within Cassandra, are triggered by system cron jobs.</p>
<ul style="list-style-type: none"> • HTTP Requests • Bytes IN • Bytes OUT 	<p>By default system configuration, HTTP Requests, Bytes IN, and Bytes OUT are not tracked.</p> <p>If you enable usage tracking for HTTP Requests and Bytes IN/OUT, then for each S3 request the S3 Service writes transactional metadata directly to the Raw column family in Cassandra's Reports keyspace. It does so asynchronously so that S3 request processing latency is not impacted.</p> <p>In recording the Bytes IN and Bytes OUT impacts of a given S3 transaction, both the request and the response are counted, and HTTP header size is counted as well as body size. For example, a GET request/response pair will count as Bytes IN (typically very small) as well as Bytes OUT (varies with object size); and conversely a PUT request/response pair will count as Bytes OUT (typically very small) as well as Bytes IN (varies with object size).</p> <p>Together with the Storage Bytes and Storage Objects data, the HTTP Request and Bytes IN/OUT data in the Raw column family is rolled up each hour. For each user and each group the roll-up calculates the hour's total for HTTP GETs, PUTs, and DELETEs, and for Bytes IN and Bytes OUT. For Bytes IN and Bytes OUT, maximums for the hour (largest single upload and largest single download) and averages for the hour (average upload size and average download size) are derived for each user and group.</p> <p>The hourly roll-up data is rolled up daily; and the daily roll-up data is rolled up monthly.</p> <p>If you fully enable HyperStore Quality of Service functionality, then for each S3 request the S3 Service also updates HTTP Request and Bytes IN/OUT counters in the Redis QoS database, in support of QoS enforcement.</p>

4.18.1.3. S3 Request Traffic Analysis and Visualization

For information on setting up an [Elastic Stack](#) node and streaming request log data to that node to support analysis and visualization of your S3 request traffic, see "[Setting Up Elastic Stack for S3 Request Traffic Analysis](#)" (page 598).

4.18.2. Enabling Non-Default Usage Reporting Features

This topic describes how to enable advanced HyperStore usage reporting features that are disabled by default: per-user and per-group traffic rates (HTTP request rates and data transfer rates); and per-bucket usage tracking.

4.18.2.1. Per-User and Per-Group Traffic Rates

By default the HyperStore system tracks Storage Bytes and Storage Objects for each user and each group. If you want the system to also track HTTP Requests, Bytes IN, and Bytes OUT for each user and group, log into the CMC and go to **Cluster** → **Cluster Config** → **Configuration Settings**, and then open the **Usage Tracking** panel. There, enable the "Track/Report Usage for Request Rates and Data Transfer Rates" setting and **Save** your change. Your change is applied to the system dynamically -- there is no need to restart services.

Once you enable this feature, this type of usage data will be tracked and available for reporting from that point in time forward. There will not be any per-user and per-group traffic rate data from prior to the time that you enabled this feature.

Enabling this feature results in additional data being stored in Cassandra, and additional work for the cron jobs that roll up usage data into hourly, daily, and monthly aggregates.

4.18.2.2. Per-Bucket Usage Tracking

HyperStore supports usage tracking and reporting on a per-bucket basis, but this feature is disabled by default. To enable per-bucket usage statistics, in the configuration file [common.csv](#) set *bucketstats_enabled* to "true". Then push your change out to the cluster and restart the S3 Service. For instructions see "[Pushing Configuration File Edits to the Cluster and Restarting Services](#)" (page 454).

Once you enable this feature, bucket usage data for storage consumption and traffic rates will be tracked and available for reporting from that point in time forward. There will not be any per-bucket usage data from prior to the time that you enabled this feature.

Note Enabling this feature results in additional data being stored in Cassandra, and additional work for the cron jobs that roll up usage data into hourly, daily, and monthly aggregates.

Note The Admin API call that returns the current total number of bytes and total number of objects in a bucket -- *POST /usage/repair/bucket* -- works even if bucket statistics are disabled in the system. The other bucket statistics related API calls only work if bucket statistics are enabled. For more information on the API calls see "[usage](#)" (page 822).

4.18.3. Validating Usage Data for Storage Levels

As described in the "[Usage Reporting Feature Overview](#)" (page 153), each time the S3 Service processes an S3 request it updates per-user and per-group counters for Storage Bytes and Storage Objects which are maintained in the Redis QoS database. These counts are regularly written over to the Cassandra Reports key-space, where they are post-processed for reporting and for bill generation.

Because the Redis QoS counters for Storage Bytes and Storage Objects can impact your billing of service users (if you charge users based on volume of storage used), it's important that the counts be accurate.

Various types of errors and conditions can on occasion result in discrepancies between the Redis QoS counts and the actual stored bytes and objects owned by particular users. The HyperStore system provides mechanisms for detecting and correcting such discrepancies.

Routine Automated Validation

Twice a day, a [system cron job](#) executes the HyperStore Admin API method [POST /usage/repair/dirtyusers](#). This operation randomly selects up to a configurable maximum number of users (*mts.properties.erb*: "["usage.repair.maxdirtyusers"](#)" (page 512); default = 1000) for whom Storage Bytes and/or Storage Objects counts in Redis have changed since the last time a validation operation was run. For those users, the operation validates the Redis counters by comparing them to the information in the Cassandra "UserData" keyspace's "CLOUDIAN_METADATA" column family, which stores metadata (including size) for every object belonging to each user of the HyperStore service. If any users' Redis counters are found to be out-of-sync with counts derived from the object metadata, the Redis counters are corrected.

In normal circumstances, this automated mechanism should suffice for maintaining the accuracy of usage data for Storage Bytes and Storage Objects.

Validation for Special Cases

If you have reason to question the accuracy of Storage Bytes and/or Storage Objects counts for a particular user — for example, if a user claims their usage report or their bill to be inaccurate — the Admin API supports a method for validating the counts for a specified user: [POST /usage/repair/user](#).

The Admin API also supports a method for validating Storage Bytes and Storage Object counts for a whole user group, a whole service region, or the whole system: [POST /usage/repair](#). Depending on how many users are in your system, this is potentially a resource-intensive operation. This operation should only be considered in unusual circumstances, such as if the Redis QoS database has been brought back online after being unavailable for some period of time.

4.18.4. Setting Usage Data Retention Periods

Data retention periods are separately configurable for raw, hourly roll-up, daily roll-up, and monthly roll-up usage data. After data has been stored for its configured retention period (also known as its "time-to-live" or TTL), it's automatically deleted from the system.

The relevant settings are all in the configuration template *mts.properties.erb*:

- "["reports.rolluphour.ttl"](#)" (page 511) (default = 5616000 seconds [65 days])

IMPORTANT: The HyperStore system calculates monthly bills for service users by aggregating **hourly** roll-up data. Once hourly data is deleted, you will not be able to generate bills for the service period covered by that data. So be sure to have `reports.rolluphour.ttl` set to a value large enough to accommodate your billing routine.

- **"reports.rollupday.ttl"** (page 511) (default = 5616000 seconds [65 days])
- **"reports.rollupmonth.ttl"** (page 511) (default = 15552000 seconds [180 days])

If you edit any of these settings, push your changes out to the cluster and restart the S3 Service. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

4.18.5. Protecting Usage Data Through Replication

All HyperStore service usage data is stored in Cassandra, in the Reports keyspace. This data is protected against loss or corruption by maintaining multiple copies of the data, with each copy stored on a different node. During system installation the install script prompts you to specify how many replicas to keep of service metadata, which includes usage data. If you are running HyperStore in multiple data centers, during install you choose how many service metadata replicas to keep in each data center.

4.18.6. Generating a Usage Report

You can generate usage report data through either the CMC or the Admin API.

Generating a Usage Report (CMC)

In the CMC's [Usage By Users & Groups](#) page you can generate usage reports for a user, a group, a whole region, or your whole system. You can choose a report interval, a report granularity (raw, hourly, daily, or monthly roll-up), and the usage type to report on (stored bytes or stored object counts, or — if [request tracking is enabled](#) — statistics for HTTP requests and data transfer).

You can display reports in the CMC in tabular format or dynamic graph format, or download report data in comma-separated value (CSV) format.

The CMC does not support per-bucket usage reporting. For that you must use the Admin API.

Generating a Usage Report (Admin API)

The Admin API method [GET /usage](#) returns usage data for a specified user, group, or bucket. You can choose a report interval, a report granularity (raw, hourly, daily, or monthly roll-up), and the usage type to report on (stored bytes or stored object counts, or — if [request tracking is enabled](#) — statistics for HTTP requests and data transfer).

Note To retrieve usage data for a whole region or the whole system, you must execute [GET /usage](#) separately for each group.

The Admin API also supports a [POST /usage/repair/bucket](#) call that returns the current total bytes count and total objects count for a bucket; and a [POST /usage/bucket](#) call that can retrieve raw usage data for multiple specified buckets at once.

Note The Admin API call that returns the current total number of bytes and total number of objects in a bucket -- *POST /usage/repair/bucket* -- works even if bucket statistics are disabled in the system. The other bucket statistics related API calls only work if bucket statistics are enabled.

Note For information on setting up an [Elastic Stack](#) node and streaming request log data to that node to support analysis and visualization of your S3 request traffic, see "["Setting Up Elastic Stack for S3 Request Traffic Analysis"](#) (page 598).

4.19. WORM (Bucket Lock)

4.19.1. WORM (Bucket Lock) Feature Overview

Subjects covered in this section:

- *Introduction (immediately below)*
- **"HyperStore Licensing for WORM (Bucket Lock)" (page 160)**
- **"How HyperStore Enforces a Bucket Lock" (page 160)**
- **"End of Life for a Locked Bucket" (page 161)**
- **"Locked Buckets and Auto-Tiering or Auto-Expiration" (page 161)**
- **"Locked Buckets and Cross-Region Replication" (page 162)**
- **"Privileged Delete User" (page 162)**
- **"WORM Audit Logging" (page 162)**

HyperStore is capable of applying a "Write Once, Read Many" (WORM) policy to a bucket. When such a policy is implemented for a bucket, objects in the bucket cannot be altered or deleted through HyperStore interfaces until the object age exceeds a specified retention period. This type of policy can be used, for example, to meet regulatory requirements that call for data to be kept in its original form throughout a mandated retention period.

The HyperStore mechanism for implementing WORM is called a "bucket lock". To apply a lock to a bucket, the **bucket must first have versioning enabled**. When versioning is enabled on a bucket, each version of objects in the bucket is retained -- not just the current version but prior versions as well. For example, if you upload a newly created document, and then on two subsequent occasions you revise the document and upload it again, the system will retain all three versions of the document.

To apply a bucket lock to a versioning-enabled bucket is **two-step process**: First you **initiate** the lock -- which includes defining a retention period to apply to objects in the bucket -- and then you have 24 hours to **complete** the lock. As soon as you initiate the bucket lock, objects in the bucket become protected against deletion (so long as they are not already older than your defined retention period). During the subsequent 24 hours, you have the option to remove the lock from the bucket, and also if you take no action at all the system will automatically remove the lock from the bucket after the 24 hours expire. But if during the 24 hours you take the next

step and complete the lock, then the lock can never be removed from the bucket or modified. For more information on how to apply a bucket lock, see "**Setting Up a Bucket Lock**" (page 163).

Note For a given bucket you can only have one bucket lock policy in place, with just one defined retention period. If you need to satisfy multiple types of data retention requirements -- such that, for example, one category of data must be retained for one year and a different category of data must be retained for ten years -- you must use a different bucket for each data type, with each bucket having just one bucket lock policy with one defined retention period.

4.19.1.1. HyperStore Licensing for WORM (Bucket Lock)

HyperStore licenses come in three types in regard to WORM capabilities:

- Default license -- WORM disabled. With this HyperStore license type, the WORM feature is Disabled and it is not possible to lock a bucket.
- SEC17 WORM license -- WORM enabled, with no support for a "privileged delete user".
- Enterprise WORM license -- WORM enabled, with support for a "privileged delete user" (see "**Privileged Delete User**" (page 162)).

If you're not sure what license type you have, you can check the CMC's [Cluster Information](#) page to find out. If you want a different license type than what you currently have, contact your Cloudian Sales representative or Support representative.

Note If you switch from one of the WORM license type to the default license type that does not support WORM, this change will **not** unlock any buckets that are currently locked. It will only prevent the locking of additional buckets.

If you switch from a SEC17 WORM license to an Enterprise WORM license, this change **will** enable you to establish a privileged delete user for buckets that were already locked at the time that you switch to the Enterprise WORM license (as well as for any buckets that you lock from that time forward).

4.19.1.2. How HyperStore Enforces a Bucket Lock

Once a bucket lock is permanently in place, then each object in the bucket is protected against deletion until the object age exceeds the defined retention period for the bucket. This is enforced at the S3 API level: If an S3 delete request for an object in the bucket is received, the system checks the object creation date-time stamp (the date-time at which the object was uploaded to HyperStore) and adds to that the retention period defined in the bucket lock policy, and if the object is still within the retention period the S3 delete request is rejected. For example, if an object's creation date-timestamp is from December 8, 2017 and the bucket's lock policy mandates a retention period of five years, then the system will not allow the object to be deleted through the S3 API until after December 8, 2022.

It's important to note that the retention period is applied on a **per-object basis**, commencing from object creation date-time. So for example, if you were to execute a bucket lock on January 15, 2018, with a five year retention period, it is not the case that all objects in the bucket would be eligible for deletion after January 15, 2023. Rather, an existing object that had been uploaded to the bucket on April 4, 2017 would be eligible for deletion after April 4, 2022; and an object that gets uploaded to the bucket on September 9, 2018 would be eligible for deletion after September 9, 2023.

In the case of **object versions** -- different versions of the same object, as there can be in a versioning-enabled bucket -- the retention period is applied separately to each object version, commencing from the object version creation date-time. So for example if there are version1 and version2 of a given object, with version1 being the older version, version1 of the object would become eligible for deletion before version2 does.

Note also that because versioning is used -- recall that bucket locks can only be applied to buckets that have versioning enabled -- a client application's request to store a modified version of an existing object does not result in the previous version being deleted from the system. Rather, all versions of an object are retained in the system, with each version being protected against deletion throughout the defined retention period.

IMPORTANT: The HyperStore bucket lock feature ensures that throughout a defined retention period objects in a locked bucket cannot be altered or deleted through the HyperStore application layer -- specifically, through the HyperStore GUI (CMC) or HyperStore APIs (S3 API, Admin API). To fully safeguard such objects you must also maintain the physical and network security of the hardware resources on which HyperStore software is deployed, in order to prevent any tampering with the data at the operating system or physical hardware layers. Also the system clocks of HyperStore host machines must be properly configured to synchronize with an external source such as NTP to prevent tampering that could result in the premature conclusion of an object's retention period. (For NTP configuration for HyperStore hosts see "**NTP Automatic Set-Up**" (page 540).)

4.19.1.3. End of Life for a Locked Bucket

Once you complete the execution of a bucket lock, that lock will be on that bucket for the life of the bucket. The lock cannot be removed, and the bucket itself cannot be deleted so long as there are any objects in the bucket. (The latter is true of buckets in general -- you cannot delete a bucket that has objects in it.)

Consequently, end of life for a locked bucket can only occur in this way:

1. All objects in the bucket reach the end of their retention period, and so are eligible for deletion.
2. You delete all objects from the bucket.
3. Once there are no longer any objects in the bucket, you can delete the bucket in the normal way.

Note You cannot delete a user who owns a locked bucket. To delete such a user you would have to first delete the bucket, as per the sequence described above.

4.19.1.4. Locked Buckets and Auto-Tiering or Auto-Expiration

If for a given bucket you have put in place both a bucket lock and a bucket lifecycle policy -- for auto-tiering or auto-expiration -- then the **bucket lock policy supersedes the bucket lifecycle policy** in terms of the processing of individual objects in the bucket. Objects that are still within their bucket lock mandated retention period will not be auto-tiered away to a remote destination system, or auto-deleted -- even if they've reached the age at which the lifecycle policy calls for their auto-tiering or auto-expiration.

Objects **can** be auto-tiered or auto-expired **after** they reach the end of their retention periods -- as would be the case, for example, if you had bucket to which you applied a bucket lock with one year retention period as well as a lifecycle policy that called for objects to be auto-tiered 13 months after their creation.

Note If you mis-configure a bucket such that objects are scheduled to be auto-tiered or auto-expired **before** a bucket lock expires, HyperStore will execute the tiering or deletion of those objects shortly **after** the bucket lock expires. The same logic applies if you configure a bucket for both a bucket lock and "proxy mode" auto-tiering (which auto-tiers objects as soon as they are uploaded to the bucket): in this case the objects will remain in the bucket until their retention period ends and then will be auto-tiered shortly thereafter.

4.19.1.5. Locked Buckets and Cross-Region Replication

If you wish you can use the bucket lock feature in combination with the [cross-region replication](#) feature, as follows:

- You can apply cross-region replication to a **source bucket** that is locked. Likewise, you can lock a bucket that is already serving as the source bucket in a cross-region replication relationship. In either case, the objects in the source bucket will be subject to the terms of the lock policy, but the object replicas that the system copies into the destination bucket will not be subject to a lock policy (unless you separately set up a lock policy on the destination bucket).
- You can have a locked bucket be the **destination bucket** for cross-region replication. Likewise, you can lock a bucket that is already serving as the destination bucket in a cross-region replication relationship. In either case, the object replicas that the system copies into the locked destination bucket will be subject to the terms of that bucket's lock policy. The original objects in the source bucket will not be subject to a lock policy (unless you separately set up a lock policy on the source bucket).

4.19.1.6. Privileged Delete User

If you have a HyperStore **Enterprise WORM** license, the system supports establishing a privileged delete user who is allowed to delete objects from a locked bucket, before the retention period of the objects has expired.

The feature is subject to the following restrictions:

- Only HyperStore system administrators can be privileged delete users
- Privileged delete is allowed only in buckets for which you explicitly enable a privileged delete user. Even with the Enterprise WORM license, by default locked buckets do not allow deletes by system administrators or any other users.

For more information see "[Setting Up a Privileged Delete User for a Bucket](#)" (page 164).

4.19.1.7. WORM Audit Logging

WORM for a bucket is set up through S3 API calls (see "[Setting Up a Bucket Lock](#)" (page 163) for detail). On each HyperStore node, all WORM related S3 requests are logged to a dedicated WORM request log named `s3-worm.log`. This includes:

- All operations relating to locking a bucket.
- Operations that attempt to delete a bucket lock, whether successful (as when the bucket locking process is in progress but not yet complete) or unsuccessful (as when the bucket lock has already been completed, in which case the bucket lock is not allowed to be deleted).
- Operations that establish a "privileged delete user" for a locked bucket.
- Object delete requests made by a privileged delete user, for objects in a locked bucket.

By default these logs are rotated daily on each node. On each node the rotated files are retained either for **180 days** or until the accumulated WORM log file size after compression reaches 100MB, whichever occurs first. This retention period is configurable.

For more information on the WORM audit log see:

- ["S3 Service Logs" \(page 586\)](#)
- ["Log Configuration Settings" \(page 593\)](#)

4.19.2. Setting Up a Bucket Lock

Setting up a HyperStore bucket lock is subject to these restrictions:

- Your HyperStore system license must support the bucket lock feature. For detail see "["HyperStore Licensing for WORM \(Bucket Lock\)" \(page 160\)](#)".
- Only the bucket owner (or an administrator using the bucket owner's S3 credentials) can apply a bucket lock to a bucket.
- Before a bucket lock can be applied to a bucket, versioning must be enabled on the bucket. If versioning has not already been enabled on the bucket, you can enable it through the CMC (see "["Set Versioning for a Bucket" \(page 209\)](#)") or the S3 API (see [PUT Bucket versioning](#)).

Also before setting up a bucket lock on one or more buckets, consider your data retention needs carefully, bearing in mind that you can only have one retention period per locked bucket and that **once you complete a bucket lock you can never remove or modify that lock**. For more background information see "["WORM \(Bucket Lock\) Feature Overview" \(page 159\)](#)".

Setting Up a Bucket Lock (CMC)

HyperStore service users can set up a bucket lock for their own buckets through the CMC's **Bucket Properties** page. Alternatively, as a system administrator you can set up a bucket lock for a user's bucket by retrieving the user in the **Manage Users** page and then clicking **View User Data** to open a **Bucket Properties** page for that user's data.

If your HyperStore license supports the bucket lock feature, and if the bucket has versioning enabled, then for that bucket the **Bucket Properties** page includes a **WORM** tab. (If either of these conditions is not met the **WORM** tab will not display.)

To apply a bucket lock to a bucket that has versioning enabled:

1. **Initiate** the bucket lock by using the **WORM** tab in the **Bucket Properties** page for that bucket. As part of creating the bucket lock you will specify a retention period for objects the bucket. Once you initiate the lock, objects in the bucket start being protected against deletion during the defined retention period.
2. Within 24 hours of initiating the bucket lock, **complete** the bucket lock by again using the **WORM** tab in the **Bucket Properties** page. Once you complete the bucket lock it can never be modified or removed from the bucket.

During the 24 hours following initiation of a bucket lock, if you wish you can use the **WORM** tab in the **Bucket Properties** page to remove the bucket lock. Also, if after initiating the bucket lock you do not complete the lock within the subsequent 24 hours, the bucket lock will be automatically removed by the system. **Only if you complete the lock within 24 hours of initiating the lock will the lock be made permanent and unalterable.**

For detail on these steps see "["Set a Lock Policy for a Bucket \(WORM\)" \(page 212\)](#)".

Setting Up a Bucket Lock (S3 API)

To apply a bucket lock to a bucket that has versioning enabled:

1. **Initiate** the bucket lock by using the HyperStore S3 API extension method *POST Bucket lock-policy* (for detail see [POST Bucket lock-policy](#)). As part of executing this method you will specify a retention period for objects the bucket. Once you initiate the lock, objects in the bucket start being protected against deletion during the defined retention period.
2. Within 24 hours of initiating the bucket lock, **complete** the bucket lock by using the HyperStore S3 API extension method *POST Bucket lockId* (for detail see [POST Bucket lockid](#)). Once you complete the bucket lock it can never be modified or removed from the bucket.

During the 24 hours following initiation of a bucket lock, if you wish you can immediately remove the bucket lock by using the [DELETE Bucket lock-policy](#) method. Also, if after initiating the bucket lock you do not complete the lock within the subsequent 24 hours, the bucket lock will be automatically removed by the system. **Only if you complete the lock within 24 hours of initiating the lock will the lock be made permanent and unalterable.**

If at any time you are unsure of the lock status of a bucket, you can check it by using the HyperStore S3 API extension method [GET Bucket lock-policy](#).

Extending Retention for Objects in a Locked Bucket

There may be circumstances where, after some time has passed since locking a bucket, it turns out that you need to retain all or some of the objects in the bucket for an additional period of time beyond the retention period defined by the bucket lock. However, you cannot modify the lock policy on the bucket (to lengthen the retention period), since this is not allowed by the system.

In such circumstances you can use the standard S3 "PUT Bucket policy" API to extend protection for those objects. Standard bucket policies can prevent objects from being deleted, but note that unlike a bucket lock policy a standard bucket policy is not unalterable. A bucket owner can change or delete a standard bucket policy. Therefore standard bucket policies do not provide the same iron-clad data protection that a bucket lock policy does.

For HyperStore support of the S3 "PUT Bucket policy" API see [PUT Bucket policy](#).

Note The CMC does not currently support this method. The CMC cannot be used to extend retention for objects in a bucket.

4.19.3. Setting Up a Privileged Delete User for a Bucket

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Creating S3 Security Credentials for HyperStore System Admins"** (page 165)
- **"Using a Bucket Policy to Enable a Privileged Delete User for a Bucket"** (page 165)
- **"Performing Object Deletes as a Privileged Delete User"** (page 166)

- **"Audit Logging for Privileged Delete Users"** (page 167)
- **"Disabling the Privileged Delete User Feature for a Locked Bucket"** (page 167)

To use the privileged delete user feature you must have a HyperStore **Enterprise WORM** license. With this license type, individual locked buckets can be configured such that HyperStore system administrators -- and only HyperStore system administrators -- are allowed to delete objects from the bucket before the objects have reached the end of their retention period.

Note For instructions on how to lock a bucket, see **"Setting Up a Bucket Lock"** (page 163). What follows below is instructions for establishing up a privileged delete user for a bucket that's locked or that you are going to lock. Note that the privileged delete user can be set up for a bucket before or after the lock is implemented on the bucket.

To establish a privileged delete user for a bucket:

- The system administrator(s) who will be the privileged delete user(s) must have S3 security credentials (described below).
- A special bucket policy must be attached to the bucket, using the S3 API operation *PUT Bucket policy* (described further below).

Note The CMC does not currently support enabling privileged delete on a bucket or performing privileged deletes. You must use a third party S3 client to implement these operations.

4.19.3.1. Creating S3 Security Credentials for HyperStore System Admins

As a prerequisite to using the privileged delete feature, any HyperStore system admins who are going to serve as privileged delete users must have their own S3 security credentials.

Each system admin user who will be a privileged delete user must do the following:

1. Log into the CMC.
2. On the right side of the CMC's top navigation bar, hold your cursor over your login name and then in the drop-down menu select **Security Credentials**.
3. In the security credentials page's **S3 Access Credentials** section, check to see if there are any access credentials listed. If there are, you are done. If there are not, click **Create New Key**. This creates S3 access credentials for you (access key ID and secret key). In the **S3 Access Credentials** section you will see the access key ID and an option to view the secret key.

Once privileged delete has been enabled on a bucket (as described in **"Using a Bucket Policy to Enable a Privileged Delete User for a Bucket"** (page 165)), privileged delete users will need to use their S3 access credentials when performing object deletes (as described in **"Performing Object Deletes as a Privileged Delete User"** (page 166)).

4.19.3.2. Using a Bucket Policy to Enable a Privileged Delete User for a Bucket

To establish a privileged delete user for a bucket that is locked or that you are going to lock, a special bucket policy must be applied to the bucket, using the S3 API operation *PUT Bucket policy*. This operation must be executed by the bucket owner (or by a system administrator using the bucket owner's S3 credentials, or by a user to whom the bucket owner has granted permissions to execute the *PUT Bucket policy* operation on the bucket). For general information about HyperStore support of this S3 API operation see [PUT Bucket policy](#).

Note The CMC does not currently support the *PUT Bucket policy* operation. Therefore, to enable the privileged delete feature for a bucket, a third party S3 client must be used.

Note Executing the *PUT Bucket policy* operation on a bucket **replaces** any existing bucket policy for the bucket.

To establish a privileged delete user for the bucket, set the policy body as follows:

- “Effect” must be “Deny”
- “NotPrincipal” must be `{"UserType": "SystemAdmin"}`
- “Action” must be `["s3:PrivilegedDelete"]`
- “Resource” must be `["arn:aws:s3:::<bucketName>/*"]`

Here is a sample policy body that establishes a privileged delete user for a bucket named “r1bn”:

```
{  
  "Id": "some-id",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "1",  
      "Effect": "Deny",  
      "NotPrincipal": {"UserType": "SystemAdmin"},  
      "Action": ["s3:PrivilegedDelete"],  
      "Resource": ["arn:aws:s3:::r1bn/*"]  
    }  
  ]  
}
```

Note that if you have multiple HyperStore system admin users provisioned in your system the bucket policy described above enables **all** of those system admins to be privileged delete users for the bucket. If you have multiple HyperStore system admin users, the system does not support specifying just one of those system admins to be the privileged delete user for a bucket.

4.19.3.3. Performing Object Deletes as a Privileged Delete User

Once the bucket policy has been applied to the bucket, HyperStore system admins can delete objects from the bucket -- including objects that have not yet reached the end of their retention period -- by using the standard S3 API operation *DELETE Object*. In doing so the system admins **must use their own S3 credentials**, not the bucket owner's credentials. For general information about HyperStore support of this S3 API operation see [DELETE Object](#).

Note that:

- *DELETE Object* is the only operation that system admins can perform on the data, using their own S3 credentials. System admins using their own S3 credentials cannot, for instance, list the bucket contents or retrieve objects.
- System admins cannot use the CMC's **Manage Users -> View User Data** functionality to perform privileged deletes. This is because this CMC functionality uses the user's (the bucket owner's) S3 credentials, not the system administrator's S3 credentials. You can use this CMC functionality to browse the bucket contents, but not to perform privileged deletes. Instead you must use a third party S3 client to perform the *DELETE Object* operations, using your own S3 security credentials.

4.19.3.4. Audit Logging for Privileged Delete Users

When a *PUT Bucket policy* operation established a privileged delete user for a bucket, this results in an entry in the *s3-worm.log* on the node that processed the *PUT Bucket policy* request. In the entry the Operation field will indicate "s3:PrivilegedDelete".

When a privileged user deletes an object from a locked bucket, this results in an entry in the *s3-worm.log* on the node that processed the *DELETE Object* request. In the entry the Operation field will indicate "s3:DeleteObject" or "s3:DeleteObjectVersion".

For more information on this log see "**S3 Service Logs**" (page 586).

4.19.3.5. Disabling the Privileged Delete User Feature for a Locked Bucket

If the privileged delete user feature has been enabled for a locked bucket (by a *PUT Bucket policy* operation), and if later it is desired to no longer have a privileged delete user for that bucket, the bucket owner -- or a system administrator using the bucket owner's S3 credentials, or a user to whom the bucket owner has granted the required S3 permission -- can execute the *DELETE Bucket policy* operation on the bucket. This will have the effect of disabling the privileged delete user capability for that bucket.

This page left intentionally blank

Chapter 5. Cloudian Management Console (CMC)

The Cloudian Management Console (CMC) is a web-based user interface for Cloudian HyperStore system administrators, group administrators, and end users. The functionality available through the CMC depends on the user type associated with a user's login ID (system admin, group admin, or regular user). System admins can perform a wide range of system maintenance and operational tasks as well as provisioning and managing user accounts. Group admins can perform a much more limited range of tasks, pertaining specifically to their group. Regular users can use the CMC to create and configure storage buckets and to upload or download data.

Just as the CMC's functionality is tailored to the logged-in user type, so too is the CMC's online help: users can only view the Help topics for the functionality that is available to them based on their user type.

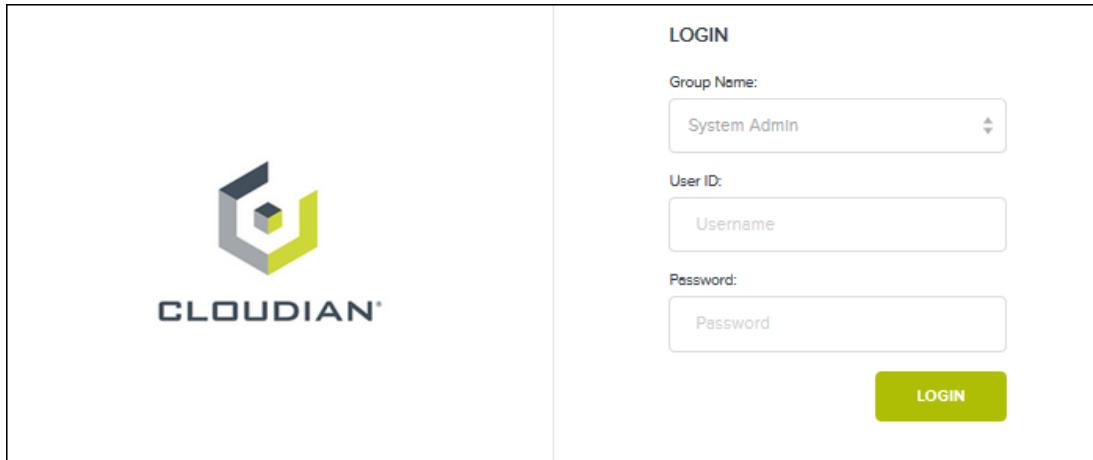
Note The CMC supports Firefox and Chrome browsers. It is recommended that you use a recent version of one of those browsers when accessing the Console. Internet Explorer is not supported — you may experience display problems with some CMC pages if you use IE.

To access the CMC for the first time, follow these steps:

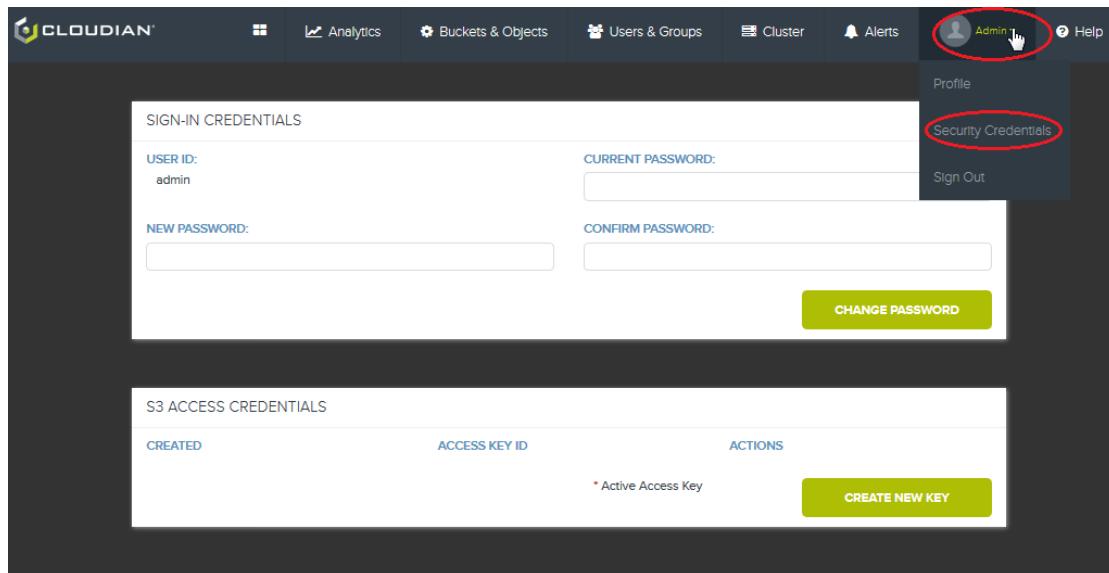
1. Point a browser to http://<CMC_host_IP_address>:8888/Cloudian

Since the CMC runs on all of your HyperStore nodes by default, you can use any node's IP address.

2. The connection will automatically switch over to SSL and you will get a certificate warning. Follow the prompts to add an exception for the certificate and accept it. You should then see the CMC's login screen.



3. Log in with the system administrator user ID *admin* and default password *public*.
4. In the upper right corner of the CMC UI, hold your cursor over your user ID ("admin"). From the drop-down menu that displays, select **Security Credentials** to open the **Sign-In Credentials** interface. For security, change the system administrator password.

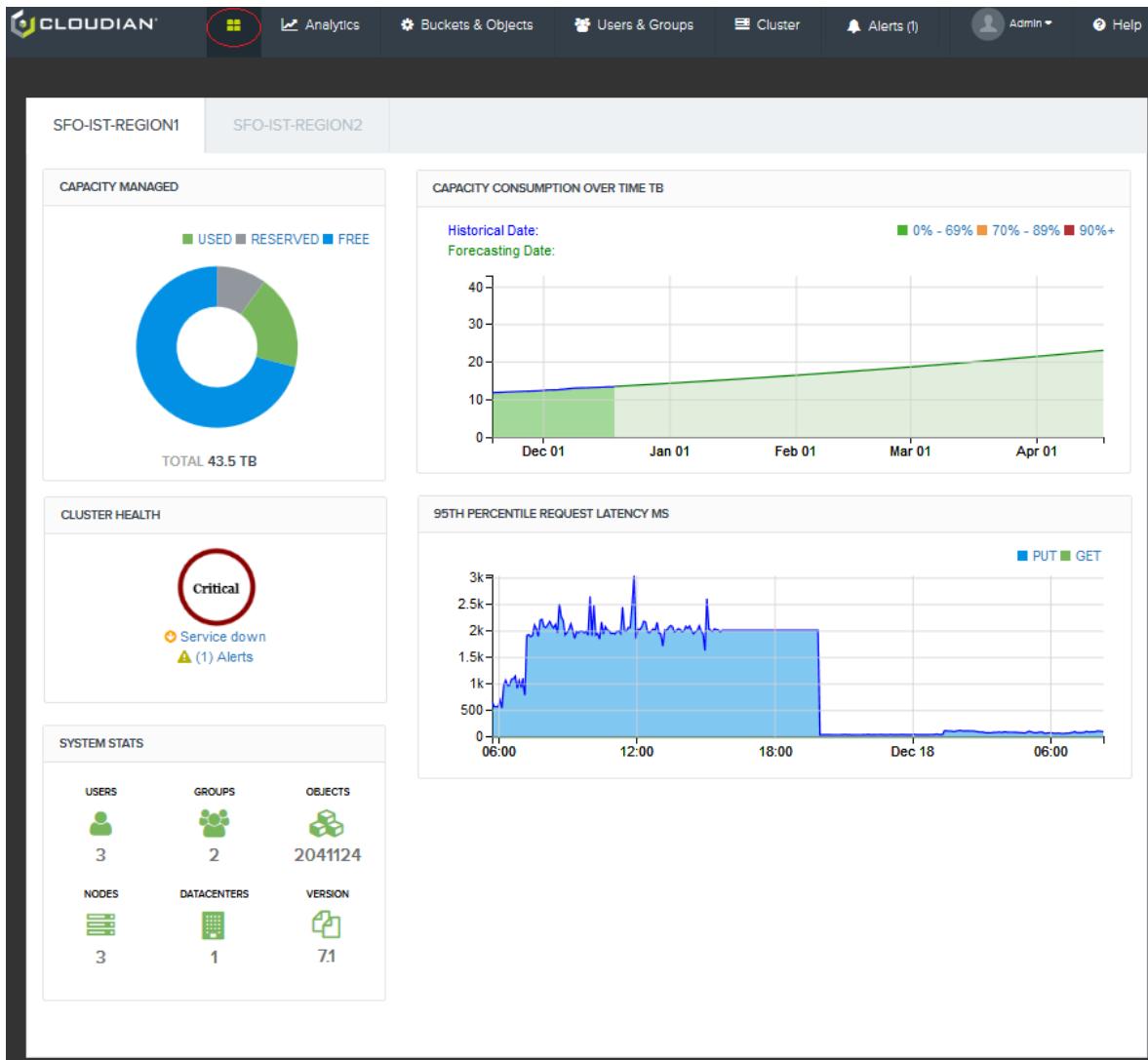


Note All user logins to the CMC are recorded in the CMC application log [cloudian-ui.log](#).

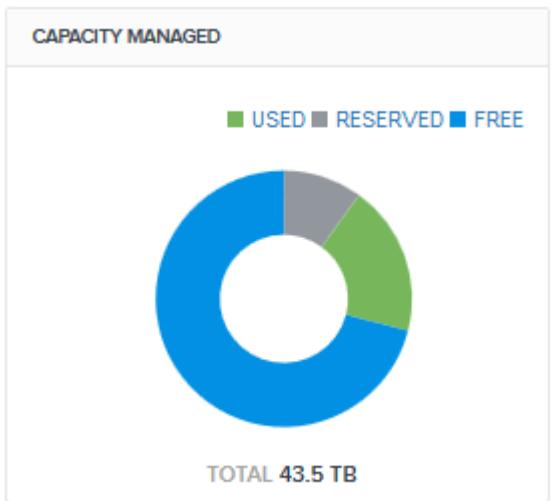
5.1. Dashboard

5.1.1. Dashboard

The CMC **Dashboard** provides a summary view of your HyperStore system status. In a multi-region system there is a separate dashboard tab for each region.



Capacity Managed



The **Capacity Managed** graphic shows the percentages of total disk space that are currently Used, Reserved, or Free, across the whole service region in aggregate. To see the percentage numbers hold your cursor over each portion of the tri-colored circle.

- The **Used** segment of the circle indicates what portion of your total system capacity is consumed by stored data. This segment displays in **green** if less than 70% of total capacity is used; or in **orange** if from 70% to 89% is used; or in **red** if 90% or more is used.

Note: The Used capacity measurement here is **raw usage** and includes overhead from object replication or erasure coding. For example a 1MB object that's replicated three times in the system counts as 3MB toward the Used capacity measurement.

- The **Reserved** segment indicates the portion of total system storage capacity that is reserved and cannot be used for data storage. The Reserved portion consists of the Linux "reserved blocks percentage" plus the HyperStore stop-write buffer:
 - By default in CentOS/RHEL the "reserved blocks percentage" for a file system (the portion of the disk space that's reserved for privileged processes) is 5% of disk capacity. In a HyperStore Appliance it's customized to 0%. See your OS documentation if you want to check or change the current reserved blocks percentage for your HyperStore host machines.
 - By default the HyperStore stop-write buffer is 10% of disk capacity. For information on this feature see "**Automatic Stop of Writes to a Disk at 90% Usage**" (page 85).

The Reserved segment always displays in **gray**.

- The **Free** segment indicates the portion of total system storage capacity that is neither used nor reserved, and is therefore available for storing new data. The Free segment always displays in **blue**.

This graphic links to the **Capacity Explorer** page, where you can see a breakdown of your available capacity by region, data center, and node.

Operation Status

OPERATION STATUS				
TYPE	OPERATION NAME	STATUS	PROGRESS	LAST UPDATE
	addNode	In progress	<div style="width: 20%; height: 10px; background-color: #ccc;"></div>	May-27-2018 10:30

The **Operation Status** section displays the status of any **in-progress** system operations that you have recently launched from the CMC. This section of the Dashboard will not appear if there are no in-progress operations.

Status reporting in this section is supported for these operation types:

- Add Node
- Add Data Center
- Add Region
- Uninstall Node
- hsstool rebalance

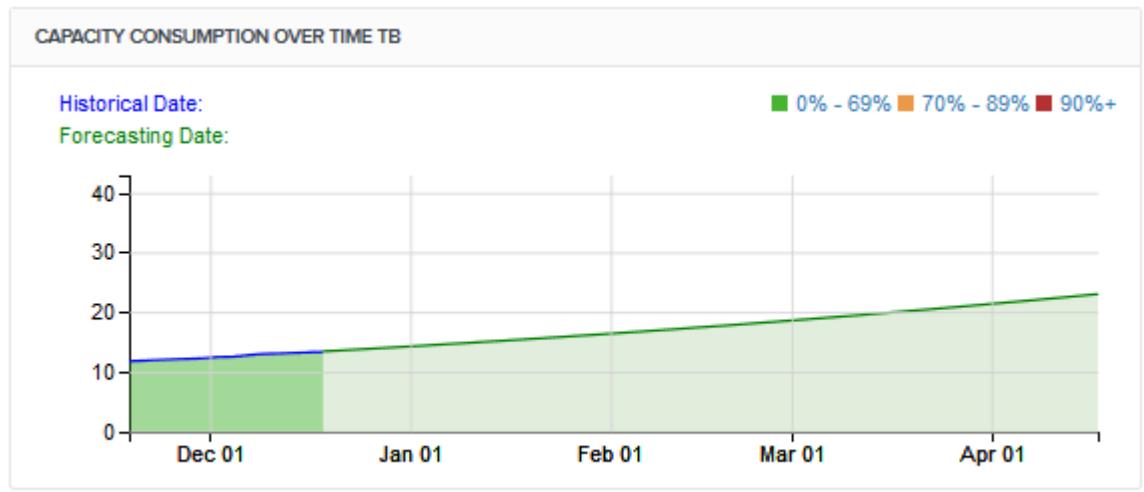
- hsstool repair or repairec
- hsstool cleanup or cleanupec

For each operation the **Operation Status** section displays the operation name, the current status, the progress (as an approximate percentage of completion), and the last update time (the last time that the CMC obtained status information for the operation).

Clicking in this section takes you to the **Operation Status** page where you can view additional status information for these in-progress operations as well as status information for operations that have recently completed.

Note For *hsstool* operations, the Dashboard's **Operation Status** section and the **Operation Status** page show status only for *hsstool* operations that you initiate through the CMC's [Node Advanced](#) page -- not for *hsstool* operations that you initiate on the command line. For commands that you launch on the command line you can use [**hsstool opstatus**](#) to track the operation status.

Capacity Consumption Over Time



The **Capacity Consumption Over Time** graphic shows the recent and forecasted raw storage consumption for the region, in GBs, TBs, PBs, or EBs (depending on the size of your system). For this graphic, storage capacity is considered to be "consumed" if it is either used or reserved. That is, **capacity "consumption" at any given time is equal to "used" capacity plus "reserved" capacity**. For more information on "reserved" capacity see **"Capacity Managed"** (page 172).

The left side of the graph (in darker shades of color) shows the storage consumption level over the past 30 days. Consumption up to 69% of total disk space capacity is shown in **green**; consumption from 70% to 89% of capacity (if applicable) is shown in **orange**, layered on top of the green portion; and consumption from 90% or above of capacity (if applicable) is shown in **red**, layered on top of the green and orange portions.

The center and right of the graph show (in lighter shades of the same colors) the 120 day forecast of raw storage consumption level for the region, based on trend analysis from the past 30 days.

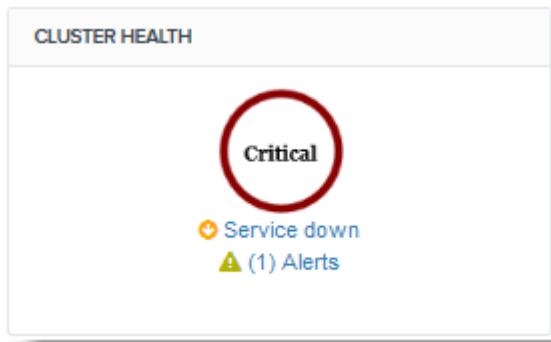
For a numerical display of the storage consumption level at a particular point in time, hold your cursor over that part of the timeline.

Links to the **Cluster Usage** page, where you can break this capacity consumption data down by data center and node.

IMPORTANT: If your cluster is projected to reach 90% usage in the next 120 days, it's time to plan for a cluster expansion. See "**Cluster Resizing Feature Overview**" (page 68).

Note A "Critical" message will display at the top of the screen if your total disk space consumption is forecasted to reach 95% of capacity in the next 90 days. The message indicates when this 95% threshold is forecasted to be reached. If an 80% threshold is forecasted to be reached, a "Warning" message displays.

Cluster Health



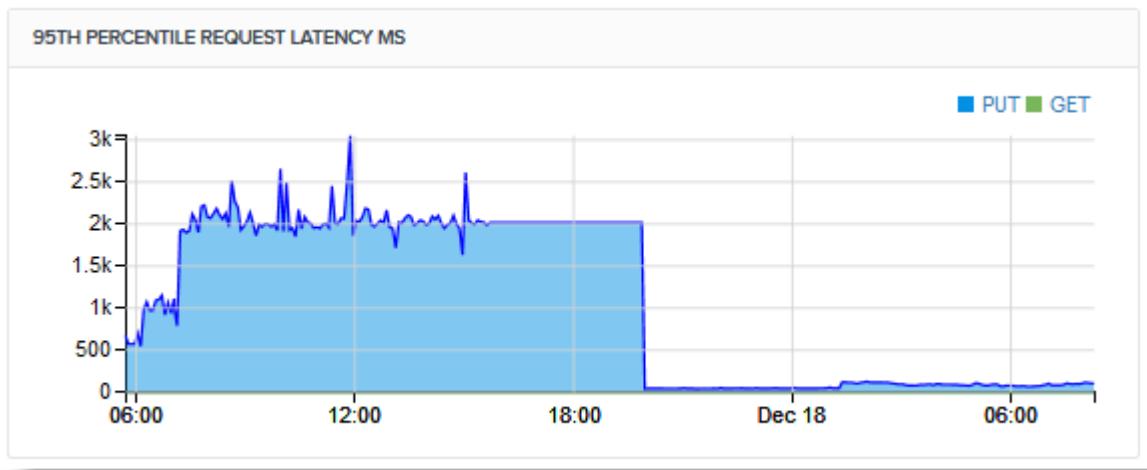
The **Cluster Health** section displays high level status information for the cluster. This section displays one or more condition messages, and an overall cluster status icon. The table below shows the possible condition messages along with the cluster status icon that displays if that condition is the most severe condition currently present in the cluster.

Cluster Status Icon	Condition Message(s)	Action to Take
	"System is X% full" (where X is 90 or more)	<p>Click the condition message to go to the Capacity Explorer page for more detail regarding system capacity usage. At this level of storage capacity utilization the system may soon stop supporting new writes if it has not done so already. It's urgent to add more nodes to your system as soon as possible. For more information see "Cluster Resizing Feature Overview" (page 68).</p> <p>Note The percentage shown in this message is the used percentage plus the reserved percentage. For more information on "reserved" capacity see "Capacity Managed" (page 172).</p>
	"Hardware issues"	<p>This message indicates that a data disk has errors or has been disabled. Click the condition message to go to the Data Centers page where you can determine the node on which the bad disk is located. Then from there click the node icon to go to the Node Status page to determine which disk is bad. You may need to replace the disk. For more information see "Replacing a Hyper-</p>

Cluster Status Icon	Condition Message(s)	Action to Take
	"Service down"	Store Data Disk " (page 430). A HyperStore service is down on a node. Click the condition message to go to the Data Centers page. By reviewing the "Service Status" section of that page you can determine which node has a service down, and which service it is. Then in the "Service Status" section click the node's hostname to go to the Node Status page where you can start the service that's down (alternatively you can use the service initialization script to start the down service as described in "Starting and Stopping Services" (page 365)).
	"Node stopped write"	A node has stopped accepting writes -- and the system has stopped directing S3 write requests to that node -- because all of the node's disks are nearly full. For more information on this condition and how to remedy it see "Automatic Stop of Writes to a Node at 90% Usage" (page 86)
	"License usage X%" (where X is 90 or more)	Contact Cloudian to request a new license. For more information see "Licensing and Auditing" (page 6). Note Only your actually used capacity counts toward this percentage -- "reserved" capacity does not count toward this.
	None	This status icon displays if the system has been disabled because your HyperStore license has expired and your grace period (if any) has ended. Contact Cloudian to request a new license. For more information see "Licensing and Auditing" (page 6).
	"System forecast 90% full in X days" (where X is 90 or less)	Aggregate disk usage in the region is forecast to reach 90% of region's total storage capacity in 90 days or less (based on usage trend analysis). Click the condition message to go to the Cluster Usage page for more detail regarding current and forecasted system capacity usage. Add more nodes to your system as soon as possible. For more information see "Cluster Resizing Feature Overview" (page 68). Note The percentage shown in this message is the used percentage plus the reserved percentage. For more information on "reserved" capacity see "Capacity Managed" (page 172).
	"Node's Disks X% Full"	A node's data disks are nearing capacity. Click the con-

Cluster Status Icon	Condition Message(s)	Action to Take
	(where X is 80 or more)	<p>dition message to go to the Capacity Explorer page for more detail regarding that node's capacity usage. Check also the capacity usage of other nodes as well, since if one node is nearing capacity other nodes may be approaching that level also. Add more nodes to your system as soon as possible. See "Cluster Resizing Feature Overview" (page 68).</p> <div style="background-color: #e0f2e0; padding: 10px; border-radius: 10px;"> <p>Note The percentage shown in this message is the used percentage plus the reserved percentage. For more information on "reserved" capacity see "Capacity Managed" (page 172).</p> </div>
	"Long proactive repair queue"	<p>A node has a long "Proactive Repair" (page 76) queue. This warning is triggered if a node's proactive repair queue has built up to the point that proactive repair needs to write data for 10,000 or more objects to the node. Click the condition message to go to the Repair Status page to see which node has status "Proactive Repair Pending". Click that node's icon to see detail about the proactive repair queue length. Then click the node's hostname to go to the Node Status page for that node to check whether any services are down on the node. From that page you can start any down services. Other possible explanations for a long proactive repair queue are that the node is in main-tanence mode or in a stop-write condition; if so then in the Data Centers page the node's icon will be blue.</p>
	"License usage X%" (where X is from 70 to 89)	<p>Storage usage in the system as a whole is at 70% of licensed usage or higher, but still below 90%. Contact Cloudian to request a new license soon. For more information see "Licensing and Auditing" (page 6).</p> <div style="background-color: #e0f2e0; padding: 10px; border-radius: 10px;"> <p>Note Only your actually used capacity counts toward this percentage -- "reserved" capacity does not count toward this.</p> </div>
	None or "(X) Alerts" (where X is the number of alerts)	<p>This status icon displays if none of the conditions noted above apply.</p> <p>Note that even if the system is considered to be Healthy, there may be unacknowledged alerts in the system. Click the condition message to go to the Alerts page where you can review the alerts and acknowledge them.</p>

95th Percentile Request Latency



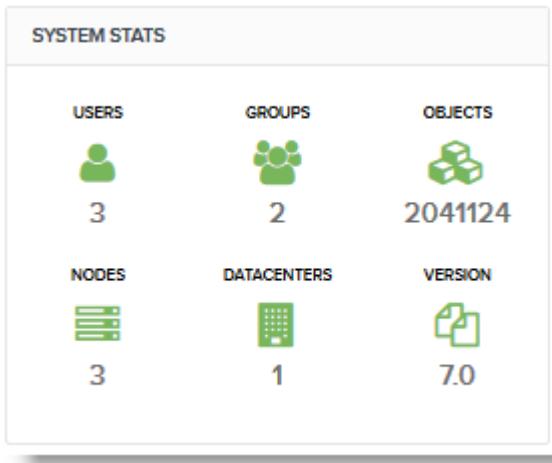
The **95th Percentile Request Latency MS** graphic shows the region-wide 95th percentile latencies for S3 GET and PUT transactions in milliseconds. New statistic values are calculated and plotted each five minutes, based on the most recent approximately 1000 GET transactions and 1000 PUT transactions. Each plotted 95th percentile latency value indicates that of the last 1000 transactions of that type, 95% completed in that many milliseconds or less.

PUT latency is shown in blue and GET latency is shown in green. Hold your cursor over any point in time to view the specific PUT and GET latency 95th percentile figure for that point in time.

Note If request activity completely stops for some period of time -- such that there is no new raw data going into the statistical calculation -- the graph will for a while continue to plot the statistic value from the last-completed 1000 transactions (resulting in a perfectly horizontal line across part or all of the inactive period) rather than the graph line immediately dropping down to zero.

Note In regard to the per-transaction latency measurements that go into calculating a 95th percentile figure over a given time period, note that S3 uploads of very large objects are typically implemented as multipart upload (MPU) operations, as S3 client applications use the S3 API calls for MPU. In terms of PUT latency metrics, each part upload of an MPU is treated as a separate transaction -- that is, a latency is recorded for each part upload rather than for the aggregate multipart upload operation. By contrast a GET of a very large object is a single transaction with a single latency measurement recorded.

System Stats



The **System Stats** graphic displays basic information about your HyperStore system:

Users

Number of users in your whole HyperStore system. If your system has multiple service regions, users are registered to the system as a whole (they are not tied to a particular region). So if you toggle through the service region tabs, this number will remain the same -- showing the number of users in the system as a whole.

Along with regular S3 service users, the total number includes system admin users and group admin users.

Links to the [Manage Users](#) page, where you can create new users or retrieve and edit existing users.

Groups

Number of user groups in your whole HyperStore system. If your system has multiple service regions, user groups are registered to the system as a whole (they are not tied to a particular region). So if you toggle through the service region tabs, this number will remain the same -- showing the number of user groups in the system as a whole.

Links to the [Manage Groups](#) page, where you can create new groups or retrieve and edit existing groups.

Objects

Number of S3 objects stored in the service region. In a multi-region system, this count is for the region for which the tab is selected at the top of the Dashboard.

This statistic counts each object, not each replica — for example, if an object is replicated three times across your cluster (to protect data durability and availability), this counts as one object, not three.

In the case of versioned objects, each version of the object counts towards this stat.

S3 objects that have been auto-tiered to Amazon S3, Amazon Glacier, or other HyperStore regions or systems do count toward this stat.

Links to the [Object Locator](#) page, where you can retrieve information about where a particular object is stored within your cluster.

Nodes

Number of nodes in the service region. In a multi-region system, this count is for the region for which the tab is selected at the top of the Dashboard.

Links to the **Node Status** page, where you can see status details for individual nodes, start and stop HyperStore services on nodes, and review and acknowledge node alerts.

Data Centers

Number of data centers in the service region. In a multi-region system, this count is for the region for which the tab is selected at the top of the Dashboard.

Links to the **Data Centers** page, where you can see your HyperStore node inventory in each of your data centers.

Version

HyperStore software version.

Links to the **Cluster Information** page, where you can view static information about your cluster such as license information, the S3 service endpoint for your system, and the mapping of specialized service roles (such as the Redis QoS database master role) to individual nodes.

5.2. Analytics

The CMC's **Analytics** tab contains the following functions:

- [Cluster Usage](#)
- [Capacity Explorer](#)
- [Usage By Users & Groups](#)
- [Object Locator](#)

5.2.1. Cluster Usage

Path: **Analytics** → **Cluster Usage**



Supported tasks:

- View recent and forecasted cluster usage

In the CMC's **Cluster Usage** page you can view cluster usage graphs that cover the past 30 days of activity (or less if the cluster has been in operation for less than 30 days). The time period is not editable. For storage capacity consumption, along with recent capacity consumption you can also view a forecast of future consumption.

IMPORTANT: See "[Cluster Resizing Feature Overview](#)" (page 68) for guidance about capacity management and the importance of early planning for cluster expansions.

The following graphs are displayed:

Capacity consumption over time

This graphic shows the recent and forecasted raw storage consumption for the region, in GBs, TBs, PBs, or EBs (depending on the size of your system). For this graphic, storage capacity is considered to be "consumed" if it is either used or reserved. That is, **capacity "consumption" at any given time is equal to "used" capacity plus "reserved" capacity**. For more information on "reserved" capacity see "[Capacity Managed](#)" (page 172).

The left side of the graph (in darker shades of color) shows the storage consumption level over the past 30 days. Consumption up to 69% of total disk space capacity is shown in **green**; consumption from 70% to 89% of capacity (if applicable) is shown in **orange**, layered on top of the green portion; and consumption from 90% or above of capacity (if applicable) is shown in **red**, layered on top of the green and orange portions.

The center and right of the graph show (in lighter shades of the same colors) the 120 day forecast of raw storage consumption level for the region, based on trend analysis from the past 30 days.

For a numerical display of the storage consumption level at a particular point in time, hold your cursor over that part of the timeline.

The drop-down list at the top of the graphic lists all the nodes in the service region. You can select from the list to view recent and forecasted usage for a specific node.

IMPORTANT: If your cluster is projected to reach 90% consumption in the next 120 days, it's time to plan for a cluster expansion. See "[Cluster Resizing Feature Overview](#)" (page 68).

Object transactions/sec (GET & PUT)

This graph shows the number of S3 transactions processed per second. The graph shows the PUT transaction activity in blue and the GET transaction activity in green. To highlight one or the other trend line, hold your cursor over the "PUT" or "GET" label off to the right of the graph.

The transactions per second values are calculated and plotted each five minutes, based on the past five minutes of activity.

Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

Throughput MB/sec (GET & PUT)

This graph shows the S3 data throughput as KB or MB or GB per second. The graph shows the PUT transaction activity in blue and the GET transaction activity in green. To highlight one or the other trend line, hold your cursor over the "PUT" or "GET" label off to the right of the graph.

The throughput per second values are calculated and plotted each five minutes, based on the past five minutes of activity.

Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

95th percentile Request Latency ms (GET & PUT)

This graph shows the 95th percentile latencies for S3 GET and PUT transactions in milliseconds. The graph shows the PUT transaction activity in blue and the GET transaction activity in green. To highlight one or the other trend line, hold your cursor over the "PUT" or "GET" label off to the right of the graph.

New statistic values are calculated and plotted each five minutes, based on the most recent approximately 1000 GET transactions and 1000 PUT transactions. Each plotted 95th percentile latency value indicates that of the last 1000 transactions of that type, 95% completed in that many milliseconds or less.

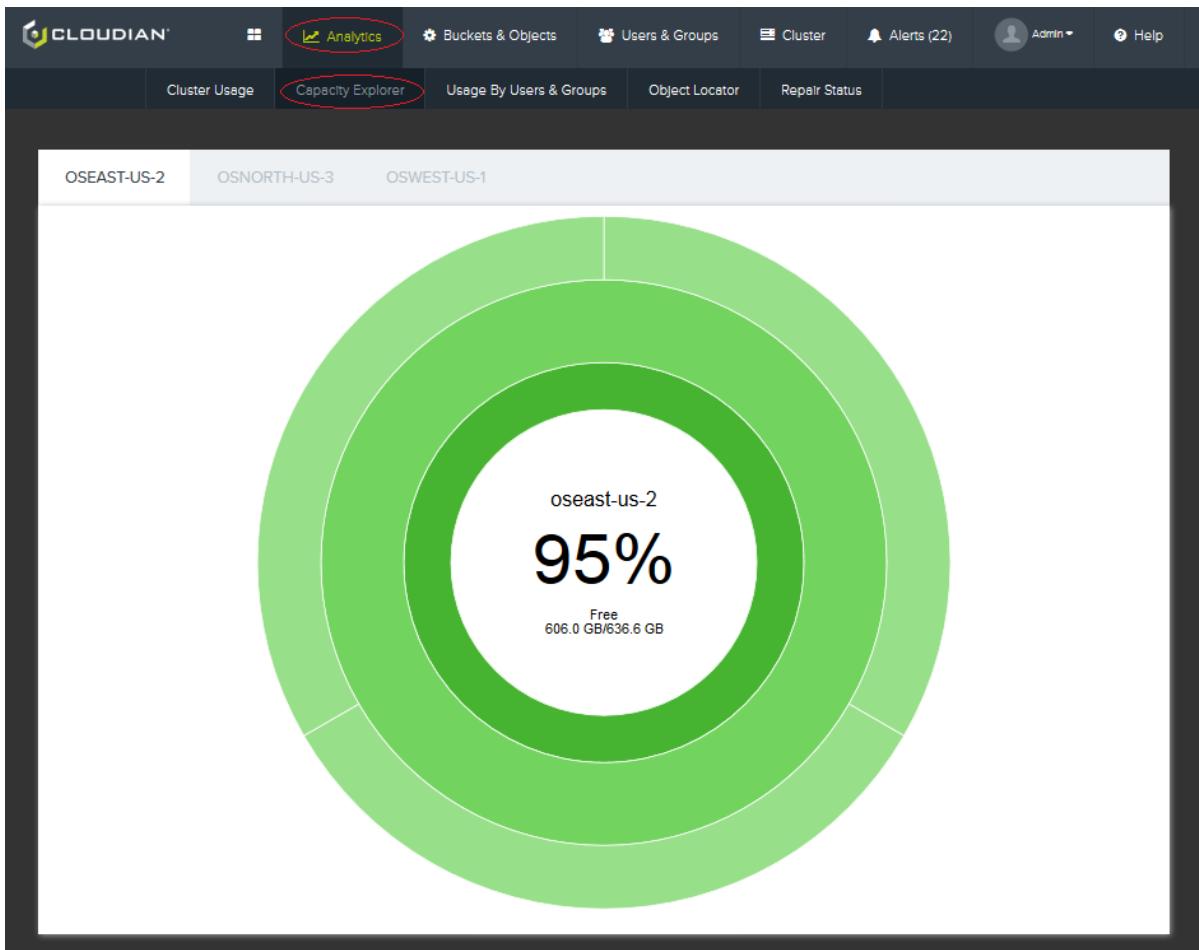
Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

Note If request activity completely stops for some period of time -- such that there is no new raw data going into the statistical calculation -- the graph will for a while continue to plot the statistic value from the last-completed 1000 transactions (resulting in a perfectly horizontal line across part or all of the inactive period) rather than the graph line immediately dropping down to zero.

Note S3 uploads of very large objects are typically implemented as multipart upload (MPU) operations, as S3 client applications use the S3 API calls for MPU. In terms of PUT latency metrics, each part upload of an MPU is treated as a separate transaction -- that is, a latency is recorded for each part upload rather than for the aggregate multipart upload operation. By contrast a GET of a very large object is a single transaction with a single latency measurement recorded.

5.2.2. Capacity Explorer

Path: [Analytics](#) → [Capacity Explorer](#)



Supported task:

- View remaining free space for S3 object data storage

With the CMC's **Capacity Explorer** page you can view your remaining free storage capacity by region, by data center, and by node. **"Free" capacity here is capacity that is neither used nor reserved** (that is, free capacity is what remains after deducting both used capacity and reserved capacity from total capacity). For more information on "reserved" capacity see **"Capacity Managed"** (page 172).

First choose a region tab at the top of the page (if you have just region there will be just one tab). Then, in the graphical display:

- The **inner circle** represents the service region as a whole
- The **middle circle** has one segment for each data center in the region
- The **outer circle** has one segment for each node in each data center

The circle segments are color-coded as follows:

- **Green** indicates that free space is 30% or more of total space for that region, data center, or node. (Slightly different shades of green are used merely to differentiate the concentric circles from each other. Green has the same meaning regardless of the particular shade of green.)
- **Orange** indicates that free space is between 10% and 29% of total space for that region, data center, or node.
- **Red** indicates that free space is less than 10% of total space for that region, data center, or node.

Hold your cursor over a segment to see specific storage space availability for that region, data center, or node, expressed as "<Free space>/<Total capacity>". Click on a segment to have the space availability information for that region, data center, or node display in the center of the circle, where it will include the free space percentage as well as the absolute numbers for free and total space. By default the free space information for the region as a whole displays in the center of the circle.

The measurements in the **Capacity Explorer** page are exclusively for HyperStore data disks -- on which S3 object data is being stored -- and do not address drives that are storing only the OS and the Cassandra and Redis databases (for system metadata). The HyperStore data disk utilization measured here is **raw** utilization and includes overhead for replication or erasure coding. For example a 1MB object that's replicated three times in the system results in 3MB of disk usage for the system.

IMPORTANT: See "**Cluster Resizing Feature Overview**" (page 68) for guidance about capacity management and the importance of early planning for cluster expansions.

5.2.3. Usage By Users & Groups

Path: **Analytics** → **Usage By Users & Group**

Supported tasks:

- "**Create a Usage Report**" (page 185)
- "**Review Usage Report Output**" (page 189)

Note For an overview of how the HyperStore system tracks service usage by groups and users, see "**Usage Reporting Feature Overview**" (page 153)

5.2.3.1. Create a Usage Report

In the CMC's **Usage By Users & Group** page you can generate service usage reports for individual users, for user groups, and for the system as a whole.

Usage reporting complies with Amazon S3 in that data storage and data transfer activity are always attributed to the bucket owner, regardless of who owns an individual object within the bucket, or who is submitting object-related requests.

To create a usage report, choose your report filtering criteria:

Group Name

- ID of the Group to report on. For a system-wide report choose "All Groups".

User

- User ID, if you want the report to be limited to a specific user's activity.

Operation

- *Storage-Bytes* — This report shows number of stored bytes. This is **net** bytes and excludes any storage policy overhead. For example, in the case of a 1MB object that's protected by 3X replication, this counts as 1MB toward Storage-Bytes -- not 3MB.
- *Storage-Objects* — This report shows number of stored objects.
- *Data Transfer-In-Bytes* — This report shows data upload activity.
- *Data Transfer-Out-Bytes* — This report shows data download activity.
- *All Requests* — This report shows HTTP PUT, GET, and DELETE activity.
- *Get/Head Requests* — This report shows HTTP GET and HEAD activity. In List form this report is identical to the Data Transfer-Out-Bytes. In Graph form, Get/Head Requests graphs a request count while Data Transfer-Out-Bytes graphs number of bytes.
- *Put/Post Requests* — This report shows HTTP PUT and POST activity. In List form this report is identical to the Data Transfer-In-Bytes. In Graph form, Put/Post Requests graphs a request count while Data Transfer-In-Bytes graphs number of bytes.
- *Delete Requests* — This report shows HTTP DELETE activity.

Note By default only the Storage-Bytes and Storage-Objects reports are enabled. To enable the other types of reports, you must enable the "**Track/Report Usage for Request Rates and Data Transfer Rates**" (page 299) setting on the CMC's Configuration Settings page.

Report Granularity

- *Hours* — Break the report period down into hourly intervals. The report will show only hours that have completed (to the top of the hour), not the currently in-progress hour.
- *Days* — Break the report period down into daily intervals. The report will show only days that have completed (midnight to midnight), not the currently in-progress day.
- *Months* — Break the report period down into monthly intervals. The report will show only months that have completed, not the currently in-progress month. For example, if today is June 25th and you set a Time Period from March 1st up through today, with Granularity of "Months", the report will show monthly usage data for the months of March, April, and May (and not June, since June hasn't completed yet).
- *Raw* — List every report-relevant transaction individually, with timestamp. For raw granularity, you must choose a custom Time Period that spans no more than 24 hours.

Time Period

- *Current Billing Period* — The current calendar month up to today's date.
- *Previous Billing Period* — The last completed calendar month.
- *Last Week* — The last completed calendar week (Monday to Sunday).
- *Last Month* — The last completed calendar month. This is equal to Previous Billing Period.

- **Custom Period** — This option opens a calendar tool in which you can specify a particular report begin date and report end date.

Region

- The Cloudian HyperStore service region for which to report usage activity. Choose "All" to show aggregate activity across all regions. This field displays only if your HyperStore system has multiple service regions.

Note "All" regions is a valid option only if you are generating a List report — not a Graph or CSV report. Multi-region Graph or CSV reports are not currently supported.

Traffic Type

- Choose "Normal" or "Whitelist". "Whitelist" refers to request traffic that originates from white-listed source IP addresses (traffic subject to special pricing), and is an option only if white-listing is enabled in the system. "Normal" refers to all other traffic.

Note The Traffic Type option does not apply to reports with Report Granularity "Raw", nor to reports with Storage-Bytes or Storage-Objects as the Operation. The Traffic Type field will not display if you choose those types of reports.

After specifying your usage report parameters, click:

- **List** to display a traditional tabular report.
- **Graph** to display a graphical report.

Note: Graphs are not supported for reports for which the selected Operation category is "All Requests", reports for which the selected Report Granularity is "Raw", or reports for which the selected Region is "All". For more on graphing functionality see Manipulating Graph Reports.

- **Download CSV** to download a comma-separated value version of the report to your computer.

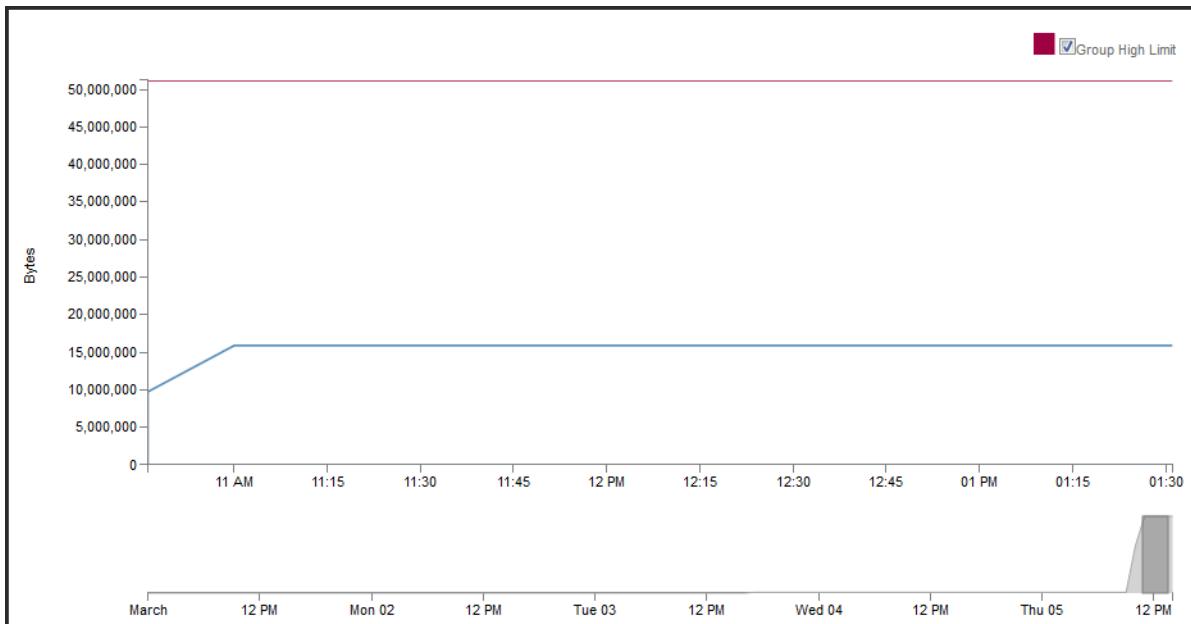
Note: In CSV-formatted report output, all data size values are expressed in bytes (rather than KBs, MBs, or GBs).

Manipulating Graph Reports

When you generate a graph report you can manipulate the graph display.



In the small graph at the bottom of the page click and drag horizontally to create a dark grey block. You can then click and drag the edges of the block to expand or contract the time period shown in the main graph. A narrower block provides a more granular view while a wider block provides a less granular view. You can also click the block's center and drag the block to shift the main graph to an earlier or later time interval (within the bounds of the Time Period that you selected when generating the graph).



For reports on usage for a single group, the graph will show a horizontal line that indicates the quality of service (QoS) limit for that group, for the service metric that's the focus of the report (storage bytes, for instance). For reports on usage for a single user, two horizontal lines display in the graph — one indicating the user's QoS limit and one indicating the user's group's QoS limit. You can uncheck the QoS display boxes to hide these lines.

Note **Hiding the QoS lines** will be desirable if the current usage indicated by the graph is just a small fraction of the QoS limit. When QoS lines are included, the graph's Y axis will necessarily scale up to include the QoS limit level, which can make it hard to view variation within the usage level if usage is confined to just a small portion of the Y axis. Without the QoS lines the Y axis will scale appropriately to the usage level.

The **metric used on a graph's Y-axis auto-scales** to units that are most appropriate for the particular quantities being conveyed. Specifically, for a given report the Y-axis may be expressed in terms of bytes, KBs, MBs, or GBs, depending on the usage level during the full 30-day graphing interval. Pay attention to the Y-axis label to see what metric is being used.

If your **report is based on granular data but covers a long time period**, the labels on the X-axis will be less granular than the source data. For example, if you generate a report based on hourly granularity and a month-long reporting period, the X-axis labels will indicate days not hours. However, the graph content — the trend line itself — will be based on hourly data points. In a month with 30 days, 720 hourly data points (30 X 24) will go into determining and drawing the trend line. Note that if you use the click-and-drag controls (below the graph) to zoom in on a shorter period of time — such as a day or a portion of a day — then the X-axis labels will show hours rather than days.

5.2.3.2. Review Usage Report Output

This topic clarifies the meaning of the data that you will see in your Cloudian HyperStore S3 usage reports. Below is an example of a "List" style report, with hourly granularity.

RESULTS						
REGION	DATE/TIME	USER	GROUP	OPERATION	AVERAGE	MAXIMUM
region1	Sep-19-2015 10:00 -0400	PubsUser1	Pubs	Storage Bytes	136.1 K	136.1 K
region1	Sep-19-2015 09:00 -0400	PubsUser1	Pubs	Storage Bytes	136.1 K	136.1 K
region1	Sep-19-2015 08:00 -0400	PubsUser1	Pubs	Storage Bytes	136.1 K	136.1 K
region1	Sep-19-2015 07:00 -0400	PubsUser1	Pubs	Storage Bytes	161.9 K	170.5 K
region1	Sep-19-2015 06:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 05:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 04:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 03:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 02:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 01:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-19-2015 00:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 23:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 22:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 21:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 20:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 19:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 18:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 17:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 16:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K
region1	Sep-18-2015 15:00 -0400	PubsUser1	Pubs	Storage Bytes	170.5 K	170.5 K

20

NEXT

Starting from the left of "List" style reports, the first several columns have consistent meaning across the different types of usage reports:

Region

The HyperStore service region in which the usage occurred.

Date/Time

- For reports with hourly granularity, this field displays each hour for which activity occurred during the

reporting period. For example, in a row with Date/Time "Jan-14-2019 07:00 <UTC offset>" the reported activity is from 07:00 through 07:59.

- For reports with daily granularity, this field displays the day as, for example, "Jan-14-2019 <UTC offset>", and the reported activity is from that day, from midnight to midnight.
- For reports with monthly granularity, this field displays the month as, for example, "Jan-2019 <UTC offset>", and the corresponding activity is from that calendar month.
- For reports generated with granularity "Raw", this field displays the transaction timestamp.

Note Times are in the local time zone of your browser.

Note Usage reports show activity only for completed intervals, not in-progress ones. For example, an hourly report shows activity only for completed hours, not the currently in-progress hour. Likewise, a daily report shows activity for completed days, not the currently in-progress day.

User

- For a report on a specific user's activity, this field displays the user's ID. For system-wide or group-wide reports, this field displays an asterisk.

Group

- Group ID. For a system-wide report this field displays "ALL".

Operation

Operation field values will be one of the following:

- HTTP PUT/POST
- HTTP GET/HEAD
- HTTP DELETE
- Storage Bytes
- Storage Objects

Additional columns display depending on the **Operation** that you selected when you chose your report parameters in the upper part of the screen:

*For Storage reports ("Storage-Bytes" or "Storage-Objects" from the **Operation** selection menu), these columns display on the right-hand side of the List report:*

Average

- Average storage level during the time period specified in the Date/Time column (in bytes or number of objects depending on the report type).

Maximum

- Maximum storage level during the time period specified in the Date/Time column (in bytes or number of objects depending on the report type).

Note In usage reports, Storage-Bytes are a measure of **net** bytes and exclude any storage

policy overhead. For example, in the case of a 1MB object that's protected by 3X replication, this counts as 1MB toward Storage-Bytes -- not 3MB.

For *Data Transfer* reports ("Data Transfer-In-Bytes" or "Data Transfer-Out-Bytes" from the **Operation** selection menu -- which in the report results will show as "HTTP PUT/POST" or "HTTP GET/HEAD" in the Operation column), these columns display on the right-hand side of the List report:

Data Transfer

- Amount of data transferred during the time period specified in the Date/Time column.

Average

- Average size of an individual data transfer during the time period specified in the Date/Time column. For example in the case of "Data Transfer-In-Bytes" reports, this would be the total data transferred in during the time period divided by the number of PUT and POST requests processed during the time period, to arrive at the average size of an inbound data transfer during the time period.

Maximum

- Maximum size of an individual data transfer during the time period specified in the Date/Time column. For example in the case of "Data Transfer-In-Bytes" reports, this would be the largest single PUT or POST processed during the time period.

For *Requests* reports ("Get/Head Requests", "Put/Post Requests", "Delete Requests", or "All Requests" from the **Operation** selection menu), this column displays on the right-hand side of the List report:

HTTP Requests

- Request count for the time period specified in the Date/Time column.

5.2.4. Object Locator

Path: **Analytics** → **Object Locator**

Supported task:

- View storage location information for an S3 object

To view storage location information for an object:

1. Enter the bucket name. For example, *bucket1*. This field is case-sensitive.
2. Enter the full object name including "folder" path, if any. For example, *file1.txt* (for a file at the root level of the bucket) or *Videos/Vacation/Italy-2014.mpg*. This field is also case-sensitive.
3. Optionally, if versioning is enabled on the bucket that contains the object, enter the version ID of a particular version of the object. Version IDs are system-generated hexadecimal values (for example, *fe1be647-5f3b-e87f-b433-180373cf31f5*). If versioning has been used for the object but you do not specify a version ID in this field, location information will be retrieved for the most recent version of the object.
4. Click **Find**.

This function executes the *hsstool whereis* command. The results that display are the same as those that display if you run the *hsstool whereis* command on the object (on the **Cluster → Nodes → Advanced** page). For description of the result elements, see [hsstool whereis](#).

5.3. Buckets

Path: **Buckets & Objects → Buckets**

NAME	REGION	POLICY	Properties	Delete
bucket1	region1	Replication-3X		
bucket2	region1	Replication-3X		

Supported tasks:

- "**Add a Bucket**" (page 192)
- "**Set Bucket Properties**" (page 194)
- "**Delete a Bucket**" (page 214)

Note System administrators do not have their own S3 account credentials and therefore cannot use the **Buckets & Objects** page for their own data storage purposes. However, system admins can access the **Buckets & Objects** page on behalf of regular service users, via the [Manage Users](#) page. System admins can also access the **Buckets & Objects** page by creating a regular user account for themselves (in the [Manage Users](#) page) and then logging into the CMC as a regular user.

5.3.1. Add a Bucket

A "bucket" is a logical container in which you can store data objects — comparable to a root folder in a conventional file system. You must create at least one bucket before you can store any data objects. Optionally you can have more than one bucket.

Note By default each user is allowed a maximum of 100 buckets. You can change this setting in the CMC's [Configuration Settings](#) page.

To create a bucket:

1. In the [Buckets](#) list view, click **Add New Bucket**. This opens the bucket creation interface.

The screenshot shows a 'BUCKETS' interface with a 'ADD NEW BUCKET' button. Below it, there are three main input fields: 'Bucket Name' (with placeholder 'Bucket Name'), 'Region' (set to 'region1'), and 'Storage Policy' (set to '*Replication-3X'). Underneath these, a 'Replication Policy Description' field contains the text 'Replication 3X In One DC'. At the bottom right are two buttons: 'CANCEL' and 'CREATE'.

2. Enter a bucket name. Your **bucket name must meet these restrictions**:

- Must be globally unique across the HyperStore service. If you choose a bucket name that's already been taken, an error message will display in the UI.
- Must be at least 3 and no more than 63 characters long.
- Only lower case letters (a-z), numbers (0-9), dash (-), period (.), and underscore (_) are allowed. The use of the non-alphanumeric characters -- dash, period, or underscore -- is subject to certain restrictions. A valid bucket name:
 - Must start with a letter or a number.
 - Must not end with a dash or a period.
 - Must not contain two or more adjacent periods.
 - Must not contain dashes next to periods (e.g., "my-.bucket.com" and "my.-bucket" are invalid).
 - Must not be in the form of an IPv4 address (e.g. "192.168.5.4").
 - Must not contain underscores if the bucket is being created in a non-default region of the S3 service. Underscores are allowed in bucket names for buckets created in the default service region. However, **if you use an underscore in a bucket name you will not be able to use auto-tiering** for the bucket (for transitioning objects to Amazon or other remote destinations on a configurable schedule). It's **best not to use underscores** when naming new buckets, in case you may want to enable auto-tiering on the bucket immediately or in the future.

3. From the "Region" drop-down list, select the HyperStore service region in which you want the bucket to be created. If you have only one service region, that will be the only region listed.
4. From the "Storage Policy" drop-down list, select a storage policy to apply to the data that will be stored in this bucket. A storage policy is a method for protecting data against loss or corruption. The policies are pre-configured by system administrators. When you select a policy from the drop-down list a brief policy description displays.

Note: If you do not select a storage policy the system default storage policy is automatically applied to the bucket. In the list of policies, the system default policy is the one listed first and pre-fixed by an asterisk (*).

After a bucket is created, it cannot be assigned a different storage policy. The storage policy

assigned to the bucket at bucket creation time will continue to be bucket's storage policy for the life of the bucket.

- Click **Create** to create the bucket.

The newly created bucket will then appear in the **Buckets** list view.

5.3.2. Set Bucket Properties

To set bucket properties, in the CMC's **Buckets** page click the **Properties** link for the bucket that you want to work with. This displays the bucket properties interface for that bucket.

BUCKET PERMISSIONS	GRANTEE	READABLE	WRITABLE	ACP READABLE	ACP WRITABLE
	Public	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Log Delivery	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Supported tasks:

- **"Set Custom S3 Permissions for a Bucket"** (page 194)
- **"Set "Canned" S3 Permissions for a Bucket"** (page 196)
- **"View a Bucket's Storage Policy Information"** (page 197)
- **"Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration"** (page 197)
- **"Configure a Bucket as a Static Website"** (page 205)
- **"Configure Cross-Region Replication for a Bucket"** (page 207)
- **"Set Versioning for a Bucket"** (page 209)
- **"Set Logging for a Bucket"** (page 211)
- **"Set a Lock Policy for a Bucket (WORM)"** (page 212)

5.3.2.1. Set Custom S3 Permissions for a Bucket

To choose who you want to be able to access your bucket using S3 protocol applications, and which specific permissions you want to give them, select the **Bucket Permissions** tab of the CMC's **Bucket Properties** interface. (This is the tab that displays by default.)

GRANTEE	READABLE	WRITABLE	ACP READABLE	ACP WRITABLE
Public	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Log Delivery	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

1. Select the S3 permissions to grant to the **public at large** (the "Public" row). Supported permission types are:

Readable

Grantee can view a list of the bucket's contents. This enables only a list view; it does not enable opening or downloading of individual files. (To allow read permissions on individual files, see "**Set Object Properties**" (page 219).)

Writable

Grantee can upload objects to the bucket, replace existing objects in the bucket, and delete objects from the bucket.

ACP Readable

Grantee can view the current permission settings for the bucket.

ACP Writable

Grantee can change the permission settings for the bucket.

2. Select the S3 permissions to grant to **all registered users of the Cloudian HyperStore service** (the "Authenticated Users" row). The supported permission types are the same as described in Step 1.
3. If you want the HyperStore system to be able to store the bucket's server access logs in the bucket, give the "Log Delivery" user Writable permission.
4. If you want to grant S3 permissions to **specific registered users of the Cloudian HyperStore service**:
 - a. Click **Add New**. A new grantee row will display.
 - b. Specify the grantee by <groupID>|<userID>, with the vertical bar separator between groupID and userID. For example, "sales|kurihara123". (Do not include the quote marks.)

Note: The CMC does not validate the user IDs that you enter. If you enter a user ID that does not belong to an active Cloudian HyperStore user, your action will have no effect.

If you want to grant permissions to **all members of a particular Cloudian HyperStore**

user group, specify the grantee as <groupID>. For example "sales". (Do not include the quote marks.)

- c. Select the S3 permissions to give the grantee. The supported permission types are the same as described in Step 1.

5. Click **Save**.

If you add a specific Cloudian HyperStore user as a grantee, and then later want to remove that grantee, click the **Delete** button for that grantee, then click **Save**.

5.3.2.2. Set "Canned" S3 Permissions for a Bucket

"Canned" S3 permissions (also known as "Canned ACLs") are pre-defined sets of S3 permissions that you can assign to a bucket. To set canned permissions on a bucket, select the **Bucket Canned ACL** tab of the CMC's [Bucket Properties](#) interface.

1. From the drop-down list, choose the canned ACL to assign to the bucket. You can only choose one. Options are:

Private

Bucket owner has full access rights and no one else has access rights. This is the default.

Public Read

Bucket owner has full access rights and the Public grantee group (the public at large) has read access. This means the public will be able to read a list of the bucket contents. (To allow read permissions on files, see "[Set Object Properties](#)" (page 219).)

Public Read Write

Bucket owner has full access rights and the Public grantee group (the public at large) has read and write access. The public will be able to read a list of contents in the bucket, and also to upload files to the bucket or delete files from the bucket.

Authenticated Read

Bucket owner has full access rights and the Cloudian User group (all users of the Cloudian HyperStore service) has read access.

Log Delivery Write

Bucket owner has full access rights and the Log Delivery group has write access and permission to read the access settings on the bucket. This canned ACL enables the HyperStore system to store the bucket's server access logs in the bucket.

2. Click **Save**.

Note A bucket can have only one canned ACL attached to it at a time. If you apply one canned ACL to a bucket, and then later apply a different canned ACL, the first canned ACL will be removed.

5.3.2.3. View a Bucket's Storage Policy Information

When you first create a bucket, you assign it a "storage policy" — a method by which data in the bucket will be protected against loss or corruption. Subsequently you can select the **Storage Policy** tab of the CMC's [Bucket Properties](#) interface to view information about the storage policy that the bucket uses.

BUCKETS			OBJECTS				
			+ ADD NEW BUCKET				
NAME	REGION	POLICY					
bucket1	region1	Replication-3X					
BUCKET PERMISSIONS	BUCKET CANNED ACL	STORAGE POLICY	LIFECYCLE POLICY	STATIC WEBSITE HOSTING	CROSS REGION REPLICATION	VERSIONING	LOGGING
NAME	DESCRIPTION	DATA DISTRIBUTION POLICY		NO OF REPLICAS	EC K+M VALUE		
Replication-3X	3X Replication in DC1	Single DC		3	N/A		

The display includes the storage policy name and description, whether data in the bucket is stored in one data center or multiple data centers, and whether data in the bucket is replicated or erasure coded.

You can control which user types can view this tab -- system admins, group admins, and/or regular users -- with the **"bucket.storagepolicy.showdetail.enabled"** (page 535) setting in *mts-ui.properties.erb*. By default all user types can view this tab.

Note You cannot change a bucket's storage policy.

5.3.2.4. Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration

The HyperStore system supports configuring bucket lifecycle policies so that objects are automatically moved (auto-tiered) to a different object storage system on a defined schedule, or automatically deleted (expired) on a defined schedule. You can also create combination lifecycle policies such that objects are auto-tiered to a different storage system and then later deleted from that system.

A lifecycle policy can apply to all objects in a bucket, or to a subset of objects as identified by object name prefix. You also have the option of creating multiple lifecycle rules for a bucket, with each rule applied to a different subset of objects based on object name prefix.

Note By default system configuration, the bucket lifecycle auto-tiering functions are disabled in the CMC. For instructions on enabling these functions, and controlling which auto-tiering options are presented to users, see "[Setting Up Auto-Tiering](#)" (page 56).

Note You cannot create an auto-tiering lifecycle rule for a bucket that has an underscore in its name.

To create a bucket lifecycle rule:

1. Select the **Lifecycle Policy** tab of the CMC's [Bucket Properties](#) interface, then click **Add New Rule**. The **Add New Bucket Lifecycle Rule** interface displays.



2. Enter a descriptive "Rule Name" for the rule you are creating. Spaces are allowed in the name.
3. Optionally enter the "Object Prefix" for the subset of objects for which you want to create a lifecycle rule. To have the lifecycle policy apply to **all objects in the bucket**, leave the "Object Prefix" field empty.

Note If you intend to use "Bridge Mode" (whereby objects are auto-tiered immediately after being uploaded to HyperStore), leave the "Object Prefix" field empty. Bridge Mode does not support filtering by prefix.

If entering an object prefix, the path should start from the root level of your bucket, and you do not need to include a leading forward slash (""). You should however include a trailing forward slash to demarcate the "folder" name from its "contents" (which will be subject to the lifecycle policy that you're configuring). For example:

Projects/archived/2016/

This object prefix would apply to all objects with names that start with *Projects/archived/2016/*, such as *Projects/archived/2016/01/JanuaryOverview.docx* and *Projects/archived/2016/02/FebruaryOverview.docx*.

4. Select:
 - The "Enable Tiering" checkbox, if you want objects to be moved to a different storage system on a schedule
 - The "Expire Objects" checkbox, if you want objects to be deleted on a schedule (and/or to configure deletion of incomplete multipart uploads and expired object delete markers)
 - Both checkboxes if you want to configure a rule that moves objects to a different storage system and then later deletes them from that system.
5. Configure the rule parameters:

Object Tiering

OBJECT TIERING

SCHEDULE

CURRENT VERSION
 PREVIOUS VERSION

OBJECT TIERING BUCKET LEVEL SETTING

DESTINATION	TIERING CREDENTIAL	
<input checked="" type="radio"/> Tier to AWS S3	Endpoint <input type="text" value="https://s3.amazonaws.com"/>	<input type="checkbox"/> Retain Local Copy
<input type="radio"/> Tier to AWS GLACIER		<input type="checkbox"/> Bridge Mode (Proxy)
<input type="radio"/> Tier to Google	Access Key: <input type="text"/> Secret Key: <input type="text"/>	GET REQUEST HANDLING
<input type="radio"/> Tier to Azure		<input checked="" type="radio"/> Stream
<input type="radio"/> Tier to Spectra	Bucket Name: <input type="text"/>	<input type="radio"/> Require Restore
<input type="radio"/> Tier to Custom Endpoint		

LIFECYCLE RULE BUCKET LEVEL SETTING

Use Creation Date/Time Use Last Access Time

Current Version

Select this checkbox to set a tiering schedule for the current version of objects in the bucket. If your bucket does not use [versioning](#), then current versions are the only versions of objects in your bucket and this is the appropriate type of schedule to set. If your bucket does use versioning, select this checkbox to set a tiering schedule specifically for the **current version** of objects in the bucket (you can set a different schedule for non-current versions of objects as described in "Previous Version" below).

- If you want tiering of the objects to occur on a particular date, choose the "After Date" radio button and enter the desired tiering date.
- If you want tiering of the objects to occur a certain number of days after the objects were last accessed (retrieved or modified) select the "Days After Last Access Date/Time" radio button and enter the number of days.
- If you want tiering of the objects to occur a certain number of days after the objects were created, at the bottom of the page choose the "Use Creation Date/Time" radio button, then back up in the scheduling section choose the "Days After Creation Date" radio button and enter the number of days.

Previous Version

This option is applicable only if the bucket uses [versioning](#). Select this checkbox to set a tiering schedule for **non-current versions** of objects (object versions that have been superseded by a newer version). Tiering scheduling for non-current versions of objects is always based on the number of days since the objects became non-current (the number of days since being superseded by a newer version of the object). Enter the desired number of days.

For example, if you enter 365 here, then if version1 of an object is made non-current by the uploading of version2 of that object on July 13, 2018, then version1 of the object will be auto-tiered a year later (regardless of whether any additional versions of the object were uploaded during the intervening year).

Destination

The destination that objects will be transitioned to.

Depending on system configuration (as described in "**Setting Up Auto-Tiering**" (page 56)), you may or may not have multiple options to choose from in this section. The system configuration may be such that all auto-tiering goes to the same default destination, in which case you will see the destination name displayed here and there will be no choice for you to make.

Alternatively, the system configuration may be such that you can choose a tiering destination from among several options such as:

- AWS S3
- AWS Glacier
- Google Cloud Storage
- Microsoft Azure
- Spectra BlackPearl

Note: The destinations and corresponding endpoint URLs that the interface displays here are configurable in *common.csv*. For details see "**Configure Tiering Destinations**" (page 58).

The system configuration may also be such that a "Tier to Custom Endpoint" option displays, in addition to the destination types listed above. If you choose this Custom option, an editable "Endpoint" field displays in which you can enter the URL of an **S3-compliant** system you want to tier to (the field is populated with *https://s3.amazonaws.com* by default but is editable). For example this could be an Amazon S3 or Google URL, or the URL of a different HyperStore region or system. Note that the system does not support specifying a Glacier, Azure or Spectra BlackPearl URL as a Custom endpoint -- the tiering operations will fail. To tier to one of those destination types select the radio button for that destination type, not the Custom Endpoint radio button.

If you enter a custom tiering URL be sure to include the *https://* part (or *http://* if the system does not support SSL).

IMPORTANT: If you use a third party cloud service such as Amazon, Google, or Azure for storage of auto-tiered objects you (or your organization) will incur charges from that provider per the terms of your service agreement with them..

Note Auto-tiering restrictions based on destination type:

- * The largest object size that can be auto-tiered to Amazon, Google, or other S3-compliant destinations is 100GB. If you want to tier objects larger than this, consult with Cloudian Support. This 100GB limit does not apply to tiering to Azure or Spectra BlackPearl.
- * Tiering to Azure or Spectra BlackPearl is not supported for source buckets that have versioning enabled or that have had versioning enabled in the past.
- * When auto-tiering to Spectra BlackPearl is used for a bucket, objects in the bucket will not be auto-tiered unless they are larger than 5MB. Objects 5MB or smaller will remain in HyperStore. To change this limit consult with Cloudian Support.

Note After you Save your lifecycle rule, you will not be able to change the tiering destination associated with this rule. Also, if you plan to create multiple auto-tiering rules for a single bucket (for different object prefixes), all such rules must use the same tiering destination system.

Tiering Credential

For any tiering destination other than a system-default endpoint, you must enter your account security credentials for accessing the destination system (your Access Key and Secret Key -- or if the destination is Azure, your Account Name and Account Key). The system will then use these account credentials when it transitions objects from the bucket to the destination account.

Note If you create multiple auto-tiering lifecycle rules for a single bucket (for different object prefixes), all such rules must use the same destination account credentials.

Bucket Name (or Container Name, for Azure)

Optionally, the name of the bucket to transition objects into, in the tiering destination system. This can be either:

- The **name of a bucket that already exists in the destination system**, and for which you have access privileges. In this case HyperStore will use this existing bucket as the tiering destination.
- The **name of a bucket that you want HyperStore to create in the destination system**, to use as the tiering destination. Be sure to choose a bucket name that is very likely to be unique in the destination system. If your supplied bucket name is not unique in the destination system, HyperStore will be unable to create the bucket and your bucket lifecycle configuration attempt will fail.

If you leave the Bucket Name field **empty**, then in the destination system HyperStore will create a tiering bucket named as follows:

`<origin-bucket-name-truncated-to-34-characters>-<28-character-random-string>`

Note Azure uses the term "Container" rather than bucket, and in the CMC the field is "Container Name" rather than Bucket Name -- but the role of the field is the same as described above.

Retain Local Copy

Select the "Retain Local Copy" checkbox if you want to keep a local copy of auto-tiered objects for a certain number of days after the objects have been auto-tiered. If you select the checkbox, then a field appears in which you can specify the number of days for which to retain the local copy.

For example, if you select this option and set the retention to 30 days, then each object in your bucket that gets auto-tiered to the destination system will also be retained locally for 30 days. After the 30 days the local copy of the object will be automatically deleted.

If you do not select the "Retain Local Copy" checkbox, then the local copies of objects will be deleted immediately after the objects have been successfully tiered to the destination system.

Note Even after local copies of auto-tiered objects have been deleted, the system retains local metadata that enables the system to retrieve a copy of the auto-tiered objects from the destination system upon your request.

Bridge Mode (Proxy) (applicable only to AWS S3, Google, Azure, or Custom S3 destinations)

Select the "Bridge Mode (Proxy)" checkbox only if, starting from when you Save your auto-tiering rule, you want **all objects newly uploaded to the bucket to be immediately transitioned** to the tiering destination system. In this mode HyperStore essentially acts as a proxy service, with uploaded objects only momentarily being held in storage before being automatically moved to the destination system.

If you select the "Bridge" checkbox, then any schedule-based tiering configuration that you set in the object tiering "Schedule" section of the interface will be applied only to objects that are already in the bucket at the time that you Save this auto-tiering rule.

For more information on bridge mode, see "**Bridge Mode**" (page 55).

Note If you set up Bridge Mode for a bucket, and then at a later time you decide you want to turn it off, the only way to do so is to edit the bucket lifecycle policy by deselecting the Bridge Mode checkbox (and then save the edited lifecycle policy). Disabling or deleting the lifecycle policy will not work to turn off Bridge Mode. See "**Editing, Disabling, or Deleting a Lifecycle Rule**" (page 204).

GET Request Handling (applicable only to AWS S3, Google, Azure, or Custom S3 destinations)

If your tiering destination is AWS S3, Google, or Azure, choose how you want the local HyperStore system to handle GET requests for objects that have been transitioned to the tiering destination system.

Options are:

- Stream -- The local HyperStore system GETs the object from the tiering destination system and streams it through to the client. This is the default for tiering to AWS S3, Google, or Azure.
- Require Restore -- If you choose this option, the only way that S3 client applications will be allowed to access tiered objects is to execute a "**POST Object restore**" (page 938) request in order to temporarily restore a copy of the object in local HyperStore storage. For information about how this works in the CMC's **Objects** interface, see "**Restore an Auto-Tiered Object**" (page 225).

Note If the bucket uses **versioning** you should use Stream as the GET handling method. If you use Require Restore you will not be able to retrieve auto-tiered non-current object versions unless you first delete the current object version. For more information see "**Restoring Auto-Tiered Object Versions**" (page 227)

Note If you create multiple auto-tiering lifecycle rules for a single bucket (for different object prefixes), all such rules must use the same GET request handling method.

For objects tiered to destinations other than AWS S3, Google, and Azure, the "Require Restore" method is always used.

Object Expiration

OBJECT EXPIRATION
SCHEDULE
<input type="checkbox"/> CURRENT VERSION <input type="checkbox"/> PREVIOUS VERSION
CLEAN UP EXPIRED OBJECT DELETE MARKERS AND INCOMPLETE MULTIPART UPLOADS
<input type="checkbox"/> CLEAN UP INCOMPLETE MULTIPART UPLOADS <input type="checkbox"/> CLEAN UP EXPIRED OBJECT DELETE MARKERS
LIFECYCLE RULE BUCKET LEVEL SETTING
<input checked="" type="radio"/> Use Creation Date/Time <input type="radio"/> Use Last Access Time

Current Version

Select this checkbox to set an expiration (deletion) schedule for the current version of objects in the bucket. If your bucket does not use [versioning](#), then current versions are the only versions of objects in your bucket and this is the appropriate type of schedule to set. If your bucket does use versioning, select this checkbox to set an expiration schedule specifically for the **current version** of objects in the bucket (you can set a different schedule for non-current versions of objects as described in "Previous Version" below).

- If you want deletion of the objects to occur on a particular date, choose the "After Date" radio button and enter the desired deletion date.
- If you want expiration of the objects to occur a certain number of days after the objects were last accessed (retrieved or modified) select the "Days After Last Access Date/Time" radio button and enter the number of days.
- If you want expiration of the objects to occur a certain number of days after the objects were created, at the bottom of the page choose the "Use Creation Date/Time" radio button, then back up in the scheduling schedule choose the "Days After Creation Date" radio button and enter the number of days.

Note If you are configuring a lifecycle rule that includes both auto-tiering and expiration, you cannot have one action use Last Access Date/Time as the basis for scheduling while the other uses Creation Date/Time. This is true too if you plan to configure multiple lifecycle rules for a bucket (applying to different object prefixes): all lifecycle rules for a given bucket must use the same type of time reference point for objects -- either Last Access Date/Time or Creation Date/Time.

Previous Version

This option is applicable only if the bucket uses [versioning](#). Select this checkbox to set an expiration (deletion) schedule for **non-current versions** of objects (object versions that have been superseded by a newer version). Expiration scheduling for non-current versions of objects is always based on the number of days since the objects became non-current (the number of days since being superseded by a newer version of the object). Enter the desired number of days.

For example, if you enter 365 here, then if version1 of an object is made non-current by the uploading of version2 of that object on July 13, 2018, then version1 of the object will be deleted a year later (regardless of whether any additional versions of the object were uploaded during the intervening year).

Clean Up Incomplete Multipart Uploads

When the CMC or other S3 client applications upload large objects into the HyperStore S3 storage system, they use a method called multipart upload during which the large object is divided into multiple parts and each part uploaded separately. In some cases a system problem may result in some of the parts being successfully uploaded while others are not. In such cases an incomplete or partial object consumes some storage space in your bucket but the partial object is not readable and will not appear in your bucket contents list.

Select the "Clean Up Incomplete Multipart Uploads" option to have the system delete such incomplete objects a certain number of days after the multipart upload was initiated. You can specify the number of days.

Clean Up Expired Object Delete Markers

This option is applicable only if the bucket uses [versioning](#) and only if you haven't set a Current Version expiration schedule. If in a versioning-enabled bucket you delete the current version of an object, a "delete marker" takes the place of that object version while all of the object's older versions are retained in the system (and are retrievable). If all the older versions are subsequently expired (by execution of your expiration rule for Previous Versions), that orphaned delete marker remains. Select the "Clean Up Expired Object Delete Markers" option to have the system automatically delete a "delete marker" if all older versions of the object have been expired.

6. Click **Save** to save the rule.

To confirm that your new rule has been saved in the system, select the **Lifecycle Policy** tab of the CMC's bucket properties interface. The new rule will display in a list along with any lifecycle rules you may have previously configured for the bucket.

Note You can create multiple lifecycle rules for a bucket (using the **Add New Rule** function each time), with each rule applying to a different object prefix. You cannot, however, have more than one lifecycle rule for the same object prefix.

IMPORTANT: Once objects have been auto-tiered to the destination system, **do not overwrite or delete tiered objects directly through the destination system's interfaces** (such as the AWS Console in the case of tiering to AWS). Doing so will cause a discrepancy between the local metadata in HyperStore and the actual data in the destination bucket. If you want to overwrite or delete tiered objects, do so through HyperStore interfaces (such as the CMC or an S3 application accessing the HyperStore S3 Service). In the case of auto-tiering from one HyperStore region to another HyperStore region, any overwriting or deleting of objects should be done through the source bucket not the destination bucket.

Editing, Disabling, or Deleting a Lifecycle Rule

To edit, disable, or delete an existing bucket lifecycle rule, select the **Lifecycle Policy** tab of the CMC's bucket properties interface. This displays the current list of lifecycle rules for the bucket.

BUCKETS		OBJECTS						
				+ ADD NEW BUCKET				
NAME	REGION	POLICY				Properties		Delete
bucket1	region1	Replication-3X						
BUCKET PERMISSIONS		BUCKET CANNED ACL		STORAGE POLICY	LIFECYCLE POLICY	STATIC WEBSITE HOSTING	CROSS REGION REPLICATION	VERSIONING
ENABLE		RULE NAME				ACTION		
<input checked="" type="checkbox"/>		Delete Logs After 90 Days				<input type="button" value="edit"/>	<input type="button" value="delete"/>	
<input checked="" type="checkbox"/>		Auto-Tier Documents After 1 Year				<input type="button" value="edit"/>	<input type="button" value="delete"/>	
+ ADD NEW RULE								
<input type="button" value="CANCEL"/> <input type="button" value="SAVE"/>								

Here you can click the appropriate button or text to add a new rule for the bucket, or to edit, disable, or delete an existing rule for the bucket.

Note that adding, editing, disabling, or deleting a lifecycle rule changes the behavior of the bucket **from that point forward** — it doesn't change the past behavior. So for example, if you disable an auto-tiering rule that had been in place for your bucket, any objects that have already been transitioned to a tiering destination system will remain in that system.

Also, there are these restrictions:

- You cannot change the tiering destination for an existing rule. While other attributes of an existing rule are editable (such as the schedule), the **tiering destination is not editable**.
- If you add a new lifecycle rule for a bucket, it must apply to a different object prefix than any existing rules. You **cannot have multiple rules for the same object prefix**.
- If you add a new tiering rule for a bucket, the new rule must use the same tiering destination as any existing tiering rules for the bucket. You **cannot have multiple tiering destinations for the same bucket**.
- Once a "Bridge Mode" tiering rule is implemented for a bucket, the only way to later turn it off is to **edit** the rule to deselect the Bridge Mode checkbox. Disabling or deleting the rule will not work to turn off Bridge Mode.

5.3.2.5. Configure a Bucket as a Static Website

You can configure a bucket as a website in order to make its content available to public users using regular web browsers (as opposed to S3 client applications). Only **static** websites are supported. A static website is a website that serves the same content to all visitors -- there is no content personalization based on cookies or other mechanisms. A static site does not use server-side scripting.

IMPORTANT: For the static website feature to work, on your DNS server you must create entries for these URLs:

```
s3-website-<region>.<your-domain>
*.s3-website-<region>.<your-domain>
```

For example:

```
s3-website-tokyo.enterprise.com IN A 10.1.1.1
*.s3-website-tokyo.enterprise.com IN A 10.1.1.1
```

For more information see "DNS Set-Up" in the *HyperStore Installation Guide*.

Note Public access to buckets configured as static websites is through HTTP not HTTPS. **The static website feature does not support HTTPS (SSL/TLS).**

To configure a bucket as a static website, select the **Static Website Hosting** tab of the CMC's [Bucket Properties](#) interface.

The screenshot shows the CMC's Bucket Properties interface. At the top, there are tabs for 'BUCKETS' (which is selected and highlighted with a red circle) and 'OBJECTS'. Below the tabs is a button '+ ADD NEW BUCKET'. The main area displays a table with columns: NAME, REGION, and POLICY. A single row is shown for 'bucket1' in 'region1' with 'Replication-3X' as the policy. To the right of this row are 'Properties' and 'Delete' buttons, with 'Properties' also highlighted with a red circle. Below the table is a horizontal bar with several tabs: BUCKET PERMISSIONS, BUCKET CANNED ACL, STORAGE POLICY, LIFECYCLE POLICY, STATIC WEBSITE HOSTING (which is highlighted with a red circle), CROSS REGION REPLICATION, VERSIONING, and LOGGING. Under the 'STATIC WEBSITE HOSTING' tab, there is a checkbox labeled 'ENABLE WEBSITE HOSTING' which is checked. At the bottom right are 'CANCEL' and 'SAVE' buttons, with 'SAVE' being the active button.

1. Select the "Enable Website Hosting" checkbox.
2. By default, when you enable website hosting on a bucket, public read access is automatically granted for all objects in the bucket. A message will pop up to warn you of this behavior -- click **OK** to close the message. If you do not want this default configuration, deselect the "Grant automatic READ access to all objects" checkbox that appears in the **Static Website Hosting** interface. You can do this if you prefer to:
 - Grant public read permissions to individual objects in the bucket, one-by-one. You can do this through the CMC's "**Set Object Properties**" (page 219) function.

OR

 - Use the S3 API's "PUT Bucket Policy" method to grant public read access to some groups of objects but not others -- such as objects that start with a certain prefix (directory path). For this you would need to use a different S3 client application to do so, since the CMC does not currently support this capability.

The more typical static website set-up would be to accept the default behavior and grant automatic read access to all objects in the bucket.

3. Complete the "Index Document" field with the name of the file that should be served when site visitors access the root URL of your site. This file is typically named "index.html".
4. Complete the "Error Document" field with the name of the file that should be served when site visitors trigger an HTTP 4xx error (such as by requesting an object that does not exist). You might name this file "404.html", for instance.
5. Click **Save**.

Note If you have not yet done so, you must upload both the index document file and the error document file into the **root level** of your bucket, and make sure their names correspond exactly to those you specified in the **Static Website Hosting** interface.

For a bucket configured as a static website, the root URL will be:

```
http://<bucketName>.<regionS3WebsiteEndpoint>
```

Each Cloudian HyperStore service region has its own website endpoint, set by system configuration. To view the website endpoint(s) for your system, go to the CMC's [Cluster Information](#) page.

IMPORTANT: Make sure that in your service center's DNS configuration this website endpoint domain is resolvable, and also that a wildcard entry has been set to resolve bucketnames under this domain. For example, if the website endpoint for the region is *s3-website-region1.cloudian.com*, then both *s3-website-region1.cloudian.com* and **.s3-website-region1.cloudian.com* must be resolvable. For more on DNS set-up see [DNS Set-Up](#).

5.3.2.6. Configure Cross-Region Replication for a Bucket

Introduction

You can configure a bucket so that any newly uploaded objects (objects uploaded **after** you configure this feature) are automatically replicated to your chosen destination bucket in a different service region. This feature enables you to enhance the protection of your data by having it stored in two geographically dispersed service regions. The feature is also useful in cases where you want to have the same set of data stored in two different regions in order to minimize read latency for users in those regions.

Along with allowing you to replicate from a source bucket to a destination bucket in a different service region in the same HyperStore system, you can also choose as your destination a bucket in an external S3 system such as Amazon S3.

HyperStore also allows you to replicate to a destination bucket that's in the same HyperStore service region as the source bucket, if you want to. Note however that replicating from a source bucket to another bucket in the same region will not provide the geographical dispersion of data copies that replicating across regions does.

Object metadata — including any access permissions assigned to an object — is replicated to the destination bucket as well as the object data itself.

The cross-region replication feature does **not** replicate:

- Objects that were already in the source bucket before you configured the bucket for cross-region replication.
- Objects that are encrypted with user-managed encryption keys.
- Object version deletions (deletions of specific object versions.)
- Objects that are themselves replicas from other source buckets.

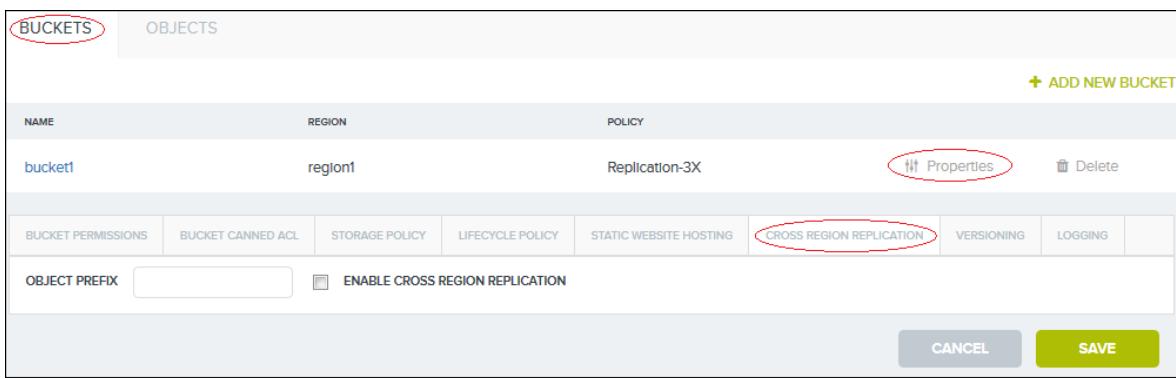
Note As is the case with Amazon S3's implementation of this feature, with HyperStore both the source bucket and the destination bucket must have **versioning enabled** in order to activate cross-region replication.

Note For additional detail on the cross-region replication feature, see "**Cross-Region Replication**" (page 132).

To Configure a Source Bucket for Cross-Region Replication

Note Before configuring cross-region replication (CRR), be sure that both the source bucket and the destination bucket have versioning enabled. This is required in order to use cross-region replication. For information about enabling versioning on a bucket in HyperStore, see "**Set Versioning for a Bucket**" (page 209).

To configure a source bucket for replication, from the CMC's [Bucket Properties](#) interface for the bucket select the **Cross Region Replication** tab.



Then do the following:

1. Select the "Enable Cross Region Replication" checkbox.
2. If you want all new objects (objects uploaded after you configure this feature) in the source bucket to be replicated, leave the "Prefix" field empty. If you only want to replicate objects whose full names (including path) start with a particular prefix, enter that prefix in the "Prefix" field. You do not need to include a leading forward slash at the start of the prefix. "Taxes/documents" is a valid example, or "profile/images/headshots".

If you want to replicate objects associated with multiple prefixes, enter a comma-separated list of prefixes, with no spaces between prefixes. For example "legal/docs,legal/audio,compliance/docs".

Note that if you are specifying multiple prefixes, all the replication must go to the same one destination bucket (you can't replicate from a single source bucket to multiple destination buckets). Note also that the source prefixes will be included in the full name of the object replicas in the destination bucket (for example, source object "legal/docs/briefing_04-17-2016" will be replicated as "legal/docs/briefing_04-17-2016" in the destination bucket).

3. Specify the Destination Bucket name.
 - If the destination bucket is in **HyperStore**, it must be a bucket that you own.
 - If the destination bucket is in an **external S3 storage system** such as Amazon S3, "versioning" must be enabled on the bucket and you must have write permissions on the bucket.
4. If the destination bucket is in an external S3 storage system such as Amazon S3 (not in the HyperStore system), enter the external system's S3 endpoint in the Destination Endpoint field, in format

`<protocol>://<endpoint>:<port>`. For example <https://s3.amazonaws.com>:443. If the destination bucket is in the HyperStore system, leave this field empty.

5. If you completed the Destination Endpoint field, then two additional fields will display in which you must enter your Access Key and Secret Key (security credentials) for the external S3 system. If the destination bucket is in the HyperStore system, leave these fields empty.
6. Click **Save**.

Note After cross-region replication is enabled on a source bucket and has been active, the system supports the option of editing the source bucket's configuration so that a different destination bucket is used. After that change, new data uploaded to the source bucket would replicate to the new destination bucket rather than to the original destination bucket. In other words, you are not permanently committed to the same destination bucket. However, after such a change, replicas already in the original destination bucket would remain there — they would not migrate to the new destination bucket.

To Disable Cross-Region Replication for a Source Bucket

To disable cross-region replication for a source bucket, from the CMC's [Bucket Properties](#) interface for the bucket select the **Cross Region Replication** tab. Then de-select the "Enable Cross Region Replication" checkbox and click **Save**.

The system will no longer replicate newly uploaded objects from the source bucket to the destination bucket. However, any replicated objects already in the destination bucket -- from during the time period that you had cross-region replication enabled on the source bucket -- will remain there unless you delete them.

5.3.2.7. Set Versioning for a Bucket

In the CMC's [Bucket Properties](#) interface, you can use the **Versioning** tab to enable object versioning for a bucket, or to suspend versioning if you enabled it previously. By default versioning is not enabled.

BUCKETS		OBJECTS					
			+ ADD NEW BUCKET				
NAME	REGION	POLICY					
bucket1	region1	Replication-3X	Properties	Delete			
BUCKET PERMISSIONS	BUCKET CANNED ACL	STORAGE POLICY	LIFECYCLE POLICY	STATIC WEBSITE HOSTING	CROSS REGION REPLICATION	VERSIONING	LOGGING
Versioning is not enabled for this bucket.							
ENABLE							

When versioning is enabled on a bucket, each version of objects in the bucket is retained -- not just the current version but prior versions as well. For example, if you upload a newly created document called *GreatIdeas.docx*; and then later you revise the document and upload it again; and then later you revise it again and upload it again -- the system will store all three versions of the document (the initial version, the next revised version, and the current version).

Versioning protects you against losing an object due to accidentally overwriting it -- since the previous version is retained as well as the (accidental) new version. Also, versioning allows you access to all the past versions

of an object throughout its history. For information about how this access appears in the CMC interface, see "[Download an Object](#)" (page 225).

By contrast, when versioning is not enabled, only the current (most recently uploaded) version of the object is retained in the system.

IMPORTANT: If you enable versioning on your bucket, each version of an object will count toward your storage space consumption for purposes of usage reporting and usage-based billing.

Note You cannot enable versioning on a bucket that is configured for auto-tiering to Amazon Glacier or Spectra BlackPearl.

To enable versioning on a bucket, with the bucket properties **Versioning** tab selected, click the **Enable** button.

For information about downloading or deleting versions of an object see "[Download an Object](#)" (page 225) and "[Delete an Object](#)" (page 228).

Lifecycle Management for Versioned Buckets

HyperStore supports configuring lifecycle rules regarding current and non-current versions of objects in buckets for which versioning is enabled. For example you could have non-current object versions auto-tiered to a different storage system -- or automatically deleted -- a certain number of days after they become non-current. For more information see "[Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration](#)" (page 197).

Suspending Versioning on a Bucket

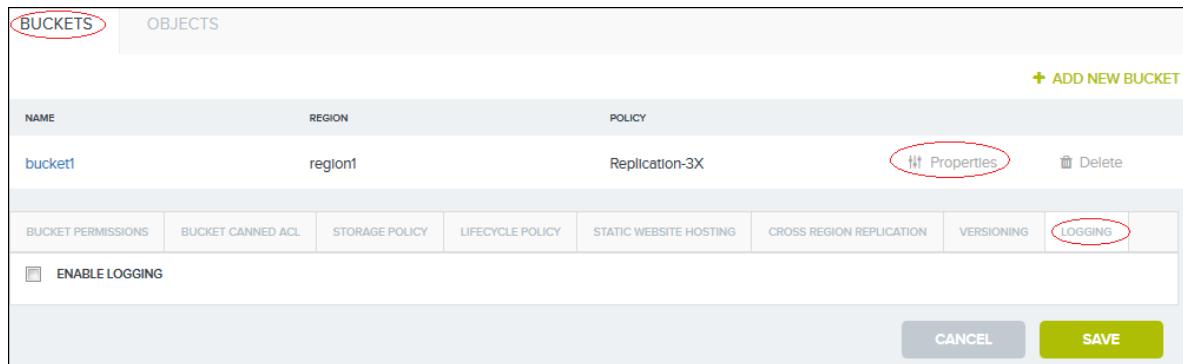
If versioning is currently enabled on a bucket, you have the option of suspending it by selecting the **Versioning** tab in the CMC's bucket properties interface and then clicking **Suspend**. The behavior in a bucket for which versioning had been enabled and is then subsequently suspended is as follows:

- For **new objects uploaded for the first time after versioning is suspended**, no versioning is applied. For example, if after suspending versioning you upload a new document *TerribleIdeas.docx*, and then later you revise that doc and upload the revised version, only the most current version will be retained.
- For **existing objects that had been uploaded during the period when versioning was enabled**:
 - All object versions that were already in the system are retained. In other words, when you suspend versioning this does not purge any existing object versions. (If you want to purge old object versions you can [delete them](#).)
 - Going forward, if you upload revised versions of objects that had been first uploaded during the period when versioning was enabled, what will be retained is:
 - All object versions from when versioning was enabled.
 - The most current of the object versions uploaded after versioning was suspended.

For example, if while versioning was enabled you upload versions 1, 2, and 3 of *GreatIdeas.docx*, and then after versioning is suspended you upload versions 4, 5, and 6 of the document, the system will retain versions 1, 2, 3, and 6.

5.3.2.8. Set Logging for a Bucket

In the CMC's [Bucket Properties](#) interface, you can use the **Logging** tab to enable access logging for a bucket, or to disable access logging if you enabled it previously. By default access logging is not enabled.



When logging is enabled on a source bucket, every 10 minutes the HyperStore system generates an access log file for the bucket (if there has been access to the bucket during the past 10 minutes) and stores the log file in a specified destination bucket. The destination bucket can be the source bucket itself -- for instance you can have the system generate logs recording activity for *bucket1*, and store the log files in *bucket1* -- or the destination bucket can be a different bucket than the source bucket. You must be the owner of the destination bucket.

The content of the bucket log files is compliant with Amazon S3's bucket logging feature. For detail about the bucket log content, see the Amazon documentation topic [Server Access Log Format](#).

Before you enable logging for a source bucket, you must establish the necessary access rights on the destination bucket so that the HyperStore system can write log files into it. You can do this through the CMC's **Bucket** page, by using the Properties interface for the destination bucket. You have two options for establishing the needed access rights on the destination bucket:

- Using the Bucket Canned ACL tab, set the canned ACL "Log Delivery Write" on the destination bucket.
OR
- Using the Bucket Permissions tab, give the "Log Delivery" grantee permissions for "Writable" and "ACP Readable" on the destination bucket.

To enable logging for a source bucket, use the Properties interface for the source bucket, and go to the Logging tab. Then:

1. Select the "Enable Logging" checkbox. When you do this, entry fields will display.
2. Enter the name of the "Destination Bucket". As discussed above, this can be the source bucket itself or some other bucket that you own, and it must have the needed permissions already set on it.
3. In the "Target Prefix" field, optionally enter a text string that will be used as a prefix for all the bucket log file names for this source bucket. Absent a prefix, the log files will simply be named by a timestamp indicating the time of the log file generation. Using a prefix is recommended as it will make it easier for you to view and manage the log files (including possibly configuring lifecycle management for the log files). For example, if you enter "LogBucket1_" as the prefix, then all log files will be named as "LogBucket1_<timestamp>".

Note: If you wish you can enable bucket logging on multiple source buckets (working through the Properties interface for each source bucket one at a time), and have all the logs written to the same destination bucket. In this case you could use prefix values to distinguish the logs from the different source buckets. For example, when enabling logging on *bucket1*, use "LogBucket1_" as the prefix, and when enabling logging on *bucket2*, use "LogBucket2_" as the prefix.

4. Click **Save**.

Once you have set up bucket logging, you may wish to use the CMC's [Bucket Lifecycle](#) feature to configure how the logs are handled as they age. For example you can configure it so that the log files are automatically deleted once they reach a certain age. To set up a bucket lifecycle rule for the logs, you must have used a Target Prefix when you set up bucket logging (Step 3 above). You will need that prefix when you set up the bucket lifecycle rule.

Editing or Disabling Bucket Logging

Once access logging is enabled for a source bucket, you can subsequently return to the bucket properties Logging tab for that bucket if you want to:

- Edit the bucket logging configuration (by changing the destination bucket or the log file name prefix)
- Disable logging for the bucket (by deselecting the "Enable Logging" checkbox)

Don't forget to click **Save**.

Note If you make any changes to the bucket logging configuration, this will affect how bucket logging is performed going forward. It will not retroactively impact log files that were previously generated.

5.3.2.9. Set a Lock Policy for a Bucket (WORM)

In the CMC's [Bucket Properties](#) interface, a **WORM** tab will display if and only if:

- The system license supports the Write Once Read Many (WORM) feature.
- Versioning has been enabled on the bucket (see "[Set Versioning for a Bucket](#)" (page 209))

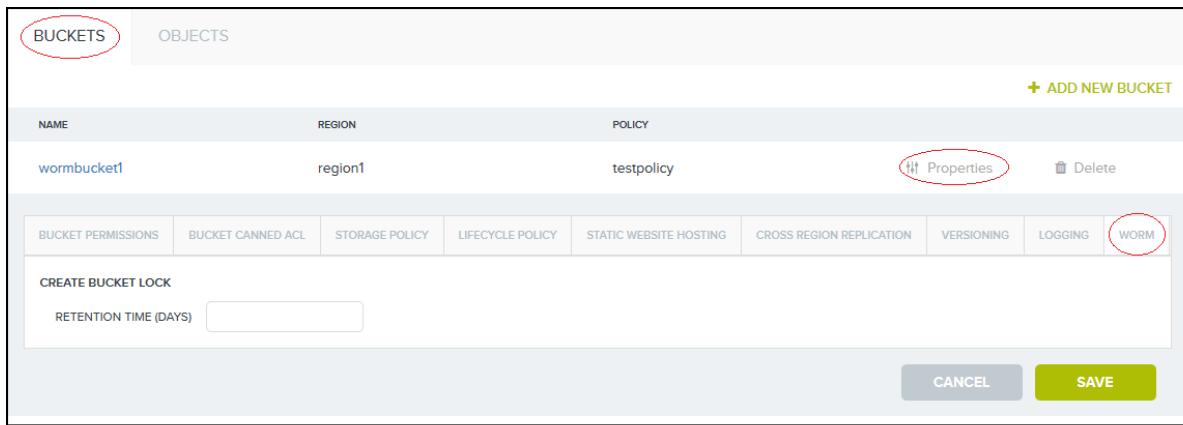
For general information about HyperStore support for WORM functionality, see "[WORM \(Bucket Lock\) Feature Overview](#)" (page 159).

You can apply a bucket lock (WORM) policy on any bucket that has versioning enabled. With versioning and a bucket lock on a bucket:

- Client requests to update an object will result in both the old version(s) and the new version of the object being retained. This is a consequence of versioning being enabled on the bucket.
- Object versions cannot be deleted until they are older than the retention period that you specify in the bucket's lock policy. This is a consequence of the bucket being locked.

IMPORTANT: Once you complete the process of setting a lock policy on a bucket, you **cannot remove or modify the lock policy**. Also, you cannot disable versioning on a locked bucket.

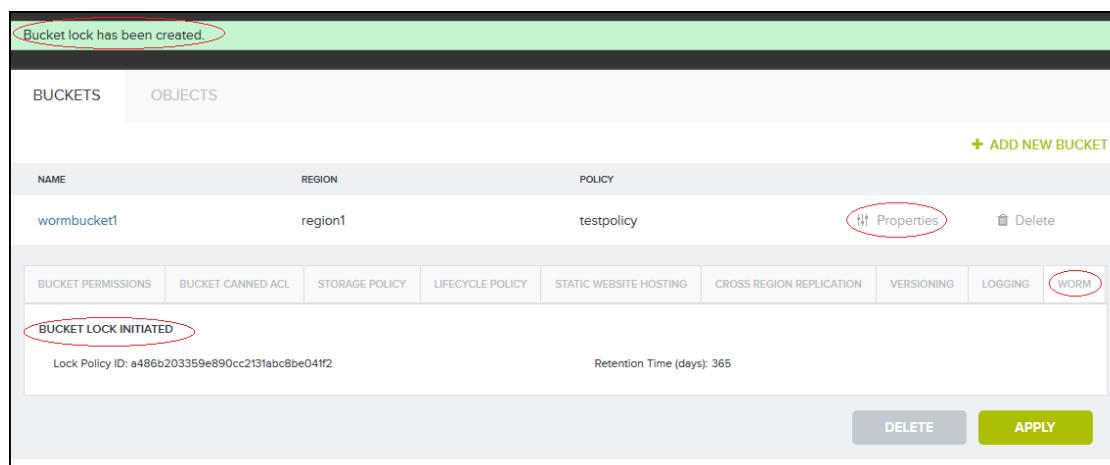
To set a bucket lock policy on a bucket, in the [Bucket Properties](#) interface select the **WORM** tab.



In the **WORM** tab there are **two stages** to applying a bucket lock policy to a bucket:

- Initiate** the bucket lock by entering a retention period (as a number of days) and then clicking **Save**.
The retention period will be applied to each object version separately, starting from the object version's creation date. For example with a retention period of 365 days, an object version created on March 17, 2019 will be not be allowed to be deleted until after March 17, 2020; and an object version created on April 5, 2019 will be not be allowed to be deleted until after April 5, 2020.

Once you click **Save** the interface will indicate that a bucket lock has been initiated.



This begins a 24 hour period during which:

- Objects in the bucket cannot be deleted if they are not yet older than the retention period that you specified in the bucket lock. You can test this if you wish.
- You have the option of removing the bucket lock, by clicking **Delete** in the **WORM** tab.

To complete (finalize and make permanent) the bucket lock, proceed to Step 2 below **within 24 hours of initiating the lock**.

- Complete** the bucket lock by clicking **Apply** in the **WORM** tab. If you do not do this within 24 hours of initiating the bucket lock, the bucket lock will automatically expire and be removed from the bucket (and therefore objects will again be deletable as usual).

Once you complete the bucket lock it can never be modified or removed from the bucket.

After you complete the bucket lock by clicking **Apply**, the bucket properties **WORM** tab indicates that the bucket is locked. Also, in the bucket list view a padlock icon appears beside the bucket name.

The screenshot shows the CMC Buckets page. A single bucket, 'wormbucket', is listed. The 'Properties' and 'WORM' tabs in the navigation bar are circled in red. A message at the bottom of the page reads: 'BUCKET LOCK COMPLETED' followed by 'Bucket objects are protected from object create date plus a retention period of 365 days.'

From now on, if you try through the CMC's **Objects** page to delete an object before its retention period has expired the interface will display an error and the object will not be deleted.

Unable to process request. BucketLockPolicyError: The bucket's lock-policy is preventing this action.

Likewise, you will not be able to remove objects from the bucket using a third party S3 client application, until the objects have reached the end of their retention period.

Note You cannot delete a bucket when there are still objects in it. So a locked bucket will not be deletable unless all the objects within it have reached the end of their retention period and have been deleted.

5.3.3. Delete a Bucket

To delete a bucket, first go to the [Objects](#) view and delete all the objects in the bucket. The system will not let you delete a bucket that has objects in it.

After deleting all objects in the bucket, you can delete the bucket by going to the [Buckets](#) list view and clicking the **Delete** button on the far right of the bucket's display row.

5.4. Objects

Path: [Buckets & Objects](#) → [Objects](#)

The screenshot shows the CMC Objects page. A bucket named 'bucket2' is selected. The 'OBJECTS' tab in the navigation bar is circled in red. At the bottom right, there are 'RESTORE' and 'DELETE' buttons.

Note The **Objects** tab does not display until you've created a bucket. In an S3-based storage system, each data entity that you store in a bucket is known as an "object". Each object has its own unique URI — derived from the object name — that S3 applications use when retrieving the object. In most respects you can think of an object as being a file.

Supported tasks:

- "[Create or Delete a "Folder"](#)" (page 215)
- "[Upload an Object](#)" (page 217)
- "[Set Object Properties](#)" (page 219)
- "[Search for an Object](#)" (page 224)
- "[Download an Object](#)" (page 225)
- "[Restore an Auto-Tiered Object](#)" (page 225)
- "[Delete an Object](#)" (page 228)

5.4.1. Create or Delete a "Folder"

In an S3-based storage system, each data entity stored in a bucket is known as an "object". Each object has its own unique URI — derived from the object name — that S3 applications use when retrieving the object. An S3 storage system does not have a hierarchical structure like a conventional file system. However, objects can be named in such a way as to imply a hierarchy. For example, in a bucket there could be four objects named:

- *Documents/resume.docx*
- *Documents/schedule.docx*
- *Images/birthday-party.png*
- *Images/sunset.png*

The CMC makes it easy to create such hierarchies by enabling you to create "folders". To arrive at the example above, through the CMC interface you could create a folder named "Documents", and then drill down into that folder and upload files named "resume.docx" and "schedule.docx". Behind the scenes, in the S3 storage layer, objects named "Documents/resume.docx" and "Documents/schedule.docx" are created. In the CMC interface, you can then retrieve a list of the contents of the "Documents" folder: the display will list the "resume.docx" and "schedule.docx" files.

Thus while the back-end is implemented differently, the CMC user experience is just like interacting with a regular file system.

To create a folder:

In the [Buckets](#) list view, click the name of the bucket that you want to work with. This will display the CMC's [Objects](#) view for that bucket. (Alternatively you can click the **Objects** tab and then choose a bucket from drop-down list.)

At the bottom of the **Objects** interface, enter the folder name and then click **Create Folder**. The new folder will then display in the object list view.

Note For folder naming restrictions, see "[Object Naming Restrictions](#)" (page 216). These restrictions apply to unusual naming schemes and are of no concern if you are using a simple alphanumeric folder name.

Note The CMC does not support renaming existing folders.

To delete a folder:

To delete a folder click **Delete** at the right side of the folder's display row. You will be asked to confirm that you want to delete the folder.

IMPORTANT: Deleting a folder will also delete all files and sub-folders within the folder.

5.4.2. Object Naming Restrictions

The following control characters cannot be used anywhere in an object name (a folder name or file name) and will result in a 400 Bad Request response: 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A ("\\n"), 0x0B, 0x0C, 0x0D ("\\r"), 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Also unsupported are:

- 0x09 ("\\t") at the beginning of an object name
- 0xBF (inverted question mark) at the end of an object name

Also, an object name may result in 400 Bad Request or be stored as a different name if the supplied object name:

- Consists only of ".."
- Contains a combination of "." and "/", or ".." and "/"

Examples of character sequences that will result in 400 Bad Request:

- ..
- ..
- ./
- ../
- ./.
- ./..
- ...
- ./a
- ./a/
- a./.../b

Examples of character sequences that will be stored as a different name:

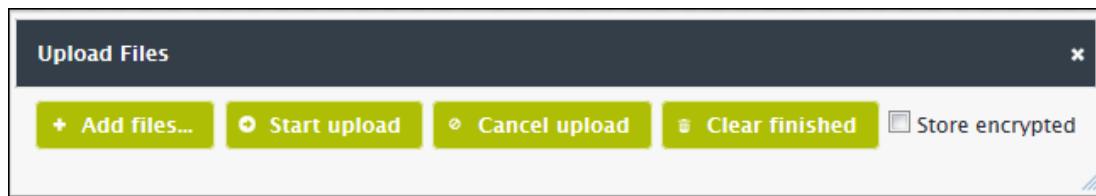
Supplied Object Name	Stored As
.a	a
.a/	a/

Supplied Object Name	Stored As
a/b	a/b
a./b	a/b
a../b	b
a ../../b	a ../../b

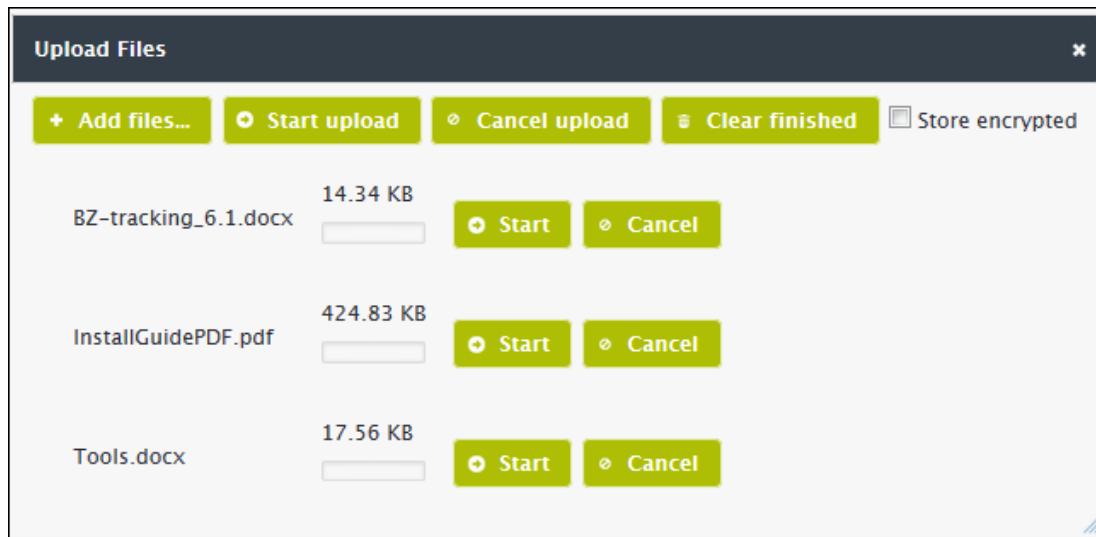
5.4.3. Upload an Object

Note If you haven't yet done so, you must "[Add a Bucket](#)" (page 192) before you can upload any objects.

1. In the CMC's [Objects](#) view click **Upload File**. The **Upload Files** interface displays.



2. Click **Add Files**, then use the browse window that opens to browse to the file(s) on your computer. You can select multiple files from within the same directory on your computer by using <Ctrl>-click or <Shift>-click. After you've selected the files and clicked **Open** in the browse window, the list of selected files displays in the **Upload Files** interface.



Note By default the maximum sized object that you can upload through the CMC interface is 5GB.

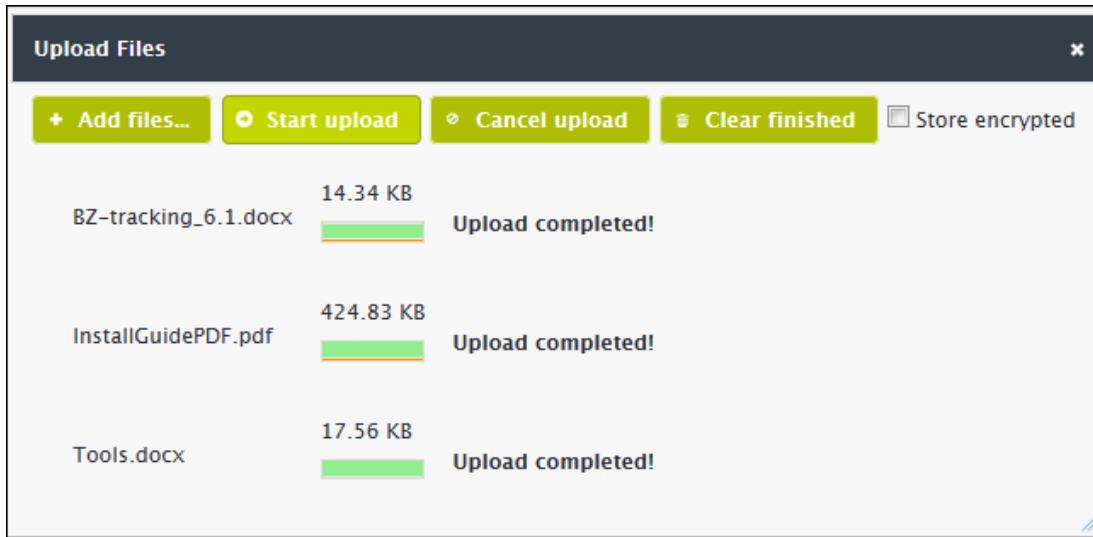
The maximum allowed S3 object name length — that is, the maximum length of the folder path plus file name, combined — is 1024 characters. For additional naming restrictions see "[Object](#)

Naming Restrictions (page 216). These additional restrictions pertain to unusual characters or character combinations and so are of no concern if you are using simple alphanumeric file names.

- If you want the Cloudian HyperStore system to encrypt the file(s) in storage, click the "Store encrypted" checkbox. The file(s) will be encrypted using a system-generated encryption key. If an encrypted file is subsequently downloaded, the system will automatically decrypt the file before transmitting it to the requesting client application.

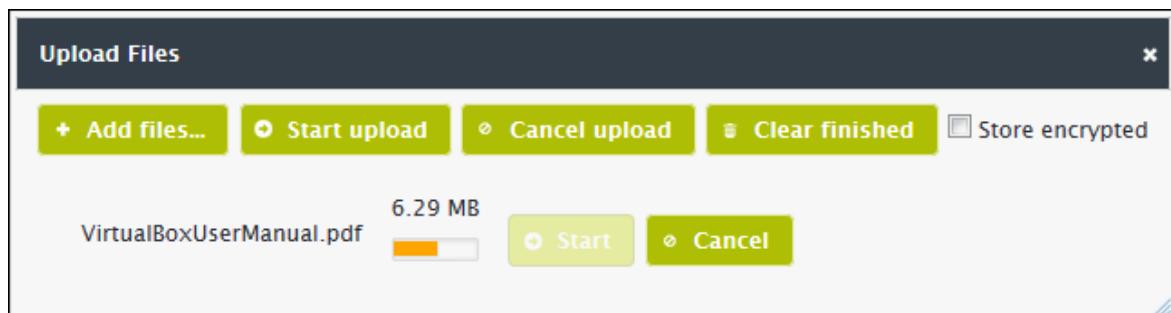
Note If you are uploading objects into a bucket that's using a storage policy that requires object encryption, then the objects will be automatically encrypted by the system regardless of whether or not you select the "Store encrypted" checkbox.

- To upload all the files click **Start Upload** at the top of the **Upload Files** interface. (Or if you want to upload files one-by-one, click **Start** to the right of individual file names). When all files have uploaded, a success message appears.



If you want to continue uploading more files, click **Clear Finished** to clear the **Upload Files** interface, then click **Add Files** to add more files to upload. If you're done uploading files, click the "X" in the upper right of the **Upload Files** interface to close it.

If you upload a large object you can monitor its upload progress by watching the progress bar beside the file name.



Objects that you have successfully uploaded appear in the **Objects** interface. If you chose to encrypt a file, a padlock icon displays to the left of the file name.

Note For a large object over a certain size threshold (16MB by default), the CMC will upload the object to the S3 Service in multiple separate parts, using multiple HTTP transactions. This is known as multipart upload. In such cases, a **Multipart Upload in Progress** section displays at the bottom of the **Objects** interface. This display indicates the time at which the multipart upload was initiated and the object name (the "Key"). If the object appears to be taking too long to upload, this may indicate that a problem has occurred while uploading the multiple parts. In this case you can use the **Abort** button to abort the incomplete multipart upload operation. By aborting the operation, you remove from storage any object parts that had been uploaded before problems occurred with the operation. You can subsequently try again to upload the object, starting over from Step 1 of the procedure above.

By default the CMC's **Objects** page will display upload progress for a maximum of 1000 multipart upload objects simultaneously. This is configurable by the "**list.multipart.upload.max**" (page 525) property in [mts-ui.properties](#).

5.4.4. Set Object Properties

To set properties for an individual object (file), in the CMC's **Objects** interface click **Properties** in the object's display row. This opens the properties interface for that object.

The screenshot shows the Cloudian CMC Objects interface. At the top, there are tabs for 'BUCKETS' and 'OBJECTS', with 'OBJECTS' being the active tab. A search bar shows 'Bucket name: bucket1'. Below the search bar, a list of objects in 'region1: bucket1' is displayed, including 'HyperStoreAdminGuide_v-7.0.pdf' (7.6 MB, last modified Mar-10-2018 09:48 AM -0800) and 'HyperStoreInstallGuide_v-7.0.pdf' (515.2 KB, last modified Mar-10-2018 09:48 AM -0800). For the selected file 'HyperStoreAdminGuide_v-7.0.pdf', a 'Properties' link is highlighted with a red circle. The 'Properties' dialog box is open, showing the file's details: 'NAME: HyperStoreAdminGuide_v-7.0.pdf', 'SIZE: 7.6 MB', and 'LAST MODIFIED: Mar-10-2018 09:48 AM -0800'. Below these details, there are tabs for 'GENERAL PERMISSIONS', 'OBJECT CANNED ACL', and 'PUBLIC URL ACCESS'. Under 'GENERAL PERMISSIONS', there are sections for 'GRANTEE', 'READABLE', 'ACP READABLE', and 'ACP WRITABLE'. Under 'GRANTEE', 'Public' and 'Authenticated Users' are listed. There is also a '+ ADD NEW' button. At the bottom of the dialog box are 'CANCEL' and 'SAVE' buttons. At the very bottom of the interface, there are 'RESTORE' and 'DELETE' buttons.

Supported tasks:

- **"Set Custom S3 Permissions on an Object" (page 220)**
- **"Set "Canned" S3 Permissions on an Object" (page 221)**
- **"Set Public URL Permissions on an Object" (page 223)**

Note The CMC supports only setting permissions on individual objects, one-by-one. If you want to make **all** objects within a bucket -- or a group of objects such as those with a common prefix (directory path) -- readable to the public or to specified individuals or groups, the best way to accomplish this is through creating a bucket policy. The S3 API supports this (see "**PUT Bucket policy**" (page 923)), and some S3 client applications may support this, but the CMC currently does not.

5.4.4.1. Set Custom S3 Permissions on an Object

To choose who you want to be able to access an object using S3 protocol applications, and which specific permissions you want to give them, in the CMC's "**Set Object Properties**" (page 219) interface for the object select the **General Permissions** tab.

GRANTEE	READABLE	ACP READABLE	ACP WRITABLE
Public	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

[+ ADD NEW](#)

CANCEL **SAVE**

1. Select the S3 permissions to grant to the **public at large** (the "Public" row). Supported permission types are:

Readable

Grantee can open or download the object.

Note Write permissions for uploading, overwriting, and deleting objects are set at the bucket properties level, not the object properties level. See "**Set Bucket Properties**" (page 194).

ACP Readable

Grantee can view the current permission settings for the object.

ACP Writable

Grantee can change the permission settings for the object.

2. Select the S3 permissions to grant to **all registered users of the Cloudian HyperStore service** (the "Authenticated Users" row). The supported permission types are the same as described in Step 1.
3. If you want to grant S3 permissions to **specific registered users of the Cloudian HyperStore service**:

- a. Click **Add New**. A new grantee row will display.
- b. Specify the grantee by <groupID>|<userID>, with the vertical bar separator between groupID and userID. For example, "sales|kurihara123". (Do not include the quote marks.)

Note The CMC does not validate the user IDs that you enter. If you enter a user ID that does not belong to an active Cloudian HyperStore user, your action will have no effect.

If you want to grant permissions to **all members of a particular Cloudian HyperStore user group**, specify the grantee as <groupID>|. For example "sales|". (Do not include the quote marks.)

- c. Select the S3 permissions to give the grantee. The supported permission types are the same as described in Step 1.
4. Click **Save**.

If you add a specific Cloudian HyperStore user as a grantee, and then later want to remove that grantee, click **Delete** at the right side of the grantee's display row, then click **Save**.

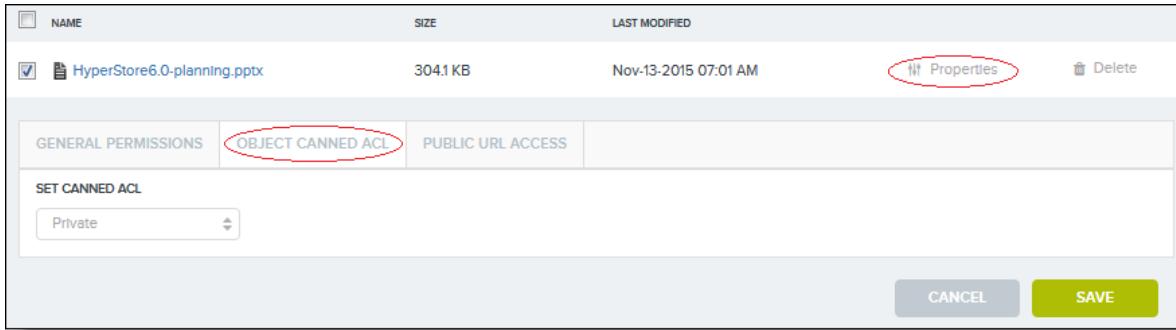
Setting Custom Permissions on an Object Version

If you have **versioning** enabled on your bucket, and you want to set permissions for a version of an object, in the **Objects** interface click **Show Versions**. Under each object name, the interface will then list all stored versions of that object, ordered from the current (most recently uploaded) version to the oldest version. Each version is identified by a system-generated alphanumeric identifier, and for each version the upload timestamp is shown (the date and time that particular version of the object was uploaded to the system). To set custom permissions for an object version, in the CMC's **Objects** interface click **Properties** in the object version's display row. This opens the properties interface for that object version. Then follow the instructions above.

IMPORTANT: When you set permissions for a version of a versioned object, the permissions apply **only to that version of the object** -- not to any past or future versions of the object. For example, if you set permissions for the current version of a versioned object, and then you later upload a revised version of the object, the permissions that you set previously will not apply to the revised version of the object. If you want there to be permissions on the revised version of the object, you must explicitly set the permissions on that object version after you upload it. This behavior is compliant with Amazon S3.

5.4.4.2. Set "Canned" S3 Permissions on an Object

"Canned" S3 permissions (also known as "canned ACLs") are pre-defined sets of S3 permissions that you can assign to an object. To set canned permissions on an object, in the CMC's "**Set Object Properties**" (page 219) interface for the object select the **Object Canned ACL** tab.



- From the drop-down list, choose the canned ACL to assign to the object. You can only choose one. Options are:

Private

Object owner has full access rights and no one else has access rights. This is the default.

Public Read

Object owner has full access rights and the Public grantee group has read access.

Authenticated Read

Object owner has full access rights and the Cloudian User group has read access.

Bucket Owner Read

Object owner has full access rights and the bucket owner has read access.

Bucket Owner Full Control

Object owner and bucket owner both have full access rights.

- Click **Save**.

Note An object can have only one canned ACL attached to it at a time. If you apply one canned ACL to an object, then later apply a different canned ACL, the first canned ACL will be removed.

Setting Canned Permissions on an Object Version

If you have [versioning](#) enabled on your bucket, and you want to set permissions for a version of an object, in the **Objects** interface click **Show Versions**. Under each object name, the interface will then list all stored versions of that object, ordered from the current (most recently uploaded) version to the oldest version. Each version is identified by a system-generated alphanumeric identifier, and for each version the upload timestamp is shown (the date and time that particular version of the object was uploaded to the system). To set canned permissions for an object version, in the CMC's **Objects** interface click **Properties** in the object version's display row. This opens the properties interface for that object version. Then follow the instructions above.

IMPORTANT: When you set permissions for a version of a versioned object, the permissions apply **only to that version of the object** -- not to any past or future versions of the object. For example, if you set permissions for the current version of a versioned object, and then you later upload a revised version of the object, the permissions that you set previously will not apply to the revised version of the

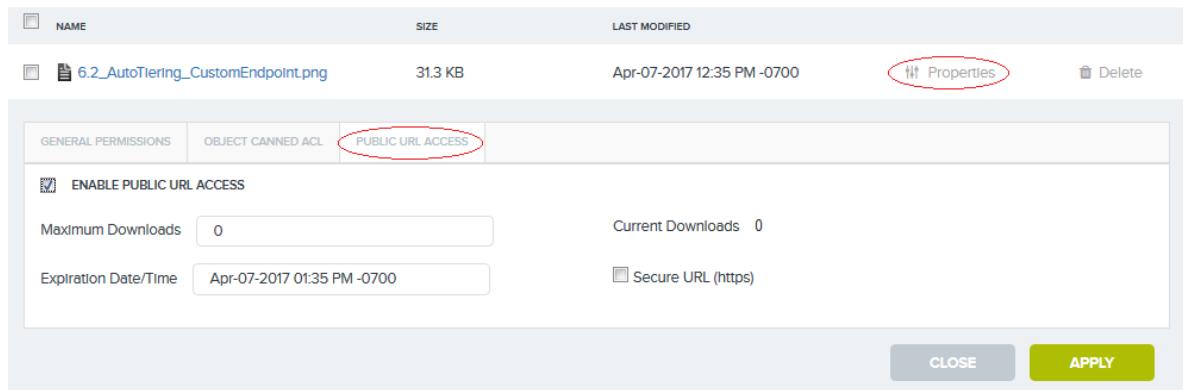
object. If you want there to be permissions on the revised version of the object, you must explicitly set the permissions on that object version after you upload it. This behavior is compliant with Amazon S3.

5.4.4.3. Set Public URL Permissions on an Object

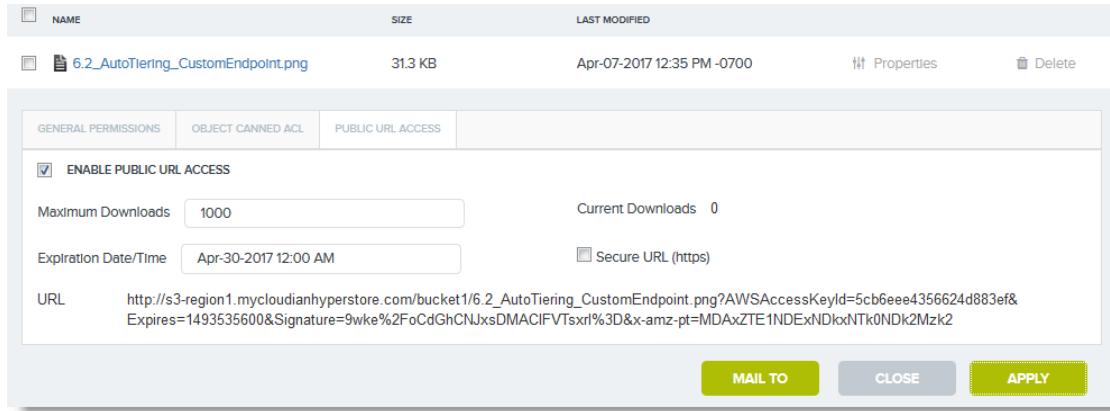
Setting public URL permissions on an object enables members of the public to access the object by using a standard web browser (rather than an S3 client application). The system will generate for the object a web URL which you can provide to whoever you want to have access to the object.

Note You cannot set public URL permissions on an object that the system has encrypted with a user-provided encryption key. By contrast, public URL permissions are supported for files that the system has encrypted with a system-generated encryption key, which is the more typical encryption method.

To set public URL permissions on an object, in the CMC's "**Set Object Properties**" (page 219) interface for the object select the **Public URL Access** tab.



1. Click the "Enable Public URL Access" checkbox.
2. Configure the public URL access for the object:
 - In the "Maximum Downloads" field, specify a maximum number of times that the object can be downloaded in total (the maximum total downloads that you want to allow, by all users combined). If you want to allow an unlimited number of downloads, set this to "-1" (negative one). Note that after enabling public URL access for the object, you can subsequently return to this interface and view the number of times the object has been downloaded to date (the "Current Downloads" field).
 - In the "Expiration Date/Time" field, choose an expiration date/time at which the public URL for the object will expire. After this time the URL will no longer be valid and the object will not be accessible via the URL. If you click in the field, a calendaring tool displays to make it easy to choose the date and time.
 - If you want HTTP access to the object to be SSL-secured, select the "Secure URL (HTTPS)" checkbox.
3. Click **Apply** and the system-generated public URL for the object will display in the interface.



The public URL for an object has the following format:

```
http[s]://<de-
faultS3Domain>/<bucketName>/<objectName>?AWSAccessKeyId=<accessKeyOfObjectOwner>
&Expires=<expiryTimeInUnix>&Signature=<signatureString>&x-amz-pt=<-
uniqueIDforMaxDownloadTracking>
```

The CMC interface makes it easy for you to share this public URL with others. Once the public URL is displayed in the interface, click **Mail To** to open your default email client with the public URL pre-populated into the message body of a new email message. Just add recipients and optionally edit the subject line (which by default is "Public URL Access Link") and then send the email message.

Note For the **Mail To** feature to work, your browser must be configured to be able to open your default web-based email client.

If you want, you can make changes to an object's public URL settings at a later point in time. For example, you can change the maximum allowed downloads or the expiration time. Note that in the "Expiration Date/Time" field the calendaring tool provides a **Now** button in case you want to terminate public URL access for the object immediately. Be sure to click **Apply** to save your changes.

Note If you change the expiration date/time of a public URL and click **Apply** this will result in the generation of a **new URL**. This is because the expiration times is part of the URL -- so changing the expiration time necessarily results in a new URL. By contrast, changing the maximum allowed downloads does not result in a new URL.

5.4.5. Search for an Object

In the CMC's **Objects** view you can use the **Search by Prefix** function to find objects with names that start with a particular prefix (set of alphanumeric characters) or to find one specific object.

To find objects that start with a common prefix:

1. Navigate to the folder that the objects are in. (Or if the desired objects are at the root level of the bucket, skip this step.)
2. Click **Search By Prefix** and then enter the prefix and click **OK**. For example, if you enter the prefix "log" the display will list all objects that have names that start with "log".

Note: In the back end of the S3 storage system, "folder" paths are implemented as being part of the full object names. Therefore you cannot do a prefix search that retrieves objects that are in different "folders" (since technically those objects have different prefixes). For example you cannot do a prefix search that would retrieve both an object named "11-14-2015_notes.docx" under a "Documents" folder and an object named "11-14-2015_sunset.jpg" under an "Images" folder — since in the S3 storage layer those two objects are really named as "Documents/11-14-2015_notes.docx" and "Images/11-14-2015_sunset.jpg" and thus do not share a common prefix.

To find one specific object:

1. Navigate to the folder that the object is in. (Or if the desired object is at the root level of the bucket, skip this step.)
2. Click **Search By Prefix** and then enter the object's full name (or enough of its name to uniquely identify it), and click **OK**. If the object exists, it will then display in the **Objects** view.

Note: The **Search By Prefix** function does not search across multiple folders. You must be in the correct folder before doing the file search. For more explanation see the Note in the **To find objects that start with a common prefix section** above.

5.4.6. Download an Object

To download an object from storage, click on the object name and then choose whether to directly open the object or to save it to your computer.

Note In your object directory display, certain object names may have beside them a left or right-pointing arrow icon that indicates that the object has been subject to auto-tiering. For information about the icons and about retrieving such objects, see "**Restore an Auto-Tiered Object**" (page 225).

5.4.6.1. Downloading an Object Version

If you have versioning enabled on your bucket, and you want to download a particular version of an object, in the CMC's **Objects** view click **Show Versions** (if object versions are not already showing in the object list). When versions are shown, under each object name the interface lists all stored versions of that object, ordered from the current (most recently uploaded) version to the oldest version. Each version is identified by a system-generated alphanumeric identifier, and for each version the upload timestamp is shown (the date and time that particular version of the object was uploaded to the system). To download a version of an object, click the alphanumeric identifier.

5.4.7. Restore an Auto-Tiered Object

If a bucket that you're browsing has been configured for auto-tiering, any objects that have been auto-tiered will display in the CMC's **Objects** view but will be marked with a special icon. Because these objects are currently stored in a remote tiered storage system (such as Amazon, Azure, or Google), you may not be able to directly download these objects. Whether they are directly downloadable or not depends on where they are stored — for example, objects in Amazon Glacier are never directly downloadable — and on the user-defined

bucket lifecycle configuration that governed the auto-tiering of the objects (specifically, [whether the lifecycle configuration supports "streaming" GETs](#) of tiered objects rather than requiring a restore operation).

You can try directly downloading an auto-tiered object by clicking on its object name. If direct download (streaming) is not supported for the object, a response message will indicate that you need to temporarily restore a local copy of the object rather than directly downloading it.

To restore one or more auto-tiered objects:

1. Click the checkbox(es) to the left of the object name(s).
2. Click the **Restore** button.
3. In the dialog that pops up, use the "Restore" number of days field to enter the number of days for which you want the restored object(s) to be locally available in your HyperStore S3 service.

Note: The dialog also presents you the option to choose between Bulk retrieval, Standard retrieval, and Expedited retrieval, but **in the current HyperStore release these retrieval settings have no effect** on how the retrieval is implemented.

4. Click the **OK** button.

Note that restore does not happen instantly — it can take up to six hours before auto-tiered objects are restored to your local HyperStore S3 service (or up to nine hours for objects that have been auto-tiered to Amazon Glacier). In the interim, the object is marked with a [special icon](#) that indicates that it's in the process of being restored. During this stage you cannot download the object.

After an object has been restored (as indicated by a different [special icon](#)), you can download it in the normal way.

Note For special considerations for versioned objects, see "[Restoring Auto-Tiered Object Versions](#)" (page 227).

Note The Restore "number of days" duration is implemented specifically as follows: Starting from the day and time at which you submit the restore request, the restore duration will end at the **first midnight after your specified number of days have passed**. For example: If on the 15th of a month, in the morning, you submit an object restore request with a specified restore period of 3 days, then -- with 3 days having passed by the morning of the 18th -- the object's restore duration will end at 00:00 (midnight) of the 19th.

Note In the case of objects that have been tiered to Spectra BlackPearl, if the tape on which an object resides has been ejected the restore attempt will fail and an error will be written to the S3 application log. The logged error message will indicate the ID of the tape that needs to be reinserted in order to restore the object.

5.4.7.1. Icons for Transitioned or Restored Objects

In your object list display, the document icon to the left of object names will indicate if an object is transitioned, transitioning, restored, or restoring. The icons feature right-pointing or left-pointing red or blue arrows.

Icon	Meaning
	Object is in the process of transitioning to a remote tiered storage system.
	Object has been transitioned to a remote tiered storage system.
	A copy of the transitioned object is in the process of being restored to the local HyperStore S3 service. Download of the object is not yet supported.
	A copy of the transitioned object has been restored to the local HyperStore S3 service. Download is supported.

5.4.7.2. Deleting an Auto-Tiered Object

If you want to **delete** an object that has been auto-tiered, you can do so by deleting the object through the **Objects** interface, just like any other object. You do not need to restore the object first. When the HyperStore system is deleting an auto-tiered object, it first triggers the deletion of the object from Amazon, and then after that succeeds it deletes the local reference to the object.

IMPORTANT: Do not overwrite or delete tiered objects directly through the destination system's interfaces. Doing so will cause a discrepancy between the local metadata in HyperStore and the actual data in the destination bucket. If you want to overwrite or delete tiered objects, do so through HyperStore interfaces (such as the CMC or an S3 application accessing the HyperStore S3 Service). In the case of auto-tiering from one HyperStore region to another HyperStore region, any overwriting or deleting of objects should be done through the source bucket not the destination bucket.

5.4.7.3. Restoring Auto-Tiered Object Versions

If you have versioning configured on your bucket as well as auto-tiering, then it may be that certain versions of your objects have been auto-tiered to an external destination system. A common configuration is to have object versions get auto-tiered after they become non-current. For example, the first version of an object would get auto-tiered after a second version of that same object is uploaded to the local system; and the second version would get auto-tiered after a third version of the object is locally uploaded.

If you have your bucket configured for auto-tiering of non-current object versions, retrieving object versions works like this:

- In the **Objects** interface, click **Show Versions** so that all object versions display. If you are using auto-tiering of non-current object versions, typically the newest version of each object will be in your local bucket and older versions of each object will have been auto-tiered to the external destination system (as indicated by the special icon next to transitioned object versions). You can retrieve the **current object version** (stored locally) in the usual way, by simply clicking on the object version name.
- If your bucket's auto-tiering configuration allows for streaming GETs of auto-tiered objects (which is the default configuration), you can retrieve auto-tiered non-current object versions by clicking on the version name of the non-current object version that you want to retrieve.

- If your bucket's auto-tiering configuration does not allow for streaming GETs of auto-tiered objects, the way to locally retrieve an auto-tiered non-current object version is to restore it by clicking the **Restore** button to the right of the object version. In the number of days field that appears, enter the number of days for which you want the restored object version to be locally available in your HyperStore S3 service.

Note that restore does not happen instantly — it can take up to six hours before an auto-tiered object version is restored to your local HyperStore S3 service (or up to nine hours for object versions that have been auto-tiered to Amazon Glacier). In the interim, the object version is marked with a [special icon](#) that indicates that it's in the process of being restored. During this stage you cannot download the object version.

After an object version has been restored (as indicated by a different [special icon](#)), you can download it in the normal way.

5.4.8. Delete an Object

To delete a single object from storage, click **Delete** at the right side of the object's display row. After you click **OK** in the confirmation dialog, the object is deleted.

To delete multiple objects from storage, click on the checkboxes to the left of the object names, and then click the **Delete** button at the bottom right of the **Objects** interface. After you click **OK** in the confirmation dialog, the objects are deleted.

5.4.8.1. Deleting an Object Version

If you have [versioning](#) enabled on your bucket, and you want to delete a particular version of an object, in the CMC's [Objects](#) view click **Show Versions**. When versions are shown, under each object name the interface lists all stored versions of that object, ordered from the current (most recently uploaded) version to the oldest version.. Each version is identified by a system-generated alphanumeric identifier, and for each version the upload timestamp is shown (the date and time that particular version of the object was uploaded to the system). To delete a version of an object, click **Delete** at the right side of the object version's display row. After you click **OK** in the confirmation dialog, the object version is deleted.

Note Deleting any one version of any object -- including the newest version -- does not delete the other versions of the object.

If you delete a versioned object when the **Objects** interface is in "Hide Versions" mode, it will appear that the object has been entirely deleted. But in fact only the newest version is deleted, and all other versions of the object remain in the system. You can see this if you click **Show Versions**.

To delete multiple versions of an object, click on the checkboxes to the left of the object versions, and then click the **Delete** button at the bottom right of the **Objects** interface. After you click **OK** in the confirmation dialog, the object versions are deleted.

5.4.8.2. Deleting Objects from a Locked Bucket

If a bucket has a bucket lock (WORM) policy applied to it, you cannot delete objects that have not yet reached the end of their retention period. If you try to do so the CMC will display this error:

Unable to process request. BucketLockPolicyError: The bucket's lock-policy is preventing this action.

For more on this topic see "[Set a Lock Policy for a Bucket \(WORM\)](#)" (page 212).

5.4.8.3. Deleting an Auto-Tiered Object

See "[Deleting an Auto-Tiered Object](#)" (page 227).

5.5. Users & Groups

The **Users & Groups** tab contains the following functions:

- [Manage Users](#)
- [Manage Groups](#)
- [Rating Plan](#)
- [Account Activity](#)
- [Whitelist](#)

5.5.1. Manage Users

Path: **Users & Groups** → **Manage Users**

The screenshot shows the Cloudian management interface. At the top, there is a navigation bar with various tabs: Analytics, Buckets & Objects, Users & Groups (which is highlighted with a red oval), Cluster, Alerts (18), Admin, and Help. Below the navigation bar, there is a secondary navigation bar with tabs: Manage Users (highlighted with a red oval), Manage Groups, Rating Plan, Account Activity, and Whitelist. The main content area is titled "MANAGE USERS". It features a search bar labeled "Search For A User By ID:" with a placeholder "Enter prefix or complete user ID". Below the search bar are three dropdown filters: "Group Name" (set to "CloudianTest"), "User Type" (set to "All"), and "User Status" (set to "All"). At the bottom right of the content area is a large green "SEARCH" button.

Supported tasks:

- "[Add a User](#)" (page 230)
- Work with existing users:
 - "[Retrieve a User or List of Users](#)" (page 231)
 - "[Edit or Suspend a User](#)" (page 232)
 - "[View or Edit a User's Security Credentials](#)" (page 234)
 - "[Set Quality of Service \(QoS\) Controls](#)" (page 250)
 - "[Manage a User's Stored Objects](#)" (page 236)
 - "[Delete a User](#)" (page 236)

5.5.1.1. Add a User

IMPORTANT: For a user who belongs to a group that is enabled for LDAP-based authentication:

* If you want the user to be authenticated against the LDAP system when he or she logs in to the CMC, **do not create the user here**. Instead, just have the user log into the CMC with their LDAP credentials and upon that first log-in the CMC will automatically create the user in the HyperStore system. Note however that the user name must meet the character restrictions described under "User ID" below. For example, spaces are not allowed in user names.

* If you want the user to be authenticated by a CMC-based password rather than authenticating against the LDAP system, create the user here. The CMC will not use LDAP-based authentication for users created through the **Add New User** interface described below.

To add a user:

1. In the CMC's [Manage Users](#) page, click **New User**. This opens the **Add New User** panel.

The screenshot shows the 'ADD NEW USER' panel. At the top right is a checked checkbox labeled 'Active User'. Below it are three input fields: 'User ID:' with a placeholder 'User ID', 'User Type:' with a dropdown menu showing 'User' selected, and 'Group Name:' with a dropdown menu showing 'Please select a Group'. Below these are 'Password:' and 'Confirm Password:' fields, both with placeholder text. A 'More' link is located below the password fields. At the bottom are 'CANCEL' and 'SAVE' buttons.

2. In the **Add New User** panel, complete the following user information:

User ID (mandatory)

- Must be unique within the group.
- Only letters, numbers, dashes, and underscores are allowed. No spaces or special characters.
- Maximum allowed length is 64 characters.
- The following user IDs are reserved for system use and are not available to individual users: "anonymous", "public", "null", "none", "admin", "0".

User Type (mandatory)

From the drop-down list select the type of user that you want to create:

- **User** -- A regular user of the storage service. He or she will be able to use the CMC to create storage buckets, upload and download objects, display reports on their service usage, and manage their service access credentials.
- **Group Admin** -- An administrator of a particular group of storage service users. He or she will be able to use the CMC to perform administrative functions for the group, such as adding users or setting user-level QoS profiles. A group admin has a storage service account and can upload their own objects to storage. A group admin can also manage stored objects on behalf of particular users within the group.

- **System Admin** -- A system administrator. He or she will be able to use the CMC to perform a variety of system administration and service administration tasks. System admins do not have a storage service account and cannot upload their own objects to storage. However, system admins can manage stored objects on behalf of particular service users.

Group Name (mandatory)

From the drop-down list choose the group in which the new user will be created.

This field is hidden and not relevant if User Type is System Admin. All system admin users belong to the System Admin group.

Password / Confirm Password (mandatory)

The user's password for logging into the CMC.

Passwords must meet the following conditions:

- Minimum of nine characters, maximum of 64 characters
- Must contain at least three of these four types of characters:
 - Lower case letters
 - Upper case letters
 - Numbers
 - Special characters such as !, @, #, \$, %, ^, etc.

More (optional; includes rating plan assignment)

When you click "More" these additional optional fields display:

- * Full Name (maximum 64 characters)
- * Address and contact information

Rating Plan

Rating plan to assign the user for billing calculation purposes. A new user's rating plan assignment defaults to whatever rating plan you've assigned to the group to which the user belongs. If you assign a user a different rating plan than the plan assigned to his or her group, the individual plan assignment will be applied to that user rather than the group default plan.

If you don't choose a rating plan for the user, the user is automatically assigned the default rating plan for the user's group.

If your HyperStore system has multiple service regions, rating plan assignment is on a per-region basis. A "Region" drop-down list displays which includes each of your region names. For each region, assign the user a rating plan that will apply to the user's service use within that region.

3. Click **Save**.

5.5.1.2. Retrieve a User or List of Users

In the CMC's [Manage Users](#) page you can retrieve a single user or a filtered list of users.

1. Select or enter your user list filtering criteria. You can filter by:

Search for User By ID

Specify a user ID prefix to retrieve a list of users whose IDs start with that prefix.

Specify a complete user ID if you want to retrieve just that one user.

This field is case-sensitive.

Leave this field blank if you want to filter exclusively by the other criteria.

Group Name

This is a drop-down list of group names in the system.

Note If you have more than 100 groups in your system this is implemented as a text input field with auto-complete (rather than a drop-down list).

User Type

- User — Regular service users.
- Group Admin — Group admin users.
- All — Regular service users, group admin users, and system admin users.

User Status

- Active — Users who are currently allowed service access.
- Inactive — Users whose service access credentials have been deactivated but who have not been deleted from the system. They may still have objects in storage.
- All — Active users and inactive users.

Note The CMC does not support retrieving a list of users who have been deleted from the system. If you need to retrieve a list of deleted users, you can do so through the Admin API -- see [GET /user/list](#).

2. Click **Search**. Your filtered search results then display in the lower part of the page.

USER ID	GROUP NAME	USER TYPE	STATUS	ACTIONS
PubsGroupAdmin	Pubs	GroupAdmin	Active	Edit Security Credentials Set QoS View User Data Delete
PubsUser1	Pubs	User	Active	Edit Security Credentials Set QoS View User Data Delete
PubsUser2	Pubs	User	Active	Edit Security Credentials Set QoS View User Data Delete

You can then take any of the following actions regarding the retrieved user(s):

- **"Edit or Suspend a User"** (page 232)
- **"View or Edit a User's Security Credentials"** (page 234)
- **"Set Quality of Service (QoS) Controls"** (page 250) for a user
- **"Manage a User's Stored Objects"** (page 236)
- **"Delete a User"** (page 236)

5.5.1.3. Edit or Suspend a User

1. Use the CMC's [Manage Users](#) page to **"Retrieve a User or List of Users"** (page 231).

2. In the "Actions" column for the user that you want to edit, click **Edit**. This drops down a panel for editing the user's attributes.

The screenshot shows the 'Edit' dialog for a user named 'PubsUser1'. The 'User Type' dropdown is set to 'User'. The 'Canonical ID' field displays the value 'a1b8a09e4ee421ffc89b00c50272f22f'. The 'Rating Plan' dropdown is set to 'Default-RP'. At the top right of the dialog, there is a checked checkbox labeled 'Active User'. At the bottom right are two buttons: 'CANCEL' and 'SAVE'.

3. Make your desired changes to the user's attributes:

Active User

The "Active User" checkbox is in the upper right corner of the Edit User panel.

- Active User checkbox checked — User is active. User has access to the storage service and the CMC.
- Active User checkbox unchecked — User is suspended. User will be denied access to the storage service and the CMC. However, the user will remain in the system and their objects will remain in storage.

Note If you want to remove a user entirely and have the system delete the user's stored objects, [delete the user](#) rather than just suspending the user.

User Type

- **User** -- Regular user of the storage service. He or she will be able to use the CMC to create storage buckets, upload and download objects, display reports on their service usage, and manage their service access credentials.
- **Group Admin** -- An administrator of a particular group of storage service users. He or she will be able to use the CMC to perform administrative functions for the group, such as adding users or setting user-level QoS profiles. A group admin has a storage service account and can upload their own objects to storage. A group admin can also manage stored objects on behalf of particular users within the group.
- **System Admin** -- System administrator. He or she will be able to use the CMC to perform a variety of system administration and service administration tasks. System admins do not have a storage service account and cannot upload their own objects to storage. However, system admins can manage stored objects on behalf of particular service users.

Note: You cannot change a regular user's or group administrator's User Type to System Admin. Also, you cannot change a system administrator's User Type to anything other than System Admin.

Canonical ID

The system automatically generates a unique canonical ID for every user. This ID cannot be edited.

Rating Plan

Use the "Rating Plan" drop-down list to assign the user a rating plan for billing calculation purposes. If you assign a user a different rating plan than the plan assigned to his or her group, the individual plan assignment will be applied to that user rather than the group default plan.

Since system administrator user IDs do not have their own S3 service accounts, the "Rating Plan" drop-down list does not display if you are editing a system admin user.

Note for multi-region systems

If your HyperStore system has multiple service regions, rating plan assignment is on a per-region basis. A "Region" drop-down list displays which includes each of your region names. For each region, assign the user a rating plan that will apply to the user's service use within that region, clicking Save each time.

Full name and contact information (click 'More' to display these fields)

User's full name (maximum 64 characters) and contact information.

Note If the user is configured to be authenticated against an external LDAP system rather than by a CMC-based password, the edit user interface will display some text indicating that the user is LDAP-enabled. This is not editable. Whether or not a user is LDAP-enabled is determined at the time that the user is created and this cannot be changed subsequently.

When LDAP is enabled for a user, the CMC authenticates the user against their password stored in the LDAP server. The password that's stored in the CMC for such a user is simply a system-generated random string that is not actually used for authentication.

4. Click **Save**.

5.5.1.4. View or Edit a User's Security Credentials

1. Use the CMC's [Manage Users](#) page to "**Retrieve a User or List of Users**" (page 231).
2. In the "Actions" column for the user that you want to edit, click **Security Credentials**. This displays a panel for changing the user's CMC sign-in password and viewing and managing the user's S3 access credentials.

User Credentials

SIGN-IN CREDENTIALS

USER ID:	PubsUser1	GROUP ID:	Pubs
NEW PASSWORD:	<input type="password"/>		
CONFIRM PASSWORD:	<input type="password"/>		

CHANGE PASSWORD

S3 ACCESS CREDENTIALS

CREATED	ACCESS KEY ID	ACTIONS
Apr 08 2018 07:49:36 GMT-0400	6ed8c97da762ec742d88 *	View Secret Key Inactivate Delete <small>* Active Access Key</small>

CREATE NEW KEY

Close

Note If the user is an LDAP-based user, in the CMC "Sign-In Credentials" section of the Security Credentials page does not display. Instead you should manage the user's password in your LDAP system.

3. Take the desired action:

- **To set a new CMC sign-in password** for the user, enter the password in the "New Password" field and in the "Confirm Password" field, then click **Change Password**.
Passwords must meet the following conditions:
 - Minimum of nine characters, maximum of 64 characters
 - Must contain at least three of these four types of characters:
 - Lower case letters
 - Upper case letters
 - Numbers
 - Special characters such as !, @, #, \$, %, ^, etc.
- **To view the S3 secret access key** that corresponds to an access key ID, to the right of the access key ID click **View Secret Key**. A secret key display box appears which enables you to view the user's secret key and to copy it using <Ctrl>-c if you want to. Note that the **OK** and **Cancel** buttons have no effect other than to close the secret key display box.
- **To activate or deactivate an S3 access key**, to the right of the displayed access key ID click **Activate** or **Inactivate**.
- **To create a new S3 access key** click **Create New Key**. A new access key ID then displays in the access key list.

- To delete an S3 access key, to the far right of the displayed access key ID click **Delete**. You will be asked to confirm that you want to delete the key.
4. When you're done click **Close** to close the **User Credentials** panel.

5.5.1.5. Manage a User's Stored Objects

1. Use the CMC's [Manage Users](#) page to "**Retrieve a User or List of Users**" (page 231).
2. In the "Actions" column for the user click **View User Data**. This opens the **Buckets & Objects** page for that user.

You can then work with the user's buckets and objects.

- "**Add a Bucket**" (page 192)
- "**Set Bucket Properties**" (page 194)
- "**Delete a Bucket**" (page 214)
- "**Create or Delete a "Folder"**" (page 215)
- "**Upload an Object**" (page 217)
- "**Set Object Properties**" (page 219)
- "**Search for an Object**" (page 224)
- "**Download an Object**" (page 225)
- "**Delete an Object**" (page 228)
- "**Restore an Auto-Tiered Object**" (page 225)

Note The **Buckets & Objects** page will continue to be populated with a view of this user's data throughout your CMC login session, unless you use the **Manage Users** page to switch to viewing a different user's data.

Note Since system administrator user IDs do not have their own S3 service accounts, the **View User Data** link does not display if you are viewing a system admin user.

5.5.1.6. Delete a User

You can delete a HyperStore service user from the system.

IMPORTANT: If you delete a user, **the user's stored buckets and objects will be deleted from the system**. If you want to temporarily deny service access to a user without deleting their stored buckets and objects, don't delete the user. Instead, suspend the user by using the user Edit function.

To delete a user:

1. Use the CMC's [Manage Users](#) page to "**Retrieve a User or List of Users**" (page 231).
2. In the "Actions" column for the user that you want to delete, click **Delete**.

Note: You cannot delete the default system administrator account. This is not allowed.

- When prompted by the system, confirm that you want to delete the user.

5.5.2. Manage Groups

Path: **Users & Groups** → **Manage Groups**

Supported tasks:

- "**Add a Group**" (page 237)
- Work with existing groups:
 - "**Retrieve a Group or a List of Groups**" (page 241)
 - "**Edit a Group**" (page 241)
 - "**Set Quality of Service (QoS) Controls**" (page 250)
 - "**Delete a Group**" (page 243)

5.5.2.1. Add a Group

- In the CMC's [Manage Groups](#) page, click **New Group**. This opens the **Add New Group** panel.

- In the **Add New Group** panel, complete the group information:

Group Name (mandatory)

- Must be unique within your entire HyperStore service.
- Only letters, numbers, dashes, and underscores are allowed.
- Maximum allowed length is 64 characters.

Note If you intend to enable LDAP authentication for this group (as described further below), the "Group Name" should if possible be the exact name of the group as it exists in the LDAP system. If the "Group Name" field's character restrictions prevent you from using the group's exact name from LDAP, enter something similar as the "Group Name" and then use the "LDAP Org Unit" field to specify the group's exact name from LDAP.

Group Description (optional)

- Maximum allowed length is 64 characters.

Rating Plan (optional)

- Rating plan to assign to the group for billing calculation purposes. This rating plan will apply to each user in the group, with the exception of any users to whom you individually assign a different rating plan.
- If you don't choose a rating plan for the group, the "**Default Rating Plan**" (page 246) (Default-RP) is automatically assigned to the group.

If your service deployment has multiple service regions, rating plan assignment is on a per-region basis. A "Region" drop-down list displays which includes each of your region names. For each region, assign the group a rating plan that will apply to the group's service use within that region.

Enable LDAP Authentication (optional)

Select this checkbox if you want to enable LDAP authentication this group. If you enable LDAP authentication for this group, then:

- Additional settings display in the **Add New Group** interface which allow you to configure LDAP authentication details for the group. These settings are described further below.
- If a user tries to log into the CMC as a member of this group and the user is not already registered in HyperStore as a member of the group, the CMC will attempt to authenticate the user against the LDAP system specified for this group. If the authentication succeeds, the CMC will automatically provision the user into HyperStore. Going forward whenever the user logs in the CMC will recognize the user as a registered HyperStore user, but will continue to authenticate the user against the LDAP system each time rather than by reference to a CMC-based password.
- Through the CMC's [**Add New User**](#) interface you will be able, if you wish, to create within this group local users who are to be authenticated by use of a CMC-based password rather than by reference to an LDAP system.

Note that within a HyperStore group that has LDAP authentication enabled you can have both LDAP-authenticated users and users who are authenticated by a CMC-based password rather than LDAP. For users who you want to be authenticated by LDAP, **do not manually create those users** through the CMC's **Add New User** interface. Instead, let the CMC create those users automatically when they log in for the first time and are successfully authenticated against the LDAP system. Only use the **Add New User** interface if you want to also have some users who are **not** LDAP-authenticated.

Note If you want the group administrator to be authenticated by LDAP, have him or her log into the CMC using their LDAP credentials. Once this occurs and the CMC

automatically provisions them into the system as a regular user, you can subsequently edit their user profile (using the CMC's [user edit](#) function) to promote them to the group admin role.

LDAP Org Unit (optional)

If the value you entered in the "Group Name" field is not the exact name of the group as it exists in the LDAP system, you can use the "LDAP Org Unit" field to specify the exact name of the group as it exists in the LDAP system. This would typically be the "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.

The "LDAP Org Unit" field does not have the character restrictions that the "Group Name" field has. For example if the group's "ou" value in LDAP is "Quality Assurance (U.S.)", in the CMC's "Group Name" field you would need to enter something like "Quality_Assurance_US" to satisfy that field's character restrictions, and then you can enter the exact name from LDAP -- "Quality Assurance (U.S.)" -- in the "LDAP Org Unit" field.

Note Users must use the "Group Name" value when completing the CMC login screen's "Group Name" field. In the example above, users would use "Quality_Assurance_US" in the CMC login screen's "Group Name" field. The CMC when authenticating such users against the LDAP system would use "Quality Assurance (U.S.)" as the user's LDAP group.

LDAP Server URL (mandatory if using LDAP authentication)

Use this attribute to specify the URL that the CMC should use to access the LDAP Server when authenticating users in this group. For example:

ldap://my.ldap.server:389

Or for another example:

ldap://my.ldap.server:389/o=MyCompany

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.

LDAP User DN Template (mandatory if using LDAP authentication)

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}" (as shown in the example 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:

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.

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

userPrincipalName={userId}

With the approach above -- such as might be used in an Active Directory environment -- LDAP-enabled users from this group will log in with their LDAP *userPrincipalName* 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 "LDAP Search", "LDAP Search User Base", and "LDAP Match Attribute" settings as described below.

LDAP Search (optional)

If you want to establish a LDAP-based user authorization filter to complement the user authentication template that you set with the "LDAP User DN Template" field, then use the "LDAP Search", "LDAP Search User Base", and "LDAP Match Attribute" fields 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 "LDAP Search" field to specify the user identifier type that you used in the "LDAP User DN Template" field, in format "*(<LDAP_user_identifier_attribute>={userId})*". This is used to retrieve a user's LDAP record in order to apply the filtering. Here are two examples:

- *(uid={userId})*
- *(userPrincipalName={userId})*

LDAP Search User Base (optional)

For background information about the purpose of this setting, see the description of the "LDAP Search" setting above.

Use the "LDAP Search User Base" field to specify the LDAP search base from which the CMC should start when retrieving the user's LDAP record in order to apply filtering. For example, *dc=my-company,dc=com*. Or for another example, *uid={userId},ou=engineering,dc=my-company,dc=com*.

LDAP Match Attribute (optional)

For background information about the purpose of this setting, see the description of the "LDAP Search" setting above.

Use the "LDAP Match Attribute" field 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>*. Here are two "LDAP Match Attribute" examples:

- *I=California*
- *memberOf=Sales*

Enable S3 endpoints display filter (optional)

- Select this option if you want to filter the S3 endpoints that this group's users will see when they log into the CMC and go to the **Security Credentials** page. If you select this checkbox you will then be able to select which of the system's configured S3 HTTP endpoints, S3 HTTPS endpoints, and S3 website endpoints will display for this group's users in the **Security Credentials** page.
- If you do not select this option, then by default all of the system's S3 HTTP endpoints, S3 HTTPS endpoints, and S3 website endpoints will display for this group's users in the **Security Credentials** page.

Note This feature does not impact a group's users' authorization to access S3 endpoints. It only impacts what S3 endpoint information is displayed to users in the CMC's **Security Credentials** page.

3. Click **Save**.

5.5.2.2. Retrieve a Group or a List of Groups

In the CMC's [Manage Groups](#) page you can retrieve a single group or a filtered list of groups.

1. In the "Search for a Group By Name" field:
 - To retrieve just one specific group, enter the full group name. This field is case-sensitive.
 - To retrieve a group list filtered by group name prefix, enter the prefix.
 - To retrieve a list of all groups, leave this field empty.
2. Click **Search**. Your filtered search results then display in the lower part of the page.

GROUP NAME	GROUP DESCRIPTION	STATUS	ACTIONS
TestGroup	Test Group Account	Active	Group QOS User QOS Group Default

5.5.2.3. Edit a Group

You can change a group's rating plan assignment or service status (active or suspended).

1. Use the CMC's [Manage Groups](#) page to "**Retrieve a Group or a List of Groups**" (page 241).
2. In the "Actions" column for the group click **Edit**. This displays a panel for editing group attributes.

GROUP NAME	GROUP DESCRIPTION	STATUS	ACTIONS
TestGroup	Test Group Account	Active	Group QOS User QOS Group Default

Active Group

Group Description:
Test Group Account

Rating Plan:
Default-RP

Enable LDAP Authentication
 Enable S3 endpoints display filter

CANCEL **SAVE**

3. Make your desired changes to the group attributes:

Active Group

- Active Group checkbox checked — Group is active. Group's users have access to the storage service and the CMC.
- Active Group checkbox unchecked — Group is suspended. All the users in the group — including the group administrator(s) — will be denied access to the storage service and the CMC. However, the group's users will remain in the system and their objects will remain in storage.

Group Description

- Maximum allowed length is 64 characters.

Rating Plan

- Use the "Rating Plan" drop-down list to assign the group a rating plan for billing calculation purposes. This rating plan will apply to each user in the group, with the exception of any users to whom you individually assign a different rating plan.

Note for multi-region systems

If your service deployment has multiple service regions, rating plan assignment is on a per-region basis. A "Region" drop-down list displays which includes each of your region names. For each region, assign the group a rating plan that will apply to the group's service use within that region.

Enable LDAP Authentication

- For information about this option, see the description of the "Enable LDAP Authentication" option in the **"Add a Group"** (page 237) topic.

Note: If you enable LDAP Authentication for an existing group to which users have already been added, those existing users will continue to be authenticated by reference to their CMC-based passwords -- not by reference to an LDAP server. For information about authentication of new users subsequent to enabling LDAP for the group, see **"Add a Group"** (page 237).

Enable S3 endpoints display filter (optional)

- Select this option if you want to filter the S3 endpoints that this group's users will see when they log into the CMC and go to the **Security Credentials** page. If you select this checkbox you will then be able to select which of the system's configured S3 HTTP endpoints, S3 HTTPS endpoints, and S3 website endpoints will display for this group's users in the **Security Credentials** page.
- If you do not select this option, then by default all of the system's S3 HTTP endpoints, S3 HTTPS endpoints, and S3 website endpoints will display for this group's users in the **Security Credentials** page.

Note This feature does not impact a group's users' authorization to access S3 endpoints. It only impacts what S3 endpoint information is displayed to users in the CMC's **Security Credentials** page.

4. Click **Save**.

5.5.2.4. Delete a Group

Note You cannot delete a group that currently has users in it. You must delete the users first, one by one (the CMC does not currently support bulk deletion of users). After deleting all the users you can delete the group.

1. Use the CMC's [Manage Groups](#) page to "**Retrieve a Group or a List of Groups**" (page 241).
2. In the "Actions" column for the group that you want to delete, click **Delete**.
3. When prompted by the system, confirm that you want to delete the group.

5.5.3. Rating Plan

Path: **Users & Groups** → **Rating Plan**

ID	NAME	ACTIONS
Default-RP	Default Rating Plan	Edit
Whitelist-RP	Whitelist Rating Plan	Edit

Supported tasks:

- "**Add a Rating Plan**" (page 243)
- "**Edit a Rating Plan**" (page 245)
- "**Delete a Rating Plan**" (page 247)

Note The **Rating Plan** page is for creating and maintaining rating plans. It is not for assigning a rating plan to users. To assign a rating plan to a group of users, use the CMC's [Manage Groups](#) interface. To assign a rating plan to a specific user, use the [Manage Users](#) interface.

For an overview of HyperStore user billing functionality, see "**Billing Feature Overview**" (page 63).

5.5.3.1. Add a Rating Plan

Rating plans specify pricing for various types and levels of user activity, to facilitate billing. Through the CMC's [Rating Plan](#) page you can create and configure new rating plans.

Note HyperStore includes a pre-configured [Default Rating Plan](#) named "Default-RP". If you wish you can [edit the contents](#) of that plan. There is also a pre-configured plan named "Whitelist-RP" which supports the HyperStore [whitelist feature](#).

To create a new rating plan:

1. In the **Rating Plans** page click **Add Rating Plan**. This opens the **Add Rating Plan** panel.

The screenshot shows the 'ADD RATING PLAN' dialog box. It has fields for 'ID' (with a placeholder 'Enter ID'), 'Name' (with a placeholder 'Enter name'), and 'Currency' (set to 'USD'). Below these are six expandable sections for specifying rates: 'Storage Size Rates', 'Data-Transfer-IN rates', 'Data-Transfer-OUT rates', 'HTTP(S) GET/HEAD Rates', 'HTTP(S) PUT/POST Rates', and 'HTTP(S) DELETE Rates'. At the bottom right are 'CANCEL' and 'SAVE' buttons.

2. Assign the plan a unique ID, and optionally a Name.
3. From the Currency drop-down list, choose the currency units on which the plan's pricing structure will be based (for example, USD for U.S. dollars).
4. Specify the billing rates that will constitute the rating plan. You can specify billing rates per:
 - Average GBs of data in storage during the calendar month for which the bill is calculated ("Storage-Size Rates")
 - GBs of data uploaded to the system. ("Data-Transfer-IN Rates")
 - GBs of data downloaded from the system. ("Data-Transfer-OUT Rates")
 - 10,000 HTTP GET or HEAD requests. ("HTTP[S] GET/HEAD Rates")
 - 10,000 HTTP PUT or POST requests. ("HTTP[S] PUT/POST Rates")
 - 10,000 HTTP DELETE requests. ("HTTP[S] DELETE Rates")

IMPORTANT: If you want to bill for data upload or download volume, or for HTTP request volume, you must enable the **"Track/Report Usage for Request Rates and Data Transfer Rates"** (page 299) setting on the CMC's **Configuration Settings** page. By default this setting is disabled and the system does not maintain per-user HTTP request counts and data transfer byte counts.

Note For an example of these rates applied to a user's bill, see **"Example of a Rating Plan Applied to Calculate a User's Monthly Bill"** (page 64)

For each rated activity you can establish either a single-tier or multi-tier pricing structure, based on activity level.

With **single-tier pricing**, the per-unit rate charged for the activity is not impacted by the activity level; whether the activity level is low or high, the per-unit charge remains the same. The initial display for each rated activity accommodates single-tier pricing.

For example, in the screen below, for Storage-Size Rates, a single-tier pricing structure can be established by simply specifying a Rate Per GB Per Month. In this example, the rate is \$2 per GB-month. That

same per GB-month rate applies whether a user stores, for instance, 10 GB-month (\$20 charge) or 100 GB-month (\$200 charge).

Storage Size Rates	
STORAGE SIZE (GB)	RATE PER GB PER MONTH
	2

ADD

To create a **multi-tier pricing** structure for a rated activity, first click on the activity type (for example, "Data-Transfer-IN Rates"). Then click **Add** once for each additional pricing tier that you want to specify, beyond a single tier. For example, in the screen below, **Add** has been clicked twice to enable a three tier pricing structure for Data-Transfer-IN Rates. In this example, the first five GB of data uploaded during the billing period are priced at \$2 per GB; the next five GB are priced at \$2.50 per GB; and any uploads beyond that level (above the 10 GBs encompassed by the first two tiers) are priced at \$3 per GB.

Data-Transfer-IN rates		
	STORAGE SIZE (GB)	RATE PER GB PER MONTH
Up to	5	2
Next	5	2.5
Above	10	3

ADD

Note that the numbers that you enter in the "Rate Per" column signify units of the currency that you chose in the upper part of the **Add Rating Plan** panel. You can use decimals if you want; for example, if your currency is dollars, you can enter ".1" in the "Rate Per" column to indicate a charge of 10 cents (.1 dollars) per unit of activity.

Note For HTTP request rates, the pricing is based on blocks of 10,000 requests. So for example, if you want the first 50,000 requests to be charged at a certain price per block of 10,000, then in the "Number of 10,000 Requests" field for that tier, **enter 5 — not 50,000**.

- When you're done creating the new rating plan, click **Save** at the bottom of the **Add Rating Plan** panel. The plan will then appear in the **Rating Plans** list.

5.5.3.2. Edit a Rating Plan

- In the CMC's **Rating Plan** page click **Edit** to the right of the plan name. This opens a panel in which you can edit the plan attributes.

The screenshot shows the 'Default Rating Plan' configuration page. It includes fields for 'Name' (set to 'Default Rating Plan') and 'Currency' (set to 'USD'). Below these are sections for 'Storage-Size Rates', 'Data-Transfer-IN Rates', 'Data-Transfer-OUT Rates', 'HTTP(S) GET/HEAD Rates', 'HTTP(S) PUT/POST Rates', and 'HTTP(S) DELETE Rates'. At the bottom right are 'CANCEL' and 'SAVE' buttons.

2. Edit the plan attributes. For attribute descriptions see .
3. Click **Save**.

Note When billing is calculated at the end of a month, the most current version of the rating plan is applied to the whole month's activity. For example, suppose you have a "Gold" rating plan. Late in January, you edit the pricing structure of the Gold plan. At the end of January when bills are generated for customers on the Gold plan, all their January activities will be billed in a way that reflects the modified pricing structure that you established in late January.

Note HyperStore includes a pre-configured [Default Rating Plan](#) named "Default-RP". If you wish you can edit the contents of that plan. There is also a pre-configured plan named "Whitelist-RP" which supports the HyperStore [whitelist feature](#).

5.5.3.3. Default Rating Plan

The Cloudian HyperStore system comes with a default rating plan (with unique ID "Default-RP". In the CMC's [Rating Plan](#) page you can [edit this plan](#) but you cannot delete it.

Groups and users that you do not explicitly assign a rating plan will automatically be assigned the default rating plan.

The default rating plan's currency is US dollars and the pricing structure is as follows:

- Storage billing
 - First tier, up to 1 GB: \$0.14 per GB-month
 - Second tier, from 1+ GB to 6 GB: \$0.12 per GB-month
 - Third tier, above 6 GB: \$0.10 per GB-month
- Data Transfer IN
 - First tier, up to 1 GB: \$0.20 per GB
 - Second tier, above 1 GB: \$0.10 per GB

- Data Transfer OUT
 - First tier, up to 1 GB: \$0.20 per GB
 - Second tier, above 1 GB: \$0.10 per GB
- HTTP GETs/HEADs
 - First tier, up to 10 blocks of 10,000 (that is, up to 100,000 GETs/HEADs): \$0.02 per 10,000 GETs/HEADs
 - Second tier, above 10 blocks of 10,000 (above 100,000 GETs/HEADs): \$0.01 per 10,000 GETs/HEADs
- HTTP PUTs/POSTs
 - No tiering: \$0.02 per 10,000 PUTs/POSTs
- HTTP DELETEs
 - No charge for DELETEs

5.5.3.4. Delete a Rating Plan

In the CMC's [Rating Plan](#) page click **Delete** to the right of the name of the plan that you want to delete. After you confirm that you want to take this action, the rating plan will be deleted from the system.

Note If you delete a plan that is currently assigned to some users, those users will be automatically switched to their group default rating plan, or to the system [Default Rating Plan](#) ("Default-RP") if no group default plan has been set.

You cannot delete the default rating plan, or the "Whitelist-RP" plan (which supports the HyperStore [whitelist feature](#)).

5.5.4. Account Activity

Path: [Users & Groups](#) → [Account Activity](#)

The screenshot shows the CMC's 'Users & Groups' menu bar with several tabs: Analytics, Buckets & Objects, Users & Groups (highlighted with a red circle), Rating Plan, Account Activity (highlighted with a red circle), and Whitelist. Below the tabs is a search bar and a 'GROUPS' section. The main content area is titled 'ACCOUNT ACTIVITY'. It contains four input fields: 'Group Name' (CloudianTest), 'User ID' (empty), 'Time Period' (May 2016), 'Region' (region1), and 'Traffic Type' (Normal). At the bottom right of the activity section is a green 'DISPLAY REPORT' button.

Supported task:

- Generate a billable activity report for a specified user

IMPORTANT: Billing calculation is derived from hourly rollup usage data. The retention period for hourly rollup usage data is configured by `mts.properties.erb`: "**reports.rolluphour.ttl**" (page 511). The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.

To generate a billable activity report for a user:

1. Choose a "Group Name" from the drop-down list and enter a "User ID" for the user.
2. Choose a "Time Period" from the drop-down list. The billing periods are calendar months. You can only create a bill for a past, completed month; you cannot create a bill for the current, in-progress month. The default is the most recent completed month.
3. Choose a "Region" from the drop-down list. Bills are calculated per service region. The CMC does not currently support multi-region aggregate bill generation.
4. Choose a "Traffic Type" from the drop-down list. The options are "Normal" or "Whitelist". "Whitelist" refers to request traffic that originates from white-listed source IP addresses (traffic subject to special pricing), and is only an option if white-listing is enabled in the system. "Normal" refers to all other traffic.
5. Click **Display Report** to display activity and charges for the selected billing period.

The image below shows a sample report. The upper part displays summary report parameters including the name of the rating plan that has been applied to the user's activity. The lower middle shows the charging rules from the rating plan (in this case, a three-tier charging structure for price per GB-month, based on the average level of storage volume used), and the lower right shows the user's usage level for the month (in this case 108 GB-months) and the resulting charges.

ACCOUNT ACTIVITY		Print
Account Info		
<u>Billing Period:</u>	10/01/2015-10/31/2015	<u>Name:</u> Test User
<u>Date Printed:</u>	Nov 25 2015	<u>Account Id:</u> test
<u>Region:</u>	region1	<u>Group Description:</u> Test Group
<u>Rating Plan:</u>	Default-RP	<u>Traffic Type:</u> Normal
Storage Bytes		
	Up to 1 GB-Month : 0.14 (USD) Next 5 GB-Month : 0.12 (USD) Above : 0.10 (USD)	108 GB-Month 10.94
	Total (USD)	10.94

5.5.5. Whitelist

Path: **Users & Groups** → **Whitelist**

WHITELIST		
ID	NAME	ACTIONS
Default-WL	Default Whitelist	Edit

Supported tasks:

- Add or remove IP addresses from the whitelist

The Cloudian HyperStore billing whitelist feature enables service providers to specify a list of IP addresses or subnets that are allowed to have free S3 traffic with the HyperStore system. For S3 requests originating from IP addresses on the whitelist, a special rating plan is used that applies zero charge to all the traffic-related pricing metrics:

- Price per GB data transferred in
- Price per GB data transferred out
- Price per 10,000 HTTP PUT requests
- Price per 10,000 HTTP GET requests
- Price per 10,000 HTTP DELETE requests

The billing whitelist feature affects only the pricing of **traffic**. It does **not** affect the pricing of data **storage**. For data storage billing, users' regular assigned rating plan pricing is applied.

The special rating plan assigned to whitelisted addresses — the rating plan that prices traffic at "0" — has rating plan ID "Whitelist-RP". If you wish, you can edit this rating plan in the [Rating Plan](#) page of the CMC (for example, if you want traffic originating from your whitelisted IP addresses to be specially priced but not free).

Note In the current release you can only have one whitelist, with only one rating plan applied to it.

Note If you are using load balancers in front of the HyperStore S3 Service, the whitelist 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 "[s3_proxy_protocol_enabled](#)" (page 475) in [common.csv](#). 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 whitelist feature. You must use PROXY Protocol if you have load balancers in front of the S3 Service and want to use the whitelist feature .

Note for multi-region systems

In a multi-region HyperStore deployment, the whitelist is applied the same across all regions. There is no support for region-specific whitelists.

To add or remove IP addresses from the whitelist:

1. In the **Whitelist** page, to the right of the whitelist name, click **Edit**. This opens the **Edit Whitelist** panel.

The screenshot shows the 'Edit Whitelist' dialog box. At the top left is the title 'EDIT WHITELIST'. Below it are three input fields: 'ID' containing 'Default-WL', 'Name' containing 'Default Whitelist', and 'Rating Plan' containing 'Whitelist-RP'. To the right of these is a dropdown arrow. Below these fields is a section titled 'IP/Subnet' with a large text input area labeled 'IP/Subnet' which is currently empty. At the bottom right of the dialog is a green rectangular button labeled 'SAVE'.

2. If you want you can change the name of the whitelist ("Default Whitelist"), but you cannot change its ID ("Default-WL").
3. In the "IP/Subnet" box, enter IP addresses or subnets one at a time, pressing your keyboard's Enter key after each entry. You can enter as many as you want. (If you are editing a list that you created through this dialog previously, you also have the option of removing IP addresses or subnets from the list.)

Note: The system will validate IP addresses and subnets for correct syntax. The entire list will be rejected if any address or subnet fails the syntax validation check.

4. After making your changes, click **Save**.

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

5.5.6. Set Quality of Service (QoS) Controls

Use the CMC's **Manage Users** page and **Manage Groups** page to set Quality of Service (QoS) limits for users.

Note By default the HyperStore system's enforcement of QoS restrictions is **disabled**. If you want to use the QoS feature, then before setting specific QoS limits for users and groups you must go to the CMC's **Configuration Settings** page and enable QoS enforcement.

To set QoS limits:

1. Navigate to the QoS configuration panel for the type of QoS controls that you want to set. The table below shows the path to each QoS panel. It also shows how each panel is pre-populated with default settings.

QoS Task	Path to Configuration Panel	Configuration Panel Name	Panel's Default Values
Set a system default user QoS profile	In the Manage Users page, click User QoS Default .	User QoS Limits: Defaults	User QoS Default. Defaults to "unlimited" for all QoS

QoS Task	Path to Configuration Panel	Configuration Panel Name	Panel's Default Values
			settings.
Set a default user QoS profile for members of a group	In the Manage Groups page, retrieve the group. Then for the group click User QoS Group Default .	User QoS Limits: Group Defaults	Defaults to system default user QoS profile.
Set a user-specific QoS profile	In the Manage Users page, retrieve the user. Then click Set QoS for the user.	User QoS Limits: Overrides	Defaults to default user QoS profile for the user's group.
Set a system default group QoS profile	In the Manage Groups page, click Group QoS Default .	Group QoS Limits: Defaults	Defaults to "unlimited" for all QoS settings.
Set a group-specific group QoS profile	In the Manage Groups page, retrieve the group. Then click Group QoS for the group.	Group QoS Limits: Overrides	Defaults to system default group QoS profile.

All QoS configuration panels have the same appearance and the same options, as shown in the sample panel below:

USER QOS LIMITS: OVERRIDES		
QOS ITEM	WARNING LIMIT	HIGH LIMIT
Storage Quota (KB)		Unlimited <input checked="" type="checkbox"/>
Storage Quota Count		Unlimited <input checked="" type="checkbox"/>
Request Rate (Requests/minute)	Unlimited <input checked="" type="checkbox"/>	Unlimited <input checked="" type="checkbox"/>
Data Bytes IN (KB/minute)	Unlimited <input checked="" type="checkbox"/>	Unlimited <input checked="" type="checkbox"/>
Data Bytes OUT (KB/minute)	Unlimited <input checked="" type="checkbox"/>	Unlimited <input checked="" type="checkbox"/>

- Make your desired edits to the QoS limits. Note that if you deselect an "Unlimited" checkbox, a field appears in which you can enter a numerical limit. The supported QoS limit types are described below.

Storage Quota (KB) — High Limit

Storage quota limit, in number of KBs

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.
- *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.

Storage Quota Count — High Limit

Storage quota 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.
- *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.

Request Rate — Warning Limit

Request rate warning 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, an INFO level message is written to the S3 Service's application log. 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. (Note that the system does not inform the user that the warning threshold has been exceeded -- it only writes the aforementioned log message.)
- *For group QoS* — The implementation is the same as for user QoS, except that it applies to requests from all users in the group. 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.

Request Rate — High Limit

Request rate maximum, 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. 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. 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.

Data Bytes IN (KB/minute) — Warning Limit

Inbound data rate warning limit, in KBs 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 kilobytes of data.

Data Bytes IN (KB/minute) — High Limit

Inbound data rate high limit, in KBs 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 kilobytes of data. Note that if a block is triggered by the Data Bytes IN (KB) High Limit being reached, the block applies to all HTTP request types (not just PUTs.)

Data Bytes OUT (KB/minute) — Warning Limit

Outbound data rate warning limit, in KBs 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 kilobytes of data.

Data Bytes OUT (KB/minute) — High Limit

Outbound data rate high limit, in KBs 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 kilobytes of data. Note that if a block is triggered by the Data Bytes OUT (KB) High Limit being reached, the block applies to all HTTP request types (not just GETs.)

3. Click **Save**.

Deleting QoS Overrides to Restore Defaults

Each QoS configuration panel includes a **Delete Overrides** button. The table below shows what this button does in each QoS panel.

QoS Configuration Panel	What the Delete Overrides Button Does
User QoS Limits: Defaults	Deletes all group-specific default user QoS profiles. Each group's default user QoS profile will revert to the system default user QoS profile. (This action will not delete QoS overrides set at the individual user level.)
User QoS Limits: Group Defaults	Deletes all user-specific QoS profiles for users within a group. All users in the group will revert to the default user QoS profile for the group.
User QoS Limits: Overrides	Deletes a user-specific QoS profile. The user will revert to the default user QoS profile for the user's group
Group QoS Limits: Defaults	Deletes all group-specific group QoS profiles. All groups will revert to the system default group QoS profile.
Group QoS Limits: Overrides	Deletes a group-specific QoS profile. The group will revert to the system default group QoS profile.

QoS Controls in a Multi-Region HyperStore System

In a multi-region HyperStore system, QoS controls are configured and enforced separately for each region.

If you have a multi-region system, a drop-down list of all your regions will display at the top of each QoS configuration panel. Use the drop-down list to choose the region for which you are setting QoS controls. Be sure to establish QoS limits for each region, clicking **Save** before moving on to the next region.

The QoS limits that you establish for a service region will be applied only to group and user activity in that particular region. For example, if you have a two-region HyperStore deployment with regions named "West" and "East", you might set your QoS limits in the "West" region so that each user is allowed to store 20GB of data.

The 20GB storage cap applies only to user activity in your "West" region. You would also have a separate cap configured for the "East" region. For example, users might also be allowed 20GB of storage in the "East" region (for a total of 40GB per user across the entire multi-region system); or a different value might be configured for the "East" region such as 10GB or 30GB.

5.6. IAM

The **IAM** tab contains the following functions:

- [IAM User](#)
- [IAM Group](#)

Note For an overview of HyperStore's IAM feature, including limitations on the scope of HyperStore's IAM support, see "[Identity and Access Management \(IAM\) Feature Overview](#)" (page 91).

5.6.1. Manage IAM User

Path: **IAM** → **IAM User**

Supported tasks:

- ["Add an IAM User"](#) (page 255)
- Work with existing IAM users:
 - ["Retrieve an IAM User or a List of IAM Users"](#) (page 255)
 - ["Edit an IAM User's Attributes, Credentials, or Memberships"](#) (page 256)
 - ["Delete an IAM User"](#) (page 258)

5.6.1.1. Add an IAM User

- In the CMC's [Manage IAM User](#) page, click **Add New User**. This opens the panel for adding a new IAM user.

The screenshot shows a modal window titled "Manage IAM User". At the top right is a green button labeled "+ ADD NEW USER". Below it are two input fields: "User Name" (containing "User Name") and "Path" (containing "Path"). At the bottom are two buttons: "CANCEL" (gray) and "SAVE" (green).

- In the new user panel, complete the user information:

User Name (mandatory)

- Must be unique within the parent HyperStore user account.
- Only letters, numbers, dashes, and underscores are allowed.

Path (optional)

- The path is an optional way of identifying the user's location within an organizational structure. For example you might specify a path such as "/MyCompany/London/".
- Specifying a path **does not join the user into an IAM group** and has no impact on the user's privileges. It's simply a way of identifying the user's location within an organization.
- Leave this field blank if you don't want to use a path for the user.

- Click **Save**.

IMPORTANT: When you create a new IAM user, the IAM user by default has no S3 service access or permissions. You must generate an S3 access key pair for the user, and join the user into one or more of the IAM groups that you have created. The user will then inherit whatever permissions you have established in those groups' inline policies. For more information see "[Edit an IAM User's Attributes, Credentials, or Memberships](#)" (page 256).

IAM users that you create under your HyperStore user account will be restricted in these ways:

- * They will not be allowed to create other IAM users or use other IAM management functions.
- * They will not be allowed to log into the CMC. They will need to use an S3 client application other than the CMC to access the HyperStore S3 Service.

5.6.1.2. Retrieve an IAM User or a List of IAM Users

In the CMC's [Manage IAM User](#) page, by default all your IAM users will be listed in alphabetical order, with 10 users displayed at a time and an ability to navigate through the alphabetized list by using the "Previous" and "Next" links in the lower right.

You can use the "Search" field to retrieve a single IAM user or a filtered list of users. The search function is not case-sensitive.

- To retrieve just one specific user, enter the user name. Since the displayed list dynamically filters down as you type, it may be that less than the full name is sufficient to isolate the user.

The screenshot shows the 'Manage IAM User' page. At the top right is a green '+' button labeled 'ADD NEW USER'. Below it is a search bar containing 'ithinktherefore'. A table lists one user entry:

User Name	Path	Actions
IthinkTherefore-IAM	/	Delete

At the bottom, it says 'Showing 1 to 1 of 1 entries (filtered from 12 total entries)' and includes 'Previous' and 'Next' links.

- To retrieve a user list filtered by text string, enter the text. This could be a prefix or a suffix for example.

The screenshot shows the 'Manage IAM User' page with a search bar containing 'iamuser'. The table lists five users:

User Name	Path	Actions
iamUser1	/	Delete
iamUser2	/	Delete
iamUser3	/	Delete
iamUser5	/	Delete

At the bottom, it says 'Showing 1 to 4 of 4 entries (filtered from 12 total entries)' and includes 'Previous' and 'Next' links.

Note The text you enter in the Search field will be used to search on the Path as well as the User Name. For example if you enter "xyz" in the Search field, the display will show all your IAM users whose user names start with "xzy" and also all your IAM users whose paths start with "xyz".

5.6.1.3. Edit an IAM User's Attributes, Credentials, or Memberships

To edit an IAM user, first access the user's detail page:

- In the CMC's [Manage IAM User](#) page, if the user that you want to edit is not among those immediately listed in the display, retrieve the user by typing the user name in the "Search" field.
- Click on the user name to open the detail page for that user.

The screenshot shows the 'Manage IAM User' detail page for the user 'IthinkTherefore-IAM'. The top right has a green '+' button labeled 'EDIT USER'. The user details are shown in a table:

User Name	IthinkTherefore-IAM
Path	/

Below are tabs for 'IAM ACCESS KEY' (selected) and 'IAM GROUP'. A table for 'Access Key ID' is shown:

Access Key ID	Status	Secret Access Key	Actions
No data available in table			

At the bottom, it says 'Showing 0 to 0 of 0 entries' and includes 'Previous' and 'Next' links.

In this page you can:

Edit the user's name or path

Click **Edit User**, then edit the User Name and/or the Path, then click **Save**.

Note If you change the user name, the IAM user's S3 access keys and group memberships will stay with the user under his or her new name.

Manage the user's S3 access keys

When you create a new IAM user, the user does not yet have any S3 access keys. The user needs an S3 access key (access key ID and corresponding secret key) to be able to use the HyperStore S3 Service.

To create a new S3 access key for the user, click **Create New Key**. During the key creation you will be shown the secret key. **Copy the secret key and keep it in a secure location.** After the key creation completes and you leave or refresh the page, you will only be able to view the Access Key ID. You will no longer be able to view the secret key.

To make an access key inactive, in the "Actions" column for the key click **Inactivate**.

Note If an IAM user has no active access key he or she will not be able to access the HyperStore S3 Service. Deactivating all of an IAM user's keys can be a means for you to temporarily suspend the IAM user.

You can subsequently click **Activate** for at least one of the user's inactive keys, and then the user will once again have access to the S3 Service.

By system default each IAM user is allowed a maximum of two keys. (This limit is controlled by the "**credentials.iamuser.max**" (page 514) setting in *mts.properties.erb*.) Inactive keys count toward this total.

To delete an access key , in the "Actions" column for the key click **Delete**.

Manage the user's group memberships

An IAM user can be a member of one or more IAM groups that you have created. It is **only through membership in IAM groups that IAM users gain permissions** regarding S3 actions and resources (these permissions are granted to the groups by inline IAM policies that you create).

Within the user detail page click the **IAM Groups** tab.

The screenshot shows the 'Manage IAM User' interface. At the top, there is a back arrow and the title 'Manage IAM User'. Below that, the 'User Name' is listed as 'IthinkTherefore-IAM' and the 'Path' is '/'. There are two tabs: 'IAM ACCESS KEY' (which is currently selected) and 'IAM GROUPS', which is circled in red. On the right side, there is a green 'EDIT USER' button. Below the tabs, there is a search bar with the placeholder 'Search: by name or path'. A table below the search bar has columns for 'Group Name', 'Path', and 'Actions'. The table displays the message 'No data available in table'. At the bottom of the table area, it says 'Showing 0 to 0 of 0 entries' and includes 'Previous' and 'Next' buttons.

To add the user to an IAM group that you have created under your HyperStore account, click **Add to Group**, then select the group from the drop-down list that displays. The list will be limited to IAM groups that the user does not already belong to. After selecting the group click **Add**.

To remove the user from an IAM group, in the list of groups that the user belongs to, under the "Actions" column for the group click **Remove User from Group**.

5.6.1.4. Delete an IAM User

Note Before deleting an IAM user you must delete the user's S3 credentials and remove the user from any IAM groups that she is in. For instructions see "**Edit an IAM User's Attributes, Credentials, or Memberships**" (page 256).

You can only delete one IAM user at a time. The CMC does not currently support bulk deletion of users.

Note Deleting an IAM user will not delete the user's stored S3 buckets and objects from the system. The S3 data is considered to belong to the parent HyperStore user account.

1. In the CMC's [Manage IAM User](#) page, if the user that you want to delete is not among those immediately listed in the display, retrieve the user by typing the user name in the "Search" field.
2. In the "Actions" column for the user that you want to delete, click **Delete**.
3. When prompted by the system, confirm that you want to delete the user.

5.6.2. Manage IAM Group

Path: **IAM** → **IAM Group**

Supported tasks:

- **"Add an IAM Group"** (page 259)
- Work with existing IAM groups:
 - **"Retrieve an IAM Group or a List of IAM Groups"** (page 260)
 - **"Edit an IAM Group's Attributes or Membership"** (page 260)
 - **"Create and Manage Inline Policies for an IAM Group"** (page 261)
 - **"Delete an IAM Group"** (page 263)

5.6.2.1. Add an IAM Group

1. In the CMC's [Manage IAM Group](#) page, click **Add New Group**. This opens the panel for adding a new IAM group.

2. In the new group panel, complete the group information:

Group Name (mandatory)

- Must be unique within the parent HyperStore user account.
- Only letters, numbers, dashes, and underscores are allowed.

Path (optional)

- The path is an optional way of identifying the group's location within an organizational structure. For example if the group name is "Quality_Assurance" you might specify a path such as "/MyCompany/Engineering/" (if the QA group is part of the Engineering division of your company).

- The path has no impact on group privileges or on which users can be put in the group. It's simply a way to help you identify the group within a broader organizational context, if you choose to do so.
- Leave this field blank if you don't want to use a path for the group.

3. Click **Save**.

5.6.2.2. Retrieve an IAM Group or a List of IAM Groups

In the CMC's [Manage IAM Group](#) page by default all your IAM groups will be listed in alphabetical order, with 10 groups displayed at a time and an ability to navigate through the alphabetized list by using the "Previous" and "Next" links in the lower right.

You can use the "Search" field to retrieve a single IAM group or a filtered list of groups. The search function is not case-sensitive.

- To retrieve just one specific group, enter the group name. Since the displayed list dynamically filters down as you type, it may be that less than the full name is sufficient to isolate the group.

The screenshot shows the 'Manage IAM Group' page with a single entry listed:

Group Name	Path	Actions
WattaFool-IAM	/	

Below the table, the status message reads: "Showing 1 to 1 of 1 entries (filtered from 14 total entries)".

- To retrieve a group list filtered by text string, enter the text. This could be a prefix or a suffix for example.

The screenshot shows the 'Manage IAM Group' page with three entries listed, filtered by the search term 'cloudian':

Group Name	Path	Actions
Cloudian1	/	
Cloudian2	/	
Cloudian3	/	

Below the table, the status message reads: "Showing 1 to 3 of 3 entries (filtered from 16 total entries)".

Note The text you enter in the Search field will be used to search on the Path as well as the Group Name. For example if you enter "abc" in the Search field, the display will show all your IAM groups whose group names start with "abc" and also all your IAM groups whose paths start with "abc".

5.6.2.3. Edit an IAM Group's Attributes or Membership

To edit an IAM group's attribute or its membership (the list of IAM users who belong to the group), first access the group's detail page:

1. In the CMC's [Manage IAM Group](#) page, if the group that you want to work with is not among those immediately listed in the display, retrieve the group by typing the group name in the "Search" field.
2. Click on the group name to open the detail page for that group.

The screenshot shows the 'Manage IAM Group' interface. At the top, there is a back arrow and the title 'Manage IAM Group'. Below that, the 'Group Name' is set to 'WattaFool-IAM' and the 'Path' is '/'. There are two tabs: 'IAM POLICY' (disabled) and 'IAM USER' (selected). On the right, there is a green '+ EDIT GROUP' button. Below the tabs, there is a search bar with 'Show 10 entries' and a 'Search' placeholder. A table follows, with columns for 'Policy Name', 'Policy Document', and 'Actions'. The table header says 'No data available in table'. At the bottom, it says 'Showing 0 to 0 of 0 entries' and has 'Previous' and 'Next' links.

In this page you can:

Edit the group's name or path

Click **Edit Group**, then edit the Group Name or the Path, then click **Save**.

Note If you change the group name, the IAM group's inline policies and its user members will stay with the group under the new name.

Manage the group's membership

Within the group detail page click the **IAM Users** tab. To add an existing IAM user to the group, click **Add IAM User**, then select the user from the drop-down list. The alphabetically ordered drop-down list will be limited to IAM users who you have created under your HyperStore account and who do not already belong to the group. After selecting the user click **Add**.

To remove a user from the IAM group, with the group detail page's **IAM Users** tab selected look through the alphabetized list of group members that displays at the bottom of the page. If the user who you want to delete is not displayed among the first batch of users, browse forward to where the user is in the alphabetized list (using the "Next" link) or use the "Search" box to retrieve the user by name or path. When you have the user displayed in the list, under the "Actions" column for the user click **Remove from Group**.

Note In this page you can also create and manage inline policies for the group -- this is covered separately in "[Create and Manage Inline Policies for an IAM Group](#)" (page 261).

5.6.2.4. Create and Manage Inline Policies for an IAM Group

The CMC supports creating one or more inline policies for an IAM group. Note however that to do so you will need to enter a JSON-formatted policy statement into a field in the GUI. The CMC does not currently support a

more highly automated or GUI-driven way of constructing IAM policies.

IAM users who belong to an IAM group are granted the permissions defined by the group's inline IAM policies. **In the CMC, this is the only mechanism by which you can grant service permissions to your IAM users.** In the CMC, newly created IAM users have no service permissions until you join them into an IAM group for which you have embedded one or more inline policies.

To create an inline policy for an IAM group, or manage existing inline policies that you've already embedded with the group, first access the group's detail page:

1. In the CMC's [Manage IAM Group](#) page, if the group that you want to work with is not among those immediately listed in the display, retrieve the group by typing the group name in the "Search" field.
2. Click on the group name to open the detail page for that group.

The screenshot shows the 'Manage IAM Group' page with the 'IAM POLICY' tab selected. The page displays a table with columns for 'Policy Name' and 'Policy Document'. A message at the top right indicates 'No data available in table'. At the bottom, it says 'Showing 0 to 0 of 0 entries' and includes 'Previous' and 'Next' navigation links. There are also 'EDIT GROUP' and '+ ADD IAM POLICY' buttons.

3. With the **IAM Policy** tab selected click **Add IAM Policy**. This displays the IAM Policy configuration panel.

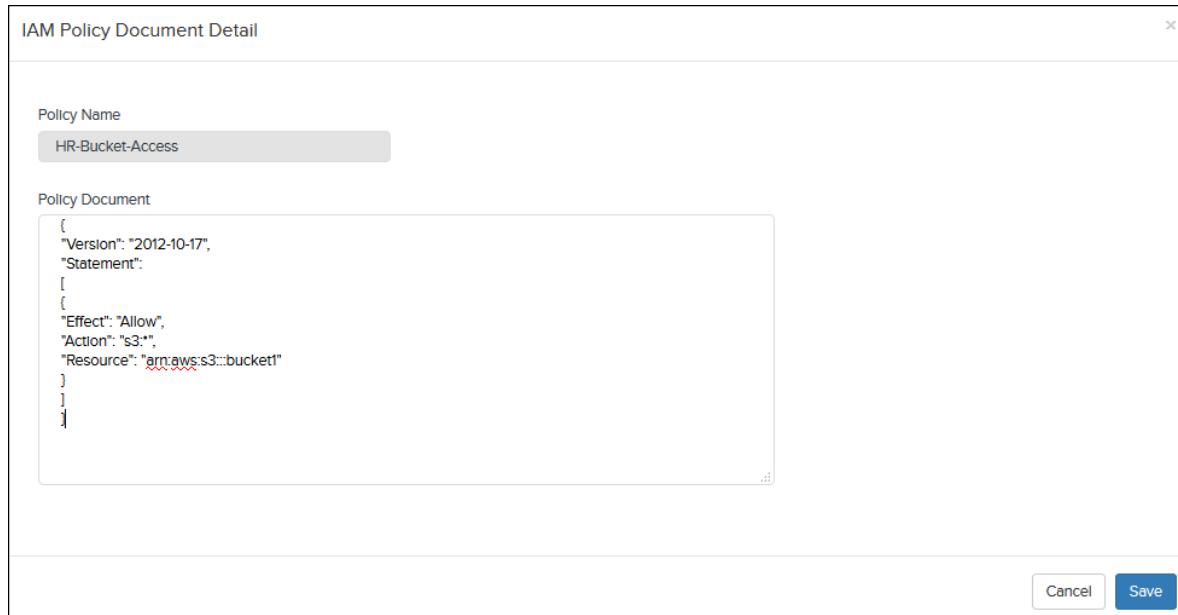
The screenshot shows the 'Manage IAM Group' page with the 'IAM POLICY' tab selected. A red circle highlights the 'IAM POLICY' tab. Another red circle highlights the '+ ADD IAM POLICY' button. The configuration panel includes fields for 'Policy Name' (with placeholder 'Name of the IAM Policy') and 'Policy Document' (with placeholder 'Detail of the IAM Policy Document'). At the bottom are 'CANCEL' and 'ADD' buttons.

4. In the "Policy Name" field enter a descriptive name for the policy. Then in the "Policy Document" field, enter a JSON-formatted IAM policy definition. For guidance on writing a policy document see "**IAM Supported Policy Document Elements**" (page 966). After entering the policy definition click **Add**. The

CMC validates your policy formatting, and if the formatting is valid the policy is embedded with the group.

Editing an Inline Policy

In the IAM group detail page, with the **IAM Policy** tab selected, a list of the group's existing inline IAM policies displays at the bottom of the page. To edit a policy, to the right of the policy name click **View Document**. The existing policy document will be shown in the policy configuration page.



Make your edits to the policy document, then click **Save**. The CMC validates your policy formatting, and if the formatting is valid the edited policy is embedded with the group.

The group's member users' permissions will automatically be updated in accordance with your policy edits.

Deleting an Inline Policy

In the IAM group detail page, with the **IAM Policy** tab selected, a list of the group's existing inline IAM policies displays at the bottom of the page. To delete a policy, in the policy's **Actions** column click **Remove**.

When you remove an inline policy from an IAM group, the group's member users lose whatever permissions had been defined in the removed policy.

5.6.2.5. Delete an IAM Group

Note You cannot delete an IAM group that currently has IAM users in it. You must delete the IAM users first (for instructions see **"Delete an IAM User"** (page 258)). Also you must delete any inline policies configured for the group (for instructions see **"Create and Manage Inline Policies for an IAM Group"** (page 261)). After deleting all of the group's IAM users and inline policies you can delete the IAM group.

1. In the CMC's [Manage IAM Group](#) page, if the group that you want to delete is not among those immediately listed in the display, retrieve the group by typing the group name in the "Search" field.
2. In the "Actions" column for the group that you want to delete, click **Delete**.
3. When prompted by the system, confirm that you want to delete the group.

5.7. Cluster

The **Cluster** tab contains the following functions:

- [Data Centers](#)
- [Node Status](#)
- [Node Activity](#)
- [Node Advanced](#)
- [Cluster Information](#)
- [Configuration Settings](#)
- [Storage Policies](#)
- [Repair Status](#)
- [Operation Status](#)

5.7.1. Data Centers

Path: **Cluster** → **Data Centers**

Supported tasks:

- View Status of All Nodes in a Data Center (below)
- Adding Nodes — For the full procedure including important steps to take before you add nodes, see "[Adding Nodes](#)" (page 369).
- Adding a Data Center — For the full procedure including important steps to take before you add a data center, see "[Adding a Data Center](#)" (page 379).
- Adding a Region — For the full procedure including important steps to take before you add a region, see "[Adding a Region](#)" (page 385).

5.7.1.1. View Status of All Nodes in a Data Center

The upper part of the **Data Centers** page displays a panel for each data center in your HyperStore system. For each data center, each HyperStore node in the data center is represented by a color-coded cube:



Red with question mark indicates that the **node is unreachable** due to a network problem.



Red with "X" indicates that the **node has experienced disk errors**. Click the node icon to jump to the **Node Status** page for the node, then check that page's [Disk Detail Info](#) panel for more information.



Red with disk stack indicates the node has **stopped accepting writes** -- and the system has stopped directing S3 write requests to that node -- because all of the node's disks are nearly full. For more information on this "stop-write" condition and how to remedy it see "**Automatic Stop of Writes to a Node at 90% Usage**" (page 86).

Orange with disk stack indicates the **node's data disks are in aggregate more than 80% full**.

Click the node icon to jump to the **Node Status** page for the node, then check that page's [Disk Detail Info](#) panel for more information.



Note For this status "more than 80% full" means that more than 80% of the node's total capacity is either used or "reserved". For more information on "reserved" capacity see "**Capacity Managed**" (page 172).



Orange with exclamation mark indicates that the **node has Medium, High, or Critical level alerts that have not yet been acknowledged** by a system administrator. This status will not display if a node has only Low level alerts. Click the node icon to jump to the **Node Status** page for the node, then at the bottom of that page see the [Alert List](#) panel for more information.



Blue with gear indicates that the **node is under maintenance**. This means either that you or another administrator put the node into maintenance mode (see "**Start Maintenance Mode**" (page 285)) in which case the node is not currently supporting S3 writes or reads.



Green with check mark indicates that the **node has no unacknowledged alerts** nor any of the conditions listed above.

Note If multiple of the above conditions apply to one node, that node's icon will reflect just the highest priority condition, with the conditions prioritized in this order: "Under Maintenance" > "Unreachable" > "Has Disk Error" > "Disks Above 80% Full" > "Has Alerts".

To view a node's hostname and summary node statistics, hold your cursor over a cube.

The hover text shows the same summary node status information as is available on the upper part of the CMC's **Node Status** page. For description of the status items, see "[View a Node's Summary Status](#)" (page 270) from the **Node Status** page documentation.

To check the status of individual services on every node in a data center, in the lower part of the **Data Centers** page view the **Services Status** panel.

For each service on each node, one of the following service status icons is displayed:



The service is **up and running**.



The service is **down**.



The service status is unknown because the node is **unreachable** due to a network problem.

Note The service statuses are automatically checked and updated each minute.

Status is shown for the following services:

- [Admin Service](#)
- [IAM Service](#) (if enabled)
- [Cassandra](#)
- [HyperStore Service](#)
- [Redis Monitor](#)
- [Redis Credentials](#)
- [Redis QoS](#)
- [S3 Service](#)

For Redis Monitor, Redis Credentials, and Redis QoS, the service status icon includes a red asterisk (*) if it is the master or primary instance of the service. For Redis Credentials or QoS, if the master instance of the service goes down, one of the slave instances becomes the new master (and that new master instance will be marked with an asterisk in the display). By contrast, if the Redis Monitor primary instance goes down, the backup instance takes over the Redis monitoring duties but it does not become the primary instance (and therefore the backup instance will not be marked with an asterisk -- instead the asterisk remains attached to the primary instance even though it's down).

5.7.2. Node Status

Path: **Cluster** → **Nodes** → **Node Status**

The screenshot shows the Cloudian Node Status interface. At the top, there are several tabs: Analytics, Buckets & Objects, Users & Groups, Cluster (highlighted with a red circle), Nodes (highlighted with a red circle), Cluster Config, Storage Policies, Repair Status, and Operation Status. Below the tabs, there are three main sections: NODE STATUS, NODE ACTIVITY, and ADVANCED. The NODE STATUS section is selected. It includes a Host dropdown set to 'DC1 RACK1 cloudian-1', a NODE CAPACITY USAGE donut chart showing 'TOTAL 1.9 TB' with segments for USED (green), RESERVED (yellow), and FREE (blue), and a CPU UTILIZATION % bar chart showing '30'. To the right, there are tables for NETWORK TRAFFIC and REQUEST THROUGHPUT. The NETWORK TRAFFIC table shows RECEIVED: 36.4 KB/SEC and TRANSMITTED: 41.6 KB/SEC. The REQUEST THROUGHPUT table shows GET: 0 and PUT: 0. Below these are sections for DISK DETAIL INFO (listing /dev/sda3 as mounted at /) and MEMORY USAGE. The SERVICES STATUS section lists various services with their status (e.g., Admin, Cassandra, HyperStore, Redis Mon, Redis Cred, Redis Qos, S3) and provides RESTART and STOP buttons. At the bottom, the ALERT LIST shows two entries: one for Redis Qos with severity Medium and another for Redis Qos with severity Medium. A 'RESTART ALL' button is located at the bottom right of the services section.

Supported tasks:

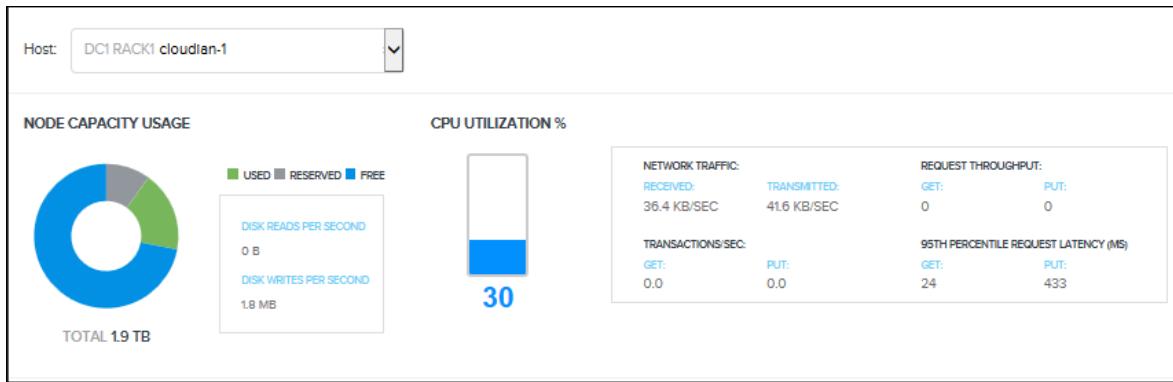
- "View a Node's Summary Status" (page 270)
- "View a Node's Disk Detail" (page 272)
- "Locate a Disk on a HyperStore Appliance" (page 274)
- "View a Node's Memory Usage" (page 276)

- **"View a Node's Services Status"** (page 277)
- **"Start, Stop, or Restart Services On a Node"** (page 277)
- **"View and Acknowledge Node Alerts"** (page 278)

Note If your HyperStore system has multiple service regions, a drop-down list displays at the top of the page so you can select a region first before selecting a node for which to view status.

5.7.2.1. View a Node's Summary Status

The upper part of the CMC's [Node Status](#) page provides a dashboard view of the health and performance of an individual node within your HyperStore system. A drop-down list lets you choose a node for which to display information.



For the selected node, the **Node Status** page displays current information for the status and performance items described below.

Note The **Node Status** page — and the status and performance information displayed on the page — automatically refreshes once each minute. In the event that the node cannot be reached by the HyperStore Monitoring Data Collector due to a network problem, the top of the **Node Status** page displays a message saying "Node is not reachable. Information on this page may not be accurate."

Node Capacity Usage

The **Node Capacity Usage** graphic shows the percentages of total disk space that are currently Used, Reserved, or Free, on the node as a whole. To see the percentage numbers hold your cursor over each portion of the tri-colored circle.

- The **Used** segment of the circle indicates what portion of the node's total disk capacity is currently consumed by stored object data. This segment displays in **green** if less than 70% of total capacity is used; or in **orange** if from 70% to 89% is used; or in **red** if 90% or more is used.
- The **Reserved** segment indicates the portion of the node's total disk capacity that is reserved and cannot be used for data storage. The Reserved portion consists of the Linux "reserved blocks percentage" plus the HyperStore stop-write buffer.
 - By default in CentOS/RHEL the "reserved blocks percentage" for a file system (the portion of the disk space that's reserved for privileged processes) is 5% of disk capacity. In a HyperStore

Appliance it's customized to 0%. See your OS documentation if you want to change the current reserved blocks percentage for your HyperStore host machines.

- By default the HyperStore stop-write buffer is 10% of disk capacity. For information on this feature see "**Automatic Stop of Writes to a Disk at 90% Usage**" (page 85).

The Reserved segment always displays in **gray**.

- The **Free** segment indicates the portion of the node's total disk capacity that is neither used nor reserved, and is therefore available for storing new data. The Free segment always displays in **blue**.

IMPORTANT: See "**Cluster Resizing Feature Overview**" (page 68) for guidance about capacity management and the importance of early planning for cluster expansions.

Disk Reads per second

Across **all** of the node's disks, the average disk read throughput per second during the past minute. The metric will auto-scale from B (bytes) to KB or MB or GB as appropriate.

This system stat is recalculated each minute, based on the most recent minute of data.

Disk Writes per second

Across **all** of the node's disks, the average disk write throughput per second during the past minute. The metric will auto-scale from B (bytes) to KB or MB or GB as appropriate.

This system stat is recalculated each minute, based on the most recent minute of data.

CPU Utilization %

Current CPU utilization on the node. In the icon that graphically shows disk usage percentage, the disk used portion of the icon displays in green if disk usage is less than 70%; in yellow if usage is between 70% and 89%; or in red if usage is 90% or higher.

Network Traffic

The aggregate network interface throughput (received and transmitted) for the node during the past minute, for all types of network traffic including but not limited to S3 request traffic. For example, data transmission associated with cluster maintenance operations would count toward these statistics. The metric will auto-scale from B (bytes) to KB or MB or GB as appropriate.

These system stats are recalculated each minute, based on the most recent minute of data.

Request Throughput

For just this node, the data throughput for S3 GET transactions and S3 PUT transactions, expressed as MB per second.

These S3 stats are recalculated each five minutes, based on the most recent five minutes of S3 transaction activity.

Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

Transactions/sec

For just this node, the number of S3 GET transactions and S3 PUT transactions processed per second.

These S3 stats are recalculated each five minutes, based on the most recent five minutes of S3 transaction activity.

Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

95th Percentile Request Latency (ms)

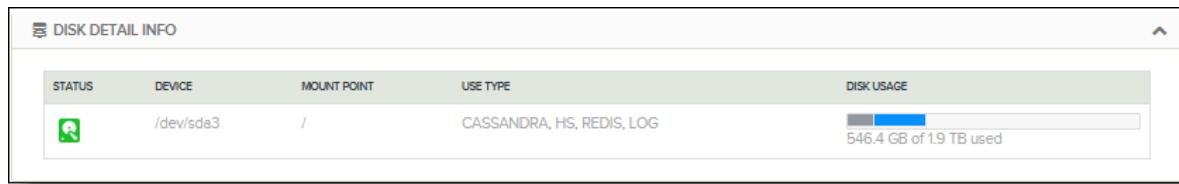
For just this node, the 95th percentile latencies for S3 PUT and S3 GET transactions in milliseconds.

These S3 stats are recalculated each five minutes, based on the most recent approximately 1000 GET transactions and 1000 PUT transactions. Each 95th percentile latency value indicates that of the last 1000 transactions of that type, 95% completed in that many milliseconds or less.

Note HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

5.7.2.2. View a Node's Disk Detail

To check the status of a node's disks, on the CMC's [Node Status](#) page open the **Disk Detail Info** panel.



The panel displays the information described below.

Status

This field displays an icon that indicates the disk's summary status.

Icon	Meaning
	OK — The green disk icon indicates that there are no errors for the disk.
	Error — The orange disk icon indicates that the disk is in an error state. For a HyperStore data ("HS") disk, an Error status indicates that read/write errors related to this disk are appearing in the HyperStore Service application log (<code>cloudian-hyperstore.log</code>), but not in excess of the configurable error rate threshold that would trigger the automatic disabling of the disk (<code>hyperstore-server.properties.erb: "disk.fail.error.count.threshold"</code> (page 494)). For a Cassandra disk, Error status is triggered if <code>cassandra.log</code> messages appear that include the string "No space left on device". If a disk is in an Error state, a Clear Error History button appears toward the bottom of the Detail Disk Info panel. You can click this button to clear the disk's Error status and return the disk status to "OK". If the error occurs again, the disk status display will revert back to showing an Error status for the disk.

Icon	Meaning
	<p>Note A disk will also show as being in an Error status if you unmount the disk without the disk having first been marked as disabled by the HyperStore system.</p>
	<p>The red disk icon means one of two things depending on whether the disk is a HyperStore data disk or an OS disk:</p> <ul style="list-style-type: none"> • Disabled (data disk) — For a data disk the red icon indicates that the HyperStore system has disabled the disk (so HyperStore doesn't try to write to or read from it any more), unmounted the disk, and transferred all of the disk's tokens to other disks on the host. A data disk can be put into Disabled status in either of two ways: <ul style="list-style-type: none"> ◦ You or another administrator disabled the disk using the HyperStore <i>disableDisk</i> function. For background information see ""Disabling a HyperStore Data Disk" (page 427). ◦ The HyperStore system detected indications of disk failure and automatically disabled the disk, in accordance with the HyperStore Disk Failure Action configuration setting. <p>Note For recovering from a disabled disk, see ""Enabling a HyperStore Data Disk" (page 428) or -- if the disk is bad -- ""Replacing a HyperStore Data Disk" (page 430).</p> <ul style="list-style-type: none"> • Disk Failure Under RAID-1 (OS disk) — For an OS disk on a HyperStore Appliance node, the red icon indicates that one of the two mirrored disks storing the OS and system metadata has failed and been marked as Offline by the RAID controller. This status detection and reporting is supported only for HyperStore Appliance machines, not for software-only deployments of HyperStore. <p>Note If this occurs, the disk will also be marked by a red exclamation mark in the appliance disk map.</p>

Device

Disk drive device.

Mount Point

File system mount point for each disk.

Use Type

For each disk, this will be one or more of the following use types:

- "HS" — The disk is storing the HyperStore File System (HSFS). The HSFS is where S3 object data is stored (in the form of object replicas and/or erasure coded fragments). Disks with mount points specified by the configuration file setting *common.csv:hyperstore_data_directory* are HS disks. In a typical configuration there will be multiple HS disks on each node.

- "Cassandra" — The disk is storing Cassandra data (the directory specified by the configuration file setting *common.csv*: *cassandra_data_directory*) and/or the Cassandra commit log (the directory specified by the configuration file setting *common.csv*: *cassandra_commit_log_directory*).
- "Redis" — The disk is storing Redis data (the directory specified by the configuration file setting *common.csv*: *redis_lib_directory*).
- "Log" — The disk is storing application logs (the directories specified by the *common.csv* settings *cloudian_log_directory*, *cassandra_log_directory*, and *redis_log_directory*).
- "UNAVAIL" -- The disk is in an error status due to having been unmounted, or is in a disabled or failed status. See the description of Status above.

Disk Usage

This colored bar graphic indicates what portion of the disk's capacity is used (in **blue**) and what portion is reserved (in **gray**). For more information about "reserved" capacity see "**Node Capacity Usage**" (page 270)

The blue portion of the graphic turns to:

- **Orange** if the reserved space and used space together consume 70 percent or more of the disk's total capacity (but less than 90 percent of total capacity)
- **Red** if the reserved space and used space together consume 90 percent or more of the disk's total capacity.

Note Below the bar graphic, in the summary text display that says for example "X GB of Y TB used", the reserved capacity is counted as "used".

5.7.2.3. Locate a Disk on a HyperStore Appliance

The **Appliance** section of the CMC's **Node Status** page is available only if you are using a HyperStore 1500 series, HyperStore 4000 series, HyperStore Extreme, or Lenovo Storage DX8200C appliance. In the event of a disk problem you can open the **Appliance** section for information and tools to help you locate the disk within your hardware environment.

The **Appliance** panel displays a drive list that shows the drive usage type ("OS" for drives that store the OS and metadata, or "Data" for drives that storage object data), device name, serial number, and slot number of the drive on the appliance. For each listed drive there is also a button that you can use to blink the ID light of the drive on the appliance. First look through the drive list to find the device name that you're looking for, then click the blink button for that drive. The **drive's light will blink on the appliance for three minutes** so you can view the appliance and physically locate the drive.

If you want help finding the appliance within a rack or cage you can click **Blink Chassis** in the upper part of the **Appliance** section and the appliance's chassis light will blink for three minutes.

Note If a red exclamation mark appears beside a device name in the drive list, this means that the device is offline (not running). A script monitors the /sys directory on the appliance to detect devices that have gone offline.

The screen shot below shows the **Appliance** section for a HyperStore 4000 appliance.

APPLIANCE

Model	HSA-4008	Serial Number	QTFCOU717013B																																																								
Front View		Back View																																																									
<table border="1"> <thead> <tr> <th>TYPE</th> <th>BLOCK DEVICE NAME</th> <th>LOCATION</th> <th>SERIAL NO.</th> <th>SLOT NO.</th> <th>BLINK</th> </tr> </thead> <tbody> <tr> <td>OS</td> <td>/dev/sda</td> <td>Internal</td> <td>BTDV714407RN800CGN</td> <td>port_5</td> <td></td> </tr> <tr> <td>OS</td> <td>/dev/sdb</td> <td>Internal</td> <td>BTDV712203BV800CGN</td> <td>port_6</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdc</td> <td>Internal</td> <td>ZA168KGD</td> <td>0</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdd</td> <td>Internal</td> <td>ZA168NZK</td> <td>1</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sde</td> <td>Internal</td> <td>ZA16MG0J</td> <td>2</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdf</td> <td>Internal</td> <td>ZA16GSQW</td> <td>3</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdg</td> <td>Internal</td> <td>ZA16MEBM</td> <td>4</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdh</td> <td>Internal</td> <td>ZA16MZ6Z</td> <td>5</td> <td></td> </tr> </tbody> </table>						TYPE	BLOCK DEVICE NAME	LOCATION	SERIAL NO.	SLOT NO.	BLINK	OS	/dev/sda	Internal	BTDV714407RN800CGN	port_5		OS	/dev/sdb	Internal	BTDV712203BV800CGN	port_6		Data	/dev/sdc	Internal	ZA168KGD	0		Data	/dev/sdd	Internal	ZA168NZK	1		Data	/dev/sde	Internal	ZA16MG0J	2		Data	/dev/sdf	Internal	ZA16GSQW	3		Data	/dev/sdg	Internal	ZA16MEBM	4		Data	/dev/sdh	Internal	ZA16MZ6Z	5	
TYPE	BLOCK DEVICE NAME	LOCATION	SERIAL NO.	SLOT NO.	BLINK																																																						
OS	/dev/sda	Internal	BTDV714407RN800CGN	port_5																																																							
OS	/dev/sdb	Internal	BTDV712203BV800CGN	port_6																																																							
Data	/dev/sdc	Internal	ZA168KGD	0																																																							
Data	/dev/sdd	Internal	ZA168NZK	1																																																							
Data	/dev/sde	Internal	ZA16MG0J	2																																																							
Data	/dev/sdf	Internal	ZA16GSQW	3																																																							
Data	/dev/sdg	Internal	ZA16MEBM	4																																																							
Data	/dev/sdh	Internal	ZA16MZ6Z	5																																																							

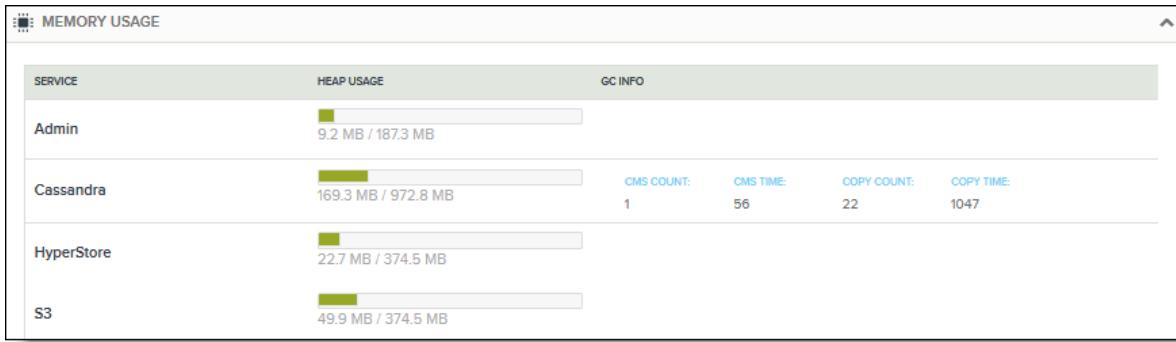
The next screen shot below shows the **Appliance** section for a HyperStore Extreme appliance.

APPLIANCE

Model	HSX-4516	Serial Number	SGFTJ18433CC2D4																																																																																																																																																																																																
<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Blink Chassis <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>Disk 96</td><td>Disk 12</td><td>Disk 24</td><td>Disk 36</td><td>Disk 48</td><td>Disk 60</td><td>Disk 72</td><td>Disk 84</td></tr> <tr><td>Disk 1</td><td>Disk 13</td><td>Disk 25</td><td>Disk 37</td><td>Disk 49</td><td>Disk 61</td><td>Disk 73</td><td>Disk 85</td></tr> <tr><td>Disk 2</td><td>Disk 14</td><td>Disk 26</td><td>Disk 38</td><td>Disk 50</td><td>Disk 62</td><td>Disk 74</td><td>Disk 86</td></tr> <tr><td>Disk 3</td><td>Disk 15</td><td>Disk 27</td><td>Disk 39</td><td>Disk 51</td><td>Disk 63</td><td>Disk 75</td><td>Disk 87</td></tr> <tr><td>Disk 4</td><td>Disk 16</td><td>Disk 28</td><td>Disk 40</td><td>Disk 52</td><td>Disk 64</td><td>Disk 76</td><td>Disk 88</td></tr> <tr><td>Disk 5</td><td>Disk 17</td><td>Disk 29</td><td>Disk 41</td><td>Disk 53</td><td>Disk 65</td><td>Disk 77</td><td>Disk 89</td></tr> <tr><td>Disk 6</td><td>Disk 18</td><td>Disk 30</td><td>Disk 42</td><td>Disk 54</td><td>Disk 66</td><td>Disk 78</td><td>Disk 90</td></tr> <tr><td>Disk 7</td><td>Disk 19</td><td>Disk 31</td><td>Disk 43</td><td>Disk 55</td><td>Disk 67</td><td>Disk 79</td><td>Disk 91</td></tr> <tr><td>Disk 8</td><td>Disk 20</td><td>Disk 32</td><td>Disk 44</td><td>Disk 56</td><td>Disk 68</td><td>Disk 80</td><td>Disk 92</td></tr> <tr><td>Disk 9</td><td>Disk 21</td><td>Disk 33</td><td>Disk 45</td><td>Disk 57</td><td>Disk 69</td><td>Disk 81</td><td>Disk 93</td></tr> <tr><td>Disk 10</td><td>Disk 22</td><td>Disk 34</td><td>Disk 46</td><td>Disk 58</td><td>Disk 70</td><td>Disk 82</td><td>Disk 94</td></tr> <tr><td>Disk 11</td><td>Disk 23</td><td>Disk 35</td><td>Disk 47</td><td>Disk 59</td><td>Disk 71</td><td>Disk 83</td><td>Disk 95</td></tr> </table> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Enclosure Front <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>Disk 96</td><td>Disk 12</td><td>Disk 24</td><td>Disk 36</td><td>Disk 48</td><td>Disk 60</td><td>Disk 72</td><td>Disk 84</td></tr> <tr><td>Disk 1</td><td>Disk 13</td><td>Disk 25</td><td>Disk 37</td><td>Disk 49</td><td>Disk 61</td><td>Disk 73</td><td>Disk 85</td></tr> <tr><td>Disk 2</td><td>Disk 14</td><td>Disk 26</td><td>Disk 38</td><td>Disk 50</td><td>Disk 62</td><td>Disk 74</td><td>Disk 86</td></tr> <tr><td>Disk 3</td><td>Disk 15</td><td>Disk 27</td><td>Disk 39</td><td>Disk 51</td><td>Disk 63</td><td>Disk 75</td><td>Disk 87</td></tr> <tr><td>Disk 4</td><td>Disk 16</td><td>Disk 28</td><td>Disk 40</td><td>Disk 52</td><td>Disk 64</td><td>Disk 76</td><td>Disk 88</td></tr> <tr><td>Disk 5</td><td>Disk 17</td><td>Disk 29</td><td>Disk 41</td><td>Disk 53</td><td>Disk 65</td><td>Disk 77</td><td>Disk 89</td></tr> <tr><td>Disk 6</td><td>Disk 18</td><td>Disk 30</td><td>Disk 42</td><td>Disk 54</td><td>Disk 66</td><td>Disk 78</td><td>Disk 90</td></tr> <tr><td>Disk 7</td><td>Disk 19</td><td>Disk 31</td><td>Disk 43</td><td>Disk 55</td><td>Disk 67</td><td>Disk 79</td><td>Disk 91</td></tr> <tr><td>Disk 8</td><td>Disk 20</td><td>Disk 32</td><td>Disk 44</td><td>Disk 56</td><td>Disk 68</td><td>Disk 80</td><td>Disk 92</td></tr> <tr><td>Disk 9</td><td>Disk 21</td><td>Disk 33</td><td>Disk 45</td><td>Disk 57</td><td>Disk 69</td><td>Disk 81</td><td>Disk 93</td></tr> <tr><td>Disk 10</td><td>Disk 22</td><td>Disk 34</td><td>Disk 46</td><td>Disk 58</td><td>Disk 70</td><td>Disk 82</td><td>Disk 94</td></tr> <tr><td>Disk 11</td><td>Disk 23</td><td>Disk 35</td><td>Disk 47</td><td>Disk 59</td><td>Disk 71</td><td>Disk 83</td><td>Disk 95</td></tr> </table> </div>				Disk 96	Disk 12	Disk 24	Disk 36	Disk 48	Disk 60	Disk 72	Disk 84	Disk 1	Disk 13	Disk 25	Disk 37	Disk 49	Disk 61	Disk 73	Disk 85	Disk 2	Disk 14	Disk 26	Disk 38	Disk 50	Disk 62	Disk 74	Disk 86	Disk 3	Disk 15	Disk 27	Disk 39	Disk 51	Disk 63	Disk 75	Disk 87	Disk 4	Disk 16	Disk 28	Disk 40	Disk 52	Disk 64	Disk 76	Disk 88	Disk 5	Disk 17	Disk 29	Disk 41	Disk 53	Disk 65	Disk 77	Disk 89	Disk 6	Disk 18	Disk 30	Disk 42	Disk 54	Disk 66	Disk 78	Disk 90	Disk 7	Disk 19	Disk 31	Disk 43	Disk 55	Disk 67	Disk 79	Disk 91	Disk 8	Disk 20	Disk 32	Disk 44	Disk 56	Disk 68	Disk 80	Disk 92	Disk 9	Disk 21	Disk 33	Disk 45	Disk 57	Disk 69	Disk 81	Disk 93	Disk 10	Disk 22	Disk 34	Disk 46	Disk 58	Disk 70	Disk 82	Disk 94	Disk 11	Disk 23	Disk 35	Disk 47	Disk 59	Disk 71	Disk 83	Disk 95	Disk 96	Disk 12	Disk 24	Disk 36	Disk 48	Disk 60	Disk 72	Disk 84	Disk 1	Disk 13	Disk 25	Disk 37	Disk 49	Disk 61	Disk 73	Disk 85	Disk 2	Disk 14	Disk 26	Disk 38	Disk 50	Disk 62	Disk 74	Disk 86	Disk 3	Disk 15	Disk 27	Disk 39	Disk 51	Disk 63	Disk 75	Disk 87	Disk 4	Disk 16	Disk 28	Disk 40	Disk 52	Disk 64	Disk 76	Disk 88	Disk 5	Disk 17	Disk 29	Disk 41	Disk 53	Disk 65	Disk 77	Disk 89	Disk 6	Disk 18	Disk 30	Disk 42	Disk 54	Disk 66	Disk 78	Disk 90	Disk 7	Disk 19	Disk 31	Disk 43	Disk 55	Disk 67	Disk 79	Disk 91	Disk 8	Disk 20	Disk 32	Disk 44	Disk 56	Disk 68	Disk 80	Disk 92	Disk 9	Disk 21	Disk 33	Disk 45	Disk 57	Disk 69	Disk 81	Disk 93	Disk 10	Disk 22	Disk 34	Disk 46	Disk 58	Disk 70	Disk 82	Disk 94	Disk 11	Disk 23	Disk 35	Disk 47	Disk 59	Disk 71	Disk 83	Disk 95
Disk 96	Disk 12	Disk 24	Disk 36	Disk 48	Disk 60	Disk 72	Disk 84																																																																																																																																																																																												
Disk 1	Disk 13	Disk 25	Disk 37	Disk 49	Disk 61	Disk 73	Disk 85																																																																																																																																																																																												
Disk 2	Disk 14	Disk 26	Disk 38	Disk 50	Disk 62	Disk 74	Disk 86																																																																																																																																																																																												
Disk 3	Disk 15	Disk 27	Disk 39	Disk 51	Disk 63	Disk 75	Disk 87																																																																																																																																																																																												
Disk 4	Disk 16	Disk 28	Disk 40	Disk 52	Disk 64	Disk 76	Disk 88																																																																																																																																																																																												
Disk 5	Disk 17	Disk 29	Disk 41	Disk 53	Disk 65	Disk 77	Disk 89																																																																																																																																																																																												
Disk 6	Disk 18	Disk 30	Disk 42	Disk 54	Disk 66	Disk 78	Disk 90																																																																																																																																																																																												
Disk 7	Disk 19	Disk 31	Disk 43	Disk 55	Disk 67	Disk 79	Disk 91																																																																																																																																																																																												
Disk 8	Disk 20	Disk 32	Disk 44	Disk 56	Disk 68	Disk 80	Disk 92																																																																																																																																																																																												
Disk 9	Disk 21	Disk 33	Disk 45	Disk 57	Disk 69	Disk 81	Disk 93																																																																																																																																																																																												
Disk 10	Disk 22	Disk 34	Disk 46	Disk 58	Disk 70	Disk 82	Disk 94																																																																																																																																																																																												
Disk 11	Disk 23	Disk 35	Disk 47	Disk 59	Disk 71	Disk 83	Disk 95																																																																																																																																																																																												
Disk 96	Disk 12	Disk 24	Disk 36	Disk 48	Disk 60	Disk 72	Disk 84																																																																																																																																																																																												
Disk 1	Disk 13	Disk 25	Disk 37	Disk 49	Disk 61	Disk 73	Disk 85																																																																																																																																																																																												
Disk 2	Disk 14	Disk 26	Disk 38	Disk 50	Disk 62	Disk 74	Disk 86																																																																																																																																																																																												
Disk 3	Disk 15	Disk 27	Disk 39	Disk 51	Disk 63	Disk 75	Disk 87																																																																																																																																																																																												
Disk 4	Disk 16	Disk 28	Disk 40	Disk 52	Disk 64	Disk 76	Disk 88																																																																																																																																																																																												
Disk 5	Disk 17	Disk 29	Disk 41	Disk 53	Disk 65	Disk 77	Disk 89																																																																																																																																																																																												
Disk 6	Disk 18	Disk 30	Disk 42	Disk 54	Disk 66	Disk 78	Disk 90																																																																																																																																																																																												
Disk 7	Disk 19	Disk 31	Disk 43	Disk 55	Disk 67	Disk 79	Disk 91																																																																																																																																																																																												
Disk 8	Disk 20	Disk 32	Disk 44	Disk 56	Disk 68	Disk 80	Disk 92																																																																																																																																																																																												
Disk 9	Disk 21	Disk 33	Disk 45	Disk 57	Disk 69	Disk 81	Disk 93																																																																																																																																																																																												
Disk 10	Disk 22	Disk 34	Disk 46	Disk 58	Disk 70	Disk 82	Disk 94																																																																																																																																																																																												
Disk 11	Disk 23	Disk 35	Disk 47	Disk 59	Disk 71	Disk 83	Disk 95																																																																																																																																																																																												
<table border="1"> <thead> <tr> <th>TYPE</th> <th>BLOCK DEVICE NAME</th> <th>LOCATION</th> <th>SERIAL NO.</th> <th>SLOT NO.</th> <th>BLINK</th> </tr> </thead> <tbody> <tr> <td>Data</td> <td>/dev/sdaa</td> <td>Internal</td> <td>ZL20399G0000C9350DS0</td> <td>024</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdab</td> <td>Internal</td> <td>ZL203F570000C9360LR1</td> <td>025</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdac</td> <td>Internal</td> <td>ZL20263Y0000G93207MS</td> <td>026</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdad</td> <td>Internal</td> <td>ZL20170E0000G93111SQ</td> <td>027</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdae</td> <td>Internal</td> <td>ZL20266H0000G93206BW</td> <td>028</td> <td></td> </tr> <tr> <td>Data</td> <td>/dev/sdaf</td> <td>Internal</td> <td>ZL2026ZW0000G93207W9</td> <td>029</td> <td></td> </tr> </tbody> </table>						TYPE	BLOCK DEVICE NAME	LOCATION	SERIAL NO.	SLOT NO.	BLINK	Data	/dev/sdaa	Internal	ZL20399G0000C9350DS0	024		Data	/dev/sdab	Internal	ZL203F570000C9360LR1	025		Data	/dev/sdac	Internal	ZL20263Y0000G93207MS	026		Data	/dev/sdad	Internal	ZL20170E0000G93111SQ	027		Data	/dev/sdae	Internal	ZL20266H0000G93206BW	028		Data	/dev/sdaf	Internal	ZL2026ZW0000G93207W9	029																																																																																																																																																					
TYPE	BLOCK DEVICE NAME	LOCATION	SERIAL NO.	SLOT NO.	BLINK																																																																																																																																																																																														
Data	/dev/sdaa	Internal	ZL20399G0000C9350DS0	024																																																																																																																																																																																															
Data	/dev/sdab	Internal	ZL203F570000C9360LR1	025																																																																																																																																																																																															
Data	/dev/sdac	Internal	ZL20263Y0000G93207MS	026																																																																																																																																																																																															
Data	/dev/sdad	Internal	ZL20170E0000G93111SQ	027																																																																																																																																																																																															
Data	/dev/sdae	Internal	ZL20266H0000G93206BW	028																																																																																																																																																																																															
Data	/dev/sdaf	Internal	ZL2026ZW0000G93207W9	029																																																																																																																																																																																															

5.7.2.4. View a Node's Memory Usage

To check a node's memory usage, in the CMC's [Node Status](#) page open the **Memory Usage** panel.



The panel displays the following information:

Service

Service name. This will be one of the HyperStore system's four major Java-based services: Admin, Cassandra, HyperStore, and S3.

Heap Usage

This column shows each service's current JVM heap memory usage and also the maximum JVM heap size allocated to each service.

GC Info

Statistics for garbage collection (GC), the recurrent automatic process within a JVM whereby memory is freed up from Java objects that are no longer in-use. These statistics are provided only for the Cassandra service. The supported statistics are:

- CMS count — 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.
- CMS time — The aggregate time (in milliseconds) spent on executing CMS garbage collections since the last start-up of the Cassandra service on this node.
- ParNew count — The number of parallel new-generation (ParNew) garbage collections executed since the last start-up of the Cassandra service on this node.
- ParNew time — The aggregate time (in milliseconds) spent on executing ParNew garbage collections since the last start-up of the Cassandra service on this node.
- Copy count — The number of Copy garbage collections executed since the last start-up of the Cassandra service on this node.
- Copy time — The aggregate time (in milliseconds) spent on executing Copy garbage collections since the last start-up of the Cassandra service on this node.

Note The statistics for a GC type (CMC, ParNew, or Copy) will display only if that GC type has been executed since the last start-up of the Cassandra service.

5.7.2.5. View a Node's Services Status

To check on the status of individual services on a node, in the CMC's [Node Status](#) page open the **Services Status** panel.

SERVICES STATUS				
STATUS	SERVICE	IP ADDRESSES	LAST UPDATE	ACTION
	Admin	██████████	Mar-09-2018 13:16	RESTART
	Cassandra	██████████	Mar-09-2018 13:16	RESTART
	HyperStore	██████████	Mar-09-2018 13:16	START
	Redis Mon (Primary)	██████████	Mar-09-2018 13:16	RESTART STOP
	Redis Cred (Master)	██████████	Mar-09-2018 13:16	RESTART STOP
	Redis QoS (Master)	██████████	Mar-09-2018 13:16	RESTART STOP
	S3	██████████	Mar-09-2018 13:16	RESTART

[RESTART ALL](#)

For each service, one of the following service status icons is displayed:



The service is **up and running**.



The service is **down**.

Status is shown for the following services:

- [Admin Service](#)
- [Cassandra](#)
- [HyperStore Service](#)
- [Redis Credentials](#) (if installed on the node)
- [Redis QoS](#) (if installed on the node)
- [Redis Monitor](#) (if installed on the node)
- [S3 Service](#)

Note The node's status is automatically checked and updated each minute, as indicated by the "Last Update" column.

5.7.2.6. Start, Stop, or Restart Services On a Node

To stop, start, or restart individual services on a node, in the CMC's [Node Status](#) page open the **Services Status** panel.

SERVICES STATUS				
STATUS	SERVICE	IP ADDRESSES	LAST UPDATE	ACTION
Green circle with checkmark	Admin	██████████	Mar-09-2018 13:16	RESTART
Green circle with checkmark	Cassandra	██████████	Mar-09-2018 13:16	RESTART
Red circle with exclamation mark	HyperStore	██████████	Mar-09-2018 13:16	START
Green circle with checkmark	Redis Mon (Primary)	██████████	Mar-09-2018 13:16	RESTART STOP
Green circle with checkmark	Redis Cred (Master)	██████████	Mar-09-2018 13:16	RESTART STOP
Green circle with checkmark	Redis QoS (Master)	██████████	Mar-09-2018 13:16	RESTART STOP
Green circle with checkmark	S3	██████████	Mar-09-2018 13:16	RESTART

RESTART ALL

For services that are currently up and running, **Restart** and **Stop** buttons display in the "Action" column. If you click either button, you will be asked to confirm that you want to proceed and then upon your confirmation the restart or stop operation will execute. If the operation succeeds you will see a success message — for example, "Stopping Redis QoS...[OK]".

For services that are not currently running, a **Start** button displays in the "Action" column. If you click **Start**, the service starts up and you should see a success message — for example, "Starting Redis QoS...[OK]".

At the bottom right of the **Services Status** section a **Restart All** button displays. If you click **Restart All** and confirm, then:

- Services that are running will be restarted.
- Services that are down will be started.

Note In a **single-region** HyperStore system, the **Node Status** page does not support stopping the Admin Service or S3 Service. In a **multi-region** HyperStore system, the **Node Status** page does not support stopping the Admin Service or S3 Service in the default region.

On a **single-node** system, in addition to the above restrictions the **Node Status** page also does not support stopping the Cassandra Service (the screen shot above is from a single-node system). In a **multi-node** system the **Node Status** page does support stopping Cassandra, but doing so may cause certain CMC functions to no longer work — depending on your configured consistency requirements for service metadata and the status of your other Cassandra nodes.

If you want to stop a service that you cannot stop on the **Node Status** page you can [do so with the HyperStore installer tool or with the HyperStore initialization scripts](#). However, the CMC may not function properly while these services are down.

5.7.2.7. View and Acknowledge Node Alerts

At the bottom of the CMC's [Node Status](#) page is an **Alert List** section that displays a list of node alerts (for the one node that you've selected at the top of the page). In this section you can review and acknowledge node alerts.

The screenshot shows the 'ALERT LIST' section of the CMC. It contains a single row of data:

SEVERITY	ALERT TYPE	ALERT TEXT	LAST UPDATE	COUNT
High	HyperStore	[Service Down or Unreachable]	Mar-09-2018 13:21	17

At the bottom right of the list area is a green rectangular button labeled 'ACKNOWLEDGE'.

This section has the same functionality as the CMC's **Alerts** page, except that the information is limited to the one node that you've selected. For more guidance on understanding and working with this display, see ["Alerts" \(page 339\)](#).

Note Acknowledged alerts are **automatically deleted from the system** after a time period configured by the `mts.properties.erb`: "`events.acknowledged.ttl`" (page 512) setting. By default this period is 86400 seconds (one day). After they are deleted acknowledged alerts will not display in the Alert List even if you click **Show Acknowledged**. If you wish you can reduce the configurable time-to-live for acknowledged alerts to as little as 1 second (so that they are deleted from your system right after acknowledgment). Note that regardless of your configured time-to-live for acknowledged alerts, a record of your system's alert history will persist in the [Smart Support](#) logs that by default are uploaded to Cloudian Support each day.

5.7.3. Node Activity

Path: **Cluster** → **Nodes** → **Node Activity**

The screenshot shows the 'NODE ACTIVITY' section of the CMC. At the top, there are tabs for 'NODE STATUS' and 'NODE ACTIVITY' (which is highlighted with a red oval). Below these are sections for 'Host' (set to 'hyperstore1') and 'Operation' (a dropdown menu showing various metrics like CPU Utilization, Disk Available, etc.).

Supported task:

- Check a node's recent activity

In the CMC's **Node Activity** page you can view a variety of health and performance statistics for individual HyperStore nodes. For each node, statistics from the past 30 days are available, and within that time period you can flexibly drill down into whatever specific interval you're interested in.

Select a node in the "Host" field and then in the "Operation" field you can choose any one of the statistics listed below.

CPU Utilization

CPU utilization percentage for the node. This is measured once per every five minutes.

Disk Available

Disk space available on the node. Note that:

- Only disks that store HyperStore data directories (as configured by *common.csv*: "**hyperstore_data_directory**" (page 462)) or the Cassandra data directory (as configured by *common.csv*: "**cassandra_data_directory**" (page 480)) count toward this stat. This stat does not count disks that are being used only for OS or application files, for example.
- This stat calculates disk usage in a way that counts a disk's "reserved-blocks-percentage" (the portion of the disk space that's reserved for privileged processes) as used space. Consequently this stat may indicate a higher degree of disk usage than would the Linux df command, which does not count reserved space towards a disk's used space. By default in Linux systems the configurable "reserved-blocks-percentage" for a file system is 5% of disk capacity.

Disk Reads/Writes

Across all the node's disks that are being used for S3 object storage, the average disk read and write throughput per second. These stats are measured each minute, based on the most recent minute of activity.

Network Throughput Outgoing/Incoming

The node's outgoing and incoming network interface throughput per second. This encompasses all types of network traffic including but not limited to S3 request traffic. For example, data transmission associated with cluster maintenance operations would count toward these statistics. These stats are measured each minute, based on the most recent minute of activity.

For throughput specifically of S3 request traffic, see the Request Throughput stats.

Transactions (GET/PUT)

Number of S3 transactions processed per second by the node. This is broken out to GET transactions and PUT transactions.

HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

These stats are measured each five minutes, based on the most recent five minutes of activity.

Request Throughput (GET/PUT)

For S3 transactions, the data volume throughput per second for the node. This is broken out to GET transactions and PUT transactions.

HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

These stats are measured each five minutes, based on the most recent five minutes of activity.

Average Request Latency (GET/PUT)

For S3 transactions, the 95th percentile request latency for the node, in milliseconds. This is broken out to GET transactions and PUT transactions.

New statistic values are calculated each five minutes, based on the most recent approximately 1000 GET transactions and 1000 PUT transactions. The latency values indicate that of the last 1000 transactions of that type (GET or PUT), 95% completed in that many milliseconds or less.

HEAD transactions are counted toward the GET stat, and POST transactions are counted toward the PUT stat.

<Service> Memory Heap Usage

The current JVM heap memory usage (in number of bytes) by each of the HyperStore system's major Java-based services on the node: Admin, Cassandra, HyperStore, and S3.

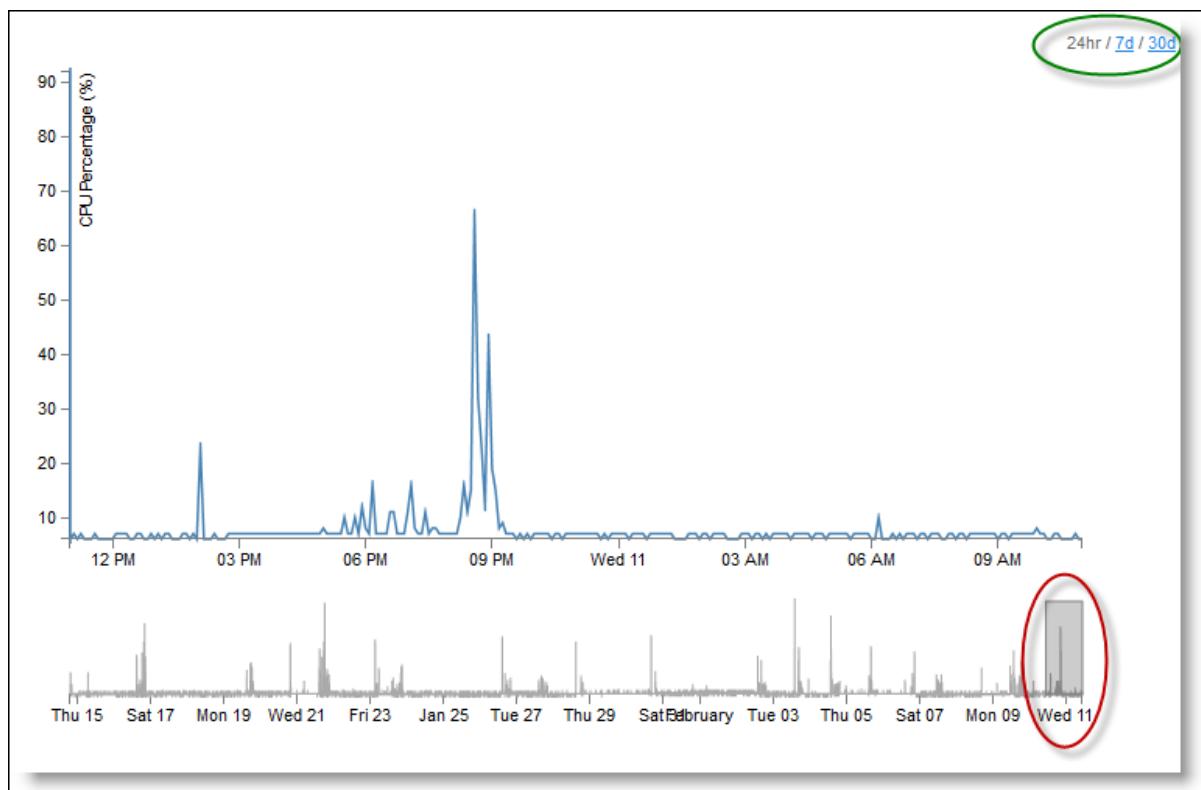
These stats are measured each five minutes.

S3 Error Count

Running count of the S3 service errors (5xx status responses) returned to S3 client applications by the node. This is a cumulative count since the last restart of the S3 Service on the node.

By default, when you choose a statistic from the "Operation" drop-down list, a graph of the statistic's movement over the **past 24 hours** displays. The interface provides you two methods for adjusting this graphing interval:

- In the upper right of the page, you can click to change the graph to a 7 day or 30 day view.
- In the small graph at the bottom of the page — which shows a compressed 30 day view — you can manipulate the gray block to control the time period that's shown in the main graph. With your mouse you can click and drag the left or right edges of the block to expand or contract the time period shown in the main graph. You can also click the block's center and drag the block to shift the main graph to an earlier or later time interval (within the bounds of the past 30 days).



Note For statistics that measure quantities of data, the metric used on a graph's Y-axis auto-scales to units that are most appropriate for the particular quantities being conveyed. Specifically, for a given

statistic the Y-axis may be expressed in terms of bytes, KBs, MBs, or GBs, depending on the activity level during the full 30-day graphing interval. Pay attention to the Y-axis label to see what metric is being used.

Note for multi-region systems

If your HyperStore system has multiple service regions, a drop-down list displays at the top of the page so you can select a region first before selecting a node for which to view activity.

5.7.4. Node Advanced

Path: Cluster → Nodes → Advanced

The screenshot shows the CMC's Node Advanced page. At the top, there are tabs for NODE STATUS, NODE ACTIVITY, and ADVANCED (which is highlighted with a red oval). Below these are input fields for Command Type (set to Info), hsstool Command (set to ring), and Target Node (set to cloudian-node1). A description field contains the text "Print information on the token ring". At the bottom right is a large green "EXECUTE" button.

Supported tasks:

- **"Info Commands"** (page 282)
- **"Maintenance Commands"** (page 283)
- **"Redis Monitor Commands"** (page 283)
- **"Disk Management Commands"** (page 283)
- **"Collect Diagnostics"** (page 284)
- **"Start Maintenance Mode"** (page 285)
- **"Uninstall Node"** (page 286)

5.7.4.1. Info Commands

In the CMC's **Node Advanced** page, with Command Type "Info" selected, you can execute any of the following **hsstool** commands which retrieve information about your system:

- [**ring**](#) — View vNode info for whole cluster (see "**hsstool ring**" (page 672))
- [**info**](#) — View vNode and data load info for a physical node (see "**hsstool info**" (page 623))
- [**status**](#) — View summary status for whole cluster (see "**hsstool status**" (page 674))
- [**opstatus**](#) — View status of repair or cleanup operations (see "**hsstool opstatus**" (page 630))

- [**proactiverepairq**](#) — View status of proactive repair queues (see "**hsstool proactiverepairq**" (page 637))
- [**repairqueue**](#) — View auto-repair schedule (see "**hsstool repairqueue**" (page 667))
- [**ls**](#) — View vNode and data load info per mount point (see "**hsstool ls**" (page 625))
- [**whereis**](#) — View storage location information for an S3 object (see "**hsstool whereis**" (page 680))
- [**metadata**](#) — View metadata for an S3 object (see "**hsstool metadata**" (page 626))
- [**trmap**](#) — View a list of active token range map snapshots (see "**hsstool trmap**" (page 676))

5.7.4.2. Maintenance Commands

In the CMC's [**Node Advanced**](#) page, with Command Type "Maintenance" selected, you can execute any of the following *hsstool* commands for acting on the data and metadata in your system:

- [**cleanup**](#) — Clean a node of replicated data that doesn't belong to it (see "**hsstool cleanup**" (page 610))
- [**cleanupec**](#) — Clean a node of erasure coded data that doesn't belong to it (see "**hsstool cleanupec**" (page 616))
- [**autorepair**](#) — Enable or disable auto-repair (see "**hsstool repairqueue**" (page 667))
- [**proactiverepair**](#) — Enable or disable or stop or start proactive repair (see "**hsstool proactiverepairq**" (page 637))
- [**repair**](#) — Repair the replicated data on a node (see "**hsstool repair**" (page 647))
- [**repairec**](#) — Repair the erasure coded data on a node (see "**hsstool repairec**" (page 658))
- [**repaircassandra**](#) — Repair the system and object metadata on a node (see "**hsstool repair-cassandra**" (page 656))
- [**rebalance**](#) — Rebalance some data from the cluster to a newly added node (see "**hsstool rebalance**" (page 642))

5.7.4.3. Redis Monitor Commands

In the CMC's [**Node Advanced**](#) page, with Command Type "Redis Monitor Operations" selected, you can execute any of the following commands for the Redis Monitor Service:

- [**setClusterMaster**](#) — Move the Redis Credentials master role or the Redis QoS master role (see "**Move the Redis Credentials Master or QoS Master Role**" (page 405))
- [**getClusterInfo**](#) — Get information about the Redis Credentials cluster or the Redis QoS cluster (see "**get cluster**" (page 686))

5.7.4.4. Disk Management Commands

In the CMC's [**Node Advanced**](#) page, with Command Type "Disk Management" selected, you can execute any of the following commands for managing HyperStore data disks:

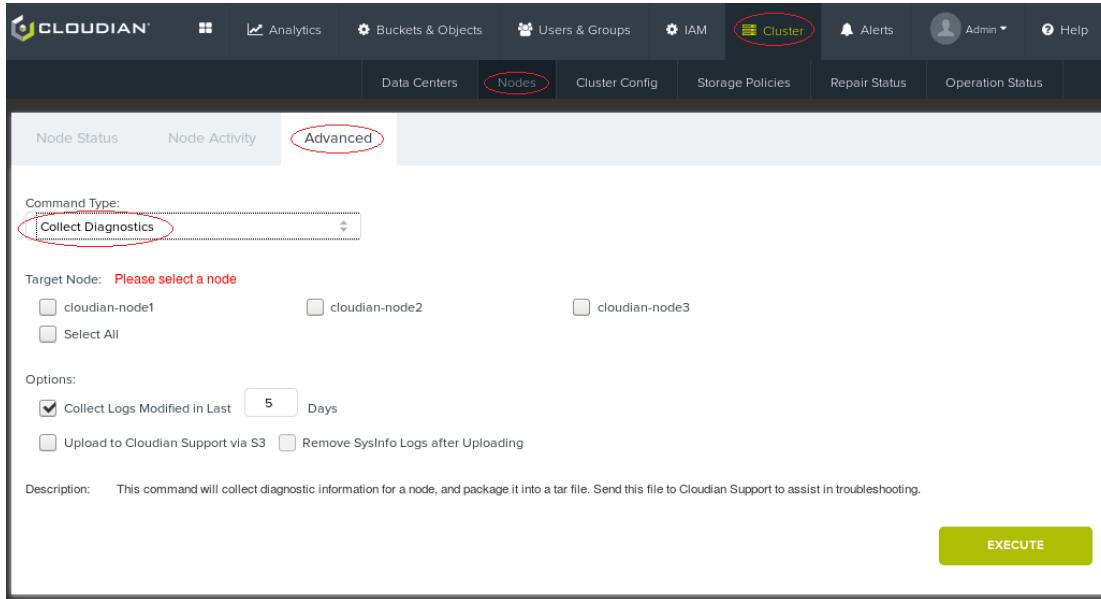
- [**disableDisk**](#) — Temporarily disable a disk (see "**Disabling a HyperStore Data Disk**" (page 427))
- [**enableDisk**](#) — Re-enable a disk that is currently disabled (see "**Enabling a HyperStore Data Disk**" (page 428))
- [**replaceDisk**](#) — Replace a disk (see "**Replacing a HyperStore Data Disk**" (page 430))

5.7.4.5. Collect Diagnostics

When you are experiencing system problems, Cloudian Support is best able to help you if you provide them with comprehensive diagnostic information. Through the CMC you can easily generate a package of diagnostic information for a specified node or nodes. Each node's diagnostics package will be created on the node itself, at path `/var/log/cloudian/cloudian_sysinfo/<hostname>_<YYYYMMDDmmss>.tar.gz`. The package will include log files, configuration files, statistics, and command outputs for that particular node. Optionally, you can have the diagnostics package file get automatically uploaded via S3 to Cloudian Support or an alternative S3 destination of your choosing.

To generate diagnostics for a node or nodes:

1. In the CMC's **Node Advanced** page, from the Command Type menu select "Collect Diagnostics":



2. In the "Target Node" section, select one or more nodes for which to collect diagnostics data.
3. Select from among the diagnostics processing options:

Collect Logs Modified in the Last X Days

Use this to specify how many days' worth of system, application, and transaction logs should be collected from the target node(s). The default behavior is to collect the last 5 days' worth of logs.

Upload to Cloudian Support via S3

Select this checkbox if you want the diagnostics from the target node(s) to be automatically uploaded to Cloudian Support, immediately after creation of the diagnostics package.

Note Optionally, you can have the automatic upload go to a different S3 destination rather than Cloudian Support. To do so requires that you first implement a system configuration change. For more information see "**Configuring Smart Support and Node Diagnostics**" (page 138).

Remove SysInfo Logs After Uploading

This option is applicable only if you are having the diagnostics automatically uploaded to an S3

destination. If you select the "Upload to Cloudian Support via S3" checkbox then the "Remove SysInfo Logs After Uploading" becomes checked also by default. With this option checked, then after the upload to the S3 destination the local copy of the diagnostics package file will be automatically deleted. If you want to retain the local copy of the diagnostics package (under a `/var/log/cloudian/cloudian_sysinfo` directory on the target node[s]) as well as uploading a copy to the S3 destination, then uncheck the "Remove SysInfo Logs After Uploading" checkbox.

Note If the "Remove SysInfo Logs After Uploading" checkbox is not selected when you perform the diagnostics collection operation, then the diagnostics package is retained on the target node(s) for **15 days** before being automatically removed. This retention period is configurable by the "`cleanup_sysinfo_logs_timelimit`" (page 461) setting in [common.csv](#).

4. Click **Execute**.
5. In the confirmation dialog that displays, confirm that you want to proceed with the operation.

Once you confirm, the Result section of the page displays a message indicating the command has been sent. After the operation completes, the Result section displays a brief summary of the operation outcome.

5.7.4.6. Start Maintenance Mode

Through the CMC you can place a node into "maintenance mode" so that no S3 write or read requests will be directed to the node and the node will not generate any alerts. If the node is hosting a Redis master, the master role will be temporarily shifted to a different node.

Within a service region, you can have no more than one node in maintenance mode at a time. While the node is in maintenance mode, in the CMC's [Data Centers](#) page a special blue icon will indicate that the node is in maintenance mode. In the CMC's [Node Status](#) page, most detailed status information will not be available for the node while it is under maintenance.

You can subsequently use the CMC to take the node out of maintenance mode and put it back into regular service.

Note For you to put a node into maintenance mode, all other nodes in the region must be up, and all services on those other nodes must be up. The system will not allow you to put a target node into maintenance mode if other nodes are down in the system, or if some services are down on other nodes in the system.

When a node is in maintenance mode, the system supports using fallback consistency levels for S3 write or read requests for which the node is an endpoint (if you have configured "[Dynamic Consistency Levels](#)" (page 29).)

To put a node into maintenance mode:

1. In the CMC's [Node Advanced](#) page, from the Command Type menu select "Start Maintenance Mode":

2. From the "Target Node" list, select the node that you want to put into maintenance mode.
3. Click **Execute**.

After confirming that all other nodes and services are up in the region, the system will put the target node into maintenance mode.

Note If any S3 requests are in processing on the target node when you submit the Start Maintenance Mode command, the processing of those requests will be completed before the node goes into maintenance mode.

Stopping Maintenance Mode

To take a node out of maintenance mode and put it back into regular service:

1. In the CMC's **Node Advanced** page, from the Command Type menu select "Stop Maintenance Mode".
2. From the "Target Node" list, select the node that you want to take out of maintenance mode
3. Click **Execute**.

The target node is put back into regular service.

5.7.4.7. Uninstall Node

In the CMC's [**Node Advanced**](#) page, you can use the Command Type "Uninstall Node" to remove a node from your cluster.

For the full procedure including important steps to take before removing a node, see "["Removing a Node"](#)" (page 390).

5.7.5. Cluster Information

Path: **Cluster** → **Cluster Config** → **Cluster Information**

VERSION INFORMATION

APPLICATION VERSION:
7.2 Compiled: 2019-08-10 16:22

LICENSE INFORMATION

EXPIRE DATE:
Dec-24-2028 02:22 +0000

LICENSED MAX NET STORAGE: 1GB	LICENSED MAX TIERED STORAGE: Unlimited
NET STORAGE USED: 0	TIERED STORAGE USED: 0

OBJECT LOCK MODE:
Disabled

BROWSE... No file selected. **UPDATE LICENSE** **+ REQUEST LICENSE**

SERVICE INFORMATION

CMC ADMIN SERVER HOST: s3-admin.landemo1.cloudian.eu:19443	CASSANDRA CLUSTER NAME: Cloudianreg-1
S3 ENDPOINT (HTTP): s3-reg-1.landemo1.cloudian.eu:80	S3 ENDPOINT (HTTPS): s3-reg-1.landemo1.cloudian.eu:443
S3 WEBSITE ENDPOINT: s3-website-reg-1.landemo1.cloudian.eu	
REDIS CREDENTIALS MASTER HOST: cloudian-node3	REDIS CREDENTIALS SLAVE HOST(S): cloudian-node1.cloudian.eu:22222, cloudian-node2.cloudian.eu:22222

Supported tasks:

- View Cluster Information (below)
- "Renew and Install a License" (page 292)

The **Cluster Information** page displays summary information about your HyperStore system. The page shows the summary information items below.

Version Information

Application Version

Your current HyperStore software version

License Information

Expire Date

Date on which your license expires.

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

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

- In the CMC, the top of the console screen displays a warning indicating that your license has expired.
- The system still accepts and processes incoming S3 requests, but every S3 response returned by the S3 Server includes an extension header indicating that the system license has expired (header name: *x-gemini-license*; value: *Expired: <expiry_time>*)

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

- No S3 service is available for end users. All incoming S3 requests will be rejected with a "503 Service Unavailable" error response. The response also includes the expiry header described above.
- You can still log into the CMC to perform system administration functions (including applying an updated license), but you will not be able to access users' stored S3 objects.

For information on renewing your license, see "**Renew and Install a License**" (page 292).

Licensed Max Net/Raw Storage

The maximum amount of object data storage your HyperStore system license allows for. This will be in terms of either Net storage or Raw storage, depending on your particular license terms.

With a license based on Net storage, the limit is on total object storage bytes excluding overhead from object replication or erasure coding. For example if a 1GB object is replicated three times in your system it counts as only 1GB toward the Net Storage limit.

With a license based on Raw storage, the limit is on total bytes stored on formatted disks in your system. This encompasses object storage bytes including overhead from object replication or erasure coding, as well as bytes of stored metadata. Note that with this type of storage limit, if a 1GB object is replicated three times in your system it counts as 3GB toward the limit.

In a multi-region HyperStore system, the licensed storage limit is for the system as a whole (all regions combined).

For more information on licensing see "**Licensing and Auditing**" (page 6).

Licensed Max Tiered Storage

The maximum amount of tiered data storage your HyperStore system license allows for.

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.

If this value is "Unlimited" then the license places no limit on tiered data volume.

For more information on licensing see "**Licensing and Auditing**" (page 6).

Net Storage Used / Raw Storage Used

If your HyperStore license is based on Net storage, this field is "Net Storage Used" and the number displayed is your current total storage usage excluding overhead from object replication or erasure coding. For example if a 1MB S3 object is replicated three times in your system it counts as only 1MB toward the "Number of Stored Bytes" count.

If your HyperStore license is based on Raw storage, this field is "Raw Storage Used" and the number displayed is your current total raw storage including overhead from object replication or erasure coding. For example if a 1MB S3 object is replicated three times in your system it counts as 3MB toward the "Raw Storage Used" count.

The storage usage count for your system is updated at the top of each hour. In a multi-region HyperStore system, the count is for the system as a whole (all regions combined).

This field displays a warning message if your current bytes count exceeds 70% of your licensed usage maximum; and a critical message if your current bytes count exceeds 90% of your licensed usage maximum

Note If some users have **versioning** enabled on their buckets -- so that the system retains rather than overwriting older versions of an object when the user uploads a new version of the object -- then each stored object version (the older versions as well as the current version) counts toward your system usage count.

Also, if some users use the **cross-region replication** feature to replicate objects from one HyperStore bucket to another HyperStore bucket in the same HyperStore system, then the original source objects and the object replicas in the destination bucket both count toward your system bytes count. For more information see "**Cross-Region Replication Impact on Usage Tracking**" (page 135)

Tiered Storage Used

The current tiered storage usage level in external systems other than HyperStore.

Data auto-tiered from one of your HyperStore regions to another region in the same HyperStore system, or from your HyperStore system to an external HyperStore system, does not count toward this figure. Auto-tiered data stored in any other type of external system -- such as the Amazon, Azure, or Google clouds -- does count toward this figure.

Note A warning message displays here if this figure exceeds 70% of your licensed maximum tiered storage limit, or a critical message if this figure exceeds 90% of your licensed maximum tiered storage limit

Bucket Lock Mode

Your license's support or non-support of the HyperStore WORM (bucket lock) feature:

- DISABLED -- WORM is not supported.
- SEC17 -- WORM is supported, but having a "privileged delete user" is not supported
- ENTERPRISE -- WORM is supported, and having a "privileged delete user" is supported

For more information see "**WORM (Bucket Lock) Feature Overview**" (page 159).

Service Information

Note If you have a [multi-region HyperStore system](#), a drop-down list lets you choose a region for which to view Service Information.

For information about moving service roles from one HyperStore node to another, see "**Change Node Role Assignments**" (page 405).

CMC Admin Server Host

Admin Service endpoint to which the CMC connects in order to submit Admin Service calls. The CMC makes calls to the [Admin Service](#) when you use the CMC to perform administrative tasks like adding users,

configuring rating plans, or viewing system monitoring data. In a multi-region system there is just one Admin Service endpoint for the whole system.

All of the HyperStore nodes in the [default service region](#) should service requests to this endpoint. In a production environment, typically you would have the Admin service endpoint resolve to the virtual IP(s) of one or more load balancers in the default service region, and the load balancers would in turn distribute traffic across all the HyperStore nodes in the default region. For more information see "DNA Set-Up" in the HyperStore Installation Guide.

Note The label "CMC Admin Server Host" is somewhat misleading. In the current version of HyperStore this is a service endpoint, not a single host or IP address as it was in earlier HyperStore versions.

Cassandra Cluster Name

Name of the [Cassandra](#) cluster in this service region. In a multi-region HyperStore system, each region has an independent Cassandra cluster. The system automatically derives a Cassandra cluster name from the local region name.

S3 Endpoint (HTTP) and S3 Endpoint (HTTPS)

S3 endpoint(s) for this service region. In a multi-region system, each region has its own S3 endpoint -- for HTTP (port 80), and optionally HTTPS (port 443). S3 client applications connect to these endpoints to submit S3 API calls.

For each region's S3 endpoint, all of the HyperStore nodes within that region should service requests to the endpoint. In a production environment, typically you would have the S3 service endpoint resolve to the IP addresses of one or more load balancers within the region, and the load balancers would in turn distribute traffic across all the HyperStore nodes in the region. For more information see "DNA Set-Up" in the HyperStore Installation Guide

Note HTTPS/SSL for the S3 Service is not implemented by default. To set this up, see "[TLS/SSL for S3 Service \(Self-Signed Certificate\)](#)" (page 543) or "[TLS/SSL for S3 Service \(CA-Verified Certificate\)](#)" (page 547).

S3 Website Endpoint

S3 website endpoint for this service region. Web browsers use the S3 website endpoint to access content in buckets that are [configured as static websites](#). In a multi-region system, each region has its own S3 website endpoint. For S3 website endpoints, only HTTP is supported -- not HTTPS.

For each region's S3 website endpoint, all of the HyperStore nodes within that region should service requests to the endpoint. In a production environment, typically you would have the S3 website endpoint resolve to the IP addresses of one or more load balancers within the region, and the load balancers would in turn distribute traffic across all the HyperStore nodes in the region. For more information see "DNA Set-Up" in the HyperStore Installation Guide

Redis Credentials Master Host

Host on which the [Redis Credentials master node](#) is located. In a multi-region system, there is just one Redis Credentials master for the whole system.

If the one Redis Credentials master is located in a service region other than the one that you are currently viewing, the display will show "Not installed".

Redis Credentials Slave Host(s)

Hosts on which the [Redis Credentials slave nodes](#) are located. This is typically two nodes per data center.

Note If you upgraded from a HyperStore version older than 6.0, then by default you will have just one Redis Credentials slave node per data center.

Redis QoS Master Host

Host on which this service region's [Redis QoS master node](#) is located. In a multi-region system, each region has its own Redis QoS master.

Redis QoS Slave Host(s)

Hosts on which the [Redis QoS slave nodes](#) are located. This is typically one node per data center.

Redis Monitor Primary Host

Host on which the primary [Redis Monitor](#) instance is located. This is just one host per whole system, even for a multi-region system.

If the one Redis Monitor primary instance is located in a service region other than the one that you are currently viewing, the display will show "Not installed".

Redis Monitor Backup Host

Host on which the backup [Redis Monitor](#) instance is located. This is just one host per whole system, even for a multi-region system. If the Redis Monitor primary instance fails, the Redis Monitor services automatically fail over to the backup host.

If the one Redis Monitor backup instance is located in a service region other than the one that you are currently viewing, the display will show "Not installed".

System Monitoring / Cronjob Primary Host

In this service region, the host on which the primary instance of the HyperStore [Monitoring Data Collector](#) and the primary instance of the HyperStore [system maintenance crontab](#) are located. In a multi-region system, each region has one System Monitoring / Cronjob Primary Host.

System Monitoring / Cronjob Backup Host

In this service region, the host on which the backup instance of the HyperStore [Monitoring Data Collector](#) and the backup instance of the HyperStore [system maintenance crontab](#) are located. In a multi-region system, each region has one System Monitoring / Cronjob Backup Host. If the System Monitoring / Cronjob Primary Host fails, the system monitoring and cron job services automatically fail over to the backup host.

External NTP Host(s)

In this service region, the external NTP servers to which the HyperStore cluster's internal NTP servers connect. For information about how HyperStore automatically implements a robust time-synchronization set-up using NTP, see "[NTP Automatic Set-Up](#)" (page 540).

Internal NTP Host

In this service region, the hosts acting as internal NTP servers. There will be four internal NTP hosts per data center. (If a HyperStore data center has only four or fewer nodes, then all the nodes in the data center are configured as internal NTP servers.) For information about how HyperStore automatically implements a robust

time-synchronization set-up using NTP, see "**NTP Automatic Set-Up**" (page 540).

Puppet Master Host

Host on which the active **Puppet Master** is located. This is just one host per whole system, even for a multi-region system.

Puppet Master Backup Host

Host that is configured to be the backup host for the **Puppet Master**. This is just one host per whole system, even for a multi-region system. If the node on which the active Puppet Master is located fails, you can **manually** fail over the Puppet Master role to the backup host. Automatic fail-over of this role is not supported. For instructions see "**Move the Puppet Master Primary or Backup Role**" (page 415).

If the one Puppet Master backup host is located in a service region other than the one that you are currently viewing, the display will show "Not installed".

Installation Staging Directory on Puppet Master

On the Puppet Master host, the installation staging directory path. In this directory resides the *cloudianInstall.sh* script, which you can use for certain cluster management tasks such as **pushing configuration file edits out to the cluster** or **restarting services**. By default the installation staging directory for HyperStore version 7.2 is */opt/cloudian-staging/7.2*.

Number of Datacenters

Number of data centers in which you are running the HyperStore system.

Number of Hosts in Each Datacenter

Number of HyperStore hosts in each data center, in format <DCName>: <#hostsInDC>.

5.7.5.1. Renew and Install a License

In the CMC's **Cluster Information** page you can request a HyperStore license renewal, and install a new or renewed license file.

Request a License Renewal

Click **Request License** to send an email to *cloudian-license@cloudian.com* to initiate the process of obtaining a new license file. Clicking this button should open your default email application (for instance, Gmail).

If nothing happens when you click **Request License**, your browser is not configured for launching your email application. You can instead open your email application manually and send a license renewal request to *cloudian-license@cloudian.com*.

Install a New License File

Note This feature is for installing a cluster-wide license, not a license specific to a particular HyperStore Appliance machine.

After you've obtained a new license file from Cloudian, you need to apply the new license file to your HyperStore system. You can do so through the **Cluster Information** page:

1. Put the license file on the computer from which you are accessing the CMC (the computer on which your browser is running).
2. On the left side of the **Cluster Information** page, click **Browse**. On your local machine, browse to and select the license file.
3. Click **Update License**.

The CMC pushes the license file to the Puppet master node, which in turn propagates the file to all your HyperStore nodes. The Puppet master then uses JMX to trigger your HyperStore nodes to dynamically reload the license file. You do not need to restart any services.

After the process completes, if you refresh the **Cluster Information** page you should see updated information in the **License Information** section.

Note The system saves your old license file on the Puppet master node as <old-license-file-name>.<-timestamp> (in directory /etc/puppet/modules/base/layout/files).

5.7.6. Configuration Settings

Path: **Cluster** → **Cluster Config** → **Configuration Settings**

The screenshot shows the Cloudian Configuration Settings page. At the top, there's a navigation bar with tabs for Analytics, Buckets & Objects, Users & Groups, Cluster (which is highlighted with a red oval), Data Centers, Nodes, Cluster Config (which is also highlighted with a red oval), Storage Policies, Repair Status, and Operation Status. Below the navigation bar, there are two main sections: 'CLUSTER INFORMATION' and 'CONFIGURATION SETTINGS'. The 'CONFIGURATION SETTINGS' section is highlighted with a red oval. It contains a list of configuration items: SMTP/Email Settings for Alerts/Notifications, SNMP Trap Destination Settings, System, Usage Tracking, Quality of Service, Auto-Tiering, S3 Request Restrictions, and Auto Repair Schedule. Each item has a 'SAVE' button at the bottom right. The 'CLUSTER INFORMATION' section is visible on the left.

Supported task:

- Edit HyperStore configuration settings

When you change settings in the CMC's **Configuration Settings** page the system applies your configuration changes dynamically — no service restart is required. The system also automatically updates and pushes the relevant configuration file settings so that your configuration changes persist across any future restarts of HyperStore services.

You can edit any of the following types of settings. **Be sure to click Save at the bottom of the page** to save your changes.

- "SMTP/Email Settings for Alerts/Notifications" (page 294)
- "SNMP Trap Destination Settings" (page 296)
- "System Settings" (page 297)
- "Usage Tracking Settings" (page 298)
- "Quality of Service Settings" (page 300)
- "Auto-Tiering Settings" (page 301)
- "S3 Request Restriction Settings" (page 304)
- "Auto-Repair Schedule Settings" (page 305)

Note In a multi-region system, the CMC's **Configuration Settings** page does not support selecting a region. Instead, these settings apply to the whole system. They are not region-specific.

5.7.6.1. SMTP/Email Settings for Alerts/Notifications

The settings in this section of the CMC's **Configuration Settings** page pertain to the SMTP service that you want the HyperStore system to use when it sends alert notification emails to system administrators. Providing the system with this information is essential for proper system monitoring and administration.

Setting	Value	Action
SMTP Server FQDN	smtp.notification.configure.me	Edit
SMTP Port	465	Edit
SMTP Protocol	smtps	Edit
SMTP Enable STARTTLS	No	Edit
SMTP From Address	noreply@smtp.notification.configure.me	Edit
Notification Message Subject Header	Cloudian HyperStore Alert	Edit
Default Email Address to Receive Notifications	admin@mycloudianhyperstore.com	Edit
SMTP Service Requires Authorization	No	Edit
User Name for SMTP Server		Edit
Password for SMTP Server	*****	Edit

To use this feature you will need information about your organization's mail server, and at least one system administrator email account must have already been set up and be able to receive email.

After making any edits in the SMTP/Email Settings section, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster. Then open the SMTP/Email Settings section again and click **Send Test SMTP Notification** to send a test email to the system administrator email address that you configured. Then confirm that the test email was received at that address. (In the event of a failed test, on the CMC node on which you conducted the test check `/var/log/cloudian/cloudian-ui.log` for error messages.)

Note You can set alert notification triggers in the [Alert Rules](#) page.

SMTP Server FQDN

Fully qualified domain name (FQDN) of the SMTP service that the HyperStore system should utilize for sending alert notification emails to system administrators.

Default = `smtp.notification.configure.me`

SMTP Port

Listening port of the SMTP service that the HyperStore system should utilize for sending alert notification emails to system administrators.

Default = 465

SMTP Protocol

Protocol to use when sending alert notification emails. Options are `smtp` or `smtpls`.

The HyperStore system uses the default SMTP ports (25 for `smtp` or 465 for `smtpls`). If you want to use non-default SMTP ports, consult with Cloudian Support.

Default = `smtpls`

SMTP Enable STARTTLS

Whether to use STARTTLS when sending alert notification emails. Options are Yes or No.

If you enable this option you should also set the SMTP Protocol setting to "smtp" (not "smtpls") and set the SMTP Port setting to 587 (if your email server is using the typical STARTTLS port).

Default = No

Note Even if you enable STARTTLS, if the recipient email account is a Gmail account the account user will still need to set the 'Allow less secure apps' option on the Gmail dashboard.

SMTP From Address

The "From" address to use when sending alert notification emails.

Default = `noreply@smtp.notification.configure.me`

Notification Message Subject Header

The "Subject" to use when sending alert notification emails.

Default = Cloudian HyperStore Notification Alert

Default Email Address to Receive Notifications

Default system administrator email address(es) to which to send alert notification emails. If you want alert

notification emails to go to multiple email addresses, enter the addresses as a comma-separated list.

Default = admin@mycloudianhyperstore.com

SMTP Service Requires Authorization

Whether the SMTP service that the HyperStore system will use to send alert notification emails requires clients to use SMTP Authentication. Options are Yes or No.

Default = No

User Name for SMTP Server

If SMTP Authentication is required by the SMTP server, the username to submit with requests to the server. Comma (,) and double-quote ("") are not supported in this setting's value.

Default = No default

Password for SMTP Server

If SMTP Authentication is required by the SMTP server, the password to submit with requests to the server. Comma (,) and double-quote ("") are not supported in this setting's value.

Default = No default

5.7.6.2. SNMP Trap Destination Settings

The settings in this section of the CMC's **Configuration Settings** page configure an SNMP trap destination, in support of having system alerts sent as SNMP traps.

The screenshot shows the Cloudian Management Console interface. The top navigation bar includes tabs for Analytics, Buckets & Objects, Users & Groups, Cluster (which is highlighted), Alerts, Admin, and Help. Below the navigation is a sub-menu for Cluster Config with options like Data Centers, Nodes, Cluster Config (highlighted), Storage Policies, Repair Status, and Operation Status. The main content area has two tabs: CLUSTER INFORMATION and CONFIGURATION SETTINGS (also highlighted with a red circle). Under CONFIGURATION SETTINGS, there is a section titled 'SNMP Trap Destination Settings' (circled in red). This section contains two input fields: 'Destination IP Address' and 'Destination Port' (set to 162), each with an 'Edit' button. Below this are sections for 'System' and 'Usage Tracking'.

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Note For information about enabling SNMP trap sending for alerts, see "**Alert Rules**" (page 344).

In the traps that HyperStore generates and sends to your specified destination, the OID is `enter-`

prises.16458.4.1.1.1. The trap payload also indicates the specific HyperStore host on which the trap-triggering event occurred. HyperStore uses SNMP version 2c.

Destination IP Address

IP address of the SNMP manager to which the CMC will send system alerts (for alert types for which you've enabled SNMP trap sending). Do not enter multiple IP addresses.

Default = None

Destination Port

Listening port used by the SNMP manager.

Default = 162

5.7.6.3. System Settings

The settings in this section of the CMC's **Configuration Settings** page configure storage system operations.

The screenshot shows the Cloudian CMC interface. At the top, there's a navigation bar with tabs for Analytics, Buckets & Objects, Users & Groups, IAM, Cluster (which is highlighted with a red circle), Alerts, Admin, and Help. Below the navigation bar, there's a secondary navigation bar with tabs for Data Centers, Nodes, Cluster Config (which is highlighted with a red circle), Storage Policies, Repair Status, and Operation Status. The main content area is titled "Cluster Information" and contains a "Configuration Settings" sub-section. This sub-section has several expandable sections: "SMTP/Email Settings for Alerts/Notifications", "SNMP Trap Destination Settings", "System" (which is highlighted with a red circle), "Usage Tracking", and "Quality of Service". Under the "System" section, there's a table with two rows: "HyperStore Disk Failure Action" (set to "Disable Disk + Move Its Tokens") and an "Edit" button. The entire configuration settings section is enclosed in a rounded rectangle.

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

HyperStore Disk Failure Action

The automated action for the system to take in the event that read or write errors occur for a particular HyperStore data disk. Supported automated actions are:

- **Disable Disk + Move its Tokens** — With this setting, if disk errors are detected the system will automatically unmount the disk and mark it as disabled. The system will also automatically transfer all of the disabled disk's storage tokens to the remaining disks on the host, in an approximately balanced way. The data from the disabled disk will **not** be recreated on the other disks (repair operations will not be automatically triggered, and even if you triggered repair operations manually the data from the disabled disk would not be recreated on the other disks because with this disk disabling approach the moved tokens are assigned updated timestamps).

When a disk is disabled in this way, writes of new or updated S3 object data that would have gone to

the disabled disk will go to the other disks on the host instead. Meanwhile existing S3 object data will be unreadable on the host. Whether the system as a whole can still provide S3 clients with read access to the affected S3 objects depends on the storage policies with which the objects are associated, and on the availability of other replicas or erasure coded fragments within the cluster.

When a disk is disabled in this way, an alert is generated and the disk will show as disabled in the Disk Detail Info section of the **Node Status** page.

Note When you subsequently perform the "**Enabling a HyperStore Data Disk**" (page 428) operation or the "**Replacing a HyperStore Data Disk**" (page 430) operation, the tokens will automatically be moved back to the re-enabled or replacement disk. Data written in association with the tokens when they were on the other disks will remain on those other disks and does not need to be moved to the re-enabled or replacement disk. For information on how HyperStore tracks token location over time so that objects can be written to and read from the correct disks, see "**Dynamic Object Routing**" (page 87).

- **None** — With this option the system will not automatically disable a disk for which errors have occurred. Each disk error will trigger an alert, however.

By default, your configured automatic disk failure action will be triggered if 10 "HSDISKERROR" error messages for the disk occur in the HyperStore Service application log within a 5 minute span. This threshold is configurable by these settings in *hyperstore-server.properties.erb*:

- "**disk.fail.error.count.threshold**" (page 494)
- "**disk.fail.error.time.threshold**" (page 495)

Your configured automatic disk failure action will also be triggered if the HyperStore drive audit feature detects that a disk is in a read-only condition.

Default = Disable Disk + Move Its Tokens

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

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

5.7.6.4. Usage Tracking Settings

The settings in this section of the CMC's **Configuration Settings** page configure the HyperStore system's functionality for tracking service usage by users and groups.

The screenshot shows the Cloudian management interface. At the top, there are tabs for Analytics, Buckets & Objects, Users & Groups, Cluster (which is circled in red), Alerts, Admin, and Help. Below these are sub-tabs for Data Centers, Nodes, Cluster Config (circled in red), Storage Policies, Repair Status, and Operation Status. The main content area has two tabs: CLUSTER INFORMATION and CONFIGURATION SETTINGS (circled in red). Under CONFIGURATION SETTINGS, there are sections for SMTP/Email Settings, SNMP Trap Destination Settings, System, and Usage Tracking. The Usage Tracking section is expanded and circled in red, showing a setting for tracking request rates and data transfer rates, which is currently disabled. There is an 'Edit' button next to this setting.

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Note For an overview of the HyperStore usage reporting feature, see "["Usage Reporting Feature Overview"](#)" (page 153).

Track/Report Usage for Request Rates and Data Transfer Rates

Whether to have the system maintain counts for number of HTTP requests, number of bytes-in, and number of bytes-out for each user and for each user group. By default this functionality is disabled, and the system only tracks and reports on stored bytes and stored object counts per user and group.

You must change this setting to "Enabled" if you want the system to support usage reports that report on number of HTTP requests (GETs, PUTs, and DELETEs), bytes-in (upload) volume, and bytes-out (download) volume per user and group. This is important if you want to bill or charge-back users or groups based in part on request volume and/or data transfer volume.

Note that enabling this feature does not produce request counts and data transfer counts for user activity that occurred during the period when the feature was disabled. Rather, if you enable this feature then request tracking and data transfer tracking starts from that point forward.

Default = Disabled

Note This setting applies only to per-user and per-group usage statistic tracking. It does not apply to per-bucket usage tracking. If you have [enabled per-bucket usage tracking](#), then all usage types -- including request rates and data transfer rates -- are available on a per-bucket basis regardless of this setting.

Note If you enable this setting, then the retention period for raw usage data in Cassandra is automatically reduced from seven days to one day. This is due to the large amount of raw usage data that can accumulate quickly if request and data transfer usage tracking is enabled. See "[reports.raw.ttl](#)"

(page 511) in *mts.properties.erb*. Note that each hour, raw usage data gets automatically rolled into hourly roll-up usage data -- which has a default retention period of 65 days.

5.7.6.5. Quality of Service Settings

The settings in this section of the CMC's **Configuration Settings** page configure the HyperStore system's quality of service (QoS) feature.

Setting	Status	Action
Enforce Configured QoS Limits for Storage Utilization	Disabled	Edit
Enforce Configured QoS Limits for Request Rates and Data Transfer Rates	Disabled	Edit

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Note For an overview of the HyperStore QoS feature, see "**Quality of Service (QoS) Feature Overview**" (page 110).

Enforce Configured QoS Limits for Storage Utilization

If this setting is **enabled** the system will enforce Quality of Service (QoS) limits for number of stored bytes and number of stored objects, if you have [set such limits for users and groups](#).

If this setting is **disabled** the system will not enforce any QoS limits, even if you have set such limits for users and groups.

Note To enforce QoS limits for stored bytes and number of stored objects **but not** QoS limits for HTTP request rate, bytes-in rate, and bytes-out rate, set:

Enforce Configured QoS Limits for Storage Utilization = enabled

Enforce Configured QoS Limits for Request Rates and Data Transfer Rates = disabled

To enforce QoS limits for stored bytes and number of stored objects **and also** QoS limits for HTTP request rate, bytes-in rate, and bytes-out rate, set:

Enforce Configured QoS Limits for Storage Utilization = enabled

Enforce Configured QoS Limits for Request Rates and Data Transfer Rates = enabled

Enforcing QoS for traffic rates but not for stored bytes and objects is not supported at the system configuration level. If you want to use QoS in this way, set both of the QoS enforcement settings to enabled, then when you're configuring QoS limits for groups and users set the stored bytes and objects controls to unlimited and the rate controls to your desired levels.

Default = Disabled

Enforce Configured QoS Limits for Request Rates and Data Transfer Rates

If this setting is **enabled** the system will enforce QoS limits for HTTP request rate, bytes-in rate, and bytes-out rate, if you have [set such limits for users and groups](#).

If this setting is **disabled**, then the system will not enforce HTTP request rate, bytes-in rate, and bytes-out rate limits, even if you have set such limits for users and groups.

Enabling this setting is supported only if the *Enforce Configured QoS Limits for Storage Utilization* setting is also enabled.

Default = Disabled

5.7.6.6. Auto-Tiering Settings

The settings in this section of the CMC's **Configuration Settings** page configure the HyperStore system's auto-tiering feature.

The screenshot shows the Cloudian Configuration Settings page. The top navigation bar includes links for Analytics, Buckets & Objects, Users & Groups, Cluster (which is highlighted with a red oval), Alerts, Admin, and Help. Below the navigation is a secondary menu with tabs for Data Centers, Nodes, Cluster Config (highlighted with a red oval), Storage Policies, Repair Status, and Operation Status. The main content area has two tabs: CLUSTER INFORMATION and CONFIGURATION SETTINGS (highlighted with a red oval). Under CONFIGURATION SETTINGS, there are several sections: SMTP/Email Settings for Alerts/Notifications, SNMP Trap Destination Settings, System, Usage Tracking, Quality of Service, and Auto-Tiering. The Auto-Tiering section is expanded and highlighted with a red oval. It contains three configuration items: 'Enable Auto-Tiering' (radio buttons for Enabled or Disabled, currently Enabled), 'Enable Per Bucket Credentials' (checkbox for Enabled, checked), and 'Enable Custom Endpoint' (checkbox for Disabled, unchecked). At the bottom of the Auto-Tiering section are 'Cancel' and 'Edit' buttons. Other collapsed sections include S3 Request Restrictions and Auto Repair.

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Note For an overview of the HyperStore auto-tiering feature, see "[Auto-Tiering Feature Overview](#)" (page 53). For a high level view of how these settings work together in combination -- or if you want to support auto-tiering to a HyperStore region or to a different HyperStore system -- see "[Setting Up Auto-Tiering](#)" (page 56).

Enable Auto-Tiering

This setting controls whether or not the CMC will support configuring auto-tiering rules for buckets. If you set "Enable Auto-Tiering" to **Enabled**, then the [CMC interface for configuring a bucket lifecycle](#) will include an option for configuring an auto-tiering rule. If "Enable Auto-Tiering" is set to **Disabled** (as it is by default), then the bucket lifecycle configuration interface will not display an option for configuring an auto-tiering rule.

Note that this setting only controls whether bucket lifecycle rules for auto-tiering can be configured **through the CMC**. Even if you have this setting disabled, bucket lifecycle rules for auto-tiering can still be configured through HyperStore extensions to the S3 API method "[PUT Bucket lifecycle](#)" (page 916), by other S3 client applications that call that API method.

If you set "Enable Auto-Tiering" to Enabled then additional settings (described below) will display in the "Auto-Tiering" section of the **Configuration Settings** page, allowing you control over the specific auto-tiering options that will be available to users configuring bucket lifecycle auto-tiering rules through the CMC.

Default = Disabled

Enable Per Bucket Credentials

This setting displays only if "Enable Auto-Tiering" is set to Enabled.

If "Enable Per Bucket Credentials" is **Enabled** (as it is by default), then users configuring bucket lifecycle auto-tiering rules through the CMC will be able to choose from several different options for tiering destination, and they will provide (through the CMC interface) their own security credentials for their selected tiering destination. By default the tiering destinations that will be available to users are:

- AWS S3 (default endpoint = <https://s3.amazonaws.com>)
- AWS Glacier (default endpoint = <https://s3.amazonaws.com>)
- Google Cloud Storage (default endpoint = <https://storage.googleapis.com>)
- Azure (default endpoint = <https://blob.core.windows.net>)
- Spectra BlackPearl (default endpoint = <https://bplab.spectralogic.com>) (note that Spectra BlackPearl is in the default list of tiering destinations only if you have upgraded from a HyperStore version older than 7.1.4 -- it is not in the default list for new installs of HyperStore 7.1.4 or later).

Note You can change this list of destinations and/or their endpoints if you wish. For details see "[Configure Tiering Destinations](#)" (page 58).

If you set "Enable Per Bucket Credentials" to **Disabled**, then all users configuring bucket lifecycle auto-tiering rules through the CMC will be presented with just one, system-default tiering destination, and all tiering to that destination will use a default set of security credentials that you supply to the system. You set this one default tiering destination and the one set of security credentials with the "Default Tiering URL" and "Default Tiering

"Credentials" settings that will display if you set "Enable Per Bucket Credentials" to Disabled. These settings are described further below.

Default = Enabled

Enable Custom Endpoint

This setting displays only if "Enable Auto-Tiering" and "Enable Per Bucket Credentials" are both set to Enabled.

If you set "Enable Custom Endpoint" to **Enabled**, then users configuring bucket lifecycle auto-tiering rules through the CMC will be able to choose from among the several tiering destinations described above under "Enable Per Bucket Credentials" **and also** they will have the option to enter an endpoint URL for an S3-compliant destination of their own choosing. Users will enter their own security credentials for their specified tiering destination. HyperStore will attempt to connect to the specified custom domain using the user's supplied credentials, and if the connection is successful the user will be able to auto-tier to that domain.

If "Enable Custom Endpoint" is **Disabled** (as it is by default), then users configuring bucket lifecycle auto-tiering rules through the CMC will not be able to specify a custom tiering destination endpoint. Instead they will choose among the several system-configured destinations (the default list of destination is described under "Enable Per Bucket Credentials" above).

Default =Disabled

Default Tiering URL

This setting displays only if "Enable Auto-Tiering" is set to Enabled and "Enable Per Bucket Credentials" is set to Disabled.

Configure a "Default Tiering URL" if you want all users setting bucket lifecycle auto-tiering rules through the CMC to tier to the same tiering destination, using the same account credentials. For example, if you want all users to tier to the same corporate account with Amazon S3 you could set the "Default Tiering URL" to <https://s3.amazonaws.com> -- or if your organization is based in California, USA, <https://s3.us-west-1.amazonaws.com>. Be sure to include the <https://> part when configuring your default tiering URL.

If you use the "Default Tiering URL" feature, you **must** set default tiering credentials.

Also, if you have a multi-region HyperStore system, note that the "Default Tiering URL" will apply to all of your service regions.

Note If you set "Enable Per Bucket Credentials" to Disabled and set a "Default Tiering URL", this configuration will **override** the list of destinations configured in the "**cmc_bucket_tiering_default_destination_list**" (page 485) setting in [common.csv](#). Only your specified "Default Tiering URL" will display for users who are using the CMC to configure auto-tiering for their buckets.

Default Tiering Credentials

This setting displays only if "Enable Auto-Tiering" is set to Enabled and "Enable Per Bucket Credentials" is set to Disabled.

Default tiering credentials (access key and secret key) to use with the "Default Tiering URL". If you configure a default tiering URL you must also configure default tiering credentials. HyperStore will then use those credentials and that tiering URL whenever users configure auto-tiering on their buckets through the CMC.

5.7.6.7. S3 Request Restriction Settings

The settings in this section of the CMC's **Configuration Settings** page place restrictions on the behaviors of S3 client applications that interface with the HyperStore system.

Put Object Maximum Size (Bytes)	5000000000	Edit
Multipart Upload Maximum Parts	10000	Edit
Maximum Buckets Per User	100	Edit
Maximum Objects Per Multi-Object Delete	1000	Edit

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Put Object Maximum Size (Bytes)

The maximize single-part object size that S3 clients will be allowed to upload to the system through a PUT Object request or POST Object request, in number of bytes. If a client submits a PUT Object or POST Object request with an object size larger than this many bytes, the S3 Service rejects the request.

In the case of Multipart Upload operations, this maximum size restriction applies to each individual uploaded part. If a single part is larger than Put Object Maximum Size, the S3 Service rejects the Upload Part request.

For best upload performance it's recommended for S3 clients to use the S3 [Initiate Multipart Upload](#) operation for objects larger than 100MB (rather than PUT Object or POST Object).

Default = 5368709120 (5 GiB)

Note In HyperStore version 7.1.x and older, the default was 5000000000 (5 GB). If you upgraded from HyperStore 7.1.x, your system retains the default of 5000000000 unless you change the setting through the CMC's **Configuration Settings** page.

Multipart Upload Maximum Parts

Maximum number of parts to allow per [Initiate Multipart Upload](#) request.

Each individual part can be no larger than the configured "Put Object Maximum Size (Bytes)" limit.

Default = 10000

Maximum Buckets Per User

Maximum number of S3 storage buckets to allow per user. When a user has this many buckets and tries to create an additional bucket, the request is rejected with an error response.

Default = 100

5.7.6.8. Auto-Repair Schedule Settings

The settings in this section of the CMC's **Configuration Settings** page configure the scheduling of the HyperStore auto-repair feature.

The screenshot shows the Cloudian Configuration Settings page. The top navigation bar includes links for Analytics, Buckets & Objects, Users & Groups, Cluster (which is highlighted with a red oval), Alerts, Admin, and Help. Below the navigation is a secondary navigation bar with links for Data Centers, Nodes, Cluster Config (highlighted with a red oval), Storage Policies, Repair Status, and Operation Status. The main content area is divided into two tabs: CLUSTER INFORMATION and CONFIGURATION SETTINGS (highlighted with a red oval). The CONFIGURATION SETTINGS tab is active and displays a list of configuration categories: SMTP/Email Settings for Alerts/Notifications, SNMP Trap Destination Settings, System, Usage Tracking, Quality of Service, Auto-Tiering, and S3 Request Restrictions. A section titled "Auto Repair Schedule" is expanded, showing three repair intervals: Replicas Repair Interval (Minutes) set to 43200, EC Repair Interval (Minutes) set to 41760, and Cassandra Full Repair Interval (Minutes) set to 10080. Each row has an "Edit" link to its right. At the bottom right of this section is a large green "SAVE" button.

After making any edits, click **Save** at the bottom of the page to dynamically apply the configuration change to the cluster.

Note For more information about the auto-repair feature see "[Data Repair Feature Overview](#)" (page 75)

Replicas Repair Interval (Minutes)

For each HyperStore node, the interval (in minutes) between scheduled auto-repairs of replicated object data.

The auto-repair feature automatically executes the [hsstool repair](#) operation to repair each node at the configured interval.

The system maintains a queue of nodes scheduled for auto-repair of replicated object data, and the queue is processed in such a way that within a service region *hsstool repair* will run on only one target node at a time. Repair of the node that is next in queue will not start until the repair operation completes on the node on which *hsstool repair* is currently running. Consequently if you have a lot of nodes and/or a high volume of data in your system, the actual time between repairs of a given node may be longer than the scheduled interval.

Note that when a replica repair operation is running on a target node, the scope of repair activity will extend to other nodes as well. In particular, repair of a target node will also make sure that for objects that fall within the target node's primary token range, the objects' replicas also are present on the other nodes where they are supposed to be.

Auto-repair of the replicated object data in your cluster is a resource-intensive operation and should be configured to run infrequently.

Default = 43200 (every 30 days)

Note If you wish, you can have some or all of the auto-repairs of replica data use the "computedigest" option to combat bit rot. This feature is controlled by the "[auto_repair_computedigest_run_number](#)" (page 468) setting in *common.csv*. By default "computedigest" is not used in auto-repair runs.

Note The system allows repair operations of different types -- such as an *hsstool repair* operation and an *hsstool repairec* operation -- to run concurrently within the same service region.

EC Repair Interval (Minutes)

For each HyperStore node, the interval (in minutes) between scheduled auto-repairs of erasure coded object data. The auto-repair feature automatically executes the [hsstool repairec](#) operation to repair each node at the configured interval.

The system maintains a queue of nodes scheduled for auto-repair of erasure coded object data, and the queue is processed in such a way that within a service region *hsstool repairec* will run on only one target node at a time. Repair of the node that is next in queue will not start until the repair operation completes on the node on which *hsstool repairec* is currently running. Consequently if you have a lot of nodes and/or a high volume of data in your system, the actual time between repairs of a given node may be longer than the scheduled interval.

Note that for erasure coded object data repair, for single data center storage policies and for multi- data center replicated EC storage policies, repair of a target node has the effect of assessing and repairing all erasure coded data on all nodes within the data center where the target node resides. And for multi- data center distributed EC storage policies, repair of a target node has the effect of assessing and repairing all erasure coded data in all of the participating data centers.

In a multi- data center HyperStore service region, the erasure coded data auto-repair queue is ordered in such a way that the target nodes alternate among the data centers -- for example after a repair completes on a target node in DC1, then the next target node will be from DC2, and then after that completes the next target node will be from DC3, and then after that completes the next target node will be from DC1 again, and so on.

Auto-repair of the erasure coded object data in your cluster is a resource-intensive operation and should be configured to run infrequently.

Default = 41760 (every 29 days)

Note If you wish, you can have some or all of the auto-repairs of erasure coded data use the "computedigest" option to combat bit rot. This feature is controlled by the "**"auto_repair_computedigest_run_number"** (page 468) setting in *common.csv*. By default "computedigest" is not used in auto-repair runs.

Note The system allows repair operations of different types -- such as an *hsstool repair* operation and an *hsstool repairrec* operation -- to run concurrently within the same service region.

Cassandra Full Repair Interval (Minutes)

For each HyperStore node, the interval (in minutes) between automatically running a full repair of the node's Cassandra data (system metadata and object metadata stored in Cassandra). It is essential to run a full Cassandra data repair on each node at an interval **less than 10 days** (less than the Cassandra *gc_grace_seconds* interval, which by default is 10 days).

The system maintains a queue of nodes scheduled for auto-repair of Cassandra data, and the queue is processed in such a way that within a service region Cassandra repair will run on only one target node at a time. Repair of the node that is next in queue will not start until the repair operation completes on the node on which a Cassandra repair is currently running.

Note that when a Cassandra repair operation is running on a target node, the scope of repair activity will extend to other nodes as well. In particular, repair of a target node will also make sure that for metadata row keys that fall within the target node's primary token range, the metadata's replicas also are present on the other nodes where they are supposed to be.

Default = 10080 (every seven days)

5.7.7. Storage Policies

Path: Cluster → Storage Policies

REGION	STATUS	NAME	DESCRIPTION	DATA DISTRIBUTION POLICY	NO OF REPLICAS	LOCAL EC
us-ca	ACTIVE	newpolicy	HSFS - DC1:3	Single DC	3	N/A

Supported tasks:

- "**Add a Storage Policy**" (page 308)
- "**Edit a Storage Policy**" (page 331)
- "**Designate a Default Storage Policy**" (page 331)
- "**Disable a Storage Policy**" (page 332)
- "**Delete a Storage Policy**" (page 332)

Note For detailed information on S3 write and read availability under various combinations of cluster size, storage policy configuration, and number of nodes down, see "**Storage Policy Resilience to Downed Nodes**" (page 148).

5.7.7.1. Add a Storage Policy

Storage policies are ways of protecting data so that it's durable and highly available to users. The HyperStore system lets you pre-configure one or more storage policies. Users when they create a new storage bucket can then choose which pre-configured storage policy to use to protect data in that bucket. **Users cannot create buckets until you have created at least one storage policy.**

For each storage policy that you create you can choose and configure either of two data protection methods: replication or erasure coding. If your HyperStore system spans multiple data centers, for each storage policy you can also choose how data is allocated across your data centers.

Note In your system you must have a [default storage policy](#). If you have not yet created any storage policies, the first policy that you create must be configured to be visible to all groups, and it will automatically become the default policy. Subsequently, after creating additional policies, you can designate a different policy as the default policy if you wish.

In a **multi-region system**, each region must have its own storage policy or policies. When you create storage policies, while configuring each policy's "Data Center Assignment" you will specify the region with which the policy is associated. Each region must have a default storage policy.

To add a storage policy:

1. In the CMC's [Storage Policies](#) page click **Create Storage Policy**. This opens the **Create New Policy** interface.

STORAGE POLICIES

CREATE NEW POLICY

Policy Name Policy Description

NUMBER OF DATACENTERS 1

DATA DISTRIBUTION SCHEME

Replicas Within Single Datacenter

EC Within Single Datacenter

NUMBER OF REPLICAS 3

DATACENTER ASSIGNMENT

REGION	DATACENTER	REPLICA	LOCAL EC
sfo-del-region1	DC1	1 of 3	
		2 of 3	disable
		3 of 3	

CONSISTENCY SETTING

CONSISTENCY LEVEL	READ	WRITE
ALL	<input type="checkbox"/>	<input type="checkbox"/>
QUORUM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ONE	<input type="checkbox"/>	

GROUP VISIBILITY

Please select a Group **ADD**

Compression Type NONE

Server-side Encryption NONE

SAVE **CANCEL**

2. Complete the sections of the **Create New Policy** interface.

- [Policy Name and Description](#)
- [Data Distribution Scheme and Data Center Assignment](#)
- [Consistency Setting](#)
- [Group Visibility](#)
- [Compression](#)
- [Server-Side Encryption](#)

Note: If you have configured Elasticsearch integration for your HyperStore system, then when you create (or edit) a storage policy you will have an option to "Enable metadata search". For information about this feature see "**Elasticsearch Integration for Object Metadata**" (page 106).

- Click **Save** to create the new policy in the system. The policy will then appear in your storage policy list in the CMC's **Storage Policies** page. The new policy will initially show a status of PENDING, but if you check again a few minutes later it should have a status of ACTIVE, at which point the policy is available to users.

Note: In the unlikely event that the new policy fails to be created in the system, in your policy list the policy will appear with status FAILED. In this case you can delete the failed policy and then try again to configure and save a new policy with your desired settings.

Create New Storage Policy -- Policy Name and Description

Enter a Policy Name (only letters, numbers, dashes, and underscores are allowed; maximum 32 characters) and also a Policy Description that will be meaningful to your users (maximum 64 characters). The Policy Name and Policy Description are important because users will see this text when they are choosing a storage policy to apply to a newly created storage bucket.

How End Users Will See the Policy Name and Description

In the CMC's **Buckets** interface, a user creating a bucket will see a drop-down menu listing all the pre-configured storage policies by name, and when they select a policy they will see its description (before they complete the process of creating the bucket). So, create policy names and descriptions that will be meaningful to your service users.

For example, suppose your HyperStore spans two data centers, one in Chicago and one in Kansas City. You create one storage policy that uses replication and stores the data in both of your data centers, and another policy that uses erasure coding and stores data only in Kansas City. If the people who will be creating storage buckets are fairly technical, then in creating these two policies you might use names and descriptions like:

Policy Name	Policy Description
Replication_Chicago-3X_KC-2X	3 replicas stored in Chicago and 2 in Kansas City
Erasure_coding_KC-only	Objects are erasure coded and stored in Kansas City only

If your users are non-technical, you might gear your policy descriptions more towards the type of data that users intend to store. Since replication is commonly used for more active data while erasure coding is suitable to "cold" data, you might do something like:

Policy Name	Policy Description
Active_High-Availability	For "live" data that is accessed often
Cold_Archival_Storage	For "cold" data that is rarely accessed

Note that if your HyperStore system has multiple service regions and/or multiple data centers, it may be important to communicate data storage location information to end users through the Policy Name and Policy Description — especially if your system spans multiple countries.

Create New Storage Policy -- Data Distribution Scheme and Data Center Assignment

In the "Data Distribution Scheme" and "Data Center Assignment" sections of the [Create New Policy](#) interface, choose and configure a scheme for protecting S3 object data and object metadata.

First, from the "Number of Data Centers" drop-down list, choose the **number of data centers in which this new policy should store data**. If you want you can specify a number here that's smaller than your total number of data centers -- for example, if you have three DCs in your system but you want to create a policy that stores data in only two of those DCs, then select 2 here. Later in the policy creation process you will be able to specify exactly which DCs to use for this policy.

Next, choose one of the following data distribution schemes. Only the schemes appropriate to your specified number of DCs will display in the interface.

Note For detailed information on S3 write and read availability under various combinations of cluster size, data distribution scheme, consistency level settings, and number of nodes down, see "[Storage Policy Resilience to Downed Nodes](#)" (page 148).

Replicas Within Single Data Center

NUMBER OF DATACENTERS

DATA DISTRIBUTION SCHEME

Replicas Within Single Datacenter



EC Within Single Datacenter



NUMBER OF REPLICAS

DATACENTER ASSIGNMENT

REGION	DATACENTER	REPLICA	LOCAL EC
region1	DC1	1 of 3	
		2 of 3	disable
		3 of 3	

This scheme protects S3 object data by replicating it within a single data center.

When you choose this option, the interface displays a "Number of Replicas" field in which you can specify how many replicas you want. For example, if you specify 3, then with this storage policy the system will maintain a total of 3 copies of each S3 data object, each on a different node.

The system will also maintain the same number of replicas of each S3 object's metadata (in Cassandra), again with each replica on a different node. The object metadata key has its own hash value (token) and thus HyperStore's token-based data distribution mechanism may allocate the object metadata to different nodes than the object data.

You **cannot choose a number of replicas that is greater than the number of HyperStore nodes in the data center**. For example, if you have only 3 nodes in the data center, then 3 is the maximum number of replicas that you can choose.

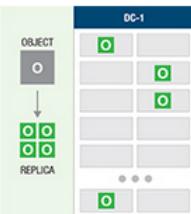
Also, you **cannot choose fewer than 3 replicas** unless you have only 1 or 2 nodes in your system (such as when doing simple evaluation or testing of HyperStore). In a production environment, 1X or 2X replication provide inadequate data protection and consequently those settings are not allowed.

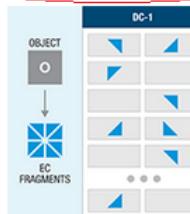
If you have only one DC in your system, then there is nothing that you need to do in the "Data Center Assignment" section of the interface. If you have more than one DC in your system, use the "Data Center Assignment" section to select which one of your DCs to use for this policy.

EC Within Single Data Center

NUMBER OF DATACENTERS

Replicas Within Single Datacenter
 EC Within Single Datacenter ?

OBJECT

REPLICA

OBJECT

EC FRAGMENTS

ERASURE CODING K+M VALUE

DATACENTER ASSIGNMENT

REGION	DATACENTER	REPLICA	LOCAL EC
region1	DC1	1 of 1	4+2

This scheme protects S3 object data by erasure coding the data, within a single data center.

When you choose this option, the interface lets you select which of the supported EC "k"+m" configuration schemes to use. For example:

- **4+2** — Each object will be encoded into 4 data fragments plus 2 parity fragments, with each fragment stored on a different node. Objects can be read so long as any 4 of the 6 fragments are available.
- **6+2** — Each object will be encoded into 6 data fragments plus 2 parity fragments, with each fragment stored on a different node. Objects can be read so long as any 6 of the 8 fragments are available.

In addition to the options above, the system by default also supports:

- 8+2
- 9+3
- 12+4

The choice among these supported EC configurations is largely a matter of how many HyperStore nodes you have in the data center. For example, compared to a 4+2 configuration, 6+2 EC provides the same degree of data availability assurance (stored objects can be read even if 2 of the involved nodes are unavailable), while delivering a higher level of storage efficiency based on the parity fragments as a percentage of the data fragments (4+2 incurs 50% overhead whereas 6+2 incurs only 33% overhead). So 6+2 may be preferable to 4+2 if you have at least 8 HyperStore nodes in the data center.

Likewise, 9+3 EC provides a higher degree of protection and availability than 6+2 EC (since with 9+3 EC, stored objects can be read even if 3 of the involved nodes are unavailable) while delivering the same level of storage efficiency (both 6+2 and 9+3 incur 33% storage overhead). So 9+3 may be preferable to 6+2 if you have at least 12 HyperStore nodes in the data center.

You **cannot choose a k+m configuration that totals to more than the number of HyperStore nodes in the data center**. For example, if you have 7 nodes in the data center, you can choose 4+2 as your k+m configuration, but not 6+2.

If you have only one DC in your system, then there is nothing that you need to do in the "Data Center Assignment" section of the interface. If you have more than one DC in your system, use the "Data Center Assignment" section to select which one of your DCs to use for this policy.

IMPORTANT: Whatever $k+m$ configuration you choose, the system will maintain either $(2m)+1$ or $(2m)-1$ replicas of each object's metadata (in Cassandra). For more detail see "**Storage of Object Metadata**" (page 144).

Consult with Cloudian Support to ensure that the disks or SSDs on which you are storing Cassandra data have sufficient capacity to support your desired $k+m$ configuration with its associated metadata storage requirements as well as the temporary storage overhead needed during Cassandra repair operations.

Note If you want to use a $k+m$ configuration other than those listed in the CMC interface, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

Replication Across Data Centers

NUMBER OF DATACENTERS

DATA DISTRIBUTION SCHEME

Replication Across Datacenters

Replicated EC

NUMBER OF REPLICAS

DATACENTER ASSIGNMENT

REGION	DATACENTER	REPLICA	LOCAL EC
	DC1	1 of 5	
	DC1	2 of 5	
region1	DC1	3 of 5	disable
	DC1	4 of 5	
	DC1	5 of 5	

This distribution scheme is supported only for HyperStore systems that span multiple data centers. Choose this option if you want S3 objects to be replicated and to have those replicas distributed across DCs.

When you choose this option, the interface displays a "Number of Replicas" field in which you can specify the **total** number of replicas that you want, across multiple data centers. For example, if for each S3 object you want 3 replicas stored in DC1 and 2 replicas in DC2, enter 5 in the "Number of Replicas" field.

You can then use the "Data Center" drop-down lists to select which one of your data centers will house **each replica**. This interface allows you to define how many replicas will be stored in each specific data center. In the example below, 3 replicas will be stored in DC1 and 2 replicas will be stored in DC2.

DATACENTER ASSIGNMENT			
REGION	DATACENTER	REPLICA	LOCAL EC
	DC1	1 of 5	
	DC1	2 of 5	
region1	DC1	3 of 5	disable
	DC2	4 of 5	
	DC2	5 of 5	

Note that the ordering of the replicas doesn't matter — what matters is the total number of replicas that you assign to each data center. (If you have a multi-region system, first choose a region then choose the DCs from within that region).

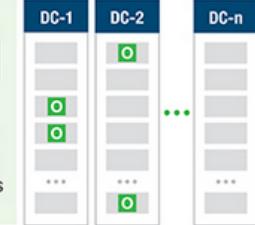
Note The system will also maintain the same number of replicas of each S3 object's metadata (in Cassandra), with the same distribution across data centers. The object metadata key has its own hash value (token) and thus HyperStore's token-based data distribution mechanism may allocate the object metadata to different nodes than the object data.

Replicated EC

NUMBER OF DATACENTERS

DATA DISTRIBUTION SCHEME

Replication Across Datacenters



Replicated EC



ERASURE CODING K+M VALUE

4+2

6+2

8+2

9+3

12+4

DATACENTER ASSIGNMENT

Please select the correct amount of DCs.

REGION	DATACENTER	SELECTED
	DC1	<input type="checkbox"/>
region1	DC2	<input type="checkbox"/>
	DC3	<input type="checkbox"/>

With Replicated EC, each object is encoded into "k" data fragments + "m" parity fragments and that set of "k"+"m" fragments is replicated in multiple data centers. With this approach, each participating data center stores the full set of an object's erasure coded fragments (each participating data center stores "k" + "m" fragments). An object can be read so long as a combined total of at least "k" fragments are available across the participating DCs.

In the "Erasure Coding k+m Value" drop-down list you can choose from among these options that the system supports by default:

- 4+2
- 6+2
- 8+2
- 9+3
- 12+4

Next, in the "Data Center Assignment" section of the interface choose which of your DCs you want to participate in this storage policy. (If you have a multi-region system, first choose a region then choose the DCs from within that region).

Note that:

- For this type of distribution scheme, each participating data center will use the same "k"+m" configuration (the configuration you selected from the "Erasure Coding k+m Value" drop-down list). You cannot create a policy that uses 4+2 in one data center while using 6+2 in a different data center, for instance.
- In each participating data center, for a given object each of the "k"+m" erasure coded fragments will be stored on a different node. Therefore you cannot choose a "k"+m" configuration that exceeds the number of nodes in any participating data center. For example, if one of the data centers that you want to use for the storage policy has 8 nodes and the second data center has 6 nodes, you can use replicated 4+2 erasure coding but not replicated 6+2 erasure coding (since the second data center doesn't have enough nodes to support 6+2 erasure coding).
- As compared to the "EC Across Data Centers" option, the "Replicated EC" option:
 - Is less efficient in using storage capacity, since there is more storage overhead per stored object. This is because for each object each DC will have a full "k"+m" set of fragment replicas, whereas for EC Across Data Centers there is just one "k"+m" set of fragments which is spread across multiple DCs. Mathematically, the storage overhead percentage (redundant data size as a percentage of original data size) for Replicated EC is #DCs(m/k) whereas for EC Across Data Centers it's m/k.
 - Reduces the chance of objects being unavailable, since there are additional redundant fragments and objects are available so long as a total of "k" fragments are readable across the participating data centers. For example, with 4+2 EC replicated in two DCs there are a total of 12 stored fragments and objects remain readable even if 8 of those fragments are unreachable. By contrast, for EC Across Data Centers an object will be unreadable if more than "m" endpoints in total are unavailable across all the participating DCs.
 - Reduces read latency since typically an object will be decodable (readable) from the fragments within the local data center (the data center processing the S3 request from a client application).

IMPORTANT: In each data center included in the policy, the system will maintain either $(2m)+1$ or $(2m)-1$ replicas of each object's metadata (in Cassandra). For more detail see "**Storage of Object Metadata**" (page 144).

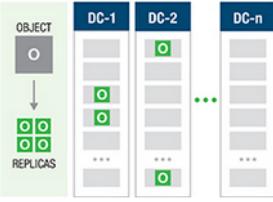
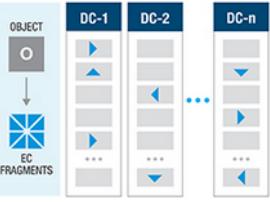
Consult with Cloudian Support to ensure that the disks or SSDs on which you are storing Cassandra data have sufficient capacity to support your desired $k+m$ configuration with its associated metadata storage requirements as well as the temporary storage overhead needed during Cassandra repair operations.

Note If you want to use a $k+m$ configuration other than those listed in the CMC interface, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

EC Across Data Centers

NUMBER OF DATACENTERS

DATA DISTRIBUTION SCHEME

- Replication Across Datacenters
 
- Replicated EC
 
- EC Across Datacenters
 

ERASURE CODING K+M VALUE

7+5

DATACENTER ASSIGNMENT

Please select the correct amount of DCs.

REGION	DATACENTER	SELECTED
region1	DC1	<input type="checkbox"/>
	DC2	<input type="checkbox"/>
	DC3	<input type="checkbox"/>

With EC Across Data Centers, each object is encoded into "k" data fragments + "m" parity fragments and then that one set of "k"+"m" fragments is spread out evenly across the participating data centers. An object can be read so long as a combined total of at least "k" fragments are available across the multiple DCs.

In the "Erasure Coding k+m Value" drop-down list you can choose your desired "k"+"m" configuration. Your options are determined by the number of DCs that you specified in the "Number of Data Centers" drop-down list. The table below shows the default supported options and how the fragments will be distributed.

# of Participating DCs	Supported "k"+"m"	How Fragments Will Be Distributed
3	5+4	3 fragments per DC
	7+5	4 fragments per DC
4	8+4	3 fragments per DC
5	6+4	2 fragments per DC
6	8+4	2 fragments per DC
	7+5	2 fragments per DC
7	10+4	2 fragments per DC
8	10+6	2 fragments per DC
9	10+8	2 fragments per DC

After selecting your "k"+m configuration, use the "Data Center Assignment" section of the interface to choose which of your DCs you want to participate in this storage policy. (If you have a multi-region system, first choose a region then choose the DCs from within that region).

Note that:

- You must have at least 3 data centers within a region to use this type of data distribution scheme.
- The number of nodes in each participating DC must be at least as many as the number of fragments that will be distributed to each DC. For example, to use 7+5 erasure coding spread across 3 data centers you need at least 4 nodes in each of those DCs (since each DC will be allocated 4 fragments from each S3 object and each fragment must be stored on a different node).
- The supported options are such that objects are readable even if one of the participating DCs goes down, so long as a sufficient number of endpoints are live in the remaining DCs. (Put differently, in all the supported options the number of fragments per DC is never greater than "m".)
- As compared to the "Replicated EC" option, the "EC Across Data Centers" option:
 - Is more efficient in using storage capacity, since there is less storage overhead per stored object. This is because there is just one "k"+m set of fragments which is spread across multiple DCs, rather than each DC having a full "k"+m set of fragments (as is the case with Replicated EC). Mathematically, the storage overhead percentage (redundant data size as a percentage of original data size) for EC Across Data Centers is m/k whereas for Replicated EC it's $\#DCs(m/k)$.
 - Entails a somewhat higher chance of objects being unavailable, since for a given object if more than a total of "m" endpoints (nodes on which the object's fragments are stored) are down across the multiple DCs, the system will not be able to decode the object. By contrast, with Replicated EC there is additional redundancy of fragments and a greater percentage of an object's fragments can be lost or unreachable without impacting the object's readability.
 - Entails a somewhat higher read latency since all object reads will necessarily require communication across DCs (since no one DC has the entire "k" set of fragments that's required for an object read).

IMPORTANT: Distributed across all the data centers included in the policy, the system will maintain a total of either $(2m)+1$ or $(2m)-1$ replicas of each object's metadata (in Cassandra). For more detail see "[Storage of Object Metadata](#)" (page 144).

Consult with Cloudian Support to ensure that the disks or SSDs on which you are storing Cassandra data have sufficient capacity to support your desired $k+m$ configuration with its associated metadata storage requirements as well as the temporary storage overhead needed during Cassandra repair operations.

Note If you want to use a $k+m$ configuration other than those listed in the CMC interface, contact Cloudian Support or your Cloudian Sales representative to see whether your desired configuration can be supported.

Create New Storage Policy -- Consistency Setting

In the "Consistency Setting" section of the [Create New Policy](#) interface, configure data consistency levels to apply to this storage policy. Consistency levels impose requirements as to **what portion of the data and metadata reads or writes associated with a given S3 request must be successfully completed within the cluster before the system can return a success response to the S3 client**. If the consistency requirements

cannot be met for a given S3 request at a given time -- for example, due to one or more endpoint nodes being inaccessible -- an HTTP 503 error response is returned to the client.

Note For detailed information on S3 write and read availability under various combinations of cluster size, data distribution scheme, and consistency level settings, see "**Storage Policy Resilience to Downed Nodes**" (page 148).

Your consistency level options depend on which data distribution scheme you chose for the policy you are creating:

Replicas Within Single Data Center

With a "Replicas within Single Data Center" scheme you can choose from the following consistency levels.

Read	Write
ALL	ALL
QUORUM (default)	QUORUM (default)
ONE	--

EC Within Single Data Center

With an "EC within Single Data Center" scheme you can choose from the following consistency levels.

Read (metadata only)	Write
ALL	ALL
QUORUM (default)	QUORUM (default)

Note For this type of scheme the consistency requirement for reading object data is always "k" unique fragments. Your "Read" consistency setting impacts only the reading of object metadata.

Replication Across Data Centers

With a "Replicas Across Data Centers" scheme you can choose from the following consistency levels.

Read	Write
ALL	ALL
--	EACH QUORUM
QUORUM (default)	QUORUM (default)
LOCAL QUORUM	LOCAL QUORUM
ONE	--

Replicated EC

With a "Replicated EC" scheme you can choose from the following consistency levels.

Read	Write
ALL	ALL

Read	Write
--	EACH QUORUM
LOCAL QUORUM	LOCAL QUORUM
ANY QUORUM (default)	ANY QUORUM (default)

Note For this type of scheme the consistency requirement for reading object data is always "k" unique fragments. If you choose "Local Quorum" for read, the system will only read from the local data center when trying to get "k" unique fragments. With the other settings the system will read from all data centers when trying to get "k" unique fragments.

EC Across Data Centers

With an "EC Across Data Centers" scheme you can choose from the following consistency levels.

Read (metadata only)	Write
ALL	ALL
QUORUM (default)	QUORUM (default)

Note For this type of scheme the consistency requirement for reading object data is always "k" unique fragments. Your "Read" consistency setting impacts only the reading of object metadata.

Note When configuring consistency levels you would typically choose just one consistency level for each operation type, by selecting one consistency level checkbox for Read and one for Write. However, HyperStore also supports an advanced option known as "**Dynamic Consistency Levels**" (page 29), whereby you can configure both a primary consistency level and a fallback consistency level to be used in instances when the primary consistency level cannot be achieved. In the CMC interface you can do this by selecting more than one consistency level checkbox for a given operation type.

Consistency Levels

To boost data durability and availability, HyperStore implements replication or erasure coding for object data and replication for object metadata. This entails distributing each object's data and metadata to multiple endpoint nodes across the cluster. When you create storage policies, along with configuring a replication or erasure coding scheme you will also configure consistency levels for writes and reads. Consistency levels impose requirements as to what portion of the data and metadata writes or reads associated with each S3 request must be successfully completed before the system can return a success response to the S3 client. If the consistency requirements cannot be met for a given S3 request at a given time -- for example, due to one or more endpoint nodes being inaccessible -- an HTTP 503 error response is returned to the client.

Below is the list of consistency levels supported by the HyperStore system. Your consistency level options when configuring a storage policy will be limited by the data distribution scheme (replication or erasure coding, single DC or multi-DC) that you have selected for that policy.

- **"Consistency Level "ALL""** (page 323)
- **"Consistency Level "QUORUM""** (page 328)
- **"Consistency Level "EACH QUORUM""** (page 325)

- "**Consistency Level "LOCAL QUORUM"**" (page 326)
- "**Consistency Level "ANY QUORUM"**" (page 324)
- "**Consistency Level "ONE"**" (page 327)

For detailed information on S3 write and read availability under various combinations of cluster size, data distribution scheme, and consistency level settings, see "**Storage Policy Resilience to Downed Nodes**" (page 148).

Note In the case of writes, if the consistency requirement is met by something less than completing writes of all replicas (or all erasure coded fragments), then after returning a success response to the client the system continues to try to complete the remaining writes. If any of these writes fail they will later be recreated by [automatic data repair](#).

Note As an advanced option you can also configure "dynamic" consistency levels, whereby the system will try to achieve a "fallback" consistency level if the primary consistency level cannot be achieved. For more information see "**Dynamic Consistency Levels**" (page 29).

Note About Object Data Replica Reads

For replication based storage policies, the descriptions and examples in this documentation sometimes state that part of the read consistency requirement is being able to read X number of object data replicas. This is a simplification. Technically, what needs to be readable is X number of object data replica **digests** (which are stored in RocksDB on the same disks as the actual replica data). If the read consistency requirements are met for an S3 GET operation -- for reading object data replica digests (in RocksDB) and for reading object metadata replicas (in Cassandra) -- then the system retrieves just one object data replica in order to return the object data to the S3 client. Because each object data replica's digest is on the same disk as the replica data itself, saying that X number of object data replicas must be readable is a convenient approximation of the actual technical requirement that X number of object data replica digests must be readable.

Note About Bucket Content List Reads

In the documentation of the supported consistency levels such as "ALL", "QUORUM", and so on (see the cross references above), when read consistency requirements are discussed the focus is on reads of individual objects -- that is, the consistency requirements for successfully implementing S3 *GET Object* requests. It's worth noting however that your configured read consistency requirements also apply to bucket content list reads -- that is, implementing S3 *GET Bucket (List Objects)* requests.

Metadata for objects is stored in two different types of record in Cassandra: object-level records (with one such record for each object) and bucket-level records that identify the objects in a bucket (along with some metadata for each of those objects). Both types of object metadata are replicated to the same degree. So for example, in a 3X replication storage policy, for each object the object-level metadata record is replicated three times in the cluster and for each bucket the bucket-level object metadata records are replicated three times in the cluster.

A *GET Object* request requires reading the object's object-level metadata record and a *GET Bucket (List Objects)* request requires reading the bucket's bucket-level object metadata records. Whatever read consistency requirements you set for a storage policy apply not only to reads of individual objects but also to reads of buckets content lists. So for example if you use a QUORUM read consistency requirement, then in order to successfully execute a *GET Bucket (List Objects)* request the system must be able to read a QUORUM of the bucket-level object metadata records for the bucket.

For more on the meaning of QUORUM and the other supported consistency levels, see the cross references above.

Consistency Level "ALL"

The consistency level "ALL" is a supported option for every type of data distribution scheme, for both reads and writes. The table below shows the general requirements of ALL in the context of replication based storage policies and erasure coding based storage policies. Following the table are examples for each supported data distribution scheme.

	Replication	Erasure Coding
Read ALL	For an S3 GET to succeed, the system must succeed in reading all object data replicas and all object metadata replicas.	For an S3 GET to succeed, the system must succeed in reading "k" object data fragments and all object metadata replicas.
Write ALL	For an S3 PUT to succeed, the system must succeed in writing all object data replicas and all object metadata replicas.	For an S3 PUT to succeed, the system must succeed in writing all ("k"+"m") object data fragments and all object metadata replicas.

Example for a "Replication Within Single Data Center" policy

Suppose a "Replication Within Single Data Center" policy uses 3X replication. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas.

If the policy's consistency level for Read is set to ALL, then during an S3 GET operation all 3 object data replicas and all 3 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to ALL, then during an S3 PUT operation all 3 object data replicas and all 3 object metadata replicas must be successfully written before a success response is returned to the client.

Example for an "EC Within Single Data Center" policy

Suppose an "EC Within Single Data Center" policy uses 4+2 erasure coding. With this configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default).

If the policy's consistency level for Read is set to ALL, then during an S3 GET operation 4 object data fragments and all 3 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to ALL, then during an S3 PUT operation all 6 object data fragments and all 3 object metadata replicas must be successfully written before a success response is returned to the client.

Example for a "Replication Across Data Centers" policy

Suppose a "Replication Across Data Centers" policy uses 3X replication in DC-East and 2X replication in DC-West. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas in DC-East, and 2 object data replicas and 2 object metadata replicas in DC-West.

If the policy's consistency level for Read is set to ALL, then during an S3 GET operation all 5 object data replicas and all 5 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to ALL, then during an S3 PUT operation all 5 object data replicas and all 5 object metadata replicas must be successfully written before a success response is returned to the client.

Example for a "Replicated EC" policy

Suppose a "Replicated EC" policy uses 4+2 erasure coding replicated in DC-East and DC-West. With this

configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default) in each of those DCs.

If the policy's consistency level for Read is set to ALL, then during an S3 GET operation a total of 4 unique object data fragments (from among all the object's fragments in the two DCs) as well as all 6 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to ALL, then during an S3 PUT operation all 12 object data fragments and all 6 object metadata replicas must be successfully written before a success response is returned to the client.

Example for an "EC Across Data Centers" policy

Suppose an "EC Across Data Centers" policy uses 7+5 erasure coding distributed across DC-East, DC-West, and DC-South. With this configuration, for each object the system will store 4 object data fragments in each of those DCs, as well as 9 object metadata replicas (2m-1, by default) spread approximately evenly across the DCs.

If the policy's consistency level for Read is set to ALL, then during an S3 GET operation a total of 7 unique object data fragments (from among all the object's fragments in the three DCs) as well as all 9 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to ALL, then during an S3 PUT operation all 12 object data fragments and all 9 object metadata replicas must be successfully written before a success response is returned to the client.

Consistency Level "ANY QUORUM"

The consistency level "ANY QUORUM" is supported only in "Replicated EC" data distribution schemes, for read and write operations. The table below shows the general requirements of ANY QUORUM, and following the table is an example.

Replicated EC	
Read ANY QUORUM	For an S3 GET to succeed the system must succeed in reading a total of "k" unique erasure coded fragments from among the storage policy's participating data centers, and also reading enough object metadata replicas to satisfy either a LOCAL QUORUM or QUORUM consistency level (implemented as dynamic consistency levels). The system automatically sets up this object metadata consistency configuration when you choose ANY QUORUM as the read consistency for the storage policy, because Cassandra (in which object metadata is stored) does not natively support an ANY QUORUM consistency level.
Write ANY QUORUM	For an S3 PUT to succeed the system must succeed in writing "k"+1 erasure coded fragments within any one of the storage policy's participating data centers and also writing enough object metadata replicas to satisfy either a LOCAL QUORUM or QUORUM consistency level

Example for a "Replicated EC" policy

Suppose a "Replicated EC" policy uses 4+2 erasure coding replicated in DC-East and DC-West. With this configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default) in each of those DCs.

If the policy's consistency level for Read is set to ANY QUORUM, and an S3 GET request is received in DC-East (for example), then in order for that request to succeed the system must succeed in reading 4 unique object data fragments, and either 2 object metadata replicas in DC-East or a total of 4 object metadata replicas from the two DCs combined.

Note For the object data, reading 3 object data fragments in DC-East (for example) and 1 object data fragment in DC-West would satisfy the requirement as long as all those fragments are unique.

If the consistency level for Write is set to ANY QUORUM, and an S3 PUT request is received in DC-East, then in order for that request to succeed the system must succeed in writing 5 object data fragments either in DC-East or in DC-West, and writing either 2 object metadata replicas in DC-East or a total of 4 object metadata replicas in the two DCs combined.

Note For the object data, writing 3 object data fragments in DC-East (for example) and 2 object data fragments in DC-West would **not** satisfy the requirement, since the object data write quorum has to be achieved **within** one of the DCs.

Consistency Level "EACH QUORUM"

The consistency level "EACH QUORUM" is supported only in "Replication Across Data Centers" or "Replicated EC" data distribution schemes and only for write operations -- not reads. The table below shows the general requirements of EACH QUORUM in the context of replication based storage policies and erasure coding based storage policies. Following the table are examples for each supported data distribution scheme.

	Replication	Erasure Coding
Write EACH QUORUM	For an S3 PUT to succeed, in each of the storage policy's participating data centers the system must succeed in writing a majority of object data replicas and a majority of object metadata replicas.	For an S3 PUT to succeed, in each of the storage policy's participating data centers the system must succeed in writing " k "+1 object data fragments and a majority of object metadata replicas.

Note For replicated data or metadata, a quorum or majority of a set of replicas is defined mathematically as $(\#replicas/2) + 1$, rounded down to the nearest integer. For example with 3 replicas a quorum would be $(3/2 = 1.5) + 1 = 2.5$, rounded down = 2. With 4 replicas a quorum would be $(4/2 = 2) + 1 = 3$, rounded down = 3. With the EACH QUORUM consistency level the quorum must be achieved in each participating DC.

Example for a "Replication Across Data Centers" policy

Suppose a "Replication Across Data Centers" policy uses 3X replication in DC-East and 2X replication in DC-West. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas in DC-East, and 2 object data replicas and 2 object metadata replicas in DC-West.

If the policy's consistency level for Write is set to EACH QUORUM, then in order for an S3 PUT operation to succeed the system must succeed in writing 2 object data replicas and 2 object metadata replicas in DC-East, **and** 2 object data replicas and 2 object metadata replicas in DC-West.

Example for a "Replicated EC" policy

Suppose a "Replicated EC" policy uses 4+2 erasure coding replicated in DC-East and DC-West. With this configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default) in each of those DCs.

If the policy's the consistency level for Write is set to EACH QUORUM, then in order for an S3 PUT operation to succeed the system must succeed in writing 5 object data fragments and 2 object metadata replicas in DC-East, **and** 5 object data fragments and 2 object metadata replicas in DC-West.

Consistency Level "LOCAL QUORUM"

The consistency level "LOCAL QUORUM" is supported only in "Replication Across Data Centers" or "Replicated EC" data distribution schemes, for write and read operations. With LOCAL QUORUM it's important to understand these two points:

- In the context of a storage policy that encompasses multiple data centers, for any given S3 request **the "local" data center is the data center in which the request is received from the S3 client**. For example in a storage policy that encompasses DC1 and DC2, for all S3 requests received in DC1 the local data center is DC1, and for all S3 requests received in DC2 the local data center is DC2.
- With a LOCAL QUORUM consistency level, whether an S3 request succeeds or not is based **solely on what happens in the local data center**. What happens in the remote data center(s) is irrelevant to achieving a LOCAL QUORUM consistency level.

With those points in mind, the table below shows the general requirements of LOCAL QUORUM in the context of replication based storage policies and erasure coding based storage policies. Following the table are examples for each supported data distribution scheme.

	Replication	Erasure Coding
Read LOCAL QUORUM	For an S3 GET to succeed, the system must succeed in reading a majority of the local DC's object data replicas and a majority of the local DC's object metadata replicas.	For an S3 GET to succeed, the system must succeed in reading "k" object data fragments within the local DC and a majority of the local DC's object metadata replicas.
Write LOCAL QUORUM	For an S3 PUT to succeed, the system must succeed in writing a majority of the local DC's object data replicas and a majority of the local DC's object metadata replicas.	For an S3 PUT to succeed, the system must succeed in writing "k"+1 object data fragments within the local DC and a majority of the local DC's object metadata replicas.

Note For replicated data or metadata, a quorum or majority of a set of replicas is defined mathematically as $(\#replicas/2) + 1$, rounded down to the nearest integer. For example with 3 replicas a quorum would be $(3/2 = 1.5) + 1 = 2.5$, rounded down = 2. With 4 replicas a quorum would be $(4/2 = 2) + 1 = 3$, rounded down = 3. With the LOCAL QUORUM consistency level the requirement is to successfully read or write a quorum among the replicas in the local DC. For example if the local DC is assigned 3 replicas, a local quorum is constituted by 2 of those replicas; or if the local DC is assigned 4 replicas, a local quorum is constituted by 3 of those replicas.

Example for a "Replication Across Data Centers" policy

Suppose a "Replication Across Data Centers" policy uses 3X replication in DC-East and 2X replication in DC-West. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata

replicas in DC-East, and 2 object data replicas and 2 object metadata replicas in DC-West.

If the policy's consistency level for Read is set to LOCAL QUORUM, and an S3 GET request is received in DC-East (for example), then in order for that request to succeed the system must succeed in reading 2 object data replicas and 2 object metadata replicas in DC-East.

If the consistency level for Write is set to LOCAL QUORUM, and an S3 PUT request is received in DC-East (for example), then in order for that request to succeed the system must succeed in writing 2 object data replicas and 2 object metadata replicas in DC-East.

Note that what happens in the non-local data center -- DC-West in the examples above -- is irrelevant to meeting the LOCAL QUORUM requirement for a given request. For an S3 request received in DC-East, DC-West could be unreachable or completely offline and LOCAL QUORUM could still be achieved so long as the read or write succeeds for 2 of the 3 replicas assigned to DC-East. Conversely, if the operation does not succeed for 2 of DC-East's 3 assigned replicas, then the S3 request fails regardless of whether or not some replicas can be read or written in DC-West.

Example for a "Replicated EC" policy

Suppose a "Replicated EC" policy uses 4+2 erasure coding replicated in DC-East and DC-West. With this configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default) in each of those DCs.

If the policy's consistency level for Read is set to LOCAL QUORUM, and an S3 GET request is received in DC-East (for example), then in order for that request to succeed the system must succeed in reading 4 object data fragments and 2 object metadata replicas in DC-East.

If the consistency level for Write is set to LOCAL QUORUM, and an S3 PUT request is received in DC-East (for example), then in order for that request to succeed the system must succeed in writing 5 object data fragments and 2 object metadata replicas in DC-East.

Note that what happens in the non-local data center -- DC-West in the examples above -- is irrelevant to meeting the LOCAL QUORUM requirement for a given request. For the S3 PUT request received in DC-East, DC-West could be unreachable or completely offline and LOCAL QUORUM could still be achieved so long as the write succeeds for 5 object data fragments and 2 object metadata replicas in DC-East. Conversely, if the system cannot write the 5 object data fragments and 2 object metadata replicas in DC-East, then the S3 PUT operation fails regardless of whether or not some data fragments and metadata replicas can be written in DC-West.

Consistency Level "ONE"

The consistency level "ONE" is supported only in "Replication Within Single Data Center" and "Replication Across Data Centers" data distribution schemes and only for read operations -- not writes.

With a read consistency of ONE, for an S3 GET to succeed the system must succeed in reading one object data replica and one object metadata replica.

Example for a "Replication Within Single Data Center" policy

Suppose a "Replication Within Single Data Center" policy uses 3X replication. With this configuration the system will for each object store 3 object data replicas and 3 object metadata replicas.

If the policy's consistency level for Read is set to ONE, then during an S3 GET operation just one object data replica and just one object metadata replica must be successfully read before the object is returned to the client.

Example for a "Replication Across Data Centers" policy

Suppose a "Replication Across Data Centers" policy uses 3X replication in DC-East and 2X replication in DC-West. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas in DC-East, and 2 object data replicas and 2 object metadata replica in DC-West.

If the policy's consistency level for Read is set to ONE, then during an S3 GET operation just one object data replica (from either DC) and just one object metadata replica (from either DC) must be successfully read before the object is returned to the client.

Consistency Level "QUORUM"

The consistency level "QUORUM" is a supported option for every type of data distribution scheme except "Replicated EC", for both reads and writes. The table below shows the general requirements of QUORUM in the context of replication based storage policies and erasure coding based storage policies. Following the table are examples for each supported data distribution scheme.

	Replication	Erasure Coding
Read QUORUM	For an S3 GET to succeed, the system must succeed in reading a majority of object data replicas and a majority of object metadata replicas.	For an S3 GET to succeed, the system must succeed in reading " k " object data fragments and a majority of object metadata replicas.
Write QUORUM	For an S3 PUT to succeed, the system must succeed in writing a majority of object data replicas and a majority of object metadata replicas.	For an S3 PUT to succeed, the system must succeed in writing " $k+1$ " object data fragments and a majority of object metadata replicas.

Note For replicated data or metadata, a quorum or majority of a set of replicas is defined mathematically as $(\#replicas/2) + 1$, rounded down to the nearest integer. For example with 3 replicas a quorum would be $(3/2 = 1.5) + 1 = 2.5$, rounded down = 2. With 4 replicas a quorum would be $(4/2 = 2) + 1 = 3$, rounded down = 3.

Example for a "Replication Within Single Data Center" policy

Suppose a "Replication Within Single Data Center" policy uses 3X replication. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas.

If the policy's consistency level for Read is set to QUORUM, then during an S3 GET operation 2 object data replicas and 2 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to QUORUM, then during an S3 PUT operation 2 object data replicas and 2 object metadata replicas must be successfully written before a success response is returned to the client.

Example for an "EC Within Single Data Center" policy

Suppose an "EC Within Single Data Center" policy uses 4+2 erasure coding. With this configuration, for each object the system will store 6 object data fragments and 3 object metadata replicas (2m-1, by default).

If the policy's consistency level for Read is set to QUORUM, then during an S3 GET operation 4 object data fragments and 2 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to QUORUM, then during an S3 PUT operation 5 object data fragments and 2 object metadata replicas must be successfully written before a success response is returned to the client.

Example for a "Replication Across Data Centers" policy

Suppose a "Replication Across Data Centers" policy uses 3X replication in DC-East and 2X replication in DC-West. With this configuration, for each object the system will store 3 object data replicas and 3 object metadata replicas in DC-East, and 2 object data replicas and 2 object metadata replicas in DC-West.

If the policy's consistency level for Read is set to QUORUM, then during an S3 GET operation any 3 object data replicas and any 3 object metadata replicas must be successfully read before the object is returned to the client. For example the 3 readable object data replicas could be constituted as 1 in DC-East and 2 in DC-West, or 2 in DC-East and 1 in DC-West, or 3 in DC-East and 0 in DC-West.

If the consistency level for Write is set to QUORUM, then during an S3 PUT operation any 3 object data replicas and any 3 object metadata replicas must be successfully written before a success response is returned to the client. For example the 3 successfully written object data replicas could be constituted as 1 in DC-East and 2 in DC-West, or 2 in DC-East and 1 in DC-West, or 3 in DC-East and 0 in DC-West.

Example for an "EC Across Data Centers" policy

Suppose an "EC Across Data Centers" policy uses 7+5 erasure coding distributed across DC-East, DC-West, and DC-South. With this configuration, for each object the system will store 4 object data fragments and 3 object metadata replicas in each of those DCs (for the object metadata it's 9 replicas [2m-1, by default] divided evenly among the three DCs).

If the policy's consistency level for Read is set to QUORUM, then during an S3 GET operation any 7 object data fragments and any 5 object metadata replicas must be successfully read before the object is returned to the client.

If the consistency level for Write is set to QUORUM, then during an S3 PUT operation a total of 8 object data fragments and a total of 5 object metadata replicas must be successfully written before a success response is returned to the client. The 8 successfully written object data fragments could be constituted as, for example, 2 in DC-East and 3 in DC-West and 3 in DC-South; or 4 in DC-East and 4 in DC-West and 0 in DC-South.

Create New Storage Policy -- Group Visibility

In the "Group Visibility" section of the [Create New Policy](#) interface, specify which user groups will be allowed to use the policy. Users from the groups that you select here will see the policy as an available storage policy when they create a storage bucket. Users are required to choose a storage policy when creating a bucket.

To make this storage policy available to **all** user groups, do nothing in the "Group Visibility" section of the interface. Making policies available to all user groups is the default behavior.

IMPORTANT: If this storage policy is going to be the [default storage policy](#), do not specify any groups.

The default storage policy must be available to all groups. If you do not yet have any storage policies in your system, the first policy that you create must be a default policy that is available to all groups.

If you want the policy to be used only by certain user groups, do the following for each group that you want the policy to be available to:

- Use the drop-down list to select a group.
- Click **Add**.

The groups that you select will then display in the "Group Visibility" section interface.

Create New Storage Policy -- Compression

In the "Compression" field of the [Create New Policy](#) interface, select the type of compression (if any) to use for S3 objects stored in the HyperStore File System. This applies to replicated S3 object data and erasure coded

S3 object data stored in the HSFS. It does not apply to data stored in Cassandra.

Supported options are:

- [Snappy](#)
- [Zlib](#)
- [LZ4](#)
- None (no compression)

When enabled, compression is applied to incoming S3 objects by the S3 Service, before those objects are transmitted to the HyperStore Service and placed into storage. Any change that you make to the compression type setting — such as changing it from disabled to a particular compression type, or from one compression type to another — is applied only to new S3 objects as they come into the system, not retroactively to objects that are already stored in the system.

Each object's compression type (if any) is stored in the object's metadata in Cassandra. Consequently, if for example you use Snappy for a while and then switch to LZ4, those objects that had been compressed with Snappy will continue to have in their metadata an indicator that they were compressed with Snappy — and so the system will be able to de-compress the objects when they are downloaded by S3 client applications.

Note For S3 service usage tracking (for purposes of QoS enforcement and billing), the **uncompressed** size of objects is always used, even if you enable HyperStore compression.

For a "**PUT Object - Copy**" (page 940) operation, the copy of the object will be subject to whatever the **current** compression type setting is (which may be different than the setting that was in effect when the original object was uploaded).

Create New Storage Policy -- Server-Side Encryption

In the "Server-Side Encryption" field of the [Create New Policy](#) interface, select the default method of server-side encryption that the system should apply to all objects in all buckets that use this storage policy.

Supported options are:

- SSE -- Regular server-side encryption using encryption keys managed by the HyperStore system
- KeySecure -- Server-side encryption using encryption keys managed by a Gemalto KeySecure key management system (KMS)

Note: To use the KeySecure encryption feature for a storage policy you must first enable KeySecure in the system configuration. By default the KeySecure option does not display in the CMC's storage policy interface. For more information see "[Using Server-Side Encryption with Gemalto KeySecure KMS](#)" (page 125).

- None -- No server-side encryption

Your chosen default method of server-side encryption for the storage policy will apply only to objects for which no server-side encryption method is specified either in the object upload request or in the bucket configuration. **Object-level and bucket-level server-side encryption settings supersede the storage policy level setting.**

For more information about HyperStore's support for server-side encryption -- including the interaction of object level, bucket level, and storage policy level encryption settings -- see "[Server-Side Encryption Feature Overview](#)" (page 117).

Note If you edit the Server-Side Encryption setting for an existing storage policy, your change applies only to objects that are uploaded from that time forward. The change does not apply to objects that are already in storage.

5.7.7.2. Edit a Storage Policy

To edit an existing storage policy, in the policy list on the CMC's [Storage Policies](#) page click **View/Edit** for the policy that you want to edit. You can then edit policy attributes such as data consistency requirements, group visibility, compression, and server-side encryption. For guidance on working with these policy characteristics, see "**Add a Storage Policy**" (page 308).

Note that any changes you make to how stored objects are handled, such as compression or server-side encryption, will apply only to objects uploaded **after** you make the storage policy edit -- not to objects that are already in storage. For example if you enable server-side encryption on an existing storage policy, then from that time forward objects that get uploaded to buckets that use that storage policy will be encrypted -- but objects that were already uploaded prior to the configuration change will not be encrypted. Conversely, if an existing storage policy has been configured for encryption and then later you disable encryption for that policy, then from that time forward newly uploaded objects will not be encrypted -- but objects that had already been uploaded (and encrypted) prior to the configuration change will remain encrypted.

Note When you click **View/Edit** for a storage policy you can also view the storage policy's data distribution scheme (such as replication factor or EC "k"+"m" values), its data center assignment, and its system-generated policy ID. However these policy attributes are not editable.

5.7.7.3. Designate a Default Storage Policy

At all times you must have one and only one **default storage policy** defined in each of your HyperStore service regions. The default policy is the one that will be applied when users create new buckets without specifying a policy.

In your storage policy list (in the CMC's [Storage Policies](#) page), the current default policy is listed first, with its Region name highlighted in green. In the example below, the policy named "HSFS-1" is the default storage policy.

STORAGE POLICIES							+ CREATE STORAGE POLICY	
	REGION	STATUS	NAME	DESCRIPTION	DATA DISTRIBUTION POLICY	NO OF REPLICAS	LOCAL EC	
<input checked="" type="checkbox"/>	Us-norcal	ACTIVE	HSFS-1	HSFS - DC1:2 - DC2:1	Multi DC	3	N/A	View/Edit
<input checked="" type="checkbox"/>	Us-norcal	ACTIVE	EC-1	EC: K2-M1	Multi DC	2	2 + 1	Set As Default View/Edit
<input checked="" type="checkbox"/>	Us-norcal	ACTIVE	Test_Policy		Single DC	3	N/A	Set As Default View/Edit
<input checked="" type="checkbox"/>	Us-norcal	ACTIVE	Test_Policy_2		Single DC	1	2 + 1	Set As Default View/Edit

[DISABLE](#) [DELETE](#)

To be eligible for being the default storage policy, a policy must be:

- Active (Status = ACTIVE)
- Visible to all user groups. If you're unsure whether a particular policy is visible to all groups, click **View/Edit** for the policy, then check in the **Group Visibility** panel. If individual group names appear in this panel, that means the policy is currently configured to be visible to only those groups. You can change this by deleting all the specific groups that are displayed and saving the edited storage policy.

To designate a policy as the default, click **Set as Default** for that policy. The interface will ask you to confirm that you want to take this action.

Note that changing the default storage policy has **no effect on which policy is used by currently existing buckets**. Buckets that have been using the old default storage policy will continue to do so. The impact is that when new buckets are created without specifying a storage policy, they will be assigned the new default storage policy.

Note When a pre-5.2 version of HyperStore is upgraded to 5.2 or newer, the upgrade process automatically creates a storage policy named "DEFAULT_<region-name>". This policy is configured with the data distribution scheme (replication factor or EC "k"+m" values) and user data read/write consistency requirements that the pre-5.2 system had been using. This "DEFAULT_<region-name>" will be your default storage policy until you designate some other policy as the default.

5.7.7.4. Disable a Storage Policy

Disabling a storage policy prevents users from choosing the policy when they create new buckets. However, buckets that are already using the policy will continue to do so even after the policy is disabled.

Note You cannot disable the default storage policy. If you want to disable that policy you must first promote a different policy to be the new default.

Note Disabled policies still count toward the configurable limit on the number of policies that can exist in the system.

To disable a policy, in the CMC's [Storage Policies](#) page select the checkbox to the left of the policy name and then click **Disable**. The interface will ask you to confirm that you want to take this action.

If subsequently you want to re-enable a disabled storage policy, select the policy and click **Enable**. When you re-enable a storage policy, the policy once again becomes available for service users to assign to newly created buckets.

5.7.7.5. Delete a Storage Policy

Deleting a storage policy completely removes the policy from the system. There are restrictions on which policies you can delete. You cannot delete:

- The default storage policy. If you want to delete that policy you must first [promote a different policy to be the new default](#).
- The system-generated policy named "DEFAULT_<regionName>" (applicable only to HyperStore systems that have upgraded from versions older than 5.2).

- Storage policies that are currently being used by one or more buckets. If a policy is being used by buckets and you want those buckets to continue using it but you no longer want the policy to be available to new buckets, you can disable the policy rather than deleting it. See "**Disable a Storage Policy**" (page 332). If you really want to delete such a policy, see the required preliminary steps below.

To delete a storage policy that has never been used (that is, the policy has never been associated with any buckets), in the CMC's [Storage Policies](#) page select the checkbox to the left of the policy name and then click **Delete**. The interface will ask you to confirm that you want to take this action, and when you confirm the policy will be deleted.

To delete a policy that has been or is currently being used by one or more buckets, do the following:

1. If any buckets are currently using the policy, first delete all objects from those buckets, then delete the buckets themselves.

Note It is not possible to reassign a different storage policy to a bucket. If you want to delete a storage policy that is currently being used by one or more buckets, you must delete those buckets.

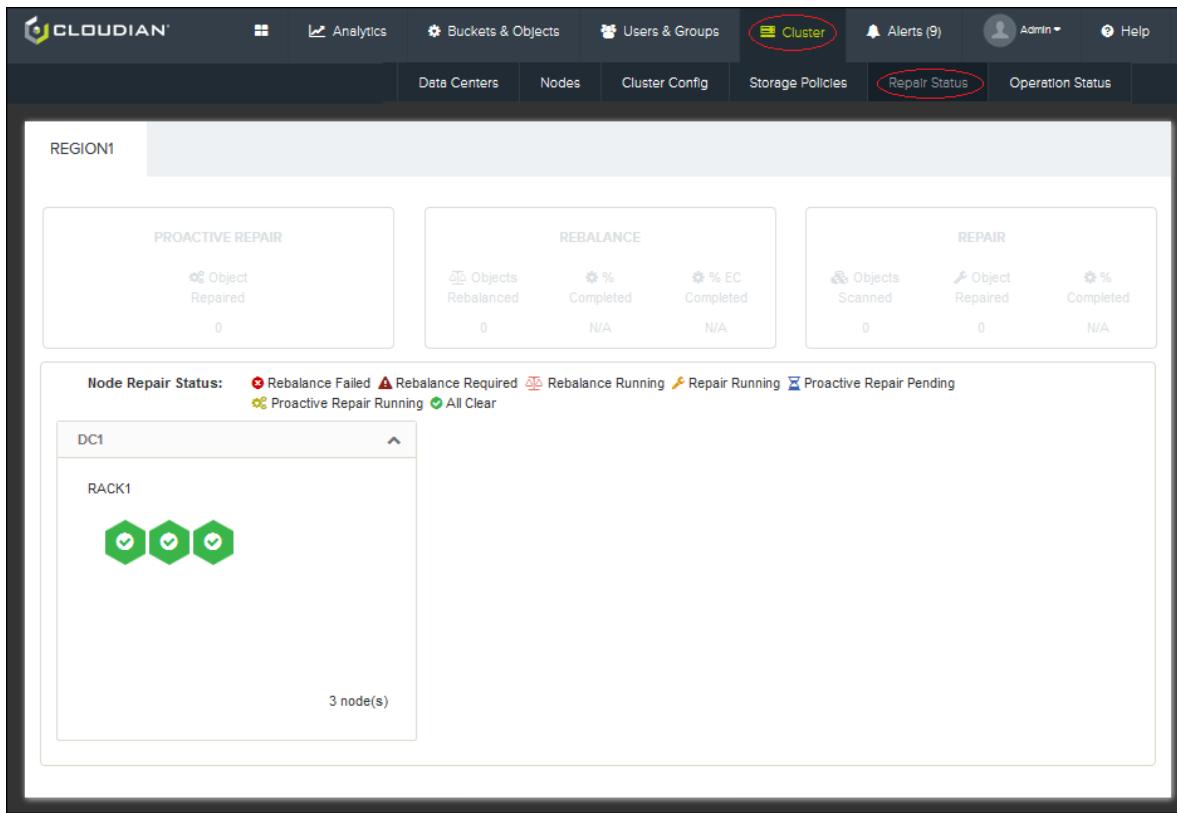
If you are not certain which buckets are currently using a given storage policy, you can use the Admin API method [GET /bppolicy/bucketsperpolicy](#) to retrieve this information.

2. Delete the policy: In the [Storage Policies](#) page select the checkbox to the left of the policy name and then click **Delete**. The interface will ask you to confirm that you want to take this action, and when you confirm the policy will be deleted.

Note: As a best practice, after deleting a storage policy wait 24 hours and then run [hsstool cleanup](#) on each node in your cluster, using the `-a` option and the `-policy` option (`hsstool cleanup -h <host> -a -policy`). This will clean up any garbage data associated with the deleted storage policy -- such as replicas or erasure coded fragments that the system failed to remove when you manually deleted all the objects from buckets that had been using the storage policy (Step 1 above). Normally you should only run `hsstool cleanup` on one node at a time, per data center. If you have circumstances where you need to clean up multiple nodes within a data center, you can try running it on two or three nodes concurrently, but pay close attention to cluster performance and also contact Cloudian Support for further guidance.

5.7.8. Repair Status

Path: **Cluster** → **Repair Status**



Supported tasks:

- View repair status for cluster
- View repair status for a node

5.7.8.1. View Repair Status for Cluster

The upper section of the **Repair Status** page shows aggregate data repair status for your cluster as a whole. The information displays in three panels:

Proactive Repair

This panel is activated if **proactive repair** is either pending or in progress on any node in your cluster.

For the cluster as a whole, this panel shows:

- Objects Repaired -- The number of objects that have been repaired by in-progress proactive repair operations

Note that an "object" in this context can be either an object replica or an erasure coded object fragment. So if for example the in-progress proactive repair has so far repaired 12 object replicas and 8 erasure coded object fragments, the Objects Repaired value would be 20.

Rebalance

This panel is activated if any of the following operations is in progress on any node:

- A **rebalance** operation (which you should be initiating toward the end of the procedure for "**Adding Nodes**" (page 369))

- A *decommission* operation (which the system would automatically initiate as part of the process of **"Removing a Node"** (page 390), if the node was accessible within the Cassandra ring)

These operations have in common that object replicas (or erasure coded fragments) are being re-located within the cluster in response to the cluster having been re-sized.

For the cluster as a whole, this panel shows:

- Objects Rebalanced -- How many object replicas or erasure coded fragments have been relocated so far, by the currently in-progress *rebalance* or *decommission* operation.
- % Completed -- For object replicas, the token ranges for which rebalancing or decommissioning has been completed as a percentage of the total token ranges impacted by the cluster resizing.
- % EC Completed -- For object erasure coded fragments, the token ranges for which rebalancing or decommissioning has been completed as a percentage of the total token ranges impacted by the cluster resizing.

Repair

This panel is activated if an [hsstool repair](#) or [hsstool repairec](#) operation is in progress on any node in your cluster. It indicates the number of objects that have been scanned and (from among the scanned objects) the number of objects that needed and received repair. It also shows the completion percentage for the operation, on the node on which it is being run.

For the cluster as a whole, this panel shows:

- Objects Scanned -- The number of objects that have been scanned by in-progress *repair* and/or *repairec* operations to determine whether or not the objects are in need of repair.
- Objects Repaired -- The number of objects that have been repaired by in-progress *repair* and/or *repairec* operations.
- % Completed -- For replica repair only, the completion percentage so far. This is calculated as the number of token ranges for which repair has been completed divided by the total number of token ranges that will be repaired by this *repair* operation. The % Completed metric does not apply to *repairec* operations -- if only a *repairec* operation is running (and not a *repair* operation), then the % Completed field displays "N/A" for not applicable.

5.7.8.2. View Repair Status for a Node

The lower section of the CMC's **Repair Status** page displays a color-coded node icon (cube) for each node in the cluster. The icon indicates the node's current repair status:



Rebalance Failed — The node has been added to the cluster and *rebalance* was run on the node, but the *rebalance* operation failed. Try running *rebalance* on the node again. See **"Adding Nodes"** (page 369).



Rebalance Required — The node has been added to the cluster, but *rebalance* has not yet been run on the node. The *rebalance* operation is required in order to shift some data to the node from other nodes in the cluster. See **"Adding Nodes"** (page 369).



Rebalance Running — Either a *rebalance* operation is in progress for the node (if the node has been recently [added to the cluster](#)); or else a *decommission* operation is in progress for the node (if the node is live and is being [removed from the cluster](#)).



Repair Running — An [hsstool repair](#) or [hsstool repairec](#) operation is running on the node. This may be as a result of the [scheduled auto-repair](#) feature, or because a system administrator initiated the operation.



Proactive Repair Pending — The system has detected that the node is in need of [proactive repair](#). The repair will occur at the next proactive repair interval (by default every 60 minutes).



Proactive Repair Running — [Proactive repair](#) is in progress on the node.



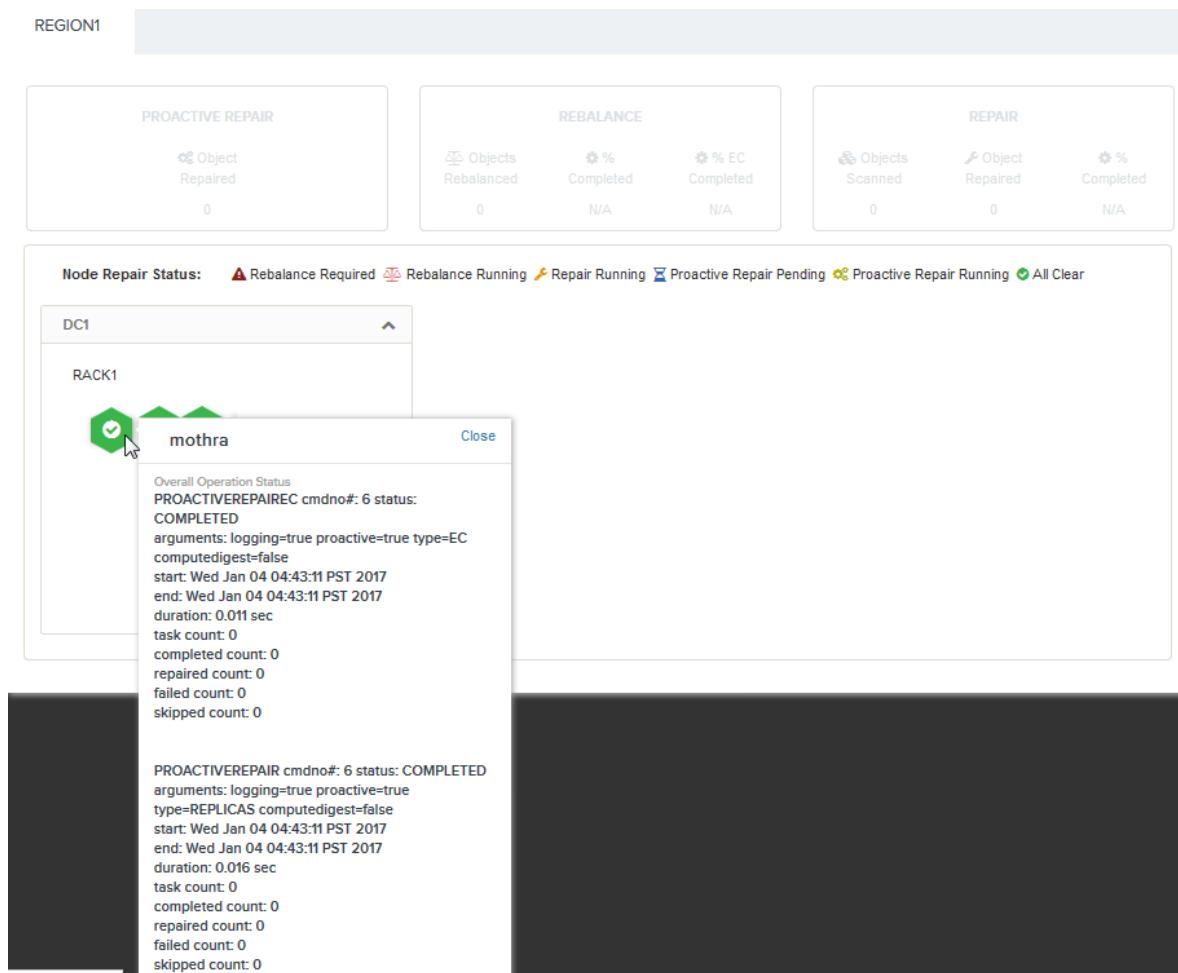
All Clear — No repair of any type is in progress on the node, and the node is not currently in need of rebalance or proactive repair.



Unavailable —The repair information for the node is unavailable, such as if the node is down or inaccessible or the HyperStore Service is down on the node.

Note In the event that multiple repair types are simultaneously running on the node, the color-coding reflects whichever in-progress repair type started first on the node.

If you click a node icon, detailed information about in-progress and recently completed repairs on that node will display.



This is the same information that is retrieved when you run the `hsstool opstatus` command on a node. For description of the available information items see the command response section of "**hsstool opstatus**" (page 630).

In the case of proactive repair, the information items will also include the **proactive repair queue length**. This indicates the number of objects for which data -- either a replica or an erasure coded fragment -- currently needs to be written to the node by the proactive repair feature. As the proactive repair of the node proceeds and fewer objects remain to repair, the proactive repair queue shrinks. (Conversely, at times when the node is down, unreachable, or otherwise unable to support writes, the node's proactive repair queue grows.)

If after clicking a node icon, you click the host name at the top of a node's status detail display you will jump to the [Node Status](#) page for that node.

Note If proactive repair is pending on a node, and if for some reason you want to trigger the proactive repair on that node immediately rather than waiting for the automatic hourly run, you can do so by using the [hsstool proactiverepairq](#) command with the "-start" option.

Note For information about stopping an in-progress repair see "**Disabling or Stopping Data Repairs**" (page 79).

5.7.9. Operation Status

Path: Cluster → Operation Status

OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE	
addNode	region1::dc1::failhost	! failed	failed	Feb-23-2018 07:27	Feb-23-2018 07:28	View Delete
cleanupec	region1::dc1::store1	✓ completed	100%	Feb-23-2018 07:25	Feb-23-2018 07:25	View Delete
repair	region1::dc1::store1	✓ completed	100%	Feb-23-2018 07:25	Feb-23-2018 07:25	View Delete
cleanup	region1::dc1::store1	✓ completed	100%	Feb-23-2018 07:24	Feb-23-2018 07:24	View Delete

Supported task:

- Check status of operations launched from the CMC

In the **Operation Status** page you can view the status of long-running operations that you have launched from the CMC. Status reporting in this page is supported for these operation types:

- Add Node
- Add Data Center
- Add Region
- Uninstall Node
- hsstool rebalance
- hsstool repair or repairrec
- hsstool cleanup or cleanupec

Note For *hsstool* operations, the CMC's **Operation Status** page reports only on *hsstool* operations that you initiate through the CMC's [Node Advanced](#) page. It does not report on *hsstool* operations that you initiate manually through the command line. For viewing status of operations that you initiate on the command line use [hsstool opstatus](#).

For each operation the **Operation Status** page displays the Operation Name, Target node (identified as <region-name>::<datacenter-name>::<hostname>), and current Status, as well as the Progress (as an approximate percentage of completion), operation Start Time, and Last Update time (the last time that the CMC obtained status information for the operation).

The operation Status will be one of In-Progress, Completed, Failed, or Terminated. A Terminated status is applicable only for *hsstool repair*, *hsstool repairrec*, *hsstool cleanup*, or *hsstool cleanupec* operations that an operator has terminated by the "stop" command option that's supported for those operation types.

- To refresh the display, click the refresh icon above the **Search** field.



- To view status detail for an operation, click the **View** button to the right of the operation status summary.

The screenshot shows the 'Operation Status' window. On the left, there's a sidebar with a tree view containing nodes like 'OPERATION LIST', 'addNode', 'cleanuppec', 'repair', and 'cleanup'. Below this is a message 'Showing 1 to 4'. The main area has a table with columns: OPERATION NAME, TARGET, STATUS, START TIME, and LAST UPDATE. A single row is shown for 'cleanuppec' with target 'region1:dct1:store1', status 'completed', start time 'Feb-23-2018 07:25', and last update 'Feb-23-2018 07:25'. Below the table is a progress bar at 100%. To the right of the table is a large text block detailing the operation command, arguments, and results. At the bottom right of the main window is a 'Close' button.

OPERATION NAME	TARGET	STATUS	START TIME	LAST UPDATE
cleanuppec	region1:dct1:store1	completed	Feb-23-2018 07:25	Feb-23-2018 07:25

CLEANUPEC cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true deletedata=true deleteobject-without-policy=false
logging=true delete-only-outofrange-objects=false
start: Fri Feb 23 04:25:54 PST 2018
end: Fri Feb 23 04:25:54 PST 2018
duration: 0.014 sec
progress percentage: 100%
time remaining: 0 ms
task count: 0
completed count: 0
deleted count: 0
failed count: 0
skipped count: 0

Operation succeeded.

In the case of *hsstool* operations these are the same operation details as are provided by [hsstool opstatus](#). To refresh the status detail, close the detail view and then reopen it.

- To filter the operation list, in the **Search** field enter an operation name, region name, data center name, or hostname.
- To delete an operation status line from the page click **Delete** to the right of the line. In the case of *hsstool* operations the status detail will still be available through the [hsstool opstatus](#) command.

5.8. Alerts

The **Alerts** tab contains the following functions:

- "Alerts" (page 339)
- [Alert Rules](#)

5.8.1. Alerts

Path: **Alerts** → **Alerts**

SEVERITY	NODE ID	ALERT TYPE	ALERT TEXT	LAST UPDATE	COUNT
High	jenkins-jd_k8	Disk space available in each device	[/dev/mapper/vg_jenkinsjdlk8-lv_root] 9.897639381318657 %	Oct-18-2016 0 6:41	1
High	jenkins-jd_k8	S3	[Service Down or Unreachable]	Oct-05-2016 1 1:16	1
High	jenkins-jd_k8	Admin	2016-10-05 07:59:55,097 ERROR [DeleteUserExecutor - bf85cd6f237c746728e015b37c2bbc80] DeleteUserExecutor: deleteRedisUserCredentials(92) bf85cd6f237c746728e015b37c2bbc80 is already deleted	Oct-05-2016 1 1:00	1
Medium	jenkins-jd_k8	Redis Cred	[4308] 05 Oct 07:47:24.636 # Server started, Redis version 2.8.23	Oct-05-2016 1 0:50	2

ACKNOWLEDGE

Supported tasks:

- **"Review Alerts"** (page 340)
- **"Acknowledge Alerts"** (page 343)

5.8.1.1. Review Alerts

In the **Alerts** page you can review active alerts that have been generated by your HyperStore nodes. Alerts are triggered by the occurrence of node events or conditions for which alert rules have been configured. Your HyperStore system comes with a set of pre-configured alert rules which are listed in the CMC's [Alert Rules](#) page. In that page you can also edit the pre-configured rules if you wish, or create additional alert rules.

The **Alerts** page auto-refreshes once per minute. For each alert the following information is displayed:

Severity

Each alert is assigned a severity level of Critical, High, Medium, or Low. The severity level assigned to each alert is configurable in the [Alert Rules](#) page.

Node ID

The HyperStore node on which the alert occurred. Note that at the top of the **Alerts** page you can filter the **Alert List** by node. By default, "ALL" is selected, to show results for all nodes in your system.

You can click on the Node ID to jump to the **"Node Status"** (page 268) page for that node.

Alert Type

- **Statistic threshold alerts** — Alerts indicating that a monitored performance statistic has crossed a threshold. For this category of alert, the "Alert Type" field indicates the statistic for which a configured threshold was crossed (such as CPU utilization, Disk space available on node, Disk space available on device, Number of S3 GET transactions per second, and so on).

- **Service down/unreachable alerts** — Alerts indicating that a service is either down or unreachable by the HyperStore monitoring system. For this category of alert, the "Alert Type" field indicates the service type that is down or unreachable -- Admin, Cassandra, HyperStore, Redis Credentials, Redis QoS, or S3. (The system also supports alerts for when such a service is restored or reachable again -- these are not pre-configured in the system but you can add alert rules for these if you wish).

Note: To determine whether a service is actually down on a node (as opposed to being up but unreachable by the monitoring system), log into the node and run:

```
systemctl is-active <servicename>
```

The <servicename> string can be one of {cloudian-s3, cloudian-cassandra, cloudian-hyper-store, cloudian-redis-credentials, cloudian-redis-qos, cloudian-redismon, cloudian-cmc, cloudian-agent, cloudian-dnsmasq}.

- **Log message alerts** — Alerts indicating that a WARN or ERROR level message has been written to a service application log. For this category of alert, the "Alert Type" field indicates the service for which the WARN or ERROR occurred -- Admin, Cassandra, HyperStore, Redis Credentials, Redis QoS, or S3.

Note: Along with alerts pertaining to providing S3 service to clients, the "S3" alerts category includes alerts pertaining to auto-tiering and cross-region replication.

- **Repair completion alerts** — Alerts indicating that a data repair operation has finished its run. For this category of alert, the "Alert Type" field displays "Repair Completion Status". Note that the completion of routine "proactive" repairs does not trigger these alerts -- only completion of scheduled auto-repairs or repairs operations that you execute yourself (from the CMC or the command line) will trigger these alerts. For more information on repair types see .
- **Disk error alerts** -- Alerts indicating that disk read/write error has been detected and/or the disk has been disabled. For this category of alert, the "Alert Type" field displays "Disk Error". To learn more about the status of a disk for which a Disk Error alert has been generated, go to the "**Node Status**" (page 268) page and select the node on which the disk resides, then view the [Disk Detail Info](#) section of the page.

Alert Text

- For **statistic threshold alerts**, this field will indicate the statistic's value which caused an alert rule to be triggered.
- For **service down/unreachable alerts**, this field will be a text string "[Service Down or Unreachable]"

Note: To determine whether a service is actually down on a node (as opposed to being up but unreachable by the monitoring system), log into the node and run:

```
/etc/init.d/<service-name> status
```

The <service-name> string can be one of {cloudian-s3, cloudian-cassandra, cloudian-hyper-store, cloudian-redis-credentials, cloudian-redis-qos, cloudian-redismon, cloudian-cmc, cloudian-agent, cloudian-dnsmasq}.

- For **disk error alerts**, this field indicates the mount point for the disk that's had an error.
- For **log message alerts**, this field will be the full text of the log entry (truncated if the log message is larger than 256 characters).
 - For general information about HyperStore log entry formatting, see "**HyperStore Logs**" (page 575).
 - Most log entries of level ERROR or higher include an alphanumeric **message code** that uniquely identifies the log message (either "HSxxxxx" or "DCxxxxx" or "RMxxxxx"). For documentation of an individual message code, click the message code text (in blue font) in the **Alerts** interface. This opens the log message code Help. The documentation for the message code that you clicked will be expanded, but you may need to scroll down the Help page to see it. (Alternatively, simply holding your cursor over the message code in the **Alerts** interface will display in-place "tool tip" text with the log level and recommended corrective action for the message code.)

Note: For log message based alerts that have occurred multiple times without being acknowledged, the "Alert Text" -- including the log entry timestamp -- will be from the most recent instance of the log message.

Last Update

This field shows the local date and time at which the alert was detected by the HyperStore monitoring system.

For alerts that have occurred multiple times without being acknowledged (as indicated by the "Count" value), the "Last Update" field indicates when the **most recent instance** of the alert was detected.

Count

The number of unacknowledged times that the same alert has occurred.

- For **statistic threshold alerts**, a Count value greater than "1" means that the monitoring system has detected the statistic to be across its configured threshold multiple times, without being acknowledged.

Note: Each node's CPU utilization, disk space availability, and total network throughput are checked each minute. For each of these statistics, each time that the once-per-minute checks finds that the statistic is across its notification threshold, the Count for the alert is incremented. For example, if you have an alert rule for "CPU utilization > 50%", and if the monitoring system's once-per-minute checks find three different instances where the node's CPU utilization was in excess of 50%, then the **Alert List** will show one line for the "CPU Utililzation" alert type, with a Count of 3.

For S3 statistics — GET/PUT transactions per second, GET/PUT throughput, and GET/PUT average latency — the statistics are calculated at the node every **five** minutes. However, the alert monitoring system retrieves the values each minute, just like for other statistics. So, the monitoring system retrieves the same calculated S3 statistics five times, until the next set of S3 statistics is calculated. If an S3 statistic value is across a notification threshold, the monitoring system retrieves that same statistic value five times, and consequently shows a Count of 5. So for alerts based on S3 statistic thresholds, the Count will overstate how often the alert has actually happened, by as much as a factor of 5.

- For **service down/unreachable alerts** the Count increments each 60 seconds for as long as the service is down or unreachable by the HyperStore monitoring system.
- For **log message alerts**, a Count value greater than "1" means that the exact same log message has occurred multiple times without being acknowledged.

Note: The monitoring system checks the logs every 30 seconds for new warning or error messages. If a log is rotated during the 30 second interval between one check and the next, it's possible that an instance of a log message may be missed by the monitoring system. In this alert the Count value may not be exactly correct, particularly in circumstances where many error messages are occurring and logs are being rotated more frequently than usual.

- For **repair completion alerts**, a Count value greater than "1" means that multiple repair completion alerts of the specified repair type (REPAIR, EC:REPAIR, or CASSANDRA:REPAIR) have occurred on the same node without being acknowledged. Note that if the Count is greater than "1", the Alert Text will show only the status of the first-finished repair operation . To view status detail on all recently run repairs use *hsstool opstatus* or check the CMC's "**Repair Status**" (page 333) page.

Note The alert list is sorted primarily by alert Severity (high to low) and secondarily by reverse chronological order (based on the time of the most recent instance of each listed alert, as indicated in the Last Update column). You can re-sort the alert list by any column except for Alert Text by clicking the column heading. Click once for ascending order, click a second time for descending order.

5.8.1.2. Acknowledge Alerts

In the **Alerts** page you can acknowledge that you have seen and reviewed the node alert notifications that display in the **Alert List**. Once you acknowledge an alert, it will no longer display in the **Alert List** (unless you choose to display acknowledged alerts as described further below).

- To acknowledge one or multiple alerts in the **Alert List**, click the checkbox to the left of the alert(s) and then click **Acknowledge** at the bottom of the list. Note that as you are selecting alerts to acknowledge (by clicking checkboxes), the **Alerts** page's auto-refresh feature will temporarily pause, so as not to clear your checkbox selections.
- To acknowledge **all** alerts, click the checkbox at the left side of the **Alert List** column heading row and then click **Acknowledge** at the bottom of the list.
- To show previously acknowledged alerts in the list as well as unacknowledged alerts, click **Show Acknowledged** at the top of the alert list. Previously acknowledged alerts then display within the list, and are distinguished from unacknowledged alerts by the absence of a checkbox to the left of the alert. You can click **Hide Acknowledged** to hide the acknowledged alerts again.

The **Alert List** can display a maximum of 50 unacknowledged alerts and acknowledged alerts combined. Note that some of these 50 alerts may have occurred multiple times as indicated by the alert's "Count" value. Even if an alert has a Count showing that it has occurred multiple times, this counts as only one alert toward the **Alert List** display maximum of 50 alerts.

Note **Acknowledged alerts are automatically deleted from the system** after a time period configured by the *mts.properties.erb*:**"events.acknowledged.ttl"** (page 512) setting. By default this period is 86400 seconds (one day). After they are deleted acknowledged alerts will not display in the Alert List

even if you click **Show Acknowledged**. If you wish you can reduce the configurable time-to-live for acknowledged alerts to as little as 1 second (so that they are deleted from your system right after acknowledgment). Note that regardless of your configured time-to-live for acknowledged alerts, a record of your system's alert history will persist in the [Smart Support](#) logs that by default are uploaded to Cloudian Support each day.

Note If you acknowledge an alert for which the underlying notification triggering condition still exists, a new alert may be generated quickly. For example, if a service is down, and you acknowledge the service down alert but do not restart the service, a new service down alert will be triggered as soon as the monitoring system polls the service status again (which occurs each minute). If you want to temporarily disable an alert rule you can do so in the [Alert Rules](#) page, but be sure to remember to re-enable the rule at the appropriate time.

5.8.2. Alert Rules

Path: **Alerts** → **Alert Rules**

ACTIVE ALERT RULES				
GENERAL STATUS				
	send email to	send snmp trap	severity level	actions
CPU Utilization greater than 90.0 %	default		Medium	Edit Delete
Disk Error	default		Critical	Edit Delete
Disk space available in node less than 10.0 %	default		High	Edit Delete
Disk space available in each device less than 15.0 %	default		High	Edit Delete
Node Unreachable	default		Critical	Edit Delete
Repair Completion Status	default		Low	Edit Delete
SERVICE STATUS				
	send email to	send snmp trap	severity level	actions
Admin service error	default		High	Edit Delete
Admin service down	default		High	Edit Delete

Supported tasks:

- "Review Supported Rule Types and Pre-Configured Rules" (page 345)
- "Add Alert Rules" (page 349)
- "Edit Alert Rules" (page 350)
- "Disable Alert Rules" (page 351)
- "Delete Alert Rules" (page 351)

5.8.2.1. Review Supported Rule Types and Pre-Configured Rules

Alert rules specify the system conditions that will trigger HyperStore alerts. HyperStore supports the alert rule types described in the tables below. As indicated by the "Pre-Configured?" column, for some alert rule types HyperStore comes with pre-configured rules that have already been created for you and are active in the system. These pre-configured rules are listed in the main part of the **Alert Rules** page when you first access the CMC. All pre-configured rules include sending a notification email to the default system administrator email address.

For all active rules, an alert is generated if the specified condition occurs **on any node in the system**. All rules are based on node conditions (as opposed to aggregate, system-wide conditions).

Network Status Rules

Rule Type	Description	Pre-Configured?
Number of GET transactions per second	For each node, every five minutes the system calculates the average number of S3 GETs processed per second, based on the last approximately 1000 S3 GET transactions. Alert rules can be based on this average exceeding or falling below a user-defined threshold.	No
Number of PUT transactions per second	Same as above, except for S3 PUTs.	No
Throughput for GET operations	For each node, every five minutes the system calculates the average throughput for S3 GETs in bytes per second, based on the last approximately 1000 S3 GET transactions. Alert rules can be based on this average exceeding or falling below a user-defined threshold.	No
Throughput for PUT operations	Same as above, except for S3 PUTs.	No
Latency for GET operations	For each node, every five minutes the system calculates the 95th percentile for transaction latencies of S3 GETs, in milliseconds, based on the last approximately 1000 S3 GET transactions. The 95th percentile latency value indicates that of the last 1000 S3 GET transactions, 95% completed in that many milliseconds or less. Alert rules can be based on this value exceeding or falling below a user-defined threshold.	No
Latency for PUT operations	Same as above, except for S3 PUTs.	No
Network throughput (incoming)	For each node, each minute the system calculates the average number of bytes of incoming network throughput per second, based on the last minute of activity. Alert rules can be based on this value exceeding or falling below a user-defined threshold.	No

Rule Type	Description	Pre-Configured?
	<p>Note Network throughput statistics are based on all data moving into or out of a node — not just S3 request data. For example, data transmission associated with cluster maintenance operations would count toward these statistics.</p>	
Network throughput (out-going)	Same as above, except for outgoing throughput.	No

General Status Rules

Rule Type	Description	Pre-Configured?
Disk space available in node	<p>For each node, each minute the system calculates the aggregate available disk space for the node's data disks, as a percentage of the total capacity of the data disks. Alert rules can be based on this value falling below a user-defined threshold.</p> <p>Note In calculating this statistic the system does not count "reserved" disk capacity as being available. Instead, available capacity is what's left after deducting both used capacity and reserved capacity from the total capacity. For information about "reserved" capacity see "Capacity Managed" (page 172).</p>	Yes -- a High severity alert if disk space available on a node is less than 10%
Disk space available in each device	Same as above, except for each individual data disk on each node.	Yes -- a High severity alert if space available on a disk is less than 15%
Disk error	With this alert rule, an alert is triggered whenever an "HSDISKERROR" or "HSDISKDISABLED" message appears in the HyperStore Service application log (<i>cloudian-hyperstore.log</i>) on any node.	Yes -- a Critical severity alert
Node unreachable	With this alert rule, an alert is triggered if a node cannot be reached by the HyperStore monitoring system.	Yes -- a Critical severity alert
Load average (5 minutes)	For each node, every five minutes the system calculates the Linux load average (average number of processes in execution or queued for execution) for the node as a whole during the past five minutes. Alert rules can be based on this average exceeding a user-defined threshold.	No

Rule Type	Description	Pre-Configured?
	<p>Note Be sure to take into account the number of processing cores on your HyperStore hosts when setting a threshold for alerts based on load average.</p>	
CPU utilization	<p>For each node, each minute the system checks the node's CPU utilization level. Alert rules can be based on this value exceeding a user-defined threshold.</p>	Yes -- a Medium severity alert if CPU utilization is greater than 90%
Repair completion status	<p>With this alert rule, an alert is triggered whenever a data repair operation finishes its run. The alert will indicate the operation's finishing status: COMPLETED, FAILED, or TERMINATED.</p> <p>Note The completion of routine "proactive" repairs does not trigger these alerts. For more information on repair types see "Data Repair Feature Overview" (page 75).</p>	Yes -- a Low severity alert

Service Status Rules

Rule Type	Description	Pre-Configured?
Admin Service status	<p>For the Admin Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down • Service logs an error • Service logs either a warning or an error 	Yes -- a High severity alert if the service is down, and a High severity alert if the service logs an error
Cassandra Service status	<p>For the Cassandra Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down • Service logs an error • Service logs either a warning or an error 	Yes -- a High severity alert if the service is down, and a High severity alert if the service logs an error
HyperStore Service status	<p>For the HyperStore Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down 	Yes -- a High severity alert if the service is

Rule Type	Description	Pre-Configured?
	<ul style="list-style-type: none"> • Service logs an error • Service logs either a warning or an error 	down, and a High severity alert if the service logs an error
Redis QoS Service status	<p>For the Redis QoS Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down • Service logs a warning 	Yes -- a High severity alert if the service is down, and a Medium severity alert if the service logs a warning
Redis Credentials Service status	<p>For the Redis Credentials Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down • Service logs a warning 	Yes -- a High severity alert if the service is down, and a Medium severity alert if the service logs a warning
Redis Monitor Service status	<p>For the Redis Monitor Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down 	Yes -- a High severity alert if the service is down
S3 Service status	<p>For the S3 Service, separate alert rules can be set for any of these statuses:</p> <ul style="list-style-type: none"> • Service is down • Service goes back up after being down • Service logs an error • Service logs either a warning or an error <div data-bbox="493 1673 1219 1882" style="background-color: #ffffcc; padding: 10px;"> <p>Note Along with alerts pertaining to providing S3 service to clients, the S3 service alerts category includes alerts pertaining to auto-tiering and cross-region replication.</p> </div>	Yes -- a High severity alert if the service is down, and a High severity alert if the service logs an error
Cron Mon Service status	<p>For the Cron and Monitoring services, separate alert rules can be set for either of these statuses:</p> <ul style="list-style-type: none"> • Service logs an error 	Yes -- a High severity alert if the

Rule Type	Description	Pre-Configured?
	<ul style="list-style-type: none"> Service logs either a warning or an error 	service logs an error
Phone Home Service status	For the Phone Home (Smart Support) service , separate alert rules can be set for either of these statuses: <ul style="list-style-type: none"> Service logs an error Service logs either a warning or an error 	Yes -- a High severity alert if the service logs an error

5.8.2.2. Add Alert Rules

To add a new alert rule, in the **Alert Rules** page do the following:

- From the "Alert Type" drop-down list, select an alert type.

	send email to	send snmp trap	severity level	actions
default		Medium	Edit Delete	
default		Critical	Edit Delete	
default		High	Edit Delete	
default		High	Edit Delete	
default		Critical	Edit Delete	

- Configure a rule for that alert type. A rule defines the conditions that will trigger the alert. The options (as presented in the rule-configuring interface) will vary according to the alert type. For descriptions of alert types see "**Review Supported Rule Types and Pre-Configured Rules**" (page 345).
- If you want the alert to include sending an **email notification** to the default system administrator email address(es), leave the "Use Default Email Address" option checked. If you want to customize the target email addresses for this particular alert, uncheck the "Use Default Email Address" option and enter the address(es) in the "Target Email" field (for multiple addresses, use comma separation). If you do not want email notification for this alert, uncheck the "Use Default Email Address" option and leave the "Target Email" field empty.

Note: The default system administrator email address is configured in the "**SMTP/Email Settings for Alerts/Notifications**" (page 294) section of the CMC's **Configuration Settings** page.

- If you want the alert to include sending an **SNMP trap** to your SNMP management system, select the "Send SNMP Trap" checkbox.

Note Trap destination settings are configured in the "**SNMP Trap Destination Settings**" (page 296) section of the CMC's **Configuration Settings** page. In the traps that HyperStore generates and sends to your specified destination, the OID is enterprises.16458.4.1.1.1. The trap payload also indicates the specific HyperStore host on which the trap-triggering event occurred. HyperStore uses SNMP version 2c.

- Select a **Severity level** to assign to the alert. You can choose from Critical, High, Medium, or Low.

Note: Subsequently, if the alert occurs and is displayed in the [Alerts](#) page, the display includes the severity level that you assigned to the alert. Also, in the [Alerts](#) page you will be able to sort the displayed alert list by severity level.

- Click **Create**.

Your new alert rule then displays in the **Active Alert Rules** section of the page.

IMPORTANT: If your HyperStore system has multiple service regions, a drop-down list displays at the top of the **Alert Rules** page so you can select a region for which to manage alert rules. Any alert rule actions that you take in one region — such as adding, editing, or disabling a rule — will **not** carry over to the other region(s). If you want an alert rule change to apply to all of your regions, you must make that change **for each region one at a time**, using the **Alert Rules** page.

5.8.2.3. Edit Alert Rules

In the **Alert Rules** page you can edit existing alert rules — for example to change a threshold value, change the severity level assigned to a particular rule, or change email notification settings for a particular rule.

Note If you want to change the default email address(es) that alert emails are sent to, go to the CMC's [Configuration Settings](#) page and edit the "Default Email Address to Receive Notifications" setting. You do not need to make any change on the **Alert Rules** page to do this. The same is true for changing the destination for sending SNMP traps: do it on the **Configuration Settings** page.

To edit a rule:

- Click **Edit** next to the rule in the **Rules** list.
- Modify the rule as desired.

Note: To change from using the default target email address to using a different email address specifically for this rule, uncheck the "Use Default" option to the right of the rule statement and

then enter the custom email address in the text field. To disable email notification for just this rule, leave the text field empty.

3. Click **Done** to apply your changes.

Your modified rule will then display in the **Rules** list.

5.8.2.4. Disable Alert Rules

To temporarily disable an alert rule, in the **Alert Rules** page select the checkbox to the left of the rule and then click **Disable** on the lower right of the page. If you want you can disable multiple rules at the same time by selecting multiple rules' checkboxes and then clicking **Disable**.

A disabled rule will remain in your **Rules** list display, but the rule will no longer be applied by the HyperStore system. In the rules list a disabled rule is distinguished by its name being grayed out.

To re-enable a rule that has been disabled, select the checkbox to the left of the rule and then click **Enable** on the lower right of the page. The HyperStore system will resume applying the alert rule.

5.8.2.5. Delete Alert Rules

To delete an alert rule, in the **Alert Rules** page click **Delete** to the right of the rule in the **Rules** list. The rule will no longer display in the **Rules** list and will no longer be applied by the HyperStore system.

If you want you can delete multiple rules at the same time by selecting the checkboxes to the left of those rules and then clicking **Delete** on the lower right of the page.

See Also:

- "How HyperStore Implements Alerts" (page 351)

5.8.3. How HyperStore Implements Alerts

In the CMC's [Alert Rules](#) page you can create rules for having the HyperStore system generate alerts when specified events occur. When an event covered by an alert rule occurs, the HyperStore system:

- Sends an SNMP trap, if the alert rule for the event includes sending an SNMP trap. The trap is sent to the SNMP destination configured in the "[SNMP Trap Destination Settings](#)" (page 296) section of the CMC's [Configuration Settings](#) page.
- Displays an alert in the CMC's [Node Status](#) page (in the **Alert List** for the node on which the event occurred) and also in the [Alerts](#) page.
- Emails an alert notification to specified system administrator addresses (unless you disable email notification, which you can do on a per-rule basis as described in).

Alerting through the CMC and through email and SNMP is implemented as follows:

- When the event that is specified by the alert rule occurs, an alert email is sent to the configured email address(es), and a trap is sent if you've enabled SNMP as part of the rule. An alert notice is also displayed on the [Node Status](#) page and on the [Alerts](#) page.
- The sending of the alert email sets off a 24 hour hold on any further alert emails in association with the same alert rule on the same node. The same hold period applies to SNMP trap sending. The 24 hour

hold works like this:

- So long as the original instance of the alert remains unacknowledged by administrators, any additional events that trigger the same alert rule on the same node will not generate additional email notifications (or SNMP traps). Such additional event instances will only generate additional alerts in the CMC interface, so that the alert's "Count" value increments.
- As soon as an administrator acknowledges the alert in the CMC's **Node Status** page or in the **Alerts** page, the 24 hour hold is lifted and any new instances of the event will trigger a new email notification (and SNMP trap if enabled).
- If 24 hours pass without the original alert being acknowledged by an administrator, then the next subsequent instance of the event will trigger the sending of a new alert email (and SNMP trap if enabled). This then initiates a new 24 hour hold period.

5.8.3.1. Handling of Alerts for ERROR or WARN Log Messages

For alert rules based on ERROR messages in service logs, an alert is written to the CMC's **Node Status** page and **Alerts** page for **each individual ERROR message** that occurs in the service logs. For alert rules based on WARN messages in service logs, the alerts are written for each individual WARN message that occurs in the service logs and also for each individual ERROR message.

However, for purposes of email notification, each additional log message from the same service is considered the same type of "event", and consequently the 24 hour hold described above applies. For example, if you have an alert rule in place for ERROR messages in the S3 service logs:

- If ERROR message "X" appears in the S3 service log on node1, an alert is written to the CMC's **Node Status** and **Alerts** pages, and an email notification is sent.
- If 10 minutes later ERROR message "X" appears again in the S3 service log on node1, another alert is written to the CMC's **Node Status** and **Alerts** pages, but no additional email notification is sent.
- If 10 minutes later ERROR message "Y" appears in the S3 service log on node1, another alert is written to the CMC's **Node Status** and **Alerts** pages, but no additional email notification is sent. Although ERROR message "Y" is different from ERROR message "X", it's still considered to be part of the same alert rule and consequently it does not trigger an additional email notification.

Note In describing alerting behavior for the second and third messages, the scenario above presumes that no administrator has yet acknowledged the earlier ERROR message alert. Recall that acknowledging an alert releases the 24-hour hold on additional emails for that alert type, in which case a new instance of the underlying event triggers a new email notification.

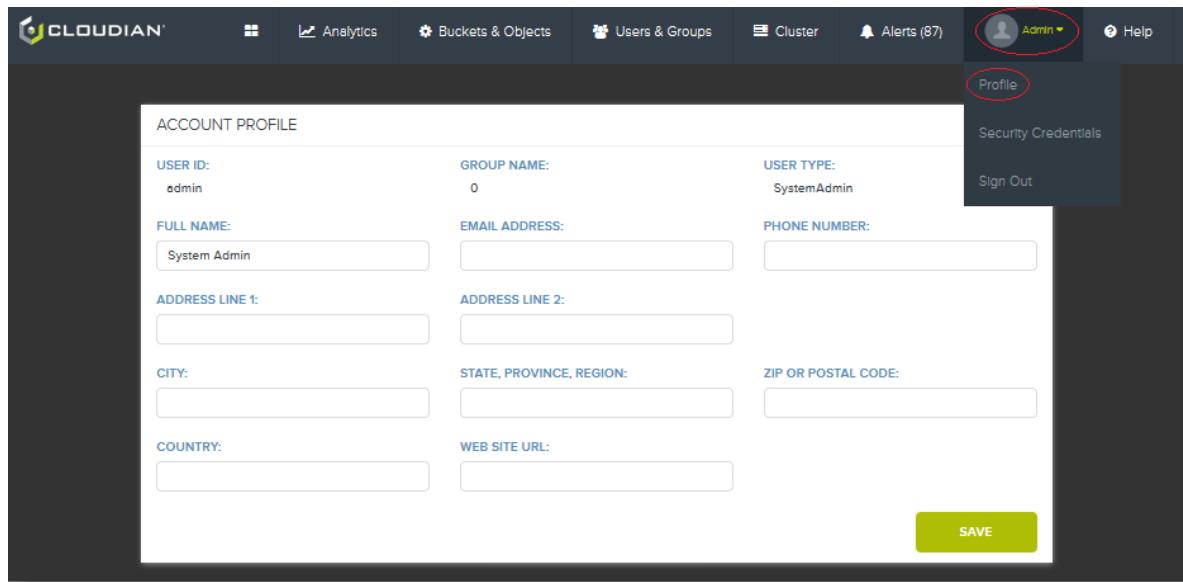
5.9. My Account

In the upper right of the CMC interface, holding your cursor over your user name displays a drop-down menu from which you can choose these options relating to your user account:

- [Profile](#)
- [Security Credentials](#)

5.9.1. Profile

Path: Drop-down menu under your user name (at top right) → **Profile**



Supported task:

- Change your contact information

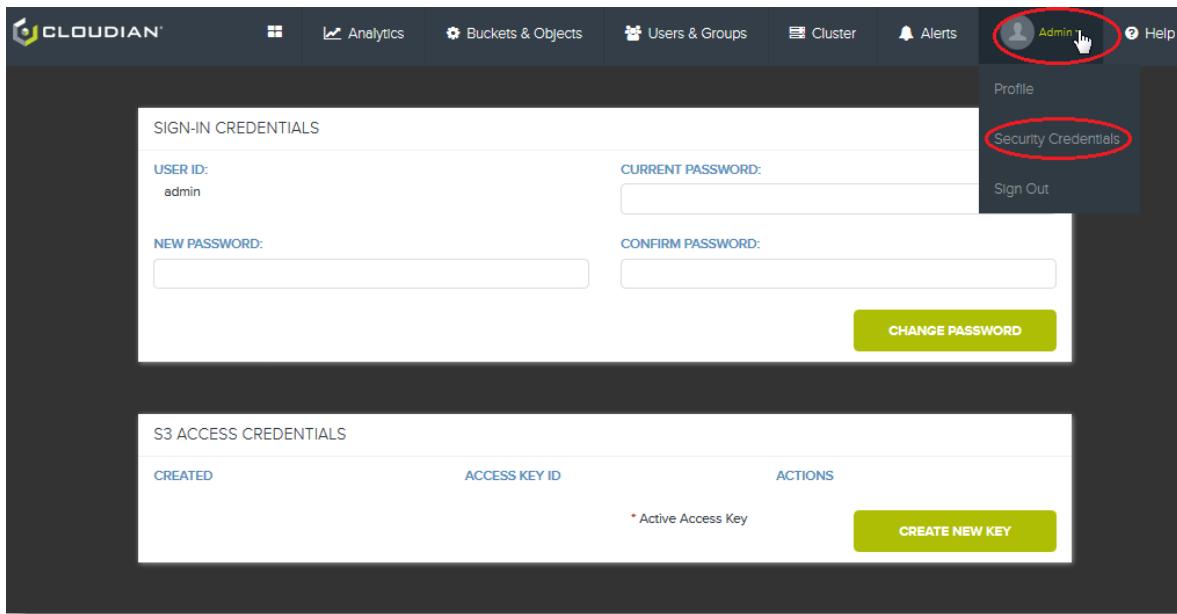
Change Your Contact Information

1. Update your contact information.
2. Click **Save**.

Note Changing your system administrator email address in this page does not impact the sending of system notification emails. To change the email address to which notification emails are sent, use the **"Default Email Address to Receive Notifications"** (page 295) setting on the **Configuration Settings** page.

5.9.2. Security Credentials

Path: Drop-down menu under your user name (at top right) → **Security Credentials**



Supported task:

- Change your Console password
- Manage your S3 access keys

5.9.2.1. Change Your Console Password

Note If the system is configured to use LDAP authentication for your user group, the CMC's **Change Password** function will not be available and the instructions below are not applicable to you. Instead, your password should be controlled through your organization's LDAP system.

1. In the "Current Password" field, enter your current password.
2. In the "New Password" field, enter a new password that you will use to log into the CMC. Then in the "Confirm Password" field, enter the new password again.

Passwords must meet the following conditions:

- Minimum of nine characters, maximum of 64 characters
- Must contain at least three of these four types of characters:
 - Lower case letters
 - Upper case letters
 - Numbers
 - Special characters such as !, @, #, \$, %, ^, etc.

3. Click **Change Password**.

5.9.2.2. Manage Your S3 Access Keys

- To create a new S3 data access key, in the **S3 Access Credentials** dialog click **Create New Key**. A new access key ID then displays in the access key list.

- To view the secret access key that corresponds to an access key ID, to the right of the access key ID click **View Secret Key**. A secret key display box appears which enables you to view your secret key and to copy it using <Ctrl>-c if you want to. Note that the **OK** and **Cancel** buttons have no effect other than to close the secret key display box.

Note: If you view your secret key(s) multiple times during one Console login session, your browser may display a "Prevent this page from creating additional dialogs" checkbox that appears as if it's part of the Console UI. Do not select this checkbox. If you select this checkbox and then click **OK** you will no longer be able to view your secret access keys during your current Console login session.

- To activate or deactivate an access key, to the right of the displayed access key ID click **Activate** or **Inactivate**.
- To delete an access key, to the far right of the displayed access key ID click **Delete**. You will be asked to confirm that you want to delete the key.

5.10. Customizing the CMC

5.10.1. Showing/Hiding CMC UI Functions

The CMC provides granular configuration control over which UI functions and sub-functions display for the three types of users that the CMC supports: system admins, group admins, and regular S3 service users. The table below summarizes the CMC's configurability for showing or hiding certain functionality. The third column indicates which user types have access to the functionality by default, or whether the function is by default hidden from all user types. The fourth column indicates the configuration setting that controls access to the functionality — all settings are in *mts-ui.properties.erb* on your Puppet master node unless otherwise noted.

Note If you make changes to a configuration file, be sure to then use the installer (*cloudianInstall.sh* on your Puppet master) to push the changes out to your cluster and then restart the CMC service.

Function Area	Functionality	Default Availability	Controlling Setting
Manage Users	Basic user management interface	System admins and group admins	"admin.manage_users.enabled" (page 526)
	Create users	System admins and group admins	"admin.manage_users.create.enabled" (page 527)
	Edit users	System admins and group admins	"admin.manage_users.edit.enabled" (page 527)
	Delete users	System admins and group admins	"admin.manage_users.delete.enabled" (page 528)
	View and manage users' security credentials	System admins and group admins	"admin.manage_users.edit.user_credentials.enabled" (page 528)
	View and manage users' stored data	System admins and group admins	"admin.manage_users.viewuserdata.enabled" (page 528)

Function Area	Functionality	Default Availability	Controlling Setting
	Edit users' Quality of Service settings	System admins and group admins	"admin.manage_users.edit.user_qos.enabled" (page 529)
Manage Groups	Basic group management interface	System admins and group admins	"admin.manage_groups.enabled" (page 529)
	Create groups	System admins	"admin.manage_groups.create.enabled" (page 530)
	Edit groups	System admins and group admins	"admin.manage_groups.edit.enabled" (page 530)
	Delete groups	System admins	"admin.manage_groups.delete.enabled" (page 530)
	Set default user QoS for a group	System admins and group admins	"admin.manage_groups.user_qos_groups_default.enabled" (page 530)
Billing whitelist	Set source IP addresses allowed free traffic	Hidden	"admin_whitelist_enabled" (page 481) in <i>common.csv</i>
User Account Self-Management	Edit own profile	System admins, group admins, and regular users	"account.profile.writeable.enabled" (page 531)
	Basic security credentials interface	System admins, group admins, and regular users	"account.credentials.enabled" (page 531)
	Manage own S3 credentials	Group admins and regular users	"account.credentials.access.enabled" (page 532)
	Change own CMC password	System admins, group admins, and regular users	"account.credentials.signin.enabled" (page 532)
Usage Reporting	Basic usage reporting interface	System admins, group admins, and regular users	"usage.enabled" (page 533)
	Reporting on HTTP request rates and byte transfer rates	Hidden	"Track/Report Usage for Request Rates and Data Transfer Rates" (page 299) in the CMC Configuration Settings page
Auto-Tiering	Allow buckets to be configured for auto-tiering	Hidden	"Enable Auto-Tiering" (page 302) in the CMC Configuration Settings page

5.10.2. Rebranding the CMC UI

If you wish you can customize various aspects of the CMC interface to reflect your organization's own branding. The interface elements that you can customize are:

- Logo
- Browser tab title

- Color scheme
- Application name in URLs

Before making any changes to the CMC's appearance you should back up existing CMC interface element files. On at least one of your CMC nodes, do the following to back up existing UI files that you may be replacing with your custom files:

Create backup directory:

```
$ mkdir /tmp/webbackup
```

Copy relevant UI files to the backup directory:

```
$ cp /opt/tomcat/webapps/Cloudian/WEB-INF/resources_en_US.properties /tmp/webbackup/
$ cp /opt/tomcat/webapps/Cloudian/css/master.css /tmp/webbackup/
$ cp /opt/tomcat/webapps/Cloudian/images/logo_new.png /tmp/webbackup/
$ cp /opt/tomcat/webapps/Cloudian/images/logo_2.png /tmp/webbackup/
$ cp /opt/tomcat/webapps/Cloudian/images/favicon.ico /tmp/webbackup/
```

Once you've backed up the default UI files, you can proceed with applying your customized branding:

1. On your Puppet Master node, replace or modify CMC files to make whichever of the following customizations you want:

Replace logos

To replace the logos that display in the UI, you will need new logo images with the following names and sizes:

- logo_new.png — Login screen logo - 225 X 225
- logo_2.png — Header logo - 144 X 28
- favicon.ico — Favicon icon for browser tab - 16 X 16

Copy these new files into the following directory on the Puppet master node:

```
/etc/cloudian-<current-version>-puppet/modules/cmc/files/Cloudian/images
```

Change browser tab title

By default the title that displays at the top of a browser tab for the UI is "Cloudian® Management Console". To change this title, do the following:

- a. On the Puppet master node, copy the *resources_en_US.properties* file from */opt/tomcat/webapps/Cloudian/WEB-INF/* to */etc/cloudian-<current-version>-puppet/modules/cmc/files/Cloudian/WEB-INF/*. For example:

```
$ cp -vp /opt/tomcat/webapps/Cloudian/WEB-INF/resources_en_US.properties
/etc/cloudian-7.2-puppet/modules/cmc/files/Cloudian/WEB-INF/resources_en_US.properties
```

- b. Edit the */etc/cloudian-<current-version>-puppet/modules/cmc/files/Cloudian/WEB-INF/resources_en_US.properties* file as follows:

- i. In the file find the following line:

```
header.title=Cloudian® Management Console
```

- ii. Change it to the desired browser tab name:

```
header.title=<NEW TAB NAME>
```

- c. Save the updated *resources_en_US.properties* file.

Change UI color scheme

To change the UI's color scheme (as defined in its CSS file), do the following:

- a. On the Puppet master node, copy the *master.css* file from */opt/tomcat/webapps/Cloudian/css/* to */etc/cloudian-<current-version>-puppet/modules/cmc/files/Cloudian/css/*. For example:

```
$ cp -vp /opt/tomcat/webapps/Cloudian/css/master.css  
/etc/cloudian-7.2-puppet/modules/cmc/files/Cloudian/css/master.css
```

- b. Edit the */etc/cloudian-<current-version>-puppet/modules/cmc/files/Cloudian/css/master.css* file to replace all occurrences of the existing color codes with the desired new values.
- c. Save the updated *master.css* file.

Change the application name in the CMC's URLs

By default all CMC URLs include the application name "Cloudian" (for example

https://<host>:8443/Cloudian/dashboard.htm). To change the application name, do the following:

- a. On the Puppet master node, in the file */etc/cloudian-<current-version>-puppet/manifests/extdata/common.csv*, edit this setting:

```
cmc_application_name,Cloudian
```

Replace "Cloudian" with a different application name string of your choosing. Use only alphanumeric characters, with no spaces, dashes, or underscores.

- b. Save the updated *common.csv* file.

2. Use the installer to push your edits to the cluster and restart the CMC to apply your changes. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

IMPORTANT: If you customize the branding of the CMC, and then subsequently upgrade your HyperStore system to a newer version, only your customized logos and your customized application name will be retained after the upgrade. After the upgrade you will need to re-implement any changes that you had made to the browser tab title and/or the color scheme, by again following the instructions above.

5.10.3. Implementing Single Sign-On for the CMC

To enable integration between a portal and the Cloudian Management Console, the Cloudian HyperStore system employs a one-way hash based Single Sign-On (SSO) solution. It allows for cross-domain sign-ons from the portal to CMC.

User provisioning is beyond the scope of the provided SSO solution. The HyperStore provides an [Admin API for user provisioning](#) but the implementation of user mapping is left to the portal application integrating with CMC.

Note for customers upgrading from a HyperStore version older than 5.0

The CMC's SSO solution was redesigned in HyperStore version 5.0. The following changes were made:

- The SSO Secure URL (*ssosecurelogin.htm*) now directly creates an authenticated CMC session instead of returning a CMCSO cookie. Therefore, it is now possible to do cross-domain sign-ons from

a portal to CMC. The portal and CMC no longer have to be on the same top level domain such as ".cloudian.com".

- *ssosecurelogin.htm* also takes an optional query string *redirect=RELATIVE_OR_ABSOLUTE_URL*, which can be used to redirect the client to a CMC interior page upon successful sign-on.
- The CMC logout URL (*logout.htm*) now takes an optional query string *redirect=RELATIVE_OR_ABSOLUTE_URL*, which can be used to redirect the client back to a portal page after signing out from the CMC.

This redesigned SSO solution provides backward compatibility. If you already have working SSO from a portal to a version of the CMC older than 5.0, it should remain working.

However, some of the SSO methods from earlier releases have been deprecated. Cloudian, Inc. recommends not using these methods, and in a future release support for them will be discontinued.

- The method for having your portal application create a CMCSSO cookie has been deprecated. Use SSO secure login API (*ssosecurelogin.htm*) instead.
- The method for having the CMC create a CMCSSO cookie using password (*ssologin.htm*) has been deprecated. Use SSO secure login API (*ssosecurelogin.htm*) instead.
- The CMC SSO logout API (*ssologout.htm*) has been deprecated. Use CMC's regular logout URL (*logout.htm*) instead.

5.10.3.1. How SSO for the CMC Works

Cloudian HyperStore SSO is ideal for sites that already have an authentication model in place using a browser-login session and that want to incorporate the Cloudian Management Console into their web portal application.

The idea is that a portal application calculates an one-way hash (also known as a signature) based on Cloudian HyperStore user identification, timestamp and the shared key. Then the user's browser accesses *ssosecurelogin.htm* with the signature. The CMC checks for this signature to determine whether a user is authenticated or not. If the signature is found valid, access to the CMC from the client will skip the login page and take the user directly to a CMC interior page such as the **Buckets & Objects** page.

IMPORTANT: To use the Cloudian HyperStore SSO feature, the following system configuration settings must be set to "true":

-- "**cmc_web_secure**" (page 482) in *common.csv* -- This is set to true by default. Leave it true if you want to use SSO.

-- "**sso.enabled**" (page 534) in *mts-ui.properties.erb* -- This is set to false by default. Change it to true if you want to use SSO.

Also in *mts-ui.properties*, if you enable SSO functionality (by setting *sso.enabled* to true), then for security reasons you should set *sso.shared.key* and *sso.cookie.cipher.key* to custom values. **Do not use the default keys.**

5.10.3.2. CMC SSO Secure Login Using One-Way Hash

The single sign-on with one-way hash method relies on a one-way hash of query string parameters (also known as a signature).

The following HTTP API, using a signature, prompts the CMC to create an authenticated session for the client that submitted the request:

Note Submit this as a GET, not a POST. POST is not supported for CMC SSO login.

```
https://<cmc_FQDN>:<cmc_port>/Cloudian/ssosecurelogin.htm?user=USERID&group=GROUPID&timestamp=TIMESTAMP&signature=SIG&redirect=RELATIVE_OR_ABSOLUTE_URL
```

- *user*: Cloudian HyperStore userId of the user
- *group*: Cloudian HyperStore groupId of the group to which the user belongs
- *timestamp*: Current Epoch time in milliseconds (eg. "1346881953440"). The timestamp is used to implement the configurable request expiration (*mts-ui.properties*: *sso.tolerance.millis*; expiration defaults to one hour).
- *signature*: This is the URI encoding of the base64 representation of the calculated signature. For further information see below.
- *redirect*: This optional parameter can be used to redirect the client to the given URL upon successful sign-in. It is typically set to a CMC interior page such as *bucket.htm*.

Each value must be URL-encoded by the client. Order of the parameters does not matter.

If the signature is found valid, the CMC creates an authenticated session for the HyperStore user, allowing the client to skip the login page and access to a CMC interior page.

How to Create the Signature

The portal server can create the signature by the following steps.

1. Assemble the query string.
 - `queryString = "user=USERID&group=GROUPID×tamp=TIMESTAMP"`
2. Calculate one-way hash for the queryString using the standard HmacSHA1 and the CMC SSO shared key. The shared key is configured by *mts-ui.properties*: *sso.shared.key*.
 - `hashresult = HmacSHA1(querystring, sharedkey)`
3. Base64 encode the resulting hash.
 - `base64string = Base64Encode(hashresult)`
4. URI encode the base64 encoded hash result.
 - `signature = encodeURIComponent(base64string)`

Note: When using the queryString to create the signature, do not URL-encode the queryString. Also do not reorder the items. (By contrast, when the client subsequently submits the SSO secure login request to the CMC, it's desirable to URL encode the request queryString.)

For a sample of a Python script that uses the one-way hash login API, see "**Cloudian HyperStore SSO Sample Script**" (page 362).

Access to a CMC's Interior Page

After creating the signature, the portal server can return an HTML page with a hyperlink to the CMC SSO secure login API. The following example will display CMC's **Buckets & Objects** page (*bucket.htm*) embedded in the inline frame on the portal's page.

```
<iframe src="https://<cmc_FQDN>:<cmc_port>/Cloudian/ssosecurelogin.htm
?user=USERID&group=GROUPID&timestamP=TIMESTAMP&signature=SIG
&redirect=bucket.htm"></iframe>
```

CMC SSO Secure Login HTTP Response

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is given, the CMC's SSO secure login API returns an HTTP redirect response.

- **If the request was successful**, the redirect response will take the client to the URL specified by *redirect*.
- **If the request failed**, the redirect response will take the client to the CMC's Login panel.

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is not given, the CMC's SSO secure login API returns an HTTP response with content-type "text/plain".

- **If the request was successful**, the HTTP response status is 200 OK.
- **If the request failed**, a 400 BAD REQUEST status is returned, along with a plain text status description. Possible reasons for failure include:
 - Missing required parameters
 - SSO token already exists (request is ignored)
 - Timestamp in request is outside of configured tolerance range
 - Invalid signature
 - Invalid credentials (group ID and/or user ID is invalid)

CMC Logout

This API method allows for immediately invalidating the CMC session.

```
https://<cmc_FQDN>:<cmc_port>/Cloudian/logout.htm&redirect=RELATIVE_OR_ABSOLUTE_URL
```

- *redirect*: This optional parameter can be used to redirect the client to the URL after logging out from the CMC. It is typically set to a portal page. The URL must be URL-encoded by the client.

CMC Logout HTTP Response

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is given, the CMC's logout API returns an HTTP redirect response to take the client to the given URL after logging out from the CMC.

If *redirect=RELATIVE_OR_ABSOLUTE_URL* is not given, the CMC's logout API returns an HTTP redirect response to take the client to the CMC's Login panel.

Logging Out from the CMC and Portal at Once

You may want the logout link on the portal page to also trigger logout from the CMC. You can achieve this by using the *redirect* parameter.

For example, if you have the portal's logout link like this:

```
<a href="/auth/logout">Logout</a>
```

You can change it to the following:

```
<a href="https://<cmc_FQDN>:<cmc_port>/Cloudian/logout.htm  
?redirect=https:%2F%2F<portal_FQDN>:<portal_port>%2Fauth%2Flogout">Logout</a>
```

- The redirect URL must be an absolute URL including the protocol (e.g. https://) and portal's FQDN.
- The redirect URL must be URL-encoded.

5.10.3.3. Cloudian HyperStore SSO Sample Script

Below is a sample Python script that outputs a HyperStore SSO secure login URL for use with the one-way hash method of having the CMC create a cookie. The script also creates an SSO logout URL.

```
#!/usr/bin/python

import time
import hmac
import hashlib
import base64
import urllib

# TODO: Move these config options to configuration file
SSO_DOMAIN = 'cmc.cloudian.com'
SSO_PORT = 8443
SSO_KEY = 'aa2gh3t7rx6d'

# TODO: Dynamically choose user/group based on the user
# and group you want to login using.
SSO_USER = 'sso@group'
SSO_GROUP = 'ssogroup'

# Do Not Change
SSO_PROTO = 'https://'
SSO_PATH = 'Cloudian/ssosecurelogin.htm'
SSO_LOGOUT_PATH = 'Cloudian/ssologout.htm'

def sso_sig(user, group, timestamp):
    # query string with no urlencoding for signature
    signme = 'user=%s&group=%s&timestamp=%s' % (user, group, timestamp)
    hmacsha1 = hmac.new(SSO_KEY, signme, hashlib.sha1).digest()
    return base64.b64encode(hmacsha1)

def sso_url(user, group):
    timestamp = int(time.time() * 1000)
    signature = sso_sig(user, group, timestamp)
    params = {'user': user,
              'group': group,
              'timestamp': timestamp,
              'signature': signature}
    query = urllib.urlencode(params)
    url = '%s%s:%d/%s?%s' % (SSO_PROTO, SSO_DOMAIN, SSO_PORT, SSO_PATH, query)
    return url
```

```
def sso_logout_url():
    url = '%s%s:%d/%s' % (SSO_PROTO, SSO_DOMAIN, SSO_PORT, SSO_LOGOUT_PATH)
    return url

print 'login: ' + sso_url(SSO_USER, SSO_GROUP)
print '\nlogout: ' + sso_logout_url()
```

Note The sample script hard-codes the SSO secret key, which is not advisable for actual practice. In practice, you should keep the secret key safely on the server side.

This page left intentionally blank

Chapter 6. Node and Cluster Operations

6.1. Starting and Stopping Services

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Start or Stop Services on All Nodes in the Cluster" (page 365)**
- **"Start or Stop Services on One Node" (page 367)**
- **"Automatic Service Start on Node Reboot" (page 368)**

You can start, restart, or stop a service across all the nodes in your cluster by using the HyperStore installer. If instead you want to start, restart, or stop a service on just one particular node, you can do so through the CMC or by using a HyperStore service initialization script. Note also that HyperStore services are configured to start automatically when you reboot a node.

6.1.1. Start or Stop Services on All Nodes in the Cluster

The interactive tool that you used to install the HyperStore system — *cloudianInstall.sh* — can also be used to manage HyperStore services. The tool enables you to start, restart, or start one or more services **on all nodes at once** (or to put it more precisely, on all nodes in rapid succession).

Note The installation tool does not support managing a service on just one particular node.

1. On your **Puppet master node**, go to the staging directory in which the HyperStore install tool resides. This is the directory in which you unpacked the HyperStore release package and initiated the software installation.

```
[root]# cd <your-installation-staging-directory>
```

2. Launch the install tool:

```
[root]# ./cloudianInstall.sh
```

This displays the top-level menu of options.

```

Cloudian HyperStore(R) 7.2 Installation/Configuration
-----
0 ) Run Pre-Installation checks
1 ) Install Cloudian HyperStore
2 ) Cluster Management
3 ) Upgrade From a Previous Version
4 ) Advanced Configuration Options
5 ) Uninstall Cloudian HyperStore
6 ) Help
x ) Exit

```

Choice: []

3. Choose "Cluster Management", then from the sub-menu that displays choose "Manage Services". This displays the "Service Management" sub-menu:

```

Service Management
-----
0 ) All services
1 ) Redis Credentials
2 ) Redis QOS
3 ) Cassandra
4 ) HyperStore service
5 ) S3 service
6 ) Redis Monitor
7 ) Cloudian Agent
8 ) DNSMASQ
9 ) Cloudian Management Console (CMC)
10) IAM
11) SQS
P ) Puppet service (status only)
X ) Quit

```

You can execute the following list of commands:
start,stop,status,restart,version,node-start,node-stop

Select a service to manage: []

Note As an alternative to launching *cloudianInstall.sh* and navigating to the "Service Management" menu, you can access this menu directly by launching the *cloudianService.sh* script in your installation staging directory.

- a. At the prompt, enter a service number from the menu. To manage all services enter option (0).

Note The Admin Service and (if you've enabled it) the IAM Service are bundled with the S3 Service. Any operation that you apply to the S3 Service applies also to these other services.

- b. At the prompt that appears after you make your service selection, enter a service command: start, stop, status, restart, or version. (The "version" option is supported only for the S3 Service.)

The service command you enter will be applied to all nodes on which the service resides. For example, if you choose Cassandra and then enter "start", this will start Cassandra on all nodes on which it is installed. Likewise if you choose S3 and then "status", this will return the status of the S3 Service on each node on which it is installed. And if you choose "All services" and then "stop", this will stop all services on all nodes.

Note: From the "Service Management" menu all you can do for the Puppet service is check its status. To stop or start the Puppet daemons, from the installer's main menu choose "Advanced Configuration Options". From the advanced sub-menu that displays you can stop or start the Puppet daemons.

6.1.2. Start or Stop Services on One Node

You can start, restart, or stop a service on just one particular node by using the CMC's [Node Status](#) page.

As an alternative to using the CMC for this task, you can use the following commands. These commands can be run from any directory on the target node.

```
systemctl start|restart|stop|is-active|version <servicename>
```

Example:

```
systemctl restart cloudian-s3
```

The table below shows all HyperStore services and their corresponding <servicename>.

Service	<servicename>
S3 Service, Admin Service, and (if you've enabled it) the IAM Service. These three services start and stop together.	<i>cloudian-s3</i>
Cassandra Service	<i>cloudian-cassandra</i>
HyperStore Service	<i>cloudian-hyperstore</i>
Redis Credentials Service	<i>cloudian-redis-credentials</i>
Redis QoS Service	<i>cloudian-redis-qos</i>
Redis Monitor	<i>cloudian-redismon</i>
Cloudian Management Console	<i>cloudian-cmc</i>
Cloudian Monitoring Agent	<i>cloudian-agent</i>
Dnsmasq	<i>cloudian-dnsmasq</i>

6.1.2.1. Stop or Start All Services on One Node

To stop all of the services on a single node, stop each individual service (as described in "Start or Stop Services on One Node" (page 367)) in this order:

1. CMC
2. Cloudian Monitoring Agent
3. Redis Monitor
4. S3 Service
5. HyperStore Service
6. Redis QoS
7. Redis Credentials
8. Cassandra
9. Dnsmasq

To start all of the services on a single node, start each individual service in this order:

1. Cassandra
2. Redis Credentials
3. Redis QoS
4. HyperStore Service
5. S3 Service
6. Redis Monitor
7. Cloudian Monitoring Agent
8. CMC
9. Dnsmasq

6.1.2.2. Checking the ISO Version on a HyperStore Appliance machine

On a HyperStore Appliance machine, you can check the ISO version (the version of the HyperStore ISO file from which CentOS was installed on the machine) by changing into the `/root/CloudianPackages` directory and then running the following command:

```
cat ISOVERSION
```

Example:

```
[root@cloudian-node3 CloudianPackages]#cat ISOVERSION
Cloudian CentOS 7.4 Rev d56af03 Custom - 09/27/17
```

6.1.3. Automatic Service Start on Node Reboot

By default all of the HyperStore services on a host are configured to automatically start when the host is booted up (the configuration setting is `common.csv`: "**service_starts_on_boot**" (page 460), which defaults to "true"). This includes `dnsmasq` if it was included in your HyperStore software installation.

Note *ntpd* is configured to automatically start on host boot-up. By design, the sequencing is such that *ntpd* starts up before major HyperStore services do.

Cloudian Inc. recommends that after booting a HyperStore host, you verify that *ntpd* is running. You can do this with the *ntpq -p* command. If *ntpd* is running this command will return a list of connected time servers.

IMPORTANT: If you are rebooting multiple nodes, make sure that each node is back up for at least one second before moving on to reboot the next node.

6.2. Adding Nodes

This procedure is for **adding one or more nodes to an existing HyperStore data center**. During this procedure you will:

- Prep the new node(s)
- Verify that your existing cluster is in a proper condition to add nodes
- Add the new node(s) to the cluster
- Rebalance data within the cluster
- Clean from the existing nodes data that they are no longer responsible for

IMPORTANT: Be sure about the node or nodes that you are adding to the cluster, before you add them to the cluster:

* Once you have added a node to the cluster's Cassandra ring **you cannot simply "undo" the process**. If after adding a new node to the Cassandra ring you were to change your mind about keeping the node in the cluster, you would still be required to rebalance data within the cluster (so that data is streamed in to the new node) and then afterwards you would need to decommission the node (so that data is streamed away from the new node). That is the only way that the system will allow you to remove the node from the cluster.

* Once you add a node to your cluster, **HyperStore does not support adding disks to the node**. Make sure that the node or nodes that you are adding have sufficient disk capacity to meet your needs.

6.2.1. Special Requirements if an Existing Node is Down

The CMC's **Add Node** function **will not work if an existing node in your cluster is down or unreachable**. If you have more nodes in your cluster than are required by your configured storage policies and one of those nodes is permanently down, follow the procedure for "**Removing a Node**" (page 390) to remove the dead node from the cluster. After you complete that procedure you can then follow this procedure for Adding Nodes. If your current cluster has **only the minimum number of nodes required by your storage policies and one of those nodes is dead**, contact Cloudian Support for guidance.

6.2.2. Preparing to Add Nodes

Before you add a node or nodes to your cluster, prepare by taking the actions below. (If the new node is a HyperStore Appliance machine you can skip Steps 1 and 4.)

1. Make sure the new host(s) **meet requirements** for:
 - Hardware specs and operating system: See "Host Hardware and OS Requirements" in the HyperStore installation Guide
 - Open listening ports: See "HyperStore Listening Ports" in the Reference section of the *Hyper-Store Installation Guide*
2. Make sure you have the **information you will need** to complete this procedure: each new node's host name, IPv4 address, internal interface name (optional), and root password.
3. **Start the new host(s)**, if not already running.
4. From your Puppet master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

More detail

- a. In your existing cluster, on your Puppet master node launch the *system_setup.sh* tool. If your Puppet master is a HyperStore Appliance the tool will be under */root/CloudianTools*, or if it's a software-only HyperStore node the will be under your installation staging directory.

```
./system_setup.sh
```

- b. In the tool's main menu select "9" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.
- c. Use the node preparation task menu to prepare the node:
 - Complete the configuration of network interfaces for the node, if you haven't already.
 - Set the timezone for the node.
 - Install and configure HyperStore prerequisites on the node.
 - Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
 - After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.
- d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the *system_setup.sh* tool.

IMPORTANT: If the new node(s) are not HyperStore Appliances and **if you do not use *system_setup.sh* to format the data disks on the new node(s)**, then in the installation staging directory on your Puppet master node you must for each new node create a text file named *<hostname>_fslist.txt* that specifies the new node's data mount points, in this format:

```
<devicename> <mountpoint>
<devicename> <mountpoint>
etc...
```

5. **Make sure the cluster is in proper condition** to add nodes:

- a. If there are any operations in-progress in the CMC's **Operation Status** page, wait for them to finish (or otherwise, the **Add Node** operation will automatically stop them).

More detail

When you initiate the **Add Node** operation in the CMC (as described in "Adding Nodes" below), the system will automatically stop any in-progress *repair*, *repairec*, *cleanup*, or *cleanupec* operations in the service region. If there is an in-progress operation that you do not want the system to stop, wait until the operation completes before you add nodes to your cluster.

OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE
cleanup	hyperstore-demo1	✓ completed	<div style="width: 100%;">100%</div>	Apr-09-2018 16:18	Apr-09-2018 17:19
repair	hyperstore-demo2	✓ completed	<div style="width: 100%;">100%</div>	Apr-08-2018 05:05	Apr-08-2018 05:14

- If a repair is running on a node because you recently initiated a "[replacedisk](#)" operation on that node, Cloudian recommends waiting until the repair completes before you add a node to your cluster.
- If you proceed with adding a node at a time when a repair is in progress, and the system automatically stops the repair, do not run the "-resume" option on that repair after you've added the node. Adding a node affects the token range distribution within the cluster, so resuming an interrupted repair operation afterward is not supported. If you want to repair a node on which repair was interrupted, wait until after you've completed the rebalance operation (as part of the Adding Nodes procedure), and then run a fresh repair operation on the node for which repair was interrupted.
- It's inconsequential to allow the system to stop an in-progress cleanup operation on a node, because you will need to run a fresh cleanup operation on each existing node anyway, at the end of the Adding Nodes procedure.

Note When you initiate the **Add Node** operation in the CMC, the system will also automatically disable the auto-repair feature and the proactive repair feature -- so that no new repairs kick off while you're expanding your cluster -- and then after you've completed the rebalance operation the system automatically re-enables auto-repair and proactive repair.

- In the **Data Centers** page, make sure that all the existing nodes and services are up and running in the service region.

More detail

The screenshot shows the Cloudian Control Manager (CMC) interface. The top navigation bar includes links for Analytics, Buckets & Objects, Users & Groups, Cluster, Alerts, Admin, and Help. The Cluster link is highlighted with a red oval. Below the navigation is a sub-menu with Data Centers (highlighted with a red oval), Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status. The main content area displays two data centers, DC1 and DC2, each containing a rack (RAC1) with five nodes. A light green cube icon with a plus sign is visible in the bottom right corner of the interface.

The **Add Node** function will not let you add nodes if any existing nodes or services in the region are down.

6.2.3. Adding Nodes

1. In the CMC's **Data Centers** page, in the display for the data center and rack in which you want to add a node, click the light green cube icon that has a plus sign on it.

More detail

The screenshot shows the Cloudian Cluster Management interface. At the top, there's a navigation bar with links for Analytics, Buckets & Objects, Users & Groups, Cluster (which is highlighted with a red oval), Alerts, Admin, and Help. Below the navigation is a secondary menu with Data Centers, Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status. A sub-menu for DEMOREG1 is open, showing options like NEW REGION and NEW DC.

The main area displays two data centers: DC1 (RAC1) and DC2 (RAC1). Each data center has five green hexagonal icons representing nodes. A red oval highlights the sixth icon in DC1, which is light green and has a plus sign (+) on it, indicating it's the target for adding a new node. Below each data center, it says "5 node(s)". To the right, there's a large light green box with a plus sign and the text "NEW DC".

Below the data centers is a "SERVICE STATUS" section with a table:

HOST	ADMIN	CASSANDRA	HYPERSTORE	REDIS MON	REDIS CRED	REDIS QOS	S3
hyperstore-demo1	✓	✓	✓		✓ *	✓	✓
hyperstore-demo3	✓	✓	✓		✓	✓ *	✓
hyperstore-demo1b	✓	✓	✓	✓	✓ *		✓
hyperstore-demo3b	✓	✓	✓				✓
hyperstore-demo5b	✓	✓	✓				✓
hyperstore-demo2	✓	✓	✓				✓
hyperstore-demo4	✓	✓	✓				✓
hyperstore-demo6	✓	✓	✓			✓	✓
hyperstore-demo2b	✓	✓	✓	✓	✓		✓
hyperstore-demo4b	✓	✓	✓				✓

Clicking the light green cube opens the **Add Node** interface.

Note If the data center has multiple racks for HyperStore -- which is possible only if your original HyperStore version installed was earlier than version 7.2 -- and if you want the new node(s) to be assigned to a new rack rather than one of the existing racks, just click on the light green cube icon for any rack in the display for the correct data center. If (and only if) the data center has multiple racks for your existing HyperStore nodes, you will have an opportunity to specify a new rack name in the next step below.

2. In the **Add Node** interface that displays, complete the fields for the new node.

More detail

Add Node [?](#)

Hostname	Region Name demoreg1
IP Address	Data Center Name DC1
Internal Network Interface Name (optional)	Rack Name RAC1
Installation User's Password	
<input type="checkbox"/> Private Key Authentication	

Description: Add a new node to the cluster. If you are adding multiple nodes, add them one at a time here, waiting for the Add Node operation to successfully complete for each node before you add the next node. For complete instructions on adding new nodes -- including steps to take before and after adding nodes -- please see the online Help.

[Cancel](#) [Execute](#)

Hostname (required)

Host name of the new node.

IP Address (required)

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

Internal Network Interface Name (optional)

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

Rack Name (required)

The behavior of this field depends on your particular HyperStore system and existing set-up:

- If your original HyperStore system is version 7.2 or later, or if your original HyperStore system was earlier than 7.2 and you have only been using one rack name for your existing nodes in the data center, this field value is fixed to the rack name that you have been using for your existing nodes. You cannot edit this field.
- If your original HyperStore system was earlier than 7.2 and you have been using multiple rack names for your existing nodes in the data center, you can select any of those rack names from the drop-down list or create a new rack name.

Installation User's Password (use this or "Private Key Authentication", not both)

For the install script (which is invoked by the **Add Node** operation) to securely connect to the new node by using the root user's password, enter the password in this field. Only the root user can be the HyperStore installation user.

Private Key Authentication (use this or "Installation User's Password", not both)

As an alternative to having the install script use a password to access the new node, you can select the "Private Key Authentication" checkbox. In this case the install script will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:

- If during installation of the cluster you let the install script generate an SSH key pair for you, then distribution of the public key to the new node will be taken care of automatically by the install script.
 - If during installation of the cluster you used your own existing key pair and you copied both the private and the public key into the installation staging directory on the Puppet master, then distribution of the public key to the new node will be taken care of automatically by the install script.
 - If during installation of the cluster you used your own existing key pair and you copied only the private key into the installation directory, while doing the distribution of the public key to the target nodes manually, then you must also copy the public key to the new node manually, before executing the **Add Node** operation.
3. Click **Execute**. This initiates a background operation that will take anywhere from a half-hour up to a full day to complete depending on your environment. When it completes successfully an orange node icon with a gear inside of it displays in the **Data Centers** page.

More detail

The **Add Node** operation entails verifying that the new host meets HyperStore requirements, installing software, updating system configuration, starting services, joining the new node into the cluster, and streaming system and object metadata to the new node (in Cassandra).

There are two CMC locations where you can monitor the **Add Node** operation progress:

- The **Data Centers** page. A new node icon appears, with color-coding to indicate the status of the **Add Node** operation. You can hold your cursor over the node icon for a text description of the status.

Note: The new node status icon will not display until the system successfully completes preliminary actions such as verifying that the new node meets HyperStore requirements and adding the node to the Cassandra ring. Prior to the appearance of the grey new node icon, if the **Add Node** operation encounters errors the system will automatically roll back the operation. Once the grey new node icon appears, the node has been added to the Cassandra ring and the operation can no longer be rolled back.



Grey node with gear -- The node has been added to the Cassandra ring, and Cassandra repair (which streams metadata to the new node) is in progress.



Orange node with gear -- Cassandra repair has completed successfully , and the node requires that you run a "rebalance" operation to stream object data to it (as described later in this procedure).



Red node with gear -- Cassandra repair has failed. If this status occurs, first check the **Data Center** page's **Service Status** section to make sure that Cassandra is up and running on all nodes (if it's down on any node, go to the **Node Status** page for that node and start Cassandra). After making sure Cassandra is up on all nodes go to the **Node Advanced** page, and from the Maintenance command type menu run `repaircassandra` on the new node.

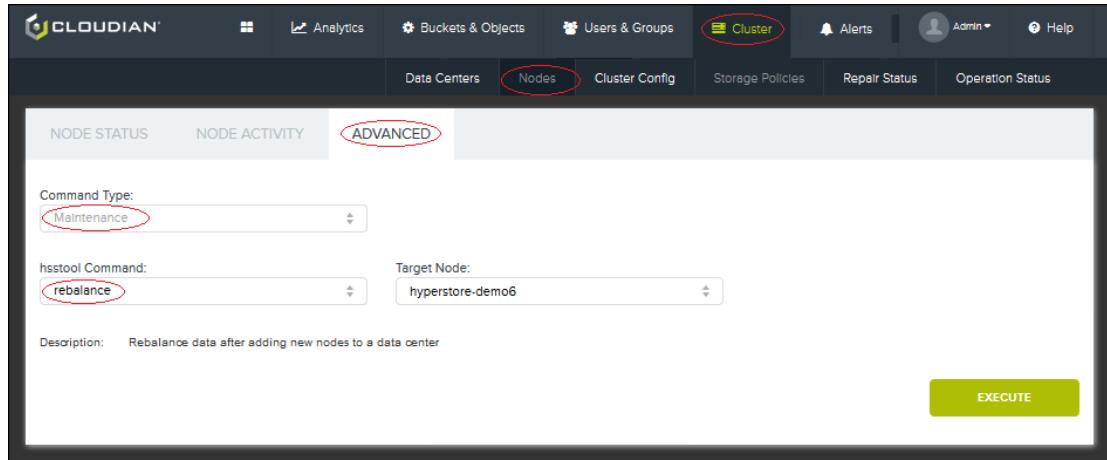
Note Do not try to add any more new nodes until the **Data Centers** page shows the orange icon for the currently added node.

- The **Operation Status** page. Click **View** to the right of the summary status line to display detailed progress information as the operation proceeds through pre-installation checks and initial set-up of the node.

The screenshot shows the Cloudian Operations Status page. The top navigation bar includes links for Analytics, Buckets & Objects, Users & Groups, Cluster (highlighted with a red circle), Alerts, Admin, and Help. Below the navigation is a sub-menu with Data Centers, Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status (also highlighted with a red circle). The main content area is titled 'OPERATION LIST' and displays a table with one row of data. The columns are: OPERATION NAME, TARGET, STATUS, PROGRESS, START TIME, and LAST UPDATE. The data row shows 'addNode' under 'OPERATION NAME', 'hyperstore-demo6' under 'TARGET', 'in progress' under 'STATUS', a progress bar at 20% under 'PROGRESS', 'Apr-30-2018 07:25' under 'START TIME', and 'Apr-30-2018 07:25' under 'LAST UPDATE'. There are 'View' and 'Delete' buttons to the right of the last column. At the bottom of the table, it says 'Showing 1 to 1 of 1 entries'.

- When the **Data Centers** page shows the orange icon for the currently added node, you can add more nodes (if you wish) by repeating Steps 1, 2, and 3 above for each new node. If you have a multi-DC cluster, be careful to click the add node icon within the data center panel for your desired DC. You can add different nodes to different DCs if you wish, as long as they're in the same service region.
- After the **Add Node** operation has completed successfully for each new node, **update your DNS and/or load balancer** configurations to include the new node(s), so that the new node(s) can participate in servicing user request traffic (if you have not already done so). For more information see "DNS Set-Up" and "Load Balancing" in the *HyperStore Installation Guide*.
- In the **Node Advanced** page, from the Maintenance command type group, **execute `hsstool rebalance` on each new node**. Rebalance is a long-running background operation that you can run concurrently on multiple new nodes that you've added. When all new nodes have completed rebalancing, for each node a green, check-marked cube icon will display in the **Data Centers** page.

More detail



The rebalance operation populates the new node(s) with their appropriate share of S3 object data. The rebalance is a background operation that may take up to several days or more to complete, depending on factors such as data volume and network bandwidth.

Note During in-progress rebalance operations the affected data remains readable by S3 client applications. Meanwhile the new node(s) are immediately available to support writes of new data, even before any rebalancing occurs.

Note When you have rebalance running on a new node or multiple new nodes, you cannot add any additional nodes to the service region (using the CMC's **Add Node** feature) until rebalance has completed successfully on all of the current new nodes.

Use the **Data Centers** page to periodically **check the progress** of the rebalance operation on each of the new nodes. The icon for the new node(s) will be color-coded, and you can hold your cursor over the icon(s) for a text description of the status.



Grey node with gear -- The rebalance operation is in progress. If you have added multiple nodes and have started rebalance operations on the nodes, each node's status icon will remain in this status until rebalance completes on all of the nodes in the batch.

Red hexagonal icon with a gear symbol -- The rebalance operation has failed for one or more token ranges. If this status displays, go to the **Nodes Advanced** page and run rebalance on the node again, using the "retry" option this time (select the retry checkbox when you run rebalance on the node). This will try the rebalance again, just for the failed token range(s).



Note This status displays -- and a rebalance retry is required -- only if rebalance has failed for one or more entire token ranges. If instead there has only been a failure to stream certain individual objects to the new node, these object stream jobs are placed in queue for automatic processing by the hourly proactive repair run.



Green node with check mark -- The rebalance operation has completed successfully. If you have added multiple nodes, this status will not display for any of the new nodes until rebalance has completed on all of the new nodes.

Another source of status information for the rebalance operation is the **Operation Status** page.

OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE
rebalance	hyperstore-demo6	in progress	<div style="width: 20%;">20%</div>	Apr-30-2018 08:27	Apr-30-2018 08:39
addNode	hyperstore-demo6	completed	<div style="width: 100%;">100%</div>	Apr-30-2018 07:25	Apr-30-2018 07:39

For status detail click **View** to the right of the summary status line.

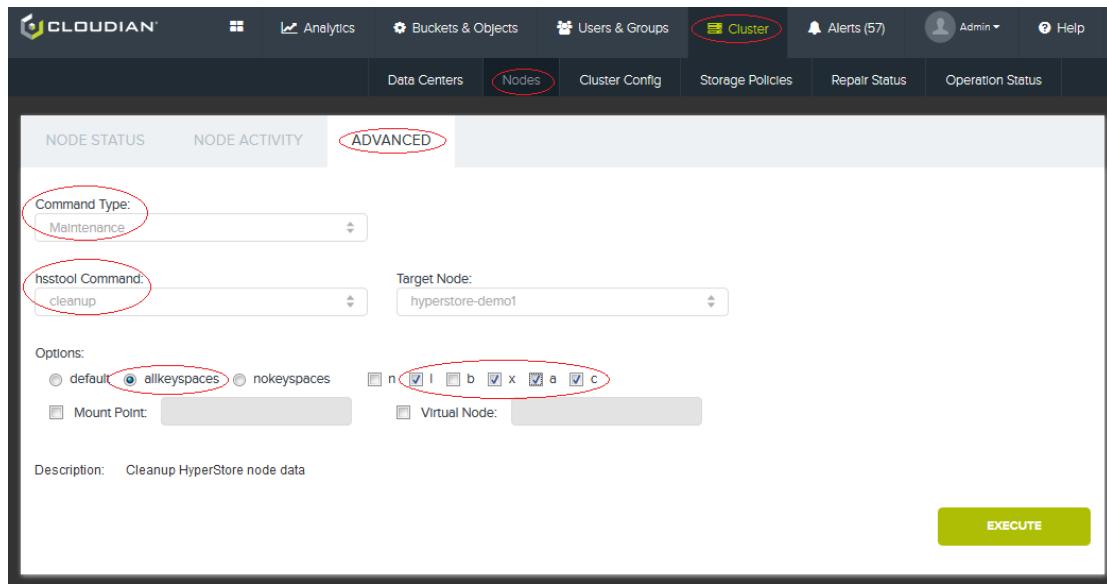
- If you removed a dead node prior to performing the Adding Nodes procedure and deferred the node repair operations that are necessary after you remove a dead node, perform those repairs now. (Skip ahead to Step 8 if this doesn't apply to you.)

More detail

- From the **Node Advanced** page, run [**hsstool repair**](#) on each node in the service region except for the new node(s), just one node at a time. When repairing each node, use the **allkeyspaces** option and also the **-pr** option. Leave the **-l** and **-m** options selected, as they are by default. Use the **Operation Status** page to track the progress of each repair. After repair of a node is complete, repair another node -- until all nodes except for the new node(s) have been successfully repaired.
 - If you have erasure coded object data in your system, from the **Node Advanced** page run [**hsstool repairec**](#) on one node in each HyperStore data center in the region. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region. Use the **Operation Status** page to track repair progress. After the repair(s) complete proceed to Step 8.
- After rebalance has completed successfully** on all new nodes, in the **Node Advanced** page from the Maintenance command group **execute hsstool cleanup on all nodes in the region except the new ones**. Use the **allkeyspaces**, **-l**, **-x**, **-a**, and **-c** options.

Cleanup is a long-running background operation. To minimize performance impact to the cluster it's best to run cleanup on just one node at a time per data center, waiting until it completes on one node before starting cleanup on another node. If you prefer to clean more than one node at a time within a data center, you can try running cleanup on two or three nodes concurrently, but pay close attention to cluster performance and contact Cloudian Support for additional guidance.

More detail



These cleanup operations remove from the target nodes data and metadata that they are no longer responsible for, thereby freeing up storage capacity on those nodes. They also serve to prevent resurrection of user-deleted objects and metadata inconsistency during any future add/remove node operations.

IMPORTANT: Do not run the *cleanup* operation on your previously existing nodes until after the *rebalance* operation has successfully completed for all of your new nodes. This is to ensure that you do not clean (delete) from the existing nodes data that had failed to be copied to the new node(s) during the *rebalance* operation.

This completes the procedure for adding nodes to your cluster.

6.3. Adding a Data Center

This procedure is for **adding a [new data center](#)** to an existing HyperStore service region. During this procedure you will:

- Prep the nodes in the new data center
- Verify that your existing cluster is in a proper condition to successfully add a data center
- Add the new data center's nodes to the cluster
- Take initial steps to start utilizing the new data center

IMPORTANT: Once you add nodes to your cluster, HyperStore does not support adding disks to those nodes. Make sure that the nodes that you are adding have sufficient disk capacity to meet your needs.

6.3.1. Special Requirements if an Existing Node is Down or Unreachable

The CMC's **Add DC** function **will not work if an existing node in your cluster is down or unreachable**. If you have more nodes in your cluster than are required by your configured storage policies and one of those nodes

is permanently down, follow the procedure for "**Removing a Node**" (page 390) to remove the dead node from the cluster. After you complete that procedure you can then follow this procedure for Adding a Data Center. If your current cluster has **only the minimum number of nodes required by your storage policies and one of those nodes is dead**, contact Cloudian Support for guidance.

6.3.2. Preparing to Add a Data Center

Before you add a new data center to a region, prepare by taking the actions below. (If the new nodes are HyperStore Appliance machines you can skip Steps 2 and 5.)

1. The HyperStore nodes in each data center will need to be able to communicate with the HyperStore nodes in the other data center(s). This includes HyperStore services that listen on the internal interface. Therefore, if you haven't already done so you must **configure your inter-DC networking so that the DCs' internal networks are connected to each other** (for example, by using a VPN).
2. Make sure the new host(s) **meet requirements** for:
 - Hardware specs and operating system: See "Host Hardware and OS Requirements" in the *HyperStore installation Guide*
 - Open listening ports: See "HyperStore Listening Ports" in the Reference section of the *HyperStore Installation Guide*
3. Make sure you have the **information you will need** to complete this procedure: the new data center's name, and each new node's hostname, IPv4 address, internal interface name (optional), rack name, and root password. For DC and rack names only alphanumeric characters and dashes are supported.
4. **Start the new host(s)**, if not already running.
5. From your Puppet master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

More detail

- a. In your existing cluster, on your Puppet master node launch the *system_setup.sh* tool. If your Puppet master is a HyperStore Appliance the tool will be under */root/CloudianTools*, or if it's a software-only HyperStore node the will be under your installation staging directory.

```
./system_setup.sh
```

- b. In the tool's main menu select "**9**" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.
- c. Use the node preparation task menu to prepare the node:
 - Complete the configuration of network interfaces for the node, if you haven't already.
 - Set the timezone for the node.
 - Install and configure HyperStore prerequisites on the node.
 - Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
 - After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.
- d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the *system_setup.sh* tool.

IMPORTANT: If the new node(s) are not HyperStore Appliances and if you do not use `system_setup.sh` to format the data disks on the new node(s), then in the installation staging directory on your Puppet master node you must for each new node create a text file named `<hostname>_fslist.txt` that specifies the new node's data mount points, in this format:

```
<devicename> <mountpoint>
<devicename> <mountpoint>
etc...
```

- In the CMC's **Data Centers** page, make sure that all the existing nodes and services are up and running in the region in which you are adding a data center. The **Add DC** function will not let you add nodes if any existing nodes or services in the region are down.

HOST	ADMIN	CASSANDRA	HYPERSTORE	REDIS MON	REDIS CRED	REDIS QOS	S3
hyperstore-demo1	✓	✓	✓		✓	✓	✓
hyperstore-demo3	✓	✓	✓		✓	✓	✓
hyperstore-demo1b	✓	✓	✓	✓			✓
hyperstore-demo3b	✓	✓	✓				✓
hyperstore-demo5b	✓	✓	✓				✓
hyperstore-demo2	✓	✓	✓				✓
hyperstore-demo4	✓	✓	✓				✓
hyperstore-demo6	✓	✓	✓				✓
hyperstore-demo2b	✓	✓	✓	✓	✓		✓
hyperstore-demo4b	✓	✓	✓				✓

6.3.3. Adding a Data Center

- In the CMC's **Data Centers** page, with the correct service region tab selected, click the large box that says **+NEW DC**.

The screenshot shows the Cloudian Cluster Management interface. At the top, there are navigation tabs: Analytics, Buckets & Objects, Users & Groups, Cluster (circled in red), Alerts, Admin, and Help. Below the tabs, there are sub-tabs: Data Centers (circled in red), Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status. The main area displays two data centers: DC1 (RAC1) and DC2 (RAC1). Each data center has 5 nodes, all of which are green (healthy). A large red circle highlights a button labeled '+ NEW DC' in the bottom right corner. Below the data centers, there is a 'SERVICE STATUS' section with a table showing the status of various hosts across different services.

HOST	ADMIN	CASSANDRA	HYPERSTORE	REDIS MON	REDIS CRED	REDIS QOS	S3
hyperstore-demo1	✓	✓	✓		✓	✓	✓
hyperstore-demo3	✓	✓	✓		✓	✓	✓
hyperstore-demo1b	✓	✓	✓	✓			✓
hyperstore-demo3b	✓	✓	✓				✓
hyperstore-demo5b	✓	✓	✓				✓
hyperstore-demo2	✓	✓	✓				✓
hyperstore-demo4	✓	✓	✓				✓
hyperstore-demo6	✓	✓	✓		✓	✓	✓
hyperstore-demo2b	✓	✓	✓	✓	✓		✓
hyperstore-demo4b	✓	✓	✓				✓

2. In the **Add DC** interface that displays, complete the top two fields for the new data center as a whole, then complete the remaining fields for each node in the data center, clicking **Add More Nodes** to display fields for additional nodes as needed.

More detail

Add DC [?](#)

System Metadata Replication Factor	Data Center Name
Hostname	Region Name demoreg1
IP Address	Data Center Name
Internal Network Interface Name (optional)	Rack Name RAC1
Installation User's Password	
<input type="checkbox"/> Private Key Authentication	

[**✚ ADD MORE NODES**](#)

Description: Add a new data center to an existing service region.

[Cancel](#) [Execute](#)

System Metadata Replication Factor (required)

In this field indicate how many replicas of system metadata (such as usage reporting data, user account information, and system monitoring data) you want to store in the new DC. For each metadata item, each replica will be stored on a different node, to protect this metadata against loss or corruption. It's recommended that you set this replication factor to 3 if you are going to have three or more nodes in the new DC. If there will be only two nodes in the new DC, set this to 2; if there will be only one node, set this to 1. You cannot set a metadata replication factor higher than the number of nodes in the new DC.

Data Center Name (required)

Name of the new data center. Maximum 256 characters. Only ASCII alphanumerical characters and dashes are allowed.

Hostname (required)

Host name of the new node.

IP Address (required)

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

Internal Network Interface Name (optional)

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

Rack Name (fixed value "RAC1")

The "Rack Name" value is automatically fixed to "RAC1" for all nodes in the new data center. You

cannot edit this value.

Note This is an internal value used by HyperStore. It does not need to correspond to any actual rack name in your data center.

Installation User's Password (use this or "Private Key Authentication", not both)

For the install script (which is invoked by the **Add DC** operation) to securely connect to the new node by using the root user's password, enter the password in this field. Only the root user can be the HyperStore installation user.

Private Key Authentication (use this or "Installation User's Password", not both)

As an alternative to having the install script use a password to access the new node, you can select the "Private Key Authentication" checkbox. In this case the install script will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:

- If during installation of the cluster you let the install script generate an SSH key pair for you, then distribution of the public key to the new node will be taken care of automatically by the install script.
- If during installation of the cluster you used your own existing key pair and you copied both the private and the public key into the installation staging directory on the Puppet master, then distribution of the public key to the new node will be taken care of automatically by the install script.
- If during installation of the cluster you used your own existing key pair and you copied only the private key into the installation directory, while doing the distribution of the public key to the target nodes manually, then you must also copy the public key to the new node manually, before executing the **Add DC** operation.

3. Click **Execute**. This initiates a background operation that will take anywhere from several minutes to several hours to complete depending on your environment. When it completes successfully the **Data Centers** page will display an additional block representing the newly added DC, with a green, check-marked cube icon for each of the new DC's nodes.

More detail

The **Add DC** operation entails verifying that the new hosts meet HyperStore requirements, installing software, updating system configuration, starting services, joining the new nodes into the cluster, and streaming system metadata to the new nodes (in Cassandra).

You can use the **Operation Status** page to monitor progress of the operation.

OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE
addDC	DC3	in progress	[progress bar]	Apr-30-2018 11:25	Apr-30-2018 11:26

For status detail click **View** to the right of the summary status line.

When the operation is complete, to see the new nodes in the **Data Centers** page you may need to refresh the page in your browser.

If you hold your cursor over each cube in the new data center the node host names will display.

Note If the **Operation Status** page indicates that the **Add DC** operation has failed, click "View" for detail. Then for more information to support troubleshooting efforts, *grep* for "ERROR" level messages in the *cloudian-installation.log* file under the installation staging directory on your Puppet master node.

4. **Update your DNS and load balancing configurations** to include the new data center and its nodes, if you have not already done so. For more information see "DNS Set-Up" and "Load Balancing" in the *HyperStore Installation Guide*.
5. Go to the CMC's [Storage Policies](#) page and **create one or more storage policies that utilize the new DC**. Until you create storage policies that use the new DC and users subsequently create buckets that use those storage policies, no S3 object data will be stored in the new DC.

Note: Because your previously existing storage policies do not include the new DC, none of the data already stored in your system in association with those storage policies will be migrated into the new DC. Accordingly, there is no need for you to run a *rebalance* operation after adding a new DC.

This completes the procedure for adding a data center to your cluster.

NOTE: If you removed a dead node prior to performing the **Adding a Data Center procedure** and deferred the node repair operations that are necessary after you remove a dead node, perform those repairs now.

More detail

- a. From the **Node Advanced** page, run [**hsstool repair**](#) on each node in the service region except for the new nodes, just one node at a time. When repairing each node, use the **allkeyspaces** option and also the **-pr** option. Leave the **-l** and **-m** options selected, as they are by default. Use the **Operation Status** page to track the progress of each repair. After repair of a node is complete, repair another node -- until all nodes except for the nodes in the new data center have been successfully repaired.
- b. If you have erasure coded object data in your system, from the **Node Advanced** page run [**hsstool repairec**](#) on one node in each HyperStore data center in the region except for the new data center. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region, except for the new DC. Use the **Operation Status** page to track repair progress.

6.4. Adding a Region

This procedure is for **adding a new service region** to your HyperStore system. During this procedure you will:

- Prep the nodes in the new service region
- Add the new region's nodes to the system
- Take initial steps to start utilizing the new region

IMPORTANT: Once you add nodes to your system, HyperStore does not support adding disks to those nodes. Make sure that the nodes that you are adding have sufficient disk capacity to meet your needs.

6.4.1. Preparing to Add a Region

Before you add a new region to your system, prepare by taking the actions below. (If the new nodes are HyperStore Appliance machines you can skip Steps 2 and 5.)

1. The HyperStore nodes in each region will need to be able to communicate with the HyperStore nodes in the other region(s). This includes HyperStore services that listen on the internal interface. Therefore, if you haven't already done so you must **configure your networking so that the internal networks of all of your data centers in all of your regions are connected to each other** (for example, by using a VPN).
2. Make sure the new host(s) **meet requirements** for:
 - Hardware specs and operating system: See "Host Hardware and OS Requirements" in the HyperStore installation Guide
 - Open listening ports: See "HyperStore Listening Ports" in the Reference section of the *HyperStore Installation Guide*
3. Make sure you have the **information you will need** to complete this procedure: the name of the new region, the name(s) of the data center(s) that will comprise the region, and each new node's hostname, IPv4 address, internal interface name (optional), rack name, and root password. For region, DC, and rack names only alphanumeric characters and dashes are supported.
4. **Start the new host(s)**, if not already running.
5. From your Puppet master node use the *system_setup.sh* tool's **Prep New Node to Add to Cluster** function to complete network interface configuration, time zone set-up, prerequisites installation, and data disk formatting for each new node.

More detail

- a. In your existing cluster, on your Puppet master node launch the *system_setup.sh* tool. If your Puppet master is a HyperStore Appliance the tool will be under */root/CloudianTools*, or if it's a software-only HyperStore node the will be under your installation staging directory.

```
./system_setup.sh
```

- b. In the tool's main menu select "**9**" for **Prep New Node to Add to Cluster**. When prompted provide the IP address of a new node, and then the password for logging into the node. A menu of node preparation tasks will then display.
- c. Use the node preparation task menu to prepare the node:
 - Complete the configuration of network interfaces for the node, if you haven't already.
 - Set the timezone for the node.
 - Install and configure HyperStore prerequisites on the node.
 - Set up data disks on the node with *ext4* file systems, if you haven't already. Make sure to format and mount **all** available data disks on the node.
 - After completing the setup tasks for the node choose the "Return to Master Node" option, which returns you to the tool's main menu.

- d. Repeat steps "b" and "c" above for each new node that you're adding. When you're done, exit the `system_setup.sh` tool.

IMPORTANT: If the new node(s) are not HyperStore Appliances and if you do not use `system_setup.sh` to format the data disks on the new node(s), then in the installation staging directory on your Puppet master node you must for each new node create a text file named `<hostname>_fslist.txt` that specifies the new node's data mount points, in this format:

```
<devicename> <mountpoint>
<devicename> <mountpoint>
etc...
```

6.4.2. Adding a Region

- In the CMC's Data Centers page, click the tab that says **+NEW REGION**.

The screenshot shows the Cloudian CMC interface. At the top, there are tabs for Analytics, Buckets & Objects, Users & Groups, Cluster (circled in red), Alerts, Admin, and Help. Below the tabs, there are sub-tabs: Data Centers (circled in red), Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status. The main content area displays two data centers: DC1 (RAC1) and DC2 (RAC1). Each data center has five green hexagonal icons representing nodes. A red circle highlights the '+ NEW REGION' button at the top left of the main content area. To the right, a dashed green box indicates where a new data center will be added, with a green plus sign and the text '+ NEW DC'. At the bottom, there is a 'SERVICE STATUS' table with columns for HOST, ADMIN, CASSANDRA, HYPERSTORE, REDIS MON, REDIS CRED, REDIS QOS, and S3. The table lists ten hosts (hyperstore-demo1 to hyperstore-demo4b) with various service status indicators.

HOST	ADMIN	CASSANDRA	HYPERSTORE	REDIS MON	REDIS CRED	REDIS QOS	S3
hyperstore-demo1	✓	✓	✓		✓ •	✓	✓
hyperstore-demo3	✓	✓	✓		✓	✓ •	✓
hyperstore-demo1b	✓	✓	✓	✓	✓		✓
hyperstore-demo3b	✓	✓	✓				✓
hyperstore-demo5b	✓	✓	✓				✓
hyperstore-demo2	✓	✓	✓				✓
hyperstore-demo4	✓	✓	✓				✓
hyperstore-demo6	✓	✓	✓			✓	✓
hyperstore-demo2b	✓	✓	✓	✓	✓		✓
hyperstore-demo4b	✓	✓	✓				✓

- In the **Add Region** interface that displays, enter the name of the new region then complete the fields for each node in the data center, clicking **Add More Nodes** to display fields for additional nodes as needed.

[More detail](#)

Add Region

Region Name

Hostname Region Name

IP Address Data Center Name

Internal Network Interface Name (optional) Rack Name

Installation User's Password

Private Key Authentication

ADD MORE NODES

Description: Add a new service region to an existing system.

Cancel **Execute**

Region Name (required)

Name of the new region. Maximum 52 characters. Only ASCII alphanumerical characters and dashes are allowed. Letters must be lower case.

Hostname (required)

Host name of the new node.

IP Address (required)

Service network IP (v4) address that the hostname resolves to. Do not use IPv6.

Data Center Name (required)

Name of the data center in which the new node is located. Maximum 256 characters. Only ASCII alphanumerical characters and dashes are allowed.

Internal Network Interface Name (optional)

If the new node will use a different dedicated interface for internal cluster traffic than other nodes in your cluster use — for example if the new node uses "eth2" for internal traffic while other nodes in your cluster are using "eth1" for internal traffic — enter the interface name in this field. If the new node will use the same internal network interface as your existing nodes you can leave this field empty.

Rack Name (fixed value "RAC1")

The "Rack Name" value is automatically fixed to "RAC1" for all nodes in the new region. You cannot edit this value.

Note This is an internal value used by HyperStore. It does not need to correspond to any actual rack name in your data center(s).

Installation User's Password (use this or "Private Key Authentication", not both)

For the install script (which is invoked by the **Add Region** operation) to securely connect to the new node by using the root user's password, enter the password in this field. Only the root user can be the HyperStore installation user.

Private Key Authentication (use this or "Installation User's Password", not both)

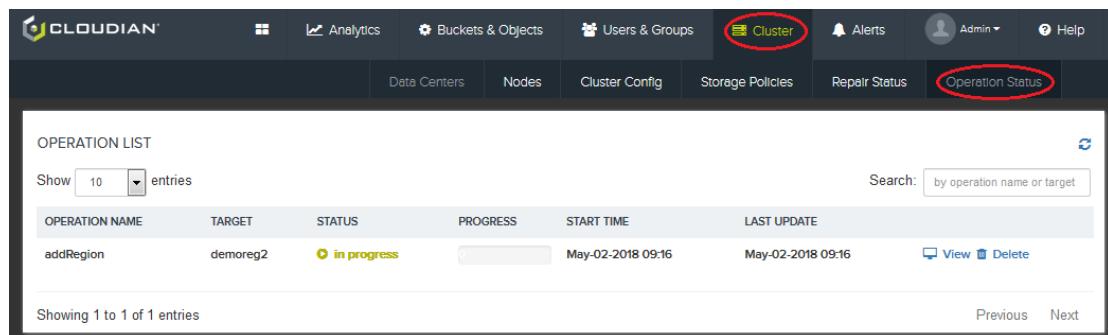
As an alternative to having the install script use a password to access the new node, you can select the "Private Key Authentication" checkbox. In this case the install script will use the same private key as was used to install the existing cluster. Distribution of the corresponding public key to the new node depends on how you handled SSH key set-up during installation of the existing HyperStore cluster:

- If during installation of the cluster you let the install script generate an SSH key pair for you, then distribution of the public key to the new node will be taken care of automatically by the install script.
 - If during installation of the cluster you used your own existing key pair and you copied both the private and the public key into the installation staging directory on the Puppet master, then distribution of the public key to the new node will be taken care of automatically by the install script.
 - If during installation of the cluster you used your own existing key pair and you copied only the private key into the installation directory, while doing the distribution of the public key to the target nodes manually, then you must also copy the public key to the new node manually, before executing the **Add Region** operation.
3. Click **Execute**. This initiates a background operation that will take anywhere from several minutes up to an hour to complete depending on your environment. When it completes successfully the **Data Centers** page will display an additional tab representing the newly added region, with a green, check-marked cube icon for each of the new region's nodes.

More detail

The **Add Region** operation entails verifying that the new hosts meet HyperStore requirements, installing software, updating system configuration, starting services, and joining the new nodes into a new Cassandra ring.

Use the **Operation Status** page to monitor progress of the operation.



The screenshot shows the Cloudian Operations List interface. At the top, there is a navigation bar with links for Analytics, Buckets & Objects, Users & Groups, Cluster (circled in red), Alerts, Admin, and Help. Below the navigation bar, there is a sub-navigation bar with tabs for Data Centers, Nodes, Cluster Config, Storage Policies, Repair Status, and Operation Status (also circled in red). The main content area is titled 'OPERATION LIST'. It includes a search bar with the placeholder 'Search: by operation name or target'. Below the search bar, there is a table with columns: OPERATION NAME, TARGET, STATUS, PROGRESS, START TIME, and LAST UPDATE. A single row is visible in the table, showing 'addRegion' as the operation name, 'demoreg2' as the target, 'in progress' as the status, a progress bar indicating completion, 'May-02-2018 09:16' as the start time, and 'May-02-2018 09:16' as the last update. At the bottom of the table, it says 'Showing 1 to 1 of 1 entries'. There are also 'View' and 'Delete' buttons for the operation.

For status detail click **View** to the right of the summary status line. (The page may at one point indicate that it cannot retrieve status information -- this is due to an S3 Server / Admin Server restart which is an automatic part of the **Add Region** operation. Wait a few minutes then refresh the page.)

When the operation is complete, to see the new nodes in the **Data Centers** page you may need to refresh the page in your browser.

If you hold your cursor over each cube in the new region the node host names will display.

Note If the **Operation Status** page indicates that the **Add Region** operation has failed, click "View" for detail. Then for more information to support troubleshooting efforts, *grep* for "ERROR" level messages in the *cloudian-installation.log* file under the installation staging directory on your Puppet master node.

4. **Update your DNS and load balancing** configurations to include the new service region and its nodes, if you have not already done so. Note that name server configurations in each of your existing region(s) and the new region must have entries for **all** your regions' S3 service endpoints, as well as for the global Admin service and CMC service endpoints. For more information see "DNS Set-Up" and "Load Balancing" in the *HyperStore Installation Guide*.
5. Go to the **Storage Policies** page and select the new region. Then **create a storage policy for the new region**. This first storage policy will become the default storage policy for the region. Later if you've created more than one storage policy in the region you can change which policy is the default policy if you wish. Until you create one or more storage policies in the new region and users subsequently create buckets that use those storage policies, no S3 object data will be stored in the new region.
6. Optionally, set Quality of Service (QoS) limits for users' and groups' activity in the new service region. Each service region has its own QoS configuration. By default, in each region no QoS limits are enforced. For more information see "**Set Quality of Service (QoS) Controls**" (page 250). When using the QoS configuration interfaces be sure to select the new service region from the interfaces' drop-down list of regions.

This completes the procedure for adding a service region to your system.

6.5. Removing a Node

This procedure is for **permanently removing a node from your cluster** so that the data storage responsibilities of that node are redistributed to the remaining nodes in the cluster. During this procedure you will:

- Verify that your existing system is in a proper condition to successfully support the removal of a node
- Remove the node
- If the node you removed was "dead" -- Cassandra down or unreachable before removing the node -- repair the remaining nodes in your cluster (this is not necessary if the node you removed was live)

Note If you are removing a node after having added a new node to your cluster, you must complete the rebalance operation for the new node before removing a node. For more information on rebalance see "**Adding Nodes**" (page 369).

IMPORTANT: Removing a node should be something that you do **only if absolutely necessary**. When you remove a live node from your cluster, during the decommissioning process the data stored on that node is unavailable to client applications. Once data has been streamed from the decommissioning node to the other nodes that data is again available to support client requests. But depending on the data volume and network bandwidth, it may take up to several days or more until the decommissioning process has completed for all of the node's data. Until all of the decommissioning node's data has been streamed to other nodes, some objects will have fewer than your configured number of replicas or erasure coded fragments available within the live system.

6.5.1. Preparing to Remove a Node

Take these actions to prepare to remove a node from your cluster:

1. **Make sure that removing the node won't leave your cluster with fewer nodes than your configured storage policies require.**

More detail

For example if 4+2 erasure coding is being used in your system you cannot reduce your cluster size to fewer than 6 nodes, even temporarily. Or for another example if you have a storage policy that for each object places 3 replicas in DC1 and 3 replicas in DC2, do not reduce the number of nodes in either data center to fewer than 3.

If you're not certain what storage policies currently exist in your system, check the CMC's [Storage Policies](#) page.

The CMC's **Uninstall Node** function checks and enforces this requirement and will not let you remove a node if doing so would leave fewer than the required number of nodes in your cluster. If your cluster is currently at the minimum size required by your storage policies and you want to remove a node:

- If the node you want to remove is **live** you can first add a new node to your cluster by following the complete procedure for "[Adding Nodes](#)" (page 369) (including rebalancing); and then after that you will be able to remove the node that you want to remove.
 - If the node you want to remove is **dead** contact Cloudian Support for guidance.
2. **Make sure that you have sufficient available storage capacity on the other nodes in your cluster.**
- The data from the removed node will be redistributed to the remaining nodes that are encompassed by the same storage policies as the removed node.

More detail

Each remaining node must have available storage capacity to take on some of the data from the removed node. You can review your cluster and per-node storage space availability in the CMC's [Capacity Explorer](#) page.

Note If any of the other nodes are in the "[stop-write](#)" condition -- or if any disks on a node are in stop-write condition -- at a time when you decommission a different node from your system, the decommissioning process overrides the stop-write restriction on the node(s) or disk(s) where it exists in order to stream to **all** nodes and disks in the cluster their share of data from the decommissioned node. Consequently, disks that were nearly full before a decommissioning operation may become completely full during the decommission operation, resulting in stream

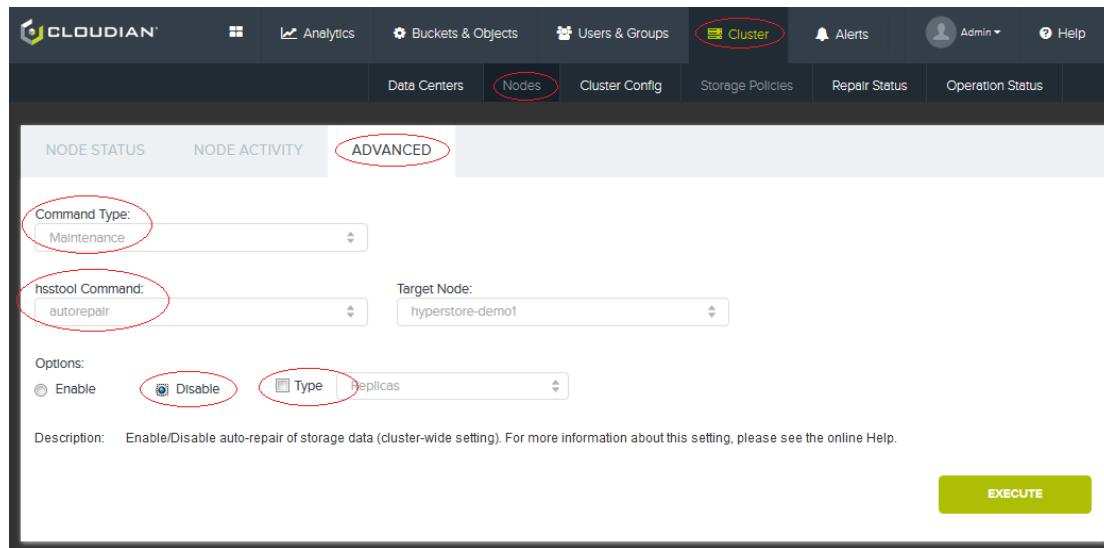
job failures once the disks can accept no more data.

To avoid this, before removing a node from your cluster make sure there is plenty of space on all the remaining nodes and disks to absorb the data from the node you intend to remove. **If you want to remove a node from your cluster at a time when some disks on the other nodes are nearly full, consult with Cloudian Support.**

3. In the **Node Advanced** page, from the Maintenance command type group, execute the *autorepair* command with the Disable option to **temporarily disable the automated repair feature** in the service region in which you are removing a node.

More detail

The target node for the command can be any node in the region. Leave the "Type" option unselected so that all automated repair types are disabled. This will prevent any new schedule-based auto-repairs or proactive repairs from launching during the node removal process.



4. In the **Operation Status** page, **make sure there are no rebalance operations or other operations currently in progress** in the service region.

More detail

If any operations are in progress, wait until the operations complete before removing a node from your cluster. The CMC's **Uninstall Node** function will not let you remove a node if a major operation such as rebalance, repair, or cleanup is running in the service region.

OPERATION NAME	TARGET	STATUS	PROGRESS	START TIME	LAST UPDATE
cleanup	hyperstore-demo1	✓ completed	100%	Apr-09-2018 16:18	Apr-09-2018 17:19
repair	hyperstore-demo2	✓ completed	100%	Apr-08-2018 05:05	Apr-08-2018 05:14

Note If you don't want to wait for an in-progress repair or cleanup of a node to complete you have the option of terminating the operation. To do so, go to the **Node Advanced** page and for that node execute [hsstool repair](#) or [hsstool repairec](#) or [hsstool cleanup](#) or [hsstool cleanupec](#) with the "stop" option selected. Note that an *hsstool rebalance* operation does not support a stop option and cannot be terminated while in progress.

5. In the **Repair Status** page, make sure there are no proactive repairs currently in progress in the service region.

More detail

If any proactive repairs are in progress, wait until they complete before removing a node from your cluster. The CMC's **Uninstall Node** function will not let you remove a node if a proactive repair is running in the service region.

Node Repair Status:	Rebalance Failed	Rebalance Required	Rebalance Running	Repair Running	Proactive Repair Pending
	✖	⚠	✖	⚡	🕒
	🕒	🕒	🕒	🕒	🕒

Note If you don't want to wait for an in-progress proactive repair of a node to complete you have the option of terminating the repair. To do so, go to the **Node Advanced** page and for that node execute [hsstool proactiverepair](#) with the "stop" option selected.

- If the node you want to remove is the active Puppet Master node, follow the instructions to "**Manually Fail Over the Puppet Master Role from the Primary to the Backup**" (page 416).

More detail

If you're not sure whether the node you want to remove is the Puppet Master host, check the CMC's **Cluster Information** page. That page also shows which host is the Puppet Master backup host.

CMC ADMIN SERVER HOST:	CASSANDRA CLUSTER NAME:
s3-admin.cloudianhyperstore.com:19443	Cloudiandemoreg1

S3 ENDPOINT (HTTP):	S3 ENDPOINT (HTTPS):
s3.cloudianhyperstore.com:80,66.109.98.221:80	s3.cloudianhyperstore.com:443,66.109.98.221:443

S3 WEBSITE ENDPOINT:	
s3-website.cloudianhyperstore.com	

REDIS CREDENTIALS MASTER HOST:	REDIS CREDENTIALS SLAVE HOST(S):
hyperstore-demo1	hyperstore-demo1,hyperstore-demo2b

REDIS QOS MASTER HOST:	REDIS QOS SLAVE HOST(S):
hyperstore-demo3	hyperstore-demo1,hyperstore-demo6

REDIS MONITOR PRIMARY HOST:	REDIS MONITOR BACKUP HOST:
hyperstore-demo1b	hyperstore-demo2b

SYSTEM MONITORING/CRONJOB PRIMARY HOST:	SYSTEM MONITORING/CRONJOB BACKUP HOST:
hyperstore-demo1b	hyperstore-demo3b

EXTERNAL NTP HOST(S):	INTERNAL NTP HOST:
0.centos.pool.ntp.org 1.centos.pool.ntp.org 2.centos.pool.ntp.org 3.centos.pool.ntp.org	DC2: [hyperstore-demo4, hyperstore-demo6, hyperstore-demo2b, hyperstore-demo4b] DC1: [hyperstore-demo1, hyperstore-demo3, hyperstore-demo1b, hyperstore-demo3b]

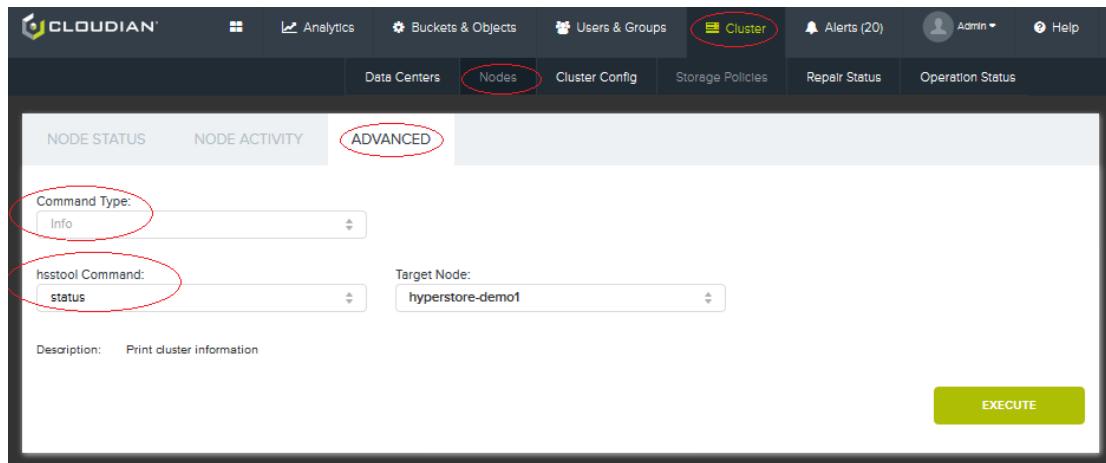
PUPPET MASTER HOST:	PUPPET MASTER BACKUP HOST:
hyperstore-demo1.cloudianhyperstore.com	hyperstore-demo3

Note Any other specialized services on the node -- such as Redis or Redis Monitor -- will be automatically moved to other nodes in the cluster by the CMC's **Uninstall Node** function.

- In the **Node Advanced** page, from the Info command type group, execute the `status` command on any healthy node to **verify that both the Cassandra Service and the HyperStore Service are up on all nodes** in the region.

More detail

The target node for the command can be any healthy node in the same service region as the node you want to remove -- the command returns status information for the cluster as a whole.



- In the command response, for the **node that you want to remove** check the status of the Cassandra Service (in the "Cassandra" column) and the HyperStore Service (in the "Hss" column).
 - If the Cassandra status and the HyperStore Service status are both Up, the data redistribution that will be required in the cluster when you remove the node will be executed by a decommissioning process that will be automatically invoked by the CMC's **Uninstall Node** function.
 - If the Cassandra status is Down or unreachable ("?"), data redistribution by decommissioning is not supported. Instead, at the end of the remove node procedure you will need to run repair on all the remaining nodes in order to redistribute data in the cluster. Details are in the procedure below.

Note If possible, start Cassandra on the node that you want to remove, so that the **Uninstall Node** function can automatically implement a decommissioning process and you won't have to perform repairs.

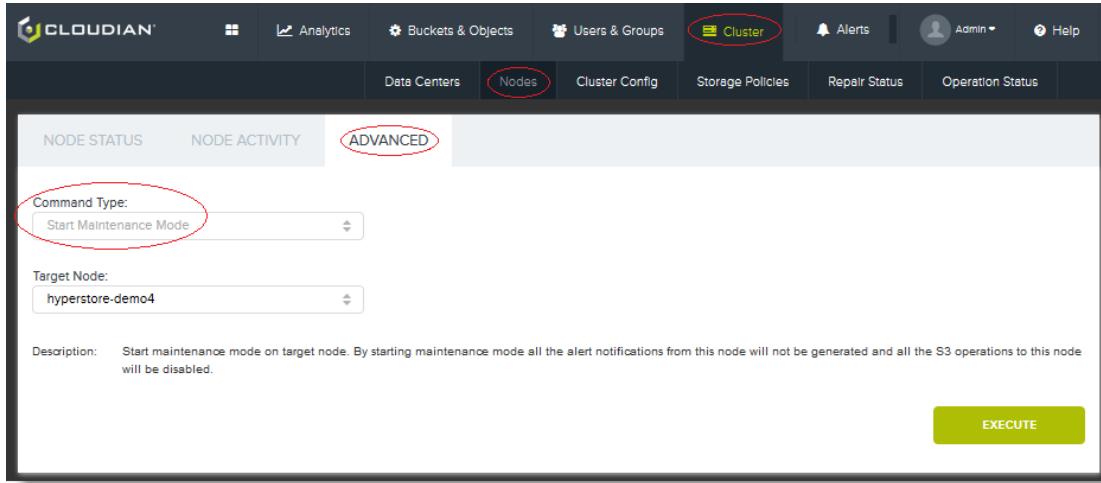
- If Cassandra is Up but HyperStore Service is Down or unreachable, the CMC's **Uninstall Node** function will abort if you try to run it. Before trying to remove the node, start the HyperStore Service on the node (or resolve the network access problem if there is one).
- For all the other nodes confirm that Cassandra and the HyperStore Service are Up. The CMC's **Uninstall Node** function will abort if Cassandra or the HyperStore Service are Down or unreachable on any other node in the cluster. If those services are down on any of the other nodes, start the services (or resolve the network access problem if there is one) before trying to remove a node.

6.5.2. Removing a Node

1. If you have not already done so, log in to a CMC instance on a node **other than the node that you are going to remove**, but in the same service region as the node that you are going to remove.

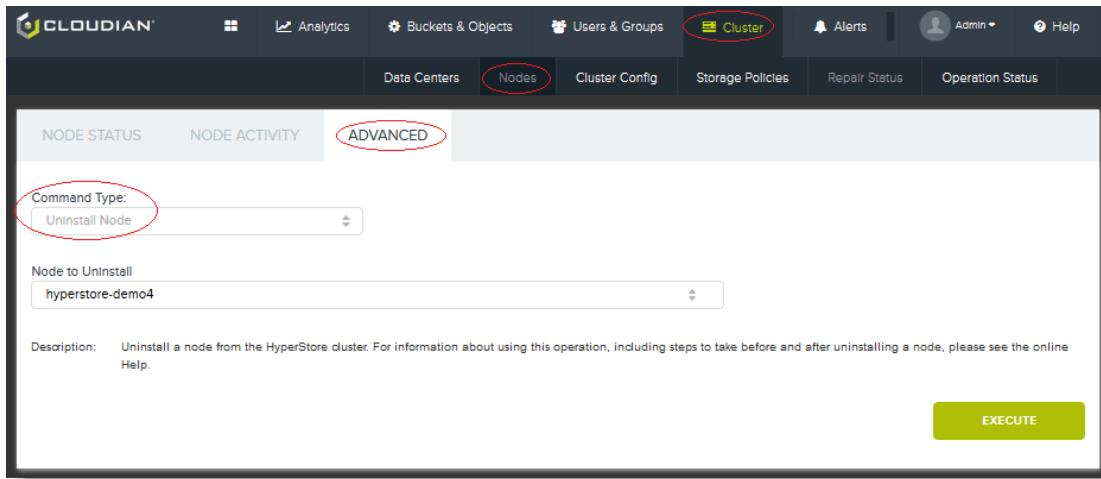
https://<IP_address_of_node_other_than_removal_node>.8443/Cloudian

2. In the CMC's **Node Advanced** page, from Command Type drop-down list select **Start Maintenance Mode**. For the target node select the node that you want to remove from the cluster.



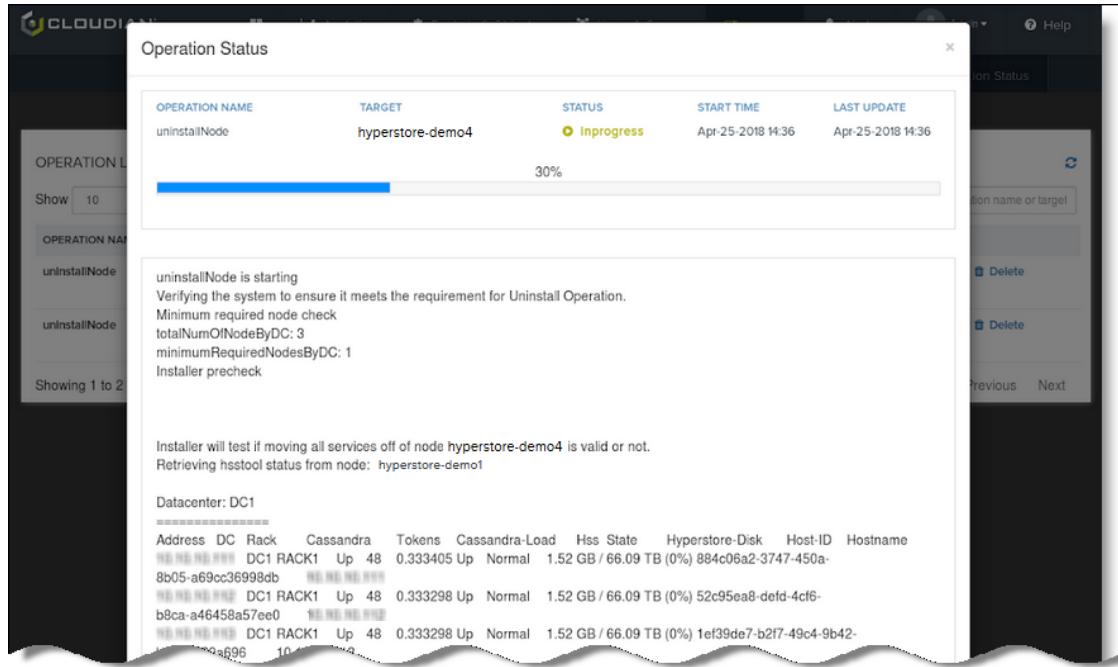
Then click **Execute**. This directs the rest of the cluster to stop sending S3 requests to the specified node.

3. In the **Node Advanced** page, from Command Type drop-down list select **Uninstall Node**. For the "Node to Uninstall" list select the node that you want to remove from the cluster.



Then click **Execute**. After you confirm that you want to proceed, the operation is initiated.

4. Use the **Operation Status** page to periodically check on the progress of the **Uninstall Node** operation. To pop up a detailed status report click **View** next to the summary status line.



If the node you are removing is live, the **Uninstall Node** operation will include decommissioning the node -- streaming copies of its data to the remaining nodes in the cluster -- and this may take up to several days or more to complete. If the node you are removing is dead, the **Uninstall Node** operation will be much briefer.

Note If you want to remove multiple nodes, wait until the **Uninstall Node** operation completes for one node before you start to uninstall the next node. The system does not support uninstalling multiple nodes concurrently.

5. After the **Operation Status** page shows that the status of the **Uninstall Node** operation is Completed, go to the [Node Status](#) page and confirm that the removed node no longer appears in the "Host" dropdown list.
6. **If the node you removed was "dead"** (meaning that the Cassandra Service on the node -- for which you checked status in **"Preparing to Remove a Node"** (page 391), Step 7 -- was down or unreachable at the time that you executed the Uninstall Node operation) you must repair each of the remaining nodes in the region.

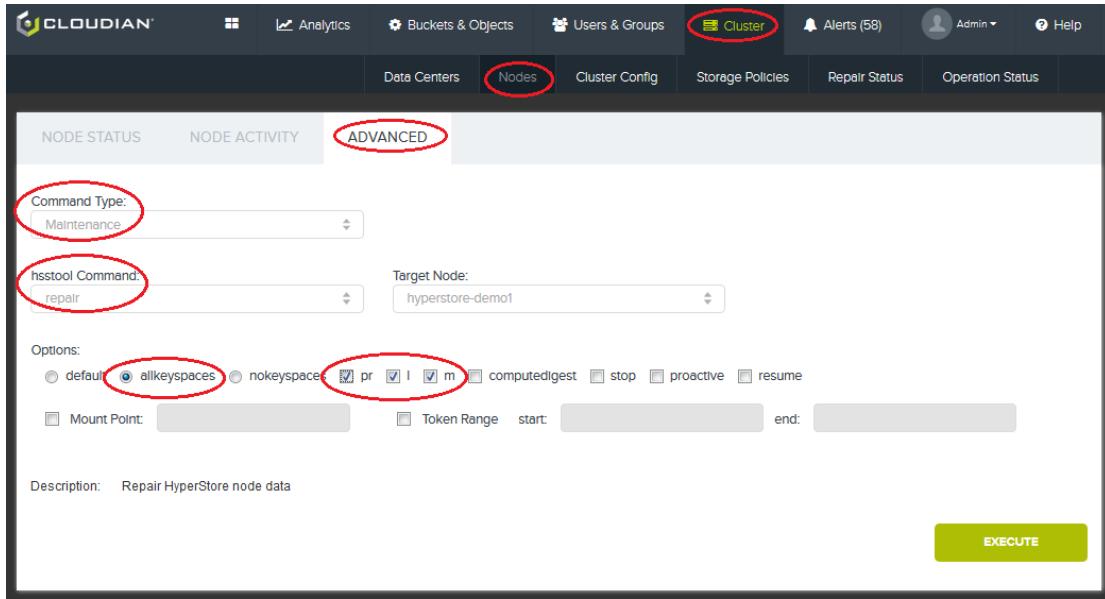
More detail

If the node you removed was "dead", take the following actions to recreate the removed node's data on the remaining nodes in the cluster. (If the node you removed was "live" -- in which case the **Uninstall Node** operation executed a decommissioning process -- these actions are not needed and you can jump down to Step 7.)

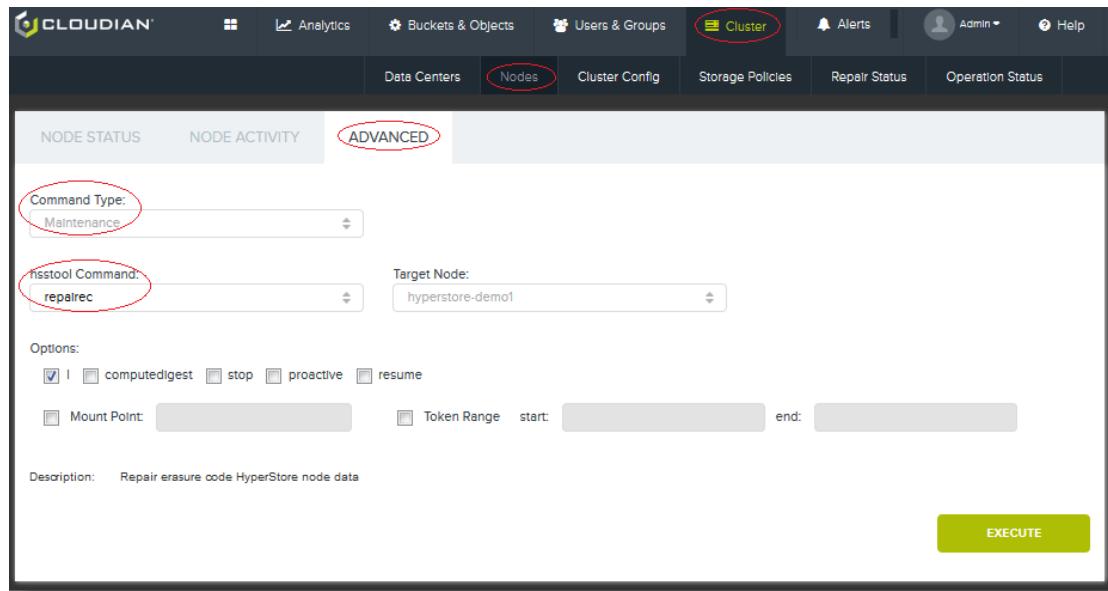
Note If you have removed a dead node from your cluster as a precursor to adding a new node to your cluster -- if you wish you can defer the time-consuming repair operations called for below until after you have added the new node(s). If that's the case, you can now perform the **"Adding Nodes"** (page 369) procedure, and that procedure indicates the point at which you should ini-

iate the deferred repairs. If you have removed a dead node and are **not** adding a new node, perform the repair now as described below.

From the **Node Advanced** page, run **hsstool repair** on each of the remaining nodes in the service region. When repairing each node, use the **allkeyspaces** option (so as to repair Cassandra metadata as well as S3 object data) and also the **-pr** option (this makes for more efficient repairs when repairing multiple nodes). Leave the **-I** and **-m** options selected, as they are by default. Since you are using the **-pr** option you can run repair on multiple nodes concurrently (in the GUI you will have to execute the repair command for each node individually, but you do not need to wait for the repair operation on one node to complete before you execute the command on the next node).



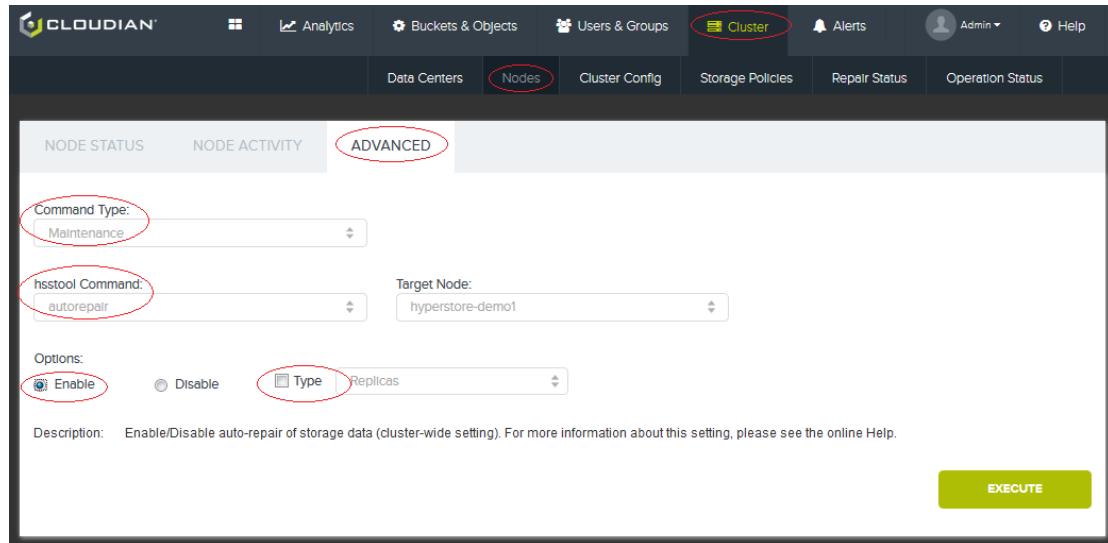
Also, if you have any erasure coded object data in your system, from the **Node Advanced** page run **hsstool repairec** on one node in each HyperStore data center in the region. It doesn't matter which node you run it on, as long as you do it for one node in each DC in the region. (Note that you do not need to wait for the in-progress *hsstool repair* operations to finish before launching *hsstool repairec* -- it's OK to run *hsstool repairec* and *hsstool repair* concurrently.)



Use the **Operation Status** page to track the progress of all repairs. After **all repairs have completed** proceed to Step 7.

Note Because the *repairec* operation repairs erasure coded data on all hosts in a data center, it's potentially a very long running operation. In a large cluster with high data volume it may take multiple weeks or even multiple months to complete.

7. In the **Node Advanced** page, from the Maintenance command group execute the *autorepair* command with the Enable option to **re-enable the HyperStore automated repair features** in the service region. The target node can be any node in the region. Leave the "Type" option unselected so that all repair types are enabled. This also re-enables proactive repair.



This completes the procedure for removing a node from your cluster.

Note that the CMC's **Uninstall Node** function deletes the node from the HyperStore cluster configuration and removes all HyperStore software from the node. It does not delete the Cassandra metadata or HyperStore object data from the node.

IMPORTANT: If the node was "live" when you removed it -- so that the **Uninstall Node** operation included a decommissioning process -- make sure that in the **Operation Status** page the **Uninstall Node** operation shows as having completed successfully with no errors, before you consider manually deleting the data that remains on the removed node. If the node was "dead" when you removed it -- so that you had to subsequently run repair operations on the remaining nodes in the cluster -- make sure that in the **Operation Status** page the **Uninstall Node** operation and also all those repair operations show as having completed successfully with no errors, before you consider manually deleting the data that remains on the removed node.

6.6. Replacing a Node

If you want to replace a **dead node** (for example a node that has failed due to hardware problems) with a new node:

- First perform the procedure for "**Removing a Node**" (page 390) (for the dead node).

Note: If your current number of nodes is at the minimum required by your storage policies, the system will not allow you to remove the node in the standard way. If this is your circumstance -- you have the minimum number of nodes required by your storage policies and one of those nodes is dead -- please contact Cloudian Support for guidance.

- Then perform the procedure for "**Adding Nodes**" (page 369) (for the new node).

If you want to replace a **live node** (a node on which the Cassandra Service and HyperStore Service are running and reachable) with a new node:

- First perform the procedure for "**Adding Nodes**" (page 369) (for the new node).
- Then perform the procedure for "**Removing a Node**" (page 390) (for the live node that you want to remove).

6.7. Restoring a Node That Has Been Offline

When a node has been down or inaccessible for a while and then you bring it back online, the HyperStore system **automatically** performs the most important actions necessary for restoring the node to a correct and up-to-date condition:

- HyperStore will use **proactive repair** to automatically populate the node with any data that the node is responsible for storing but that is missing due to the node having been offline.

IMPORTANT: The longest node outage that a proactive repair can cover for is four hours (by default configuration). **If a node is down for more than four hours you need to take manual steps to fully repair it.** For detail see "**6.7.1 Repairing a Node That's Been Down for Longer than the Proactive Repair Limit**" (page 402).

- If the node that you are restoring is a Redis slave node (for either the Redis Credentials DB or the Redis QoS DB), when you bring the node back online it will automatically sync with the Redis master node to get the most current data.

If you made any configuration changes to your cluster while the node was down, from the Puppet master node [do a Puppet push](#) out to the cluster after the node is back up.

Optionally, you can run a cleanup on the node in order to remove from the node any data that should no longer be there. This would apply, for example, if service users deleted some of their stored objects from the system while the node was down. In this case after being brought back into service the node will still be storing replicas or erasure coded fragments of those objects, resulting in wasted use of disk space. Cleaning the node removes this "garbage" data. For cleanup command details see "[hsstool cleanup](#)" (page 610) (if you have erasure coded data in your system use the command's -a option so that it cleans erasure coded data as well as replica data).

Note Before cleaning a node you should wait until any proactive repair that's automatically run on the node has completed. You can check this on the CMC's [Repair Status](#) page. Wait until the node's status displays as "All Clear", and then you can clean the node.

Also optionally, you can return to the node any specialized service role that the node was playing before it went down. If the node had been acting as a "master" or "primary" within one of the HyperStore system's specialized services, then when the node went offline that role would have failed over to a different node. If you want you can return the master or primary role to the restored node after it's back online — though it is not necessary to do so.

How going down and then being brought back up affects a node's specialized service roles...

The table below shows how having been down impacts a node's specialized service role(s).

If before going down the node was...	Then while the node was down that role...	And when brought back online the node is now...
Redis Credentials master	Automatically failed over to a Redis Credentials slave	Redis Credentials slave
Redis QoS master	Automatically failed over to a Redis QoS slave	Redis QoS slave
Redis Monitor primary	Automatically failed over to the Redis Monitor backup	Redis Monitor backup
Cron job primary	Automatically failed over to the Cron job backup	Cron job backup
Puppet Master primary	An operator may (or may not) have manually triggered a fail-over to the Puppet Master backup	Still Puppet Master primary, or else Puppet Master backup

To see what specialized service role(s) a restored node is currently playing, go to the CMC's [Cluster Information](#) page.

If you want to change the node's current role assignment(s), see the instructions for "[Change Node Role Assignments](#)" (page 405).

6.7.1. 6.7.1 Repairing a Node That's Been Down for Longer than the Proactive Repair Limit

In [mts.properties.erb](#) the setting "`hyperstore.proactiverepair.queue.max.time`" (page 522) sets the maximum time for which proactive repair jobs can be queued for a node that is unavailable. The default is 4 hours. This time limit prevents Cassandra from being over-loaded with metadata relating to proactive repair, and ensures that proactive repair is used only for its designed purpose, which is to repair object data from a relatively brief time period.

If a node is unavailable for longer than `hyperstore.proactiverepair.queue.max.time`, then the metadata required for implementing proactive repair on the node will stop being written to Cassandra, and an alert will display in the CMC's [Alerts](#) page. When the node comes back online you will need to:

1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until the proactive repair completes. You can check the CMC's [Repair Status](#) page periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the new and changed objects from during the period when proactive repair metadata was still being written to the Cassandra for the node.
2. After proactive repair on the node completes, manually initiate a full repair of the node. For information on manually initiating a repair see [hsstool repair](#) and [hsstool repairc](#). This will repair the new and changed objects from the period after the proactive repair queueing time maximum was reached and before the node came back online.

6.8. Changing a Node's IP Address

Changing the IP address of an existing node in your cluster would have many repercussions for the cluster and is not recommended. There is no standardized, supported way of changing a HyperStore node's IP address through the CMC or through command line tools. For more detail on this topic contact Cloudian Support.

6.9. Backing Up and Restoring a Cluster

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Backing Up an Entire Cluster"** (page 403)
- **"Restoring an Entire Cluster"** (page 404)

This procedure is for backing up and restoring an entire HyperStore cluster (i.e. an entire HyperStore service region). This procedure should not be used for partial backups and restores.

IMPORTANT: Make sure you have enough space for the backup. If you back up on to nodes in your HyperStore cluster, this will double disk usage within the cluster.

Note Throughout this procedure, it's assumed that you have used the default installation directories for HyperStore binaries. If not, adjust the command paths stated in the procedure.

6.9.1. Backing Up an Entire Cluster

Note Daily at 11PM HyperStore implements a Redis cron job (configured in `/etc/cron.d/redis-crontab`) that invokes a script (`/opt/redis/redisBackup.sh`) on your Redis Credentials master node and on your Redis QoS master node. The script backs up the Redis Credentials database into a `/var/lib/redis/dump-credentials.rdb` file and backs up the Redis QoS database into a `/var/lib/redis/dump-qos.rdb` file. Prior to doing so it first moves the previous day's `dump-credentials.rdb` or `dump-qos.rdb` file into a `/var/lib/redis/backup` directory, compresses the file, and appends a date-time stamp to its name (for example `/var/lib/redis/backupdump-credentials.rdb.2017111616.gz`). In the `/var/lib/redis/backup` directory, files are rotated such that the most recent seven days worth of Redis backups are retained and older backups are deleted.

So you always have access to Redis backup files that are generated at 11PM daily (more specifically, the backup starts at 11PM -- it may take some time to complete). However, if you want a more current Redis backup you can follow steps 1a and 1b below to re-generate the `dump-credentials.rdb` and `dump-qos.rdb` files.

Logging output from the `redisBackup.sh` script runs is written to `/var/log/redis/redisBackup.log`. (A separate log file `cloudian-redisBackup.log` merely records information having to do with `cron.d` launching the script.)

1. Back up the Redis Credentials and Redis QoS databases.

- a. On the Redis Credentials master node, use the Redis CLI to perform a [BGSAVE](#) operation:

```
[root]# /opt/redis/redis-cli -h <hostname> -p 6379 BGSAVE
```

The above command will update the `dump-credentials.rdb` file in the Redis data directory (`/var/lib/redis` by default).

Note The BGSAVE operation saves the database in the background. To check when the save is done you can use the Redis CLI [LASTSAVE](#) command, which returns the Unix time of the most recently completed save.

- b. On the Redis QoS master node, use the Redis CLI to perform a [BGSAVE](#) operation:

```
[root]# /opt/redis/redis-cli -h <hostname> -p 6380 BGSAVE
```

The above command will update the `dump-qos.rdb` file in the Redis data directory.

- c. Place a copy of the `dump-credentials.rdb`, `dump-qos.rdb`, `appendonly_cred.aof`, and `appendonly_qos.aoffiles` in a safe place.

2. Back up Cassandra and Hyperstore data **on each of your HyperStore nodes**, using the third party backup tool of your choice:

- For Cassandra data, on each node:

- First flush Cassandra:

```
[root]# /opt/cassandra/bin/nodetool -h <hostname> flush
```

- Then with your third party tool, back up the Cassandra data directory. (To check which directory is the Cassandra data directory, see the configuration setting *common.csv*: "**cas-sandra_data_directory**" (page 480)).
 - For HyperStore data, on each node: With your third party tool, back up the HyperStore data directories. (To check which directories are the HyperStore data directories, see the configuration setting *common.csv*: "**hyperstore_data_directory**" (page 462)).
3. Back up HyperStore binaries and configuration files:
- On the Puppet master node:
 - */etc/cloudian-<version>-puppet* for HyperStore version 5.2 or later, or */etc/puppet* for versions older than 5.2
 - The current HyperStore installation staging directory (where the *cloudianInstall.sh* tool is)
 - On each of your HyperStore nodes:
 - */opt/cloudian*
 - */opt/cassandra*
 - */opt/redis*
 - */opt/tomcat*
 - */opt/cloudianagent*
 - */opt/dnsmasq*

6.9.2. Restoring an Entire Cluster

This restore procedure assumes that the restored cluster is the same configuration as what you backed up — i.e. the same IP addresses and mount points.

1. Restore HyperStore binaries and configuration files.
 - On the Puppet master node:
 - */etc/cloudian-<version>-puppet* for HyperStore version 5.2 or later, or */etc/puppet* for versions older than 5.2
 - The HyperStore installation staging directory (where the *cloudianInstall.sh* tool is)
 - On each of your HyperStore nodes:
 - */opt/cloudian*
 - */opt/cassandra*
 - */opt/redis*
 - */opt/tomcat*
 - */opt/cloudianagent*
 - */opt/dnsmasq*
2. Restore Cassandra and HyperStore data directories **on each of your HyperStore nodes**.
3. Restore the Redis Credentials and Redis QoS databases.

6.10. Change Node Role Assignments

In a HyperStore cluster there are certain specialized service roles that are assigned to some nodes and not to others. When you install a HyperStore system, the installer assigns these roles automatically and in a way that's appropriate to your cluster topology. The roles are implemented in such a way that no node constitutes a single point of failure.

For a summary of where specialized service roles are currently assigned in your cluster, go to the CMC's [Cluster Information](#) page.

If you wish you can change node role assignments by using the HyperStore installer on the Puppet master node. The installer supports the role assignment operations listed below.

Note Each of the role assignment change operations listed below entails doing a Puppet push and a restart of the affected service (as described in the instructions for each operation). If you need to perform multiple of these role assignment change operations, do them one operation at a time and with a Puppet push and affected service restart at the end of each operation. Do **not** perform multiple different role assignment change operations while deferring the Puppet push and service restart.

- "[Move the Redis Credentials Master or QoS Master Role](#)" (page 405)
- "[Move or Add a Redis Credentials Slave or Redis QoS Slave](#)" (page 408)
- "[Move the Cassandra Seed Role](#)" (page 410)
- "[Reduce or Change the List of CMC Hosts](#)" (page 411)
- "[Move the Redis Monitor Primary or Backup Role](#)" (page 412)
- "[Move the Cron Job Primary or Backup Role](#)" (page 414)
- "[Move the Puppet Master Primary or Backup Role](#)" (page 415)
- "[Change Internal NTP Servers or External NTP Servers](#)" (page 418)

6.10.1. Move the Redis Credentials Master or QoS Master Role

The [Redis Credentials master](#) role is assigned to just one node in your entire HyperStore system (the system has just one master even if there are multiple service regions). The [Redis QoS master](#) role is assigned to one node in each of your service regions (each region has its own master). The HyperStore installer automatically makes these role assignments when you install your system.

If you wish you can move the Redis Credentials master role to a current Redis Credentials slave node, or move a Redis QoS master role to a current Redis QoS slave node, by following the procedure in this section. If you do so, the node that had been master will automatically become a slave.

Note The system does not support moving the master role to a node that is not currently a slave.

6.10.1.1. Preparing to Move the Redis Master Role

For safety, before moving a Redis master role make a backup copy of the current database from the master. You can do so by using the Redis CLI command [BGSAVE](#) as described below.

To back up a **Redis Credentials** master database:

```
[root]# /opt/redis/redis-cli -h <hostname_of_master_node> -p 6379 BGSAVE
```

The above command will update the *dump-credentials.rdb* file in the Redis data directory (*/var/lib/redis* by default).

To back up a **Redis QoS** master database:

```
[root]# /opt/redis/redis-cli -h <hostname_of_master_node> -p 6380 BGSAVE
```

The above command will update the *dump-qos.rdb* file in the Redis data directory (*/var/lib/redis* by default).

Note The BGSAVE operation saves the database in the background. To check to see whether the save is completed yet you can use the Redis CLI [LASTSAVE](#) command, which returns the Unix time of the most recently completed save.

6.10.1.2. Moving the Redis Master Role

1. Submit a Redis CLI *info* command to the **Redis slave node** to which you want to move the master role. Confirm that the slave node returns the following status information:

- *role* is slave
- *master_host* is the current master
- *master_link_status* is up
- *master_sync_in_progress* is 0

When submitting the Redis CLI command, use port 6379 if the target node is a Redis Credentials slave, or use port 6380 if the target node is a Redis QoS slave.

For example, if the Redis Credentials master role is currently on host "cloudian-node2" and you want to move it to host "cloudian-node7" where there is currently a Redis Credentials slave:

```
# /opt/redis/redis-cli -h cloudian-node7 -p 6379 info | egrep "role|master_host|master_link_status|master_sync_in_progress"

role:slave
master_host:cloudian-node2
master_link_status:up
master_sync_in_progress:0
```

2. Dynamically switch the master role to the slave:

Log into the CMC and go to **Cluster** → **Nodes** → **Advanced**. Select Command Type "Redis Monitor Operations", then select a Cluster type (Credentials or QoS) and select Command "setClusterMaster".

The screenshot shows a web-based interface for changing node role assignments. At the top, there are tabs for 'NODE STATUS', 'NODE ACTIVITY', and 'ADVANCED' (which is circled in red). Below these are input fields: 'Command Type' (set to 'Redis Monitor Operations'), 'Cluster' (set to 'credentials'), 'Hostname' (set to 'jenkins-jdk8'), and 'Command' (set to 'setClusterMaster'). A description below the fields states: 'Description: Assign a Redis cluster's master role to a different node within the cluster'. At the bottom right is a green 'EXECUTE' button.

For "Hostname" select the host to which you want to move the Redis master role. The drop-down list will show only nodes that are currently slaves within the Cluster type that you selected.

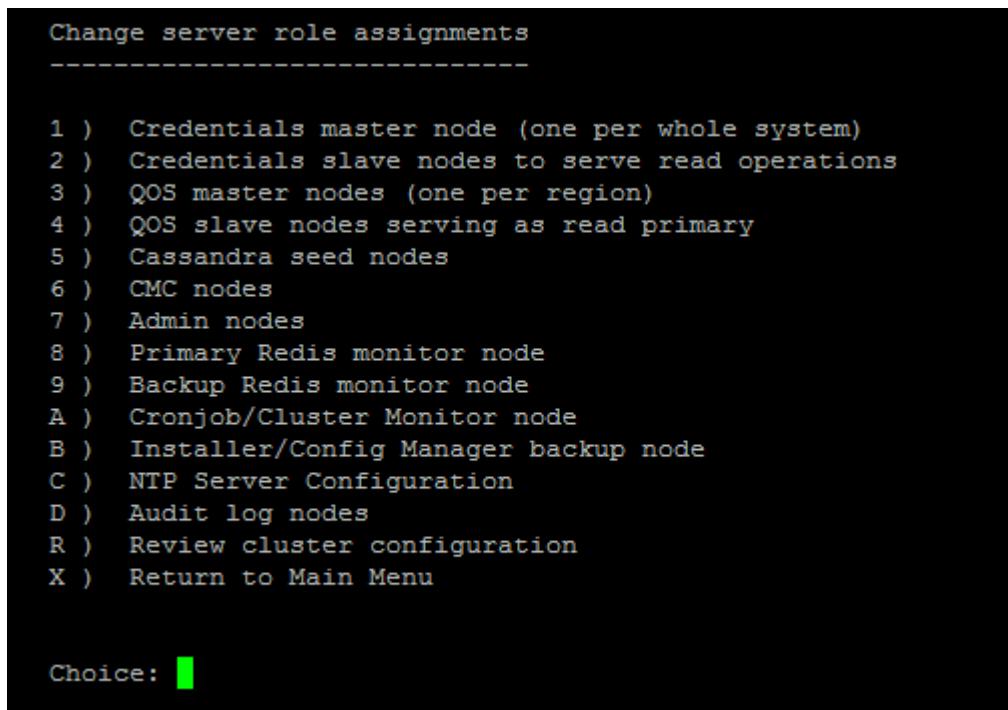
After making your selections, click **Execute**. The chosen slave will become the master, while the master becomes a slave. The change happens immediately upon command execution.

3. Make the switch of the master and slave permanent by updating your system configuration:

- a. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <installation-staging-directory>
[root]# ./cloudianInstall.sh
```

- b. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.



- c. From the Change Server Role Assignments menu select the option for the master that you want to move — Credentials master or QoS master.
- d. At the prompt indicate the host to which you want to move the Redis master role. This should be the same host that you checked in Step 1.
- e. After completing the interaction for specifying the new master host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
- f. Return to the installer's main menu, then choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster.
- g. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the following services:
 - The Redis service for which you've changed the master role (either Redis Credentials or Redis QoS)
 - Redis Monitor
 - S3 Service (restarting this service also results in a restart of the Admin Service)

After these services have successfully restarted you can exit the installer.

To verify your change, log into the CMC and go to the [Cluster Information](#) page. Review the service information section to confirm that the Redis master that you moved is now where you want it to be.

Now, the former Redis slave has been promoted to master and the former master has been demoted to slave.

IMPORTANT: If you want to remove the demoted host (the former master that's now a slave) from your cluster, you must first move its slave role to a different host in your cluster. For instructions see "[Move or Add a Redis Credentials Slave or Redis QoS Slave](#)" (page 408).

6.10.2. Move or Add a Redis Credentials Slave or Redis QoS Slave

By default the HyperStore installer deploys two [Redis Credentials slaves](#) and one [Redis QoS slave](#) per data center. The system supports moving a slave from its current host to a different host -- for example, moving a Redis Credentials slave to a host that is not currently running the Redis Credentials service. It also supports adding a new Redis Credentials or QoS slave rather than moving one — so that you end up with more slaves than when you started.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

```

Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [ ]
```

3. From the Change Server Role Assignments menu select the option for Credentials slave nodes or the option for QoS slave nodes.
4. The installer prompts you to specify a comma-separated list of all the hosts on which you want slaves to run. The installer's prompt text indicates [in brackets] which hosts are the **current** slaves. You can use your entry at the prompt to either move a slave or add a slave.

Example of moving a slave:

```
Enter a comma separated list of Credentials slave hosts in region1's
DC1 data center [deneb,vega]: deneb,altair
```

Example of adding a slave:

```
Enter a comma separated list of Credentials slave hosts in region1's
DC1 data center [deneb,vega]: deneb,vega,altair
```

Note If your HyperStore system has multiple data centers, the installer will prompt you separately for each data center's slave host list. If for some data centers you want to continue using the same slave(s) you can just press enter at the prompt rather than entering a host list.

5. After completing the interaction for specifying the slave locations, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu, then choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the following services:
 - Redis Credentials or Redis QoS (whichever service you moved or added a slave for)

- Redis Monitor
- S3 Service (restarting this service also results in a restart of the Admin Service)

After these services have successfully restarted you can exit the installer.

To verify your change, log into the CMC and go to the [Node Status](#) page. Review the service status information for the node(s) on which you've located the slave(s). Among the listed services on the node(s) should be "Redis Cred" (for a Redis Credentials slave) or "Redis QoS" (for a Redis QoS slave). The absence of "(Master)" appended to the service name indicates it's a slave instance, not a master.

6.10.3. Move the Cassandra Seed Role

Cassandra "seed" nodes serve to bootstrap the Gossip process for new nodes when you add new nodes to your cluster. Gossip is the peer-to-peer communication protocol by which Cassandra nodes regularly exchange state information. The recommended configuration is to have three of your Cassandra nodes act as "seed" nodes in each data center in which you have deployed HyperStore. The HyperStore installer automatically makes these role assignments when you install your system.

If you wish you can change the list of Cassandra seed nodes for your system, by following the steps below.

Note Cassandra runs on every one of your HyperStore nodes. So any of your nodes can play the "seed" role. But bear in mind the recommendation that you have three seed nodes per data center. For performance reasons it's not advisable to have more seed nodes than this.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

```

Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [redacted]

```

3. From the Change Server Role Assignments menu select "Cassandra seed nodes".
4. At the prompt enter a comma-separated list of hosts that you want to serve as Cassandra seed nodes. If you want to keep the same host just press Enter rather than specifying a different host. (If you have a multi-region system, you will be prompted for a list of Cassandra seed nodes for each region.)
5. After completing the interaction for specifying NTP Server Configuration, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the Cassandra service. After the Cassandra service has successfully restarted you can exit the installer.

To verify your change, you can look at the `/opt/cassandra/conf/cassandra.yaml` configuration file on any one of your nodes and confirm that the "seeds:" parameter is set to the list of hosts that you specified.

6.10.4. Reduce or Change the List of CMC Hosts

By default the [Cloudian Management Console](#) (CMC) is installed and runs on all of your HyperStore hosts. If you wish, after initial deployment of your system you can use the installer to specify a list of HyperStore hosts on which to have the CMC run, rather than having it run on all nodes.

below.

Note Cassandra runs on every one of your HyperStore nodes. So any of your nodes can play the "seed" role. But bear in mind the recommendation that you have three seed nodes per data center. For performance reasons it's not advisable to have more seed nodes than this.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

The screenshot shows a terminal window with the following content:

```
Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [
```

The menu lists various server roles with their descriptions. At the bottom, there is a prompt "Choice:" followed by a cursor key input field.

3. From the Change Server Role Assignments menu select "CMC nodes".
4. At the prompt enter a comma-separated list of hosts on which you want the CMC to run.
5. After completing the interaction for specifying your CMC host list, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster.
7. Return to the "Cluster Management" menu, then choose "Manage Services" and restart the CMC service. After the CMC service has successfully restarted you can exit the installer.

6.10.5. Move the Redis Monitor Primary or Backup Role

The [Redis Monitor](#) runs on two nodes in your cluster, with one being the primary Redis Monitor instance and the other being a backup instance. In the event that the Redis Monitor primary goes offline the Redis Monitor backup **automatically** detects this and takes over as the active Redis Monitor.

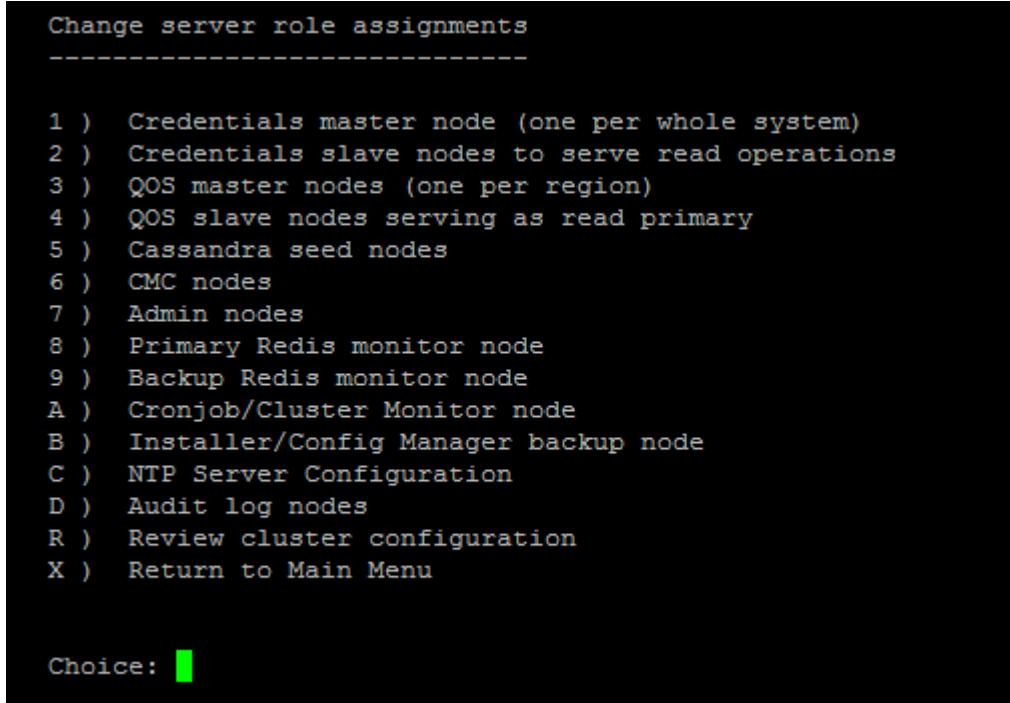
Note In a multi- data center HyperStore system, the Redis Monitor backup should remain in the same data center as the Redis Monitor primary, and this should be the same data center as where the Redis Credentials master is located.

The system supports moving the primary or backup Redis Monitor to a different host as described below.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.



```
Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [ ]
```

3. From the Change Server Role Assignments menu select the option for the Redis Monitor instance that you want to move (Primary Redis Monitor or Backup Redis Monitor).
4. At the prompt specify the host to which you want to move the Redis Monitor instance.
5. After completing the interaction for specifying the new Redis Monitor location, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster.
7. From the "Cluster Management" menu choose "Manage Services" and restart the Redis Monitor service. After the Redis Monitor successfully restarts you can exit the installer.

To verify your change, log into the CMC and go to the [Cluster Information](#) pagee. Review the service information section to confirm that your Redis Monitor primary and backup hosts are what you want them to be.

6.10.6. Move the Cron Job Primary or Backup Role

Certain HyperStore [system maintenance tasks are implemented by cron jobs](#) that run on a regular schedule, as configured by a crontab. When your HyperStore system is installed the install script designates one node to host the crontab configuration and run the cron jobs, and a second node to serve as a backup. In the event that the primary cron job host goes offline (or if *crond* goes down) the backup host **automatically** takes over as the active cron job host.

The HyperStore [Monitoring Data Collector](#) resides on the same primary host and backup host as the cron jobs. If the cron jobs role automatically fails over from the primary host to the backup host, the Monitoring Data Collector role also fails over to the backup.

The system supports moving the primary cron job role (and with it, the primary Monitoring Data Collector role) to a different host as described below. The same procedure also supports moving the backup cron job role (and with it, the backup Monitoring Data Collector role) to a different host.

Note Do not assign the primary cron job role to the same host as your Puppet Master role. If the cron job primary and the Puppet Master are on the same host and that host goes down, automated fail-over from the cron job primary to the cron job backup will not work.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

```
Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu
```

```
Choice: [
```

3. From the Change Server Role Assignments menu select "Cronjob/Cluster Monitor node".
4. At the prompts specify your desired primary host and backup host for the cron jobs / cluster monitor.
5. After completing the interaction for specifying cron job hosts, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. When Puppet pushes the current configuration settings to the cluster it will also automatically restart *cron.d* on the affected nodes. You do not need to manually restart any services..When the Puppet push completes you can exit the installer.

To verify your change, log into the CMC and go to the [Cluster Information](#) page. Review the service information section to confirm that your System Monitoring/Cronjob primary and backup hosts are what you want them to be.

6.10.7. Move the Puppet Master Primary or Backup Role

When you install your HyperStore system, you choose the node that you want to serve as the Puppet Master for cluster configuration management, and you run the installer on that node. The installer configures that node to be the primary Puppet Master, and also configures a second node to be the Puppet Master backup. Any edits that you make to configuration templates on the Puppet Master primary are automatically sync'd to the Puppet Master backup. If the primary goes down, you can **manually** fail over the active Puppet Master role to the backup host.

There are two different operations that you can perform in regard to the Puppet Master role:

Move the Puppet Master Backup

In this scenario your primary Puppet Master is fine, and you're simply looking to relocate the Puppet Master backup role to a different host than it is currently on.

Note You can find out which host is currently the Puppet Master backup host in the CMC's [Cluster Information](#) page.

1. On the Puppet Master primary host, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

```

Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [ ]
```

3. From the Change Server Role Assignments menu select "Installer/Config Manager backup node".
4. At the prompt specify the host to which you want to move the Puppet Master backup role.
5. After completing the interaction for specifying the Puppet Master backup host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
6. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster. There is no need to restart any services.
7. Exit the installer, wait at least one minute, then log into the CMC and go to the [Cluster Information](#) page. Review the service information section to confirm that your Puppet Master primary and backup hosts are what you want them to be.

Manually Fail Over the Puppet Master Role from the Primary to the Backup

In this scenario there is a problem with your primary Puppet Master, and you want the backup Puppet Master to become active.

IMPORTANT: For this procedure you **log into the current Puppet Master backup host** and implement the whole procedure from that host. You can find out which host is currently the Puppet Master backup host in the CMC's [Cluster Information](#) page.

1. **On the Puppet Master backup host**, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Start or stop Puppet daemon".

3. At the prompt specify that you want to stop Puppet. This stops any Puppet daemons that are currently running in your cluster.
4. After the installer indicates that Puppet has been stopped, return to the Advanced Configuration Options menu and select "Remove existing Puppet SSL certificates". This will remove existing Puppet SSL certificates, with no further prompts.
5. Return to the Advanced Configuration Options menu and select "Change server role assignments". This displays the Change Server Role Assignments menu.

```

Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [ ]

```

6. From the Change Server Role Assignments menu select "Installer/Config Manager backup node".
7. At the prompt specify the host to which you want to move the Puppet Master (config manager) backup role. You need a new backup because you are converting the original backup into the primary (by running through this procedure using the installer on the original backup -- this has the effect of turning it into the new primary).
8. After completing the interaction for specifying the Puppet Master backup host, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
9. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. This triggers Puppet to push your configuration change out to the cluster. Then do "Cluster Management" → "Manage Services" and restart the CMC.
10. Optionally, if you want to leave the Puppet daemons running, from the installer's main menu select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Start or stop Puppet daemon", and choose to start the daemons. After the daemons have successfully started you can exit the installer.

To verify your change, log into the CMC and go to the [Cluster Information](#) page. Review the service information section to confirm that your Puppet Master primary and backup hosts are what you want them to be. The former backup (from which you implemented the above procedure) should now be the primary and the new backup should be as you specified during the procedure.

Note If the Puppet Master primary is now on the same host as the cron job primary role you should [move the cron job primary role](#) to a different host. If the cron job primary and the running Puppet Master are on the same host and that host goes down, automated fail-over from the cron job primary to the cron job backup will not work.

6.10.8. Change Internal NTP Servers or External NTP Servers

When you install your HyperStore cluster, for each of your data centers the installation script automatically configures four of your HyperStore nodes to act as internal NTP servers for the cluster. These internal NTP servers synchronize with external NTP servers. By default the set of external servers is:

- 0.centos.pool.ntp.org
- 1.centos.pool.ntp.org
- 2.centos.pool.ntp.org
- 3.centos.pool.ntp.org

Note For more on how HyperStore automatically configures an NTP set-up for the cluster, see "[NTP Automatic Set-Up](#)" (page 540)

As described below, the system supports changing the list of internal NTP servers (for data centers in which you have more than four HyperStore nodes) and/or changing the list of external NTP servers.

1. On the Puppet master node, launch the HyperStore installer.

```
[root]# cd <your-installation-staging-directory>
[root]# ./cloudianInstall.sh
```

2. From the installer's main menu, select "Advanced Configuration Options". Then from the Advanced Configuration Options menu select "Change server role assignments". This displays the Change Server Role Assignments menu.

```

Change server role assignments
-----
1 ) Credentials master node (one per whole system)
2 ) Credentials slave nodes to serve read operations
3 ) QOS master nodes (one per region)
4 ) QOS slave nodes serving as read primary
5 ) Cassandra seed nodes
6 ) CMC nodes
7 ) Admin nodes
8 ) Primary Redis monitor node
9 ) Backup Redis monitor node
A ) Cronjob/Cluster Monitor node
B ) Installer/Config Manager backup node
C ) NTP Server Configuration
D ) Audit log nodes
R ) Review cluster configuration
X ) Return to Main Menu

Choice: [ ]

```

3. From the Change Server Role Assignments menu select "NTP Server Configuration".
4. At the first prompt specify the HyperStore hosts that you want to act as the internal NTP servers, as a comma-separated list. You must use host names, not IP addresses. If you want to keep the same hosts that the system is currently using, just press Enter rather than specifying different hosts.

Note: If you have multiple HyperStore data centers you will be prompted separately for the internal NTP host list for each data center.

5. At the next prompt specify the external NTP servers with which you want the internal NTP servers to synchronize, as a comma-separated list. For these external servers you can use either FQDNs or IP addresses. If you want to keep the same external servers that the system is currently using, just press Enter rather than specifying a different server list.
6. After completing the interaction for specifying NTP Server Configuration, return to the Change Server Role Assignments menu and select "Review cluster configuration". Then at the prompt confirm that you want to apply the updated configuration to the Puppet master.
7. Go to the installer's main menu and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts. When Puppet pushes the current configuration settings to the cluster it will also automatically restart *ntpd* on the affected nodes. You do not need to manually restart *ntpd*. When the Puppet push completes you can exit the installer.

To verify your change, log into the CMC and go to the [Cluster Information](#) page. Review the service information section to confirm that your internal NTP hosts and external NTP hosts are what you want them to be.

6.11. Cron Jobs and Automated System Maintenance

The HyperStore system automatically executes a variety of maintenance tasks. This section describes the automated maintenance jobs that the system implements. The covered topics are:

- "**System cron Jobs**" (page 420)
- "**Scheduled Auto-Repair**" (page 426)
- "**Cassandra Data Compaction**" (page 426)

6.11.1. System cron Jobs

Most HyperStore automated maintenance routines are implemented as cron jobs. Maintenance cron jobs are run from one HyperStore node in each of your service regions. To see which of your nodes is running the cron jobs, go to the CMC's [Cluster Information](#) page. In the Service Information section you will see the identity of the System Monitoring/Cronjob Primary Host, from which the cron jobs are run.

Note You will also see the identity of the System Monitoring/Cronjob Backup Host. If the primary cron job instance goes down (due to the host going down or *crond* going down on the host) and remains down for 10 minutes, the backup cron job instance will automatically take over the primary role and start running the system cron jobs.

The cron jobs themselves are configured in the */etc/cron.d/cloudian-crontab* file on the host node. If you want to adjust the scheduling of these cron jobs you should do so via Puppet configuration management, by editing the configuration template file */etc/cloudian-<version>-puppet/modules/cloudians3/templates/cloudian-crontab.erb* on the Puppet master node. For general information on how to configure cron job scheduling, refer to any reputable resource on the topic.

Note that most of the cron jobs configured in *cloudian-crontab.erb* have "> /dev/null 2>&1" appended to them and therefore they direct all output to */dev/null*.

System cron jobs are implemented for the following system maintenance tasks:

System Monitoring Data Collection

Scope: One job per region

Frequency: Every 1 minute

Operation invoked: S3 Service internal process */bin/snapshotData*

This cron job invokes a process that logs "snapshot" system statistics each minute in support of the Cloudian Management Console's system monitoring functionality.

Diagnostic Log Upload

Scope: One job per region

Frequency: Once per day

Operation invoked: S3 Service internal process */bin/phoneHome*

This cron job uploads a daily diagnostics file to a configurable S3 URI, if the HyperStore "Phone Home" feature (also known as "Smart Support") is enabled. Typically this would be the S3 URI of Cloudian Support.

Note The upload will occur within an hour of the time specified in the crontab. A random wait time is built into the upload process so that not all Cloudian customer environments are uploading to Cloudian Support at the same time.

Usage Data Processing

Several cron jobs are involved in processing service usage data for users and groups.

Note For an overview of how the HyperStore system tracks service usage by groups and users, see "["Usage Reporting Feature Overview"](#) (page 153).

Saving Storage Usage Data for Active Users

Scope: One job per region

Frequency: Every 5 minutes

Operation invoked: Admin API method [POST /usage/storage](#)

This cron job writes snapshots of per-user and per-group counts for stored bytes and number of stored objects from the Redis QoS database over to the "Raw" column family in the Cassandra "Reports" keyspace. The operation is applied only to users and groups that have uploaded or deleted objects in the time since the operation was last executed.

Saving Storage Usage Data for All Users

Scope: One job per region

Frequency: Once a day

Operation invoked: Admin API method [POST /usage/storageall](#)

This cron job writes snapshots of per-user and per-group counts for stored bytes and number of stored objects from the Redis QoS database over to the "Raw" column family in the Cassandra "Reports" keyspace. The operation is applied to **all** users and groups.

Repairing Usage Data for Active Users

Scope: One job per region

Frequency: Every 12 hours

Operation invoked: Admin API method [POST /usage/repair/dirtyusers](#)

This cron job repairs Redis QoS stored bytes and stored object counts for up to 1000 active or "dirty" users. See the Admin API method description for more detail.

Creating Usage Roll-Up Reports

Scope: One job per region for each roll-up granularity (hour, day, month)

Frequency: Hourly, daily, monthly

Operation invoked: Admin API method [POST /usage/rollup](#)

- One job with granularity=hour (runs once per hour)
- One job with granularity=day (runs once per day)
- One job with granularity=month (runs once per month)

These three cron jobs create summary (or "roll-up") usage reports data from more granular reports data. The hourly roll-up data is derived from the raw data (the transactional data and stored bytes/stored object count snapshot data). The daily roll-up data and monthly roll-up data are both derived from the hourly roll-up data.

Note Hourly rollup jobs that fail -- such as if a relevant service is down or unreachable -- will be retried. The time span for which failed rollup jobs are eligible for retry is configuration by the "**"usage.rollup.hour.maxretry"** (page 513) property in [mts.properties.erb](#). The default is 6 hours.

Bucket Log Processing

Scope: One job per region

Frequency: Once every 10 minutes

Operation invoked: S3 Service internal process `./system/dumpbucketlogs`

This cron job moves bucket logging data from the Cassandra BucketLog column family into the S3 storage system, in support of the S3 Bucket Logging feature.

Bucket Lifecycle Processing

Two cron jobs are involved in S3 Bucket Lifecycle implementation.

Auto-Tiering and Auto-Expiring Objects

Scope: One job per region

Frequency: Once a day

Operation invoked: S3 Service internal process `./system/autoexpire`

This cron job performs two tasks in support of S3 bucket lifecycle policies:

- Transitions (auto-tiers) objects to a tiering destination system, if the objects have reached their scheduled auto-tiering interval or date.

Note: For more information on the HyperStore auto-tiering feature, see ["Auto-Tiering Feature Overview"](#) (page 53).

- Deletes objects from HyperStore storage (or from the remote tiered storage system if they've been auto-tiered), if the objects have reached their scheduled expiration interval or date.

Restoring Auto-Tiered Objects

Scope: One job per region

Frequency: Every six hours

Operation invoked: S3 Service internal process `./system/restore`

This cron job executes queued object-restore jobs. Restore jobs are placed in queue when S3 clients invoke the S3 API method ["POST Object restore"](#) (page 938), to restore a local copy of objects that have been auto-tiered to a remote storage system. The cron job executes queued restored jobs every six hours.

Object Deletion Queue Processing

Scope: One job per region

Frequency: Every hour

Operation invoked: S3 Service internal process `./system/deleteobjects`

When S3 client applications delete S3 objects, the HyperStore system deletes the object metadata immediately but does not delete the actual objects immediately. Instead the objects are marked for deletion by the next run of the `./system/deleteobjects` cron job.

Similarly, when S3 clients overwrite S3 objects, the HyperStore system writes the new version of the object immediately, and updates the object metadata immediately, but does not delete the outdated version of the object immediately. Instead the outdated object versions are marked for deletion by the next run of the `./system/deleteobjects` cron job. (Note that in a bucket that has [versioning](#) enabled, the old object versions would be retained rather than deleted.)

Each run of the `./system/deleteobjects` cron job deletes all the S3 objects that are queued for deletion unless throttling is configured by the `mts.properties.erb`: **"cloudian.s3.batch.delete.maxkeys"** (page 509) setting. The speed of the operation execution can also be throttled, by using the `mts.properties.erb`: **"cloudian.s3.batch.delete.delay"** (page 509) setting.

If a run of `./system/deleteobjects` is still in progress at the time of the next scheduled run (one hour later by default), the latter run is skipped.

Note This cron job also processes the deletion of objects that you have purged from a bucket using the Admin API operation [POST /bucketops/purge](#).

Dealing with Excessive Tombstone Build-Up

Under normal circumstances the hourly running of the `./system/deleteobjects` cron job should ensure that there is no excessive build-up of Cassandra "tombstones" in connection with the deletion of objects from users' storage buckets. However, it is possible to encounter `TombstoneOverwhelmingException` errors in Cassandra logs and an inability to successfully execute an S3 **"GET Bucket (List Objects) Version 1"** (page 896) or **"GET Bucket (List Objects) Version 2"** (page 897) operation against a specific bucket, in either of these unusual circumstances:

- An S3 client application has attempted to delete more than 100,000 objects from the bucket in less than an hour.
- Over the course of multiple hours an S3 client application has attempted to delete more than 100,000 objects from the bucket and during that time the hourly `./system/deleteobjects` cron job has failed to purge tombstones for one reason or another.

In such circumstances you can trigger tombstone removal by connecting to any S3 server's JMX port (19080) and submitting a `purgeTombstone` command with the bucket name as input. If you are using JConsole, after connecting to port 19080 on an S3 node select the `MBeans` tab, then select `com.cloudian.ss.cassandra.cl → BatchJobs → Operations → purgeTombstone`. On the right side of the console enter the bucket name as the `p1` value and then execute the operation (by clicking `purgeTombstone` on the right side of the console).

User Deletion Cleanup

Scope: One job per HyperStore system (run only in default region)

Frequency: Once a day

Operation invoked: Admin API method `POST /system/deletedUserCleanup` (for system use only; this API method is not documented)

This cron job attempts to complete the user deletion process for users for whom the deletion process failed to complete on the original attempt. These are users who are stuck in "Deleting" status for one reason or another.

Storage Policy Deletion or Creation Processing

Scope: One job per HyperStore system (run only in default region)

Frequency: Once a day

Operation invoked: Admin API method [`POST /system/processProtectionPolicy`](#)

This cron job processes any pending storage policy delete jobs. System operators can initiate the deletion of an unused storage policy (a storage policy that is not assigned to any buckets) through the CMC's [Storage Policies](#) page. This operator action marks the policy with a "DELETED" flag and makes it immediately unavailable to service users. However, the full deletion of the storage policy from the system (specifically, the deletion of the Cassandra keyspace associated with the policy) is not processed until this cron job runs.

This cron job also processes any pending storage policy creation jobs, in the event that multiple storage policy creation requests have been initiated in a short amount of time -- which can result in queueing of storage policy creation jobs. More typically, storage policy creation completes shortly after the creation is initiated through the CMC.

Retries of Pending Cross-Region Replication Jobs

Scope: One job per region

Frequency: Every four hours

Operation invoked: S3 Service internal process `./system/processcrr`

This cron job processes any pending [cross-region replication](#) jobs. These pending jobs result when an attempt to replicate an object from the source bucket to the destination bucket results in a temporary error such as a connection failure or an HTTP 5xx error. This cron job retries the replication of such objects. For such objects, the retries will recur once every four hours until either the objects are successfully replicated to the destination system or a permanent error is encountered.

Retries of Pending "Bridge Mode" Auto-Tiering Jobs

Scope: One job per HyperStore system (run only in default region)

Frequency: Every six hours

Operation invoked: S3 Service internal process `./system/processproxy`

This cron job processes any pending "bridge mode" auto-tiering jobs. When a HyperStore source bucket has auto-tiering configured in bridge mode, HyperStore moves objects to the auto-tiering destination immediately after they are uploaded to the source bucket. This cron job is for retrying to move objects for which the original attempt at moving the objects failed. For such objects, the once-every-six-hours retries will continue indefinitely until the objects are successfully moved to the tiering destination system.

For more information on this feature see "["Bridge Mode"](#) (page 55).

Processing of Pending Elasticsearch Update Requests

Scope: One job per HyperStore system (run only in default region)

Frequency: Every hour

Operation invoked: S3 Service internal process `./system/processes/elasticsearch`

All insertions, updates, and deletes of HyperStore object metadata in your Elasticsearch cluster are implemented by this cron job. Until the cron job runs the Elasticsearch update requests are queued in Cassandra. Any requests that fail when this cron job runs are retried at the next running of the cron job.

Note In the event that your Elasticsearch cluster is unavailable for more than a few hours, the request queue will become full and when the cluster is available again you should use the `elasticsearchSync` tool to update the metadata in Elasticsearch. For more information about this tool see "[Using the elasticsearchSync Tool](#)" (page 109). For general information about the Elasticsearch integration feature see "[Elasticsearch Integration for Object Metadata](#)" (page 106).

6.11.2. Scheduled Auto-Repair

On a configurable schedule, the HyperStore "auto-repair" feature automatically checks and if necessary repairs S3 object data in the HyperStore File System as well as metadata in Cassandra. For more information on scheduled auto-repair and other HyperStore mechanisms for ensuring that data in your system is complete and correct, see "**Data Repair Feature Overview**" (page 75).

6.11.3. Cassandra Data Compaction

Cassandra stores data to disk in the form of "Sorted String Tables" (SSTables), a type of append-only commit log. During Cassandra operations, many SSTables may be written to disk for each column family. It's important that SSTables be compacted regularly to minimize the amount of disk space that they use. Compaction merges multiple SSTables into one.

Cassandra regularly implements a "minor" compaction process that occurs **automatically**. This automatic compaction process is sufficient in the context of the HyperStore system. For the HyperStore system, **you do not need perform "major" compactions** using the *nodetool* utility.

You can monitor the compaction process by using JConsole or another JMX client to connect to a Cassandra node's JMX listening port (7199 by default). By accessing the *CompactionManagerMBean* through the JMX console, you can check the progress of any compactions that are currently executing, and also check on the number of completed and pending compactions.

Chapter 7. Disk Operations

7.1. Disabling a HyperStore Data Disk

Subjects covered in this section:

- *Introduction (immediately below)*
- **"The Impact of Disabling a Disk" (page 427)**
- **"Disabling a Disk" (page 427)**

HyperStore supports a method for **temporarily disabling a HyperStore data disk drive** so that you can perform planned maintenance such as a firmware upgrade.

Note If you are **replacing** a disk, follow the instructions for "**Replacing a HyperStore Data Disk**" (page 430) rather than the instructions below.

7.1.1. The Impact of Disabling a Disk

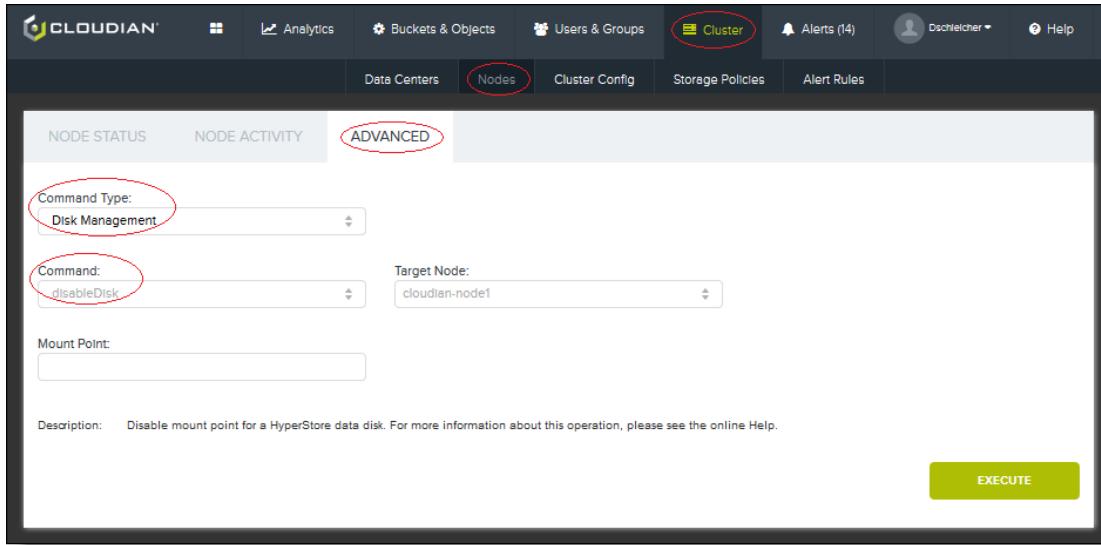
When you execute the HyperStore *disableDisk* function, the system automatically does the following:

- Unmounts the disk's file system, comments out its entry in */etc/fstab*, and marks the disk as unavailable for HyperStore reads and writes.
- Moves the disk's assigned storage tokens to the remaining HyperStore data disks on the same host, in a way that's approximately balanced across those disks. This is a temporary migration of tokens which will be reversed when you later re-enable or replace the disk. While the tokens are on the other disks, writes of new or updated S3 object data that would have gone to the disabled disk will go to the other disks on the host instead.

The existing object data on the disabled disk is **not** recreated on the other disks and therefore that data will be unreadable on the host. Whether the system as a whole can still provide S3 clients with read access to the affected S3 objects depends on the availability of other replicas or erasure coded fragments for those objects, elsewhere within the cluster.

7.1.2. Disabling a Disk

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "disableDisk" command.



2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the disk that you want to disable (for example `/cloudian6`).
3. Click **Execute**.

After the `disableDisk` operation completes, go to the CMC's [Node Status](#) page. In the "Disk Detail Info" section, the device that you disabled should now have this red status icon (indicating that it's disabled and its tokens have been migrated to other disks on the same host):



Later, after completing your maintenance work on the disk, follow the instructions for "[Enabling a HyperStore Data Disk](#)" (page 428). When you re-enable the disk, the tokens that had been moved away from the disk will be moved back to it.

7.2. Enabling a HyperStore Data Disk

Subjects covered in this section:

- *Introduction (immediately below)*
- **"The Impact of Enabling a Disk"** (page 429)
- **"Enabling a Disabled Disk"** (page 429)

HyperStore supports a method for **enabling an existing HyperStore data disk that is currently disabled**. You can tell that a disk is disabled by viewing its status in the "[Disk Detail Info](#)" section of the CMC's **Node Status** page. A disk can go into a disabled state either because you disabled it by using the HyperStore `disableDisk` function (as described in "[Disabling a HyperStore Data Disk](#)" (page 427)) or because the HyperStore system automatically disabled it in response to disk errors (as described in "[Disk Failure Handling Feature Overview](#)" (page 82)).

You can enable a disk if you know that the disk problem was only temporary and that the disk is still healthy enough to use.

Note If you are replacing a disk, follow the instructions for "Replacing a HyperStore Data Disk" (page 430) rather than the instructions below.

7.2.1. The Impact of Enabling a Disk

When you execute the HyperStore `enableDisk` function, the system automatically does the following:

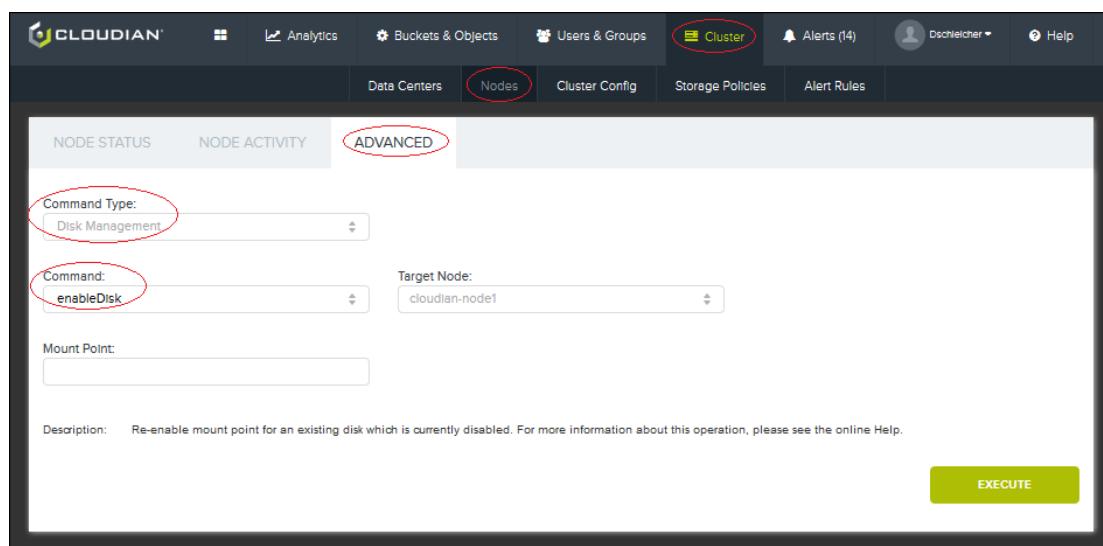
- Remounts the disk (using the same mount point that the disk previously had), uncomments its entry in `/etc/fstab`, and marks the disk as available for HyperStore reads and writes.
- Moves back to the disk the same set of storage tokens that were automatically moved away from the disk when it was disabled.
- Repairs the data on the disk (replacing any object replicas or erasure coded object fragments that should be on the disk but are missing).

After a disk is re-enabled in this way, writes of new or updated S3 object data associated with the returned tokens will go to the re-enabled disk. And the existing object data that was already on the disk will once again be readable. Meanwhile object data that was written to the affected token ranges while the disk was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks. That data will not be moved to the re-enabled disk.

Note For information on how HyperStore tracks token location over time so that objects can be written to and read from the correct disks, see "Dynamic Object Routing" (page 87).

7.2.2. Enabling a Disabled Disk

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "enableDisk" command.



2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the disk that you want to enable.
3. Click **Execute**.

After the `enableDisk` operation completes, go to the CMC's [Node Status](#) page. In the "Disk Detail Info" section, the device that you enabled should now have this green status icon (indicating that its status is OK):



If instead the disk icon is displaying in red (indicating an "Error" status), click the **Clear Error History** button. Doing so should return the disk to OK status.

Note If the CMC continues to show status information for the disk's old device address (as well as a new device address), and if clicking the **Clear Error History** button fails to clear the old information, *ssh* into the node on which you enabled the disk and run the command `systemctl restart clouddian-agent`. Then wait at least one minute, and check the CMC again. If the old information is still displaying, click the **Clear Error History** button again.

Note You do not need to manually run a repair operation after you enable a disk. The system automatically runs *repair* and *repairec* on the disk mount point. You can monitor the repair progress in the [Operation Status](#) page.

7.3. Replacing a HyperStore Data Disk

Subjects covered in this section:

- *Introduction (immediately below)*
- **"The Impact of Replacing a Disk" (page 431)**
- **"Replacing a Disk" (page 431)**

HyperStore supports a method for **activating a replacement HyperStore data disk and restoring data to it**.

This procedure applies to either of these circumstances:

- You are replacing a disk that is currently disabled. You can tell that a disk is disabled by viewing its status in the "[Disk Detail Info](#)" section of the CMC's **Node Status** page. A disk can go into a disabled state either because you disabled it by using the HyperStore *disableDisk* function (as described in "**Disabling a HyperStore Data Disk**" (page 427)) or because the HyperStore system automatically disabled it in response to disk errors (as described in "**Disk Failure Handling Feature Overview**" (page 82)).
- You are replacing a disk that is not currently disabled. In this case, it is not necessary for you to use the *disableDisk* function before replacing the disk. When you pull the disk from the host machine HyperStore will automatically disable the associated mount point, and you can proceed to replace the disk.

After you've pulled the bad disk and physically installed the replacement disk, HyperStore will take care of the rest when you follow the steps in this section.

Note If you have a HyperStore HSA1500, HSA4000, or HSX-4500 series appliance, or a Lenovo Storage DX8200C appliance, and you need assistance physically locating the failed disk that you want to pull and replace, you can use the blink light feature on the CMC **Node Status** page.

Note The procedure below is for HyperStore data disks only. For an OS/Cassandra drive (typically an SSD), see "**Replacing a Cassandra Disk**" (page 432).

7.3.1. The Impact of Replacing a Disk

When you physically install a new disk and then execute the HyperStore `replaceDisk` function, the system automatically does the following:

- Creates a primary partition and an `ext4` file system on the new disk.
- Establishes appropriate permissions on the mount.
- Remounts the new disk (using the same mount point that the prior disk had), uncomments its entry in `/etc/fstab`, and marks the disk as available for HyperStore reads and writes.
- Moves back to the new disk the same set of storage tokens that were automatically moved away from the prior disk when it was disabled.
- Performs a data repair for the new disk (populating the new disk with its correct inventory of object replicas and erasure coded object fragments).

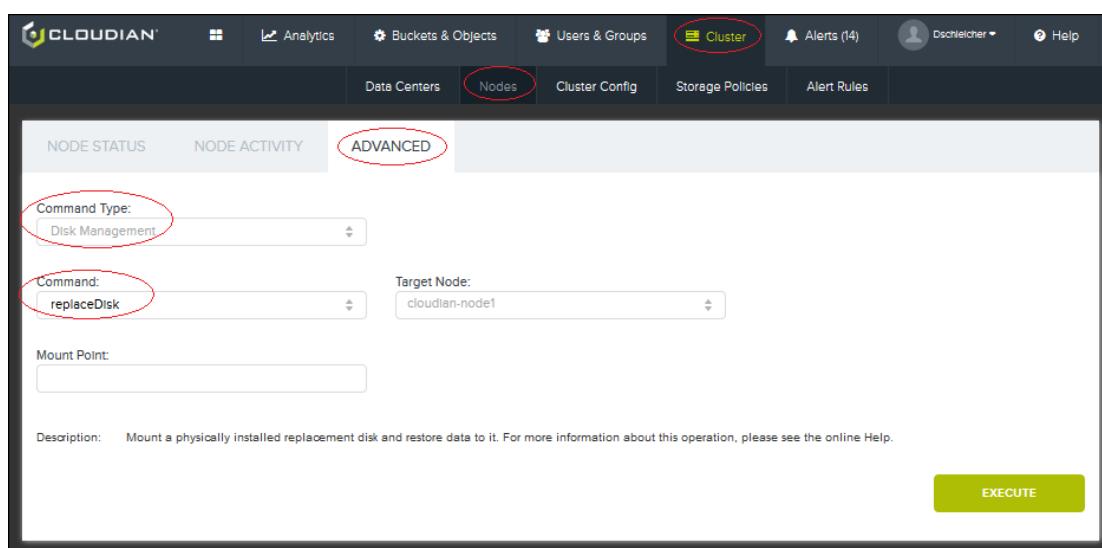
Going forward, writes of new or updated S3 object data associated with the returned tokens will go to the new disk. Meanwhile object data that was written to the affected token ranges while the mount point was disabled — while the tokens were temporarily re-assigned to other disks on the host — will remain on those other disks and will be readable from those disks. That data will not be moved to the new disk.

Note For information on how HyperStore tracks token location over time so that objects can be written to and read from the correct disks, see "**Dynamic Object Routing**" (page 87).

7.3.2. Replacing a Disk

After you've physically installed the replacement disk, follow these steps to activate the replacement disk and restore data to it:

1. In the CMC's **Node Advanced** page, select command type "Disk Management" and then select the "replaceDisk" command.



2. Choose the Target Node (the node on which the disk resides), and enter the Mount Point of the replacement disk. This must be the same as the mount point of the disk that you replaced.
3. Click **Execute**.

After the *replaceDisk* operation completes, go to the CMC's [Node Status](#) page. In the "Disk Detail Info" section, the replacement disk should now have this green status icon (indicating that its status is OK):



If instead the disk icon is displaying in red (indicating an "Error" status), click the **Clear Error History** button. Doing so should return the disk to OK status.

Note If the CMC continues to show status information for the old disk (as well as the new disk), and if clicking the **Clear Error History** button fails to clear the old information, ssh into the node on which you replaced the disk and run the command *systemctl restart clouidian-agent*. Then wait at least one minute, and check the CMC again. If the old information is still displaying, click the **Clear Error History** button again.

Note You do not need to manually run a repair operation after you replace a disk. The system automatically runs *repair* and *repairec* on the disk mount point. You can monitor the repair progress in the [Operation Status](#) page.

7.4. Replacing a Cassandra Disk

Note The procedure that follows is valid for systems on CentOS/RHEL 6.x. For systems on CentOS/RHEL 7.x the procedure that follows is **not** valid. Please contact Cloudian Support if your system is on CentOS/RHEL 7.x and you need to replace a Cassandra disk.

Cloudian, Inc. recommends using RAID mirroring (RAID1) for the drives that store the OS and Cassandra. HyperStore Appliance machines are configured in this way, using software RAID1. If a single drive fails, the other mirrored drive will continue to serve I/O. The recommended procedure for replacing a failed OS/Cassandra drive in the context of software RAID1 is described below.

The procedure described below uses the following sample setup:

- */dev/sda* with partitions */dev/sda1* and */dev/sda2*
- */dev/sdb* with partitions */dev/sdb1* and */dev/sdb2*
- */dev/sda1 + /dev/sdb1 = /dev/md0* (RAID1 array */dev/md0* mounted on */boot*)
- */dev/sda2 + /dev/sdb2 = /dev/md1* (RAID1 array */dev/md1* mounted on */*)

In this example */dev/sda* has failed, and we want to replace it.

1. Identify the failed drive by checking */proc/mdstat*.

```
# cat /proc/mdstat
Personalities : [raid1]
```

```

md0 : active raid1 sdb1[3]
      511936 blocks super 1.0 [2/1] [_U]

md1 : active raid1 sdb2[2]
      468206592 blocks super 1.1 [2/1] [_U]
      bitmap: 3/4 pages [12KB], 65536KB chunk

unused devices: <none>

```

To identify the status of the RAID array, look at the string containing [UU]. Each "U" is a representation of a healthy partition member of the array.

From the above output, you can tell that RAID arrays *md0* and *md1* are missing a "U" [_U]. This means that they are degraded and faulty. (For an example of the output for healthy arrays, look ahead to Step 4-d of this procedure.)

2. Remove the failed drive.

- a. If the drive is not in the failed state yet, fail the partitions belonging to the drive:

```

# mdadm --manage /dev/md0 --fail /dev/sda1
# mdadm --manage /dev/md1 --fail /dev/sda2

```

- b. Remove the partitions from the RAID array:

```

# mdadm --manage /dev/md0 --remove /dev/sda1
# mdadm --manage /dev/md1 --remove /dev/sda2

```

3. Physically replace the failed drive with a new drive of equal or greater capacity.

4. Add the new drive to the RAID array.

- a. Create the exact same partitioning on the new drive.

```
# sfdisk -d /dev/sdb | sfdisk --force /dev/sda
```

Note */dev/sda* is the newly replaced drive.

- b. Check that both drives have the same partition:

```

# fdisk -l /dev/sdb

# fdisk -l /dev/sda

```

- c. Add the new partitions to the RAID array:

```

# mdadm --manage /dev/md0 --add /dev/sda1
# mdadm --manage /dev/md1 --add /dev/sda2

```

- d. Wait for the arrays to be synchronized. You can check status of synchronization by executing:

```
# cat /proc/mdstat
```

The output should look like this:

```

Personalities : [raid1]
md0 : active raid1 sda1[2] sdb1[3]

```

```
      511936 blocks super 1.0 [2/2] [UU]

md1 : active raid1 sda2[0] sdb2[2]
      468206592 blocks super 1.1 [2/1] [_U]
      [======>.....] recovery = 55.5% (260239360/468206592)
      finish=13.4min speed=258048K/sec
      bitmap: 2/4 pages [8KB], 65536KB chunk

unused devices: <none>
```

Once synchronization is complete, it will look like the output below:

```
Personalities : [raid1]
md0 : active raid1 sda1[2] sdb1[3]
511936 blocks super 1.0 [2/2] [UU]

md1 : active raid1 sda2[0] sdb2[2]
      468206592 blocks super 1.1 [2/2] [UU]
      bitmap: 0/4 pages [0KB], 65536KB chunk

unused devices: <none>
```

5. Install [GRUB bootloader](#) on the new hard drive.

- Check if GRUB bootloader exists on the mirrored drives:

```
# file -s /dev/sdb

/dev/sdb: x86 boot sector; GRand Unified Bootloader, stage1 version 0x3,
stage2 address 0x2000,
stage2 segment 0x200, GRUB version 0.94; partition 1: ID=0xfd, active,
starthead 32,
startsector 2048, 1024000 sectors; partition 2: ID=0xfd, starthead 221,
startsector 1026048,
936675328 sectors, code offset 0x48

# file -s /dev/sda

/dev/sda: x86 boot sector; partition 1: ID=0xfd, active, starthead 32,
startsector 2048,
1024000 sectors; partition 2: ID=0xfd, starthead 221, startsector 1026048,
936675328 sectors,
code offset 0x0
```

From the above outputs, it looks like GRUB is only installed on `/dev/sdb`. It also needs to be installed on `/dev/sda` to ensure that the system will boot regardless of which drive fails in the future.

- Check the device maps.

```
# cat /boot/grub/device.map
```

The output will look like this:

```
# this device map was generated by anaconda
(hd0)      /dev/sda
(hd1)      /dev/sdb
```

c. Locate GRUB setup files.

Enter the grub command line by executing:

```
# grub
```

At the *grub>* prompt, type *find /grub/stage1*

For example:

```
grub> find /grub/stage1
      find /grub/stage1
      (hd0,0)
      (hd1,0)
```

d. Install GRUB on */dev/sda*.

First:

```
grub> root (hd0,0)
```

You will see the following output:

```
root (hd0,0)
Filesystem type is ext2fs, partition type 0xfd
```

Then:

```
grub> setup (hd0)
```

You will see the following output:

```
setup (hd0)
  Checking if "/boot/grub/stage1" exists... no
  Checking if "/grub/stage1" exists... yes
  Checking if "/grub/stage2" exists... yes
  Checking if "/grub/e2fs_stage1_5" exists... yes
  Running "embed /grub/e2fs_stage1_5 (hd0)"... 27 sectors are
  embedded.
succeeded
  Running "install /grub/stage1 (hd0) (hd0,0)/grub/stage2
  /grub/grub.conf"...
succeeded
Done.
```

Then:

```
grub> quit
```

e. Check that the GRUB bootloader is installed on */dev/sda*.

```
# file -s /dev/sda
/dev/sda: x86 boot sector; GRand Unified Bootloader, stage1 version 0x3,
stage2 address 0x2000,
stage2 segment 0x200, GRUB version 0.94; partition 1: ID=0xfd, active,
```

```
starthead 32, startsector 2048,  
1024000 sectors; partition 2: ID=0xfd, starthead 221, startsector 1026048,  
936675328 sectors,  
code offset 0x48
```

This completes the process of replacing a mirrored OS/Cassandra drive.

7.5. Responding to Data Disks Nearing Capacity

HyperStore implements an [automatic mechanism for helping to ensure balanced disk usage](#) among the disks on a host. However, if service utilization is heavy for the size of your cluster, there may be times when one or more disks nears their capacity.

Note For guidance about HyperStore capacity management and cluster resizing, see "[Cluster Resizing Feature Overview](#)" (page 68).

If an individual disk or a node as a whole is running low on available capacity, HyperStore alerts administrators:

- Alerts are triggered if an individual disk drops below 15% available capacity or if a node as a whole drops below 10% available capacity. When such alerts are triggered, they appear in the CMC's [Node Status](#) page and [Alerts](#) page as well as being sent to the system administrator email address(es). Optionally, alerts can also be transmitted as SNMP traps. Alert thresholds and options are configurable in the [Alerts Rules](#) page.
- If a node as a whole reaches 80% utilization of its data disk capacity, a Warning is displayed on the CMC's [Dashboard](#) page.

If a **HyperStore data disk** (a disk storing S3 object data) is nearing capacity, the first two things to try are:

- Use [hsstool cleanup](#) (and [hsstool cleanupec](#)) if you use erasure coding in your system) on the node to clear it of any data that's no longer supposed to be there. Such "garbage data" may be present if, for example, S3 objects have been deleted from the system as a whole but the deletion operations on the node in question failed.
- Delete S3 objects. Note that the associated files will not be deleted from the disk immediately since HyperStore uses batch processing for deletion of S3 object data. The batch processing is triggered hourly by a cron job (see "[Object Deletion Queue Processing](#)" (page 422)).

Note: For additional guidance on removing data to free up disk space, consult with Cloudian Support.

If these measures do not free up sufficient disk space, the solution is to increase system capacity by adding one or more new nodes to your cluster. For the procedure see "[Adding Nodes](#)" (page 369). When you add a node, a portion of the data on your existing nodes is copied to the new node and then (when you subsequently run a cleanup operation) deleted from the existing nodes — thereby freeing up space on the existing nodes. The degree to which space will be freed up on existing nodes depends on the number of new nodes that you add in proportion to the size of your existing cluster — for example, adding two nodes to a four node cluster would free up a larger percentage of the existing nodes' disk space than adding two nodes to a twenty node cluster.

For information about managing a **Cassandra data disk** (a disk storing system and object metadata stored in Cassandra) that is nearing capacity see "**Responding to Cassandra Disks Nearing Capacity**" (page 437).

7.6. Responding to Cassandra Disks Nearing Capacity

If a Cassandra data drive (a disk or SSD storing the Cassandra database) is nearing capacity, the first two things to try are:

- Use [**hsstool cleanup**](#) on the host node, using the *allkeyspaces* option. This will clear any Cassandra data that is no longer supposed to be on the node.
- Selectively delete Cassandra data. To do this, consult with Cloudian Support.

If these measures do not free up sufficient space, the solution is to increase system capacity by adding one or more new nodes. For the procedure see "**Adding Nodes**" (page 369). When you add a node, a portion of the Cassandra data on your existing nodes is copied to the new node and then (when you subsequently run a cleanup operation) deleted from the existing nodes — thereby freeing up space on the existing nodes.

7.7. Adding Disks is Not Supported

HyperStore does not support adding disks to an existing node within the cluster. To increase cluster storage capacity, the only supported method is to add one or more nodes as described in "**Adding Nodes**" (page 369).

Note For guidance about HyperStore capacity management and cluster resizing, see "**Cluster Resizing Feature Overview**" (page 68).

This page left intentionally blank

Chapter 8. System Monitoring

8.1. Using the CMC to Monitor Your HyperStore System

The Cloudian Management Console (CMC) provides extensive support for monitoring the health and performance of your HyperStore system. The CMC also supports a configurable mechanism for alerting administrators when system events occur.

The table below shows the system monitoring and alert management tasks that are supported by the CMC.

Category	Tasks	CMC Page
System Monitoring	Check high level status information for each service region, including current storage capacity usage and remaining available capacity, project capacity usage for the next 120 days, recent S3 transaction performance, and whether there are any system alerts	Dashboard
	Check the remaining available storage capacity in each service region, as well as capacity remaining in each data center and on each node	Capacity Explorer
	Check how your storage capacity usage has changed over the past 30 days, per service region	Cluster Usage
	For each data center, see which nodes have any alerts and whether any HyperStore services are down on any node	Data Centers
	Check a specific node's storage capacity usage and remaining available capacity, CPU and memory usage, recent S3 transaction performance, and HyperStore services status. Also check capacity usage and status information for individual disks on a node.	Node Status
	Check a specific node's performance trends over the past 30 days, for metrics such as CPU utilization, disk read and write throughput, and S3 transactions per second.	Node Activity
System Alerts Management	Configure system alert rules, for having the system automatically notify administrators regarding system events. Also view pre-configured alert rules that come with the system.	Alert Rules
	Review and acknowledge alerts generated by any node in the system	Alerts
	Review and acknowledge alerts generated by a specific node	Node Status

8.2. Additional Monitoring Tools

8.2.1. Using the Admin API to Monitor HyperStore

The HyperStore Admin API supports methods for monitoring HyperStore health and performance. The CMC invokes these Admin API methods in implementing the CMC's system monitoring functions. If you wish you can invoke these Admin API methods directly, through a client application of your own making or through third party command line tools that enable you to construct HTTP requests.

For more information see the Admin API methods associated with the "**monitor**" (page 735) resource.

8.2.2. Doing an HTTP Health Check

The HyperStore system supports a health check that lets you quickly determine whether certain HTTP interfaces are responsive on particular hosts. This function is supported for:

- **S3 Service** (HTTP port 80 by default; or HTTPS port 443 if you've enabled SSL for the S3 service)
- **Admin Service** (HTTPS port 19443 by default; or HTTP port 18081 if you've configured the Admin Service to accept regular HTTP connections)
- **HyperStore Service** (HTTP port 19090 by default)

To do the health check on a particular host, submit an HTTP(S) HEAD request to `<host>:<port>/healthCheck`. If the service is up and running and listening on its assigned port, you will receive back an HTTP 200 OK status. If not, your request will time out.

For the **CMC**, you can check responsiveness by submitting a HEAD request to the CMC login URL (`https://localhost:8443/Cloudian/login.htm`).

The example below shows a health check of an S3 Service instance that is responsive.

```
HEAD http://192.168.2.16:80/.healthCheck
Status Code: 200 OK
Content-Length: 0
Date: Wed, 14 Aug 2019 12:51:50 GMT
Server: CloudianS3
```

Note To do health checks against an HTTPS port, the client executing the check must support TLS/SSL. Note also that health checks against HTTPS ports are a more expensive operation (in terms of resource utilization) than health checks against HTTP ports.

8.2.3. Using JMX to Monitor Java-Based HyperStore Services

IMPORTANT: Cloudian recommends that you do not use JMX for monitoring a HyperStore production system, as it may negatively impact performance (particularly if you run JConsole on one of your production nodes). However, JMX may be useful for monitoring a HyperStore testing or evaluation system.

The S3 Service, Admin Service, HyperStore Service, and Cassandra Service support monitoring via Java Management Extensions (JMX). You can access JMX statistics using the graphical JMX client JConsole, which

comes with your Java platform. By default the full path to the JConsole executable is `/usr/java/latest/bin/jconsole`.

After launching JConsole, in the JConsole GUI specify the `<host>:<jmx-port>` that you want to connect to. Each of the HyperStore system's Java-based services has a different JMX listening port as indicated in the sections that follow. **The statistics that you view will be only for the particular node to which you are connected via JConsole.**

Note By default a JConsole connection does not require a user name and password, so in the JConsole GUI these fields can be left empty. For general information about using JConsole, including password protection and security options, see the JConsole Help.

Note This section on JMX statistics presumes that you are using JConsole, but there are other JMX clients available. Your HyperStore system comes with two command-line JMX clients — `cmdline-jmx-client` and `jmxterm` — which are in the `/opt/cloudian/tools` directory.

S3 Service JMX Statistics

Default JMX port: 19080

For the S3 Service, these categories of JMX statistics are supported:

S3 operation timing and rate stats

In JConsole's MBeans tab, timing performance statistics for S3 operations are available under the `metrics` MBean. Under `metrics` there is `com.cloudian.s3.stats.<operation>`, where `<operation>` is an S3 API operation such as `putObject`, `getObject`, `deleteObject`, `getBucket`, and so on. For each operation type, under `Attributes` there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the S3 Service. Statistics will only be available for operations that have been performed since the last S3 Service restart — for example, if no `deleteObject` operations have been performed since the last restart, then no `deleteObject` statistics will be available.

Note These timing and rates stats are implemented with the Metrics Core library.

S3 operation success stats

In JConsole's MBeans tab, success rate statistics for S3 operations are available under `com.gemini.cloudian.s3 → Accounting → Success Rate → Attributes`. For each S3 operation, the success rate is

expressed as a double between 0 and 1 indicating the percentage of successful processing for that S3 operation. The success rate statistics are cumulative since the last start of the S3 Service on the node to which you are connected.

For example, the statistic "DeleteBucket" would have a value such as 1.0 or 0.92, indicating a 100% or 92% success rate for S3 "DELETE Bucket" operations since the last start-up of the S3 Service.

S3 Service's Cassandra client stats

For its client interface to Cassandra, the S3 Service leverages open source Hector technology. In JConsole's MBeans tab, Hector statistics are available under `me.prettyprint.cassandra.service_Cloudian<regionName> → hector → hector → Attributes`. Descriptions of these statistics are available in the [Health Check Attributes Available for Hector](#) section of the online Hector user guide.

The list of supported statistics is below.

- ExhaustedPoolNames
- KnownHosts
- NumActive
- NumBlockedThreads
- NumConnectionErrors
- NumExhaustedPools
- NumIdleConnections
- NumPoolExhaustedEventCount
- NumPools
- NumRenewedIdleConnections
- NumRenewedTooLongConnections
- ReadFail
- RecoverableErrorCount
- RecoverableLoadBalancedConnectErrors
- RecoverableTimedOutCount
- RecoverableTransportExceptionCount
- RecoverableUnavailableCount
- SkipHostSuccess
- StatisticsPerPool
- SuspendedCassandraHosts
- WriteFail

HTTP server thread pool stats

For serving HTTP requests, the S3 Service leverages open source Jetty technology. In JConsole's MBeans tab, Jetty thread pool statistics are available under `org.eclipse.jetty.util.thread → queuedthreadpool → 0 → Attributes`. The statistics available are:

- threads
- idleThreads
- queueSize

Admin Service JMX Statistics

Default JMX port: 19081

In JConsole's MBeans tab, timing performance statistics for Admin Service operations are available under the *metrics* MBean. Under *metrics* there is *com.cloudian.admin.stats.<operation>*, where *<operation>* is an S3 API operation such as *getUser*, *createUser*, and so on. For each operation type, under *Attributes* there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the S3 Service (the Admin Service stops and starts together with the S3 Service). Statistics will only be available for operations that have been performed since the last S3 Service restart — for example, if no *createUser* operations have been performed since the last restart, then no *createUser* statistics will be available.

Note These timing and rates stats are implemented with the Metrics Core library.

HyperStore Service JMX Statistics

Default JMX port: 19082

For the HyperStore Service, these categories of JMX statistics are supported:

HyperStore operation timing and rate stats

In JConsole's MBeans tab, timing performance statistics for HyperStore Service operations are available under the *metrics* MBean. Under *metrics* there is *com.cloudian.hybrid.stats.<operation>*, where *<operation>* is a HyperStore Service operation such as *put*, *getBlob*, *getDigest*, or *delete*. For each operation type, under *Attributes* there is a set of timing statistics including:

- Count — The total number of executions that were timed.
- Max — The maximum value in milliseconds of the logged execution times.
- Mean — The mean execution time, in milliseconds.
- Min — The minimum value in milliseconds of the logged execution times.
- StdDev — The standard deviation, in milliseconds.

For each operation type there are also rate stats including:

- MeanRate — Average transactions-per-second (TPS) since last restart.
- FifteenMinuteRate — 15 minute exponentially weighted moving average rate for TPS.

The statistics are initialized at each restart of the HyperStore Service. Statistics will only be available for operations that have been performed since the last HyperStore Service restart — for example, if no *delete* operations have been performed since the last restart, then no *delete* statistics will be available.

Note These timing and rates stats are implemented with the Metrics Core library.

HyperStore operation success stats

In JConsole's MBeans tab, HyperStore file system operations statistics are available under *com.gemini.cloudian.hybrid.server* → *HyperstoreOperationsStats* → *Attributes*. The statistics available are:

- *NumberOfSuccessfulDeleteOperations*
- *NumberOfSuccessfulReadOperations*
- *NumberOfSuccessfulWriteOperations*
- *TotalNumberOfDeleteOperations*
- *TotalNumberOfReadOperations*
- *TotalNumberOfWriteOperations*

Cassandra client stats

For its client interface to Cassandra, the HyperStore Service leverages open source Hector technology. In JConsole's MBeans tab, Hector statistics are available under *me.prettyprint.cassandra.service_Cloudian<regionName>* → *hector* → *hector* → *Attributes*.

The list of supported statistics is the same as indicated for the [S3 Service's Hector statistics](#). Descriptions of these statistics are available in the [Health Check Attributes Available](#) for Hector section of the online Hector user guide.

HTTP server thread pool stats

For serving HTTP requests, the HyperStore Service leverages open source Jetty technology. In JConsole's MBeans tab, Jetty thread pool statistics are available under *org.eclipse.jetty.util.thread* → *queuedthreadpool* → *0* → *Attributes*. The statistics available are:

- *threads*
- *idleThreads*
- *queueSize*

Cassandra Service JMX Statistics

Default JMX port: 7199

In JConsole's MBeans tab, under *org.apache.cassandra.metrics*, Cassandra supports a wide range of statistics.

The *Compaction* MBean exposes statistics including:

- Number of completed compactions.
- Number of pending compaction tasks.
- Progress of currently running compactions.

Note If the number of pending compaction tasks grows over time, this is an indicator of a need to increase cluster capacity.

The *ColumnFamily* MBean exposes statistics for active, pending, and completed tasks for each of Cassandra's thread pools. This is the same information as is available through the [nodetool tpstats](#) command.

Note A significant and sustained increase in the pending task counts for the Cassandra thread pools is an indicator of a need to increase cluster capacity.

The *ColumnFamily* MBean exposes individual column family statistics including:

- Memtable data size
- Number of live SSTables
- Average read latency
- Average write latency

Note A sustained increase in read and write latencies may indicate a need to increase cluster capacity.

8.2.4. Using Native Linux Utilities for System Resource Monitoring

The Linux OS on which the HyperStore system runs includes several useful commands for checking on system resource utilization on individual host machines. For example, the *top* command returns a summary of host-wide resource utilization as well as a breakdown of resource usage per process. Using *top -c* (which returns more information in the "Command" column than *top* alone) makes it easier to distinguish between the several Java-based HyperStore processes that will be running on each host — including Cassandra, the S3 Service (*cloudian_s3*), the Admin Service (*cloudian_admin*), the HyperStore Service (*storage_s3*), and the CMC (*tomcat*).

Important statistics are the memory usage and CPU usage:

- VIRT: Virtual memory.
- RES: Resident memory. VIRT minus RES is the amount of memory swapped out.
- %CPU: Percentage of CPU used

Other useful Linux utilities for monitoring system resource usage include:

- *vmstat*
- *iostat*
- *dstat*

8.2.5. Using nodetool to Monitor Cassandra

Through the CMC's [Node Status](#) page you can monitor high-level status information for the Cassandra service instance on a particular node. For more granular Cassandra monitoring you can optionally use the native Cassandra utility *nodetool*.

The *nodetool* utility resides in each Cassandra host's */opt/cassandra/bin* directory. From that directory, the general syntax for *nodetool* is:

```
$ ./nodetool -h <host> [-p <CassandraJMXport>] <COMMAND>
```

The Cassandra JMX port defaults to 7199.

Some *nodetool* commands that you may find useful for monitoring a Cassandra cluster are summarized below.

cfstats [<Keyspace.ColumnFamily>]

Returns information about each keyspace and each column family, including:

- Read count
- Read latency
- Write count
- Write latency
- Pending tasks

For column families only (not keyspaces):

- Memtable stats
- Key cache capacity
- Key cache hit rate

cflhistograms <Keyspace> <ColumnFamily>

Returns information for a specific column family, including:

- Read latency
- Write latency
- Row size
- Column count

tpstats

Returns information for each thread pool, including:

- Active tasks
- Completed tasks
- Pending tasks

Note: A significant and sustained increase in the pending task counts for the Cassandra thread pools is an indicator of a need to increase cluster capacity.

compactstats

Returns information for an in-progress compaction, including:

- Compaction type (major or minor)
- Column family for which the compaction is being performed
- Bytes compacted so far

netstats

Returns network information, including:

- Status of streaming operations such as bootstrap, repair, move, or decommission
- Active, pending, and completed command counts

For more information about *nodetool*, see either the Apache Cassandra online documentation or the DataStax Cassandra online documentation.

8.2.6. Using the Redis CLI to Monitor Redis

Through the CMC's [Node Status](#) page you can monitor high-level status information for the Redis service (the Redis Credentials master instance or slave instance or the Redis QoS master instance or slave instance) on a particular node. For more granular Redis monitoring you can optionally use the native Redis CLI.

To launch the Redis CLI on a Redis node, change into the node's */opt/redis* directory and do either of the following:

```
# to connect to the CLI for the Redis Credentials DB:  
$ ./redis-cli  
redis 127.0.0.1:6379>  
  
# to connect to the CLI for the Redis QoS DB:  
$ ./redis-cli -p 6380  
redis 127.0.0.1:6380>
```

Once you're in the Redis CLI mode, you can issue commands. Use "quit" to exit the CLI.

With the Redis [INFO command](#) you can retrieve statistics such as:

- Uptime since last start
- Memory usage as allocated by Redis
- Memory usage as seen by OS
- Memory fragmentation ratio
- Number of dataset-changing operations processed since last save
- Keyspace hits
- Keyspace misses
- Total keys
- Total expired keys

You can reset Redis statistics with the [CONFIG RESETSTAT command](#).

To check just on the number of keys currently in the database, you can use the [DBSIZE command](#).

Also useful is the [MONITOR command](#), which enables you to monitor the complete sequence of commands received by the DB in near real-time.

In addition to monitoring Redis through its CLI, you should also monitor the size of the Redis data files (in default directory */var/lib/redis/*) to ensure files are not growing excessively.

This page left intentionally blank

Chapter 9. System Configuration

9.1. CMC's Configuration Settings Page

Through the CMC's **Configuration Settings** page you can dynamically change a variety of HyperStore system configuration settings. For detail see "**Configuration Settings**" (page 293).

9.2. Installer Advanced Configuration Options

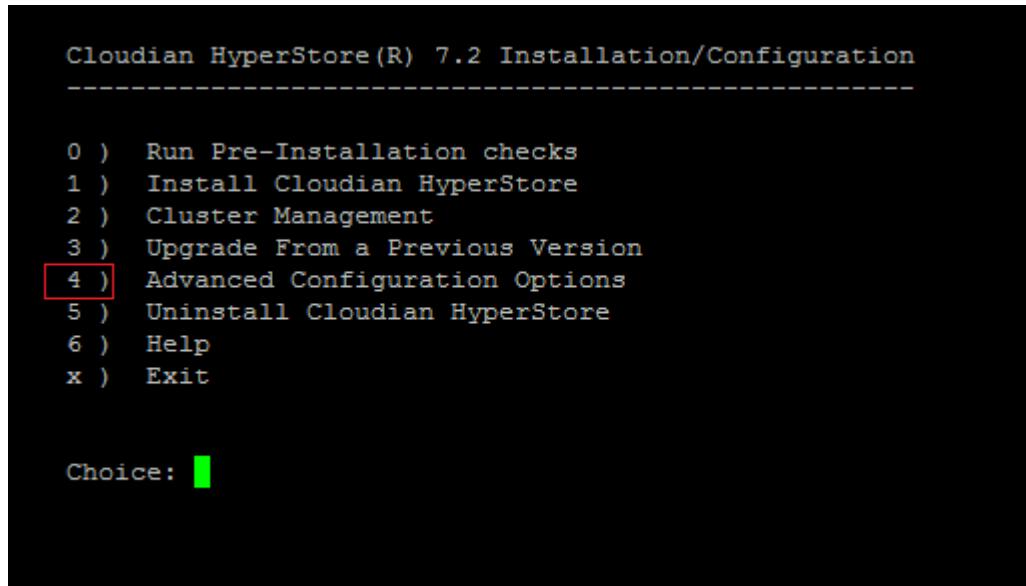
The HyperStore installation tool supports several types of advanced system configurations which can be implemented at any time after initial installation of the system. In most cases, these are configuration tasks that can also be performed manually by editing system configuration files. The installer makes these tasks easier by enabling you to perform the configurations by responding to prompts from the installer.

Note As a best practice, you should complete basic HyperStore installation first and confirm that things are working properly (by running the installer's Validation Tests, under the "Cluster Management" menu) before you consider using the advanced configuration options to make changes such as customizing port assignments or implementing SSL for the S3 Service.

To access the advanced configuration options, on your Puppet master node change into your installation staging directory and launch the installer.

```
[7.2]# ./cloudianInstall.sh
```

At the installer main menu's Choice prompt enter **4** for Advanced Configuration Options.



```
Cloudian HyperStore(R) 7.2 Installation/Configuration
-----
0 ) Run Pre-Installation checks
1 ) Install Cloudian HyperStore
2 ) Cluster Management
3 ) Upgrade From a Previous Version
4 ) Advanced Configuration Options
5 ) Uninstall Cloudian HyperStore
6 ) Help
x ) Exit

Choice: [
```

This opens the "Advanced Configuration Options" sub-menu.

```
Advanced Configuration Options
-----
a ) Change server role assignments
b ) Change S3, Admin and CMC ports
c ) Change S3, S3-Website, Admin, or CMC endpoints
d ) Configure diagnostic data collection options
e ) Configure SSL for S3
f ) Configure SSL for CMC
g ) Configure SSL for IAM
h ) Remove existing Puppet SSL certificates
i ) Start or stop Puppet daemon
j ) Remove Puppet access lock
k ) Enable or disable DNSMASQ
l ) Configure Performance Parameters on Nodes
m ) Disable the root password
r ) Exclude host(s) from configuration push and service restarts
s ) Configure Firewall
x ) Return to Main Menu

Choice: [ ]
```

From this menu you can choose the type of configuration change that you want to make and then proceed through the interactive prompts to specify your desired settings.

a) Change server role assignments

This is for shifting role assignments — such as the Redis QoS master role or Redis Credentials master role — from one of the hosts in your cluster to another. For the complete procedures, see "**Change Node Role Assignments**" (page 405).

b) Change S3, Admin or CMC ports

This lets you interactively change listening ports for the S3, Admin, and CMC services. For instructions see "**Changing S3, Admin, or CMC Listening Ports**" (page 541).

c) Change S3, S3-Website, Admin, or CMC endpoints

This lets you interactively change the S3 service endpoint, S3 static website endpoint, Admin service endpoint, or CMC endpoint. For instructions see "**Changing S3, Admin, or CMC Service Endpoints**" (page 542).

d) Configure diagnostic data collection options

This lets you interactively configure Phone Home (Smart Support) settings.

1. After selecting this option from the Advanced Configuration Options menu, follow the prompts to specify your desired settings. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value. The final prompt will ask whether you want to save your changes -- type yes to do so.

2. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
3. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service.

Generate a self-signed certificate in a JKS keystore

This generates a self-signed certificate that you can use for implementing TLS/SSL for the S3 Service. You can also use this option to generate a self-signed certificate and then submit it to a CA for signing. For instructions on using this option, see either of these procedures, depending on what type of certificate you want to use:

- "**TLS/SSL for S3 Service (Self-Signed Certificate)**" (page 543)
- "**TLS/SSL for S3 Service (CA-Verified Certificate)**" (page 547)

Enable and configure HTTPS access on S3 server

This configures the S3 Service to use HTTPS, using a specified TLS/SSL keystore. For instructions on using this option, see either of these procedures, depending on what type of certificate you want to use:

- "**TLS/SSL for S3 Service (Self-Signed Certificate)**" (page 543)
- "**TLS/SSL for S3 Service (CA-Verified Certificate)**" (page 547)

Import Java keystore to CMC

This enables you to use a specified TLS/SSL keystore for the CMC server rather than its default keystore. For instructions on using this option, see either of these procedures, depending on what type of certificate you want to use:

- "**TLS/SSL for the CMC (Self-Signed Certificate)**" (page 553)
- "**TLS/SSL for the CMC (CA-Verified Certificate)**" (page 556)

Generate a self-signed certificate for IAM in a JKS keystore

This generates a self-signed certificate that you can use for implementing TLS/SSL for the IAM Service. You can also use this option to generate a self-signed certificate and then submit it to a CA for signing. For instructions on using this option, see either of these procedures, depending on what type of certificate you want to use:

- "**TLS/SSL for IAM Service (Self-Signed Certificate)**" (page 561)
- "**TLS/SSL for IAM Service (CA-Verified Certificate)**" (page 563)

Enable and configure HTTPS access for IAM

This configures the IAM Service to use HTTPS, using a specified TLS/SSL keystore. For instructions on using this option, see either of these procedures, depending on what type of certificate you want to use:

- "**TLS/SSL for IAM Service (Self-Signed Certificate)**" (page 561)
- "**TLS/SSL for IAM Service (CA-Verified Certificate)**" (page 563)

h) Remove existing Puppet SSL certificates

This is a troubleshooting measure in the event of Puppet connectivity problems, as described in [Installation Troubleshooting](#).

This operation is specifically in regard to Puppet certificates and has nothing to do with SSL for the S3 Service.

i) Start or stop Puppet daemon

By default, after HyperStore installation the Puppet master and agent daemons are left running, and every 10 minutes the agents check the master to see if there are updated HyperStore configuration settings to download to the agent nodes. Alternatively you can use this option from the Advanced menu to stop the Puppet agent daemons.

Note that if you don't leave the Puppet agent daemons running, you must remember to trigger a one-time Puppet sync-up each time you make a HyperStore configuration change on the Puppet master. By contrast, if you leave the Puppet agent daemons running, a sync-up will happen automatically every 10 minutes even if you don't specifically trigger a sync-up after making a configuration change.

If you ever want to check on the current status of your Puppet master and agent daemons, you can do so by choosing "Cluster Management" from the installer's main menu; then choose "Manage Services"; then in the Service Management sub-menu choose "Puppet service (status only)".

j) Remove Puppet access lock

When Puppet is performing a sync-up, the system temporarily locks Puppet access (so that no additional Puppet instances are launched by other operators or systems). If the sync-up operation is terminated unexpectedly — such as by a <CTRL>-c command, or a Puppet master node shutdown — the temporary lock may fail to release. This unreleased lock would prevent any subsequent sync-ups from being implemented.

You can clear the lock by running the "Remove Puppet access lock" operation.

Afterwards, to confirm that the lock is cleared you can check to make sure that no */tmp/cloudian.installer.lock* directory exists on the Puppet master node.

k) Enable or disable DNSMASQ

This menu option is for either of these circumstances:

- When you ran the *cloudianInstall.sh* script to install your HyperStore system, you had the script automatically install and configure [dnsmasq](#) to perform DNS resolution for HyperStore service domains. (That is, you used the *configure-dnsmasq* option when you launched the install script.) However, now you want to use your own DNS solution for resolving HyperStore domains, and you've completed your DNS configuration as described in DNS Set-Up. You can use the installer's "Enable or disable DNSMASQ" menu option to disable dnsmasq. This stops *dnsmasq* on all HyperStore nodes and disables *dnsmasq* by configuration.
- When you ran the *cloudianInstall.sh* script to install your HyperStore system, you did **not** have it install *dnsmasq* (the default installer behavior is not to install *dnsmasq*). However, now you want to use *dnsmasq* for HyperStore domain resolution. With the installer's "Enable or disable DNSMASQ" menu option you can enable *dnsmasq*:
 1. After selecting this option from the Advanced Configuration Options menu, follow the prompts to choose to enable *dnsmasq*.
 2. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
 3. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the DNSMASQ service.

l) Configure Performance Parameters on Nodes

This lets you run a HyperStore performance configuration optimization script. The script runs automatically

when you install your cluster and when you add nodes, so under normal circumstances you should not need to use this installer Advanced Configuration option. For instructions on using the option see "**Tuning HyperStore Performance Parameters**" (page 568).

r) Exclude host(s) from configuration push and service restarts

Use this option if you want to temporarily exclude a particular node or nodes from installer-driven configuration pushes and service restarts. One scenario where it would be appropriate to do this is if you have a node that's down and can't be brought back up anytime soon, and in the interim you want to make a configuration change to your system. With a node down, the installer's "Cluster Management" → "Push Configuration Settings to Cluster" function will fail if you try to push to the whole cluster. So before doing a push, use the "Advanced Configuration Options" → "Exclude host(s) from configuration push and service restarts" to specify the down node. Then, when you subsequently do a configuration push to the cluster, the installer will automatically exclude that node and the push to the cluster will succeed.

The excluded host will also be excluded when you use the installer's "Cluster Management" → "Manage Services" menu to stop, start, or restart particular services in the cluster (such as the S3 Service or the Cassandra Service).

Note This "exclude from configuration push and service restarts" status is different than "maintenance mode" (see "**Start Maintenance Mode**" (page 285)). The "excluded" status merely excludes a node from the list of nodes that the installer uses when it pushes out configuration changes to the cluster or restarts services across the cluster; it has no effect other than that. By contrast, when you put a node into "maintenance mode" the system stops sending S3 requests to the node and stops collecting alerts from the node.

If you put a node into "excluded" status, and you later exit the installer, then if you subsequently launch the installer again it will display a message indicating that there is a node in excluded status. In the sample below the node "cloudian-node6" is in this status.

```
[7.2]# ./cloudianInstall.sh
The following nodes have been excluded for configuration updates and service
restarts: cloudian-node6
Press any key to continue ...
```

When the node is back up again and you are ready to have it again be eligible for installer-managed configuration pushes and service restarts, return to the "Advanced Configuration Options" → "Exclude host(s) from configuration push and service restarts" function and enter "none" at the prompt.

If you made a system configuration change when a node was down and excluded, then after the node is back up and you've taken the node out of excluded status, do a configuration push and a service restart (whichever service restart is appropriate to the configuration change you made). This will bring the node's configuration up to date with the rest of the cluster.

Note There are a small number of circumstances where the system will **automatically** place a down node into the "excluded" status. One such circumstance is when the system is executing automatic fail-over of the system cronjob host role, in the event that the primary cronjob host goes down. The next time that you launch the installer it will display a message identifying the node that's in "excluded" status.

s) Configure Firewall

This lets you enable and configure the built-in HyperStore firewall, on all the HyperStore nodes in your system. For instructions see "[Configuring the HyperStore Firewall](#)" (page 569).

9.3. Puppet Overview

During HyperStore installation, the open source version of the cluster configuration management application [Puppet](#) is automatically installed and set up. The Puppet framework consists of:

- A **Puppet master node** on which all the HyperStore configuration templates reside. This is one of your HyperStore nodes -- specifically, the node on which you ran the HyperStore installation script.
- A **Puppet agent on every node** in your HyperStore system (including the node on which the master is running).

The Puppet system enables you to edit HyperStore configuration templates in one location — on the Puppet master node — and have those changes propagate to all the nodes in your HyperStore cluster, even across multiple data centers and multiple service regions. There are two options for implementing Puppet sync-up: you can trigger an immediate sync-up by using the HyperStore installer, or you can wait for an automatic sync-up which by default occurs on a 10 minute interval. With either approach, after the sync-up **you must restart the affected service(s) to apply the configuration changes**.

For more information see "[Pushing Configuration File Edits to the Cluster and Restarting Services](#)" (page 454).

IMPORTANT: Do not directly edit configuration files on individual HyperStore nodes. If you make edits on an individual node, those local changes will be overwritten when the local Puppet agent does its next sync-up with the Puppet master.

Note To ensure high availability of the Puppet master role, HyperStore automatically sets up a backup Puppet master node and supports a method for manually failing over the master role in the event of problems with the primary Puppet master node. See "[Move the Puppet Master Primary or Backup Role](#)" (page 415)

9.4. Pushing Configuration File Edits to the Cluster and Restarting Services

Subjects covered in this section:

- "[Using the Installer to Push Configuration Changes and Restart Services](#)" (page 454)
- "[Option for Triggering a Puppet Sync-Up from the Command Line](#)" (page 456)
- "[Excluding a Down Node from Installer-Driven Configuration Push](#)" (page 457)
- "[Automatic Puppet Sync-Up on an Interval](#)" (page 457)

9.4.1. Using the Installer to Push Configuration Changes and Restart Services

If you've edited configuration files on the Puppet master and you don't want to wait for the [automatic sync-up](#) -- or if automatic sync-up is disabled because you're not leaving the Puppet daemons running -- you can

manually trigger a Puppet sync-up using the HyperStore installer's interactive menu:

1. On your Puppet master node, change into your installation staging directory. Once in the staging directory, launch the HyperStore installer:

```
[7.2]# ./cloudianInstall.sh
```

This displays the installer's main menu:

```
Cloudian HyperStore(R) 7.2 Installation/Configuration
-----
0 ) Run Pre-Installation checks
1 ) Install Cloudian HyperStore
2 ) Cluster Management
3 ) Upgrade From a Previous Version
4 ) Advanced Configuration Options
5 ) Uninstall Cloudian HyperStore
6 ) Help
x ) Exit

Choice: [ ]
```

2. Enter "2" for Cluster Management. This displays the Cluster Management menu.

```
Cluster Management
-----
a ) Review Cluster Configuration
b ) Push Configuration Settings to Cluster
c ) Manage Services
d ) Run Validation Tests
x ) Return to Main Menu

Choice: [ ]
```

3. Enter "b" for Push Configuration Settings to Cluster. You will then be prompted to list the hosts on which you want the Puppet agents to sync up with the Puppet master. The default is all your HyperStore hosts; for this you can just press enter at the prompt. In a multi-region system you are also given the option to sync-up only the agents in a particular region.
4. After the Puppet run completes for all the Puppet agents (and a success message displays on the console), **restart the affected service(s) to apply your configuration change**:
 - a. From the Cluster Management menu, enter "c" for Manage Services. This displays the Service

Management menu.

```

Service Management
-----
0 ) All services
1 ) Redis Credentials
2 ) Redis QOS
3 ) Cassandra
4 ) HyperStore service
5 ) S3 service
6 ) Redis Monitor
7 ) Cloudian Agent
8 ) DNSMASQ
9 ) Cloudian Management Console (CMC)
10) IAM
11) SQS
P ) Puppet service (status only)
X ) Quit

You can execute the following list of commands:
start,stop,status,restart,version,node-start,node-stop

Select a service to manage: [
```

- b. From the Service Management menu, enter the number for the service to restart. The service to restart will depend on which configuration setting(s) you edited. For example:

File in Which You Edited Setting(s)	Service to Restart
<i>mts.properties.erb</i>	S3 Service
<i>hyperstore-server.properties.erb</i>	HyperStore Service
<i>mts-ui.properties.erb</i>	CMC
<i>common.csv</i>	Depends on the specific setting(s); see the documentation for com-mon.csv

- c. After entering the service to manage, enter "restart". Watch the console for messages indicating a successful stop and restart of the service. Note that the service restart occurs on each node on which the service is running.

9.4.2. Option for Triggering a Puppet Sync-Up from the Command Line

HyperStore supports an alternative means of triggering a Puppet sync-up, from the command line. To use this method you would run the install script like this:

```
[7.2]# ./cloudianInstall.sh runpuppet=" [<region>,<host>,<host>,... ] "
```

Here are some examples for how to specify the *runpuppet* option:

- *runpuppet=""*— Sync-up all the agents in your whole HyperStore system.
- *runpuppet="region1"*— Sync-up only all the agents in region1.

- *runpuppet="region1,host1"* — Sync-up only host1 in region1.
- *runpuppet="region1,host1,host2"* — Sync-up only host1 and host2 in region1.

Note If you use this method, you would still need to subsequently launch the installer in the normal way and then **restart the affected service(s)** as described in Step 4 above, in order to apply your changes.

9.4.3. Excluding a Down Node from Installer-Driven Configuration Push

If you use the installer to push a configuration change out to your cluster (by triggering a Puppet sync-up throughout the cluster) and the installer detects that a node is down or unreachable, the whole configuration push operation will fail. Therefore, if you want to make a system configuration change at a time when a node is down or unreachable you need to configure the installer to exclude that node from the cluster configuration push. For instructions on place a node into "excluded" status (and how to take a node out of this status), see "**r**) **Exclude host(s) from configuration push and service restarts**" (page 453).

Note There are a small number of circumstances where the system will **automatically** place a down node into the "excluded" status. One such circumstance is when the system is executing automatic fail-over of the system cronjob host role, in the event that the primary cronjob host goes down. The next time that you launch the installer it will display a message identifying the node that's in "excluded" status.

9.4.4. Automatic Puppet Sync-Up on an Interval

If you leave the Puppet master and Puppet agents running as background daemons (as is the default behavior after HyperStore installation), the Puppet agents on all of your up HyperStore nodes will automatically check every 10 minutes to see if changes have been made to the HyperStore configuration templates on the Puppet master node. If configuration changes have been made, each Puppet agent will download those changes to its local host machine.

Note however that this automatic Puppet sync-up **does not apply the configuration changes to the currently running services**. Applying the configuration changes requires a **service restart**.

To apply your changes, first wait long enough to be sure that the automatic Puppet sync-up has occurred (again the default is every 10 minutes). Then, restart the affected service(s) by logging into your Puppet master node, changing into the installation staging directory, launching the installer, then going to the Cluster Management menu and restarting the affected service(s) as described in Step 4 of the procedure above.

9.5. HyperStore Configuration Files

The main configuration file for your HyperStore system is [common.csv](#). Under typical circumstances that is the only configuration file that you might edit.

On rare occasions you might -- in consultation with Cloudian Support -- edit the [mts.properties.erb](#) file, the [hyperstore-server.properties.erb](#) file, or the [mts-ui.properties.erb](#) file.

There is no requirement to manually edit any HyperStore configuration file. Configuration customizations that are required in order to tailor HyperStore to your environment are implemented automatically by the installation script during the initial installation of your HyperStore system and during cluster expansions or contractions.

Note The configuration settings that operators would most commonly want to adjust during the operation of a HyperStore system are editable through the CMC's [Configuration Settings](#) page. The only reason to manually edit configuration files is for less-frequently-used settings that are not in the CMC's [Configuration Settings](#) page.

9.5.1. common.csv

The *common.csv* file is the main HyperStore configuration file and under typical circumstances this is the only configuration file that you may want to edit. On the Puppet master the path to the file is:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

The settings in *common.csv* are grouped according to which HyperStore component they configure. The documentation that follows describes the settings in each section of *common.csv*, in the order in which they appear in the file.

IMPORTANT: If you make any edits to *common.csv*, be sure to push your edits to the cluster and restart the affected services to apply your changes. For instructions see "[Pushing Configuration File Edits to the Cluster and Restarting Services](#)" (page 454).

Note The HyperStore system's interactive installation script *cloudianInstall.sh* writes to this file. All *common.csv* settings that **require** environment-specific customization are automatically pre-configured by the install script, based on information that you provided during the installation process. Making any further customizations to this configuration file is optional.

GENERAL Section

release_version

Current HyperStore release version.

Default = 7.2

Do not edit.

cloudian_license_file_name

Name of your Cloudian license file.

Default = Set during installation based on operator input.

Do not edit manually. To apply an updated license file, use the CMC's Update License function (**Cluster → Cluster Config → Cluster Information → Update License**). That function will automatically update this configuration setting as appropriate and dynamically apply the change to your live system. No service restart is necessary.

default_region

The default [service region](#) for your S3 service. Must be one of the regions listed for the regions setting. In a multi-region HyperStore system, the default region plays several roles in the context of S3 storage bucket LocationConstraint functionality. For example:

- For PUT Bucket requests that lack a CreateBucketConfiguration element in the request body, the bucket will be created in the default region.
- For PUT Bucket requests that do include a CreateBucketConfiguration element (with LocationConstraint attribute), the PUT requests typically resolve to the default region, and S3 Service nodes in the default region then initiate the process of creating the bucket in the requested region.

If your HyperStore system has only one service region, make that region the default region.

Default = Set during installation based on operator input.

Do not change this setting after your system is installed and running. If for some reason you need to change which region is your default region, please consult with Cloudian Support.

regions

Comma-separated list of [service regions](#) within your HyperStore system. Region names must be lower case with no dots, dashes, underscores, or spaces. Even if you have only one region, you must give it a name and specify it here.

Default = Set during installation based on operator input.

javahome

Home directory of Java on your HyperStore nodes.

Default = Path to OpenJDK (set automatically during installation)

java_minimum_stack_size

Memory stack size to use for the HyperStore system's Java-based services (Cassandra, S3 Service, HyperStore Service, Admin Service)

Default = 256k

Note Optionally you can override this stack size value on a per-server-type basis by adding any of the following settings to the configuration file and assigning them a value:

```
cassandra_stack_size
cloudian_s3_stack_size
cloudian_hss_stack_size
cloudian_admin_stack_size
```

installation_root_directory

Directory in which HyperStore service packages will be physically installed.

Default = /opt/cloudian-packages

run_root_directory

Root directory in which HyperStore services will be run. Links will be created from this directory to the physical installation directory.

Default = /opt

pid_root_directory

Root directory in which HyperStore service PID (process ID) files will be stored. A */cloudian* sub-directory will be created under this root directory, and the PID files will be stored in that sub-directory.

Default = /var/run

cloudian_user

User information for the user as which to run Cloudian HyperStore services, in format <user_name>,<group_name>,<optional_numeric_UID>,<login_shell>. If you want this user to be something other than the default, edit this setting and also the *cloudian_runuser* setting.

Default ="cloudian,cloudian,,/bin/bash"

cloudian_runuser

User name of the user as which to run Cloudian HyperStore services. This must match the first field from the *cloudian_user* setting. If you edit the *cloudian_runuser* setting you must also edit the first field from the *cloudian_user* setting.

Default = cloudian

user_home_directory

Directory under which to create the home directory of the HyperStore services runtime user. The system will append the *cloudian_runuser* value to the *user_home_directory* value to get the full home directory path. For example, with "cloudian" as the *cloudian_runuser* and "/export/home" as the *user_home_directory*, the Cloudian user's home directory is "/export/home/cloudian".

Default = /export/home

service_starts_on_boot

Whether to have HyperStore services start on host reboot, true or false.

Default = true

cloudian_log_directory

Directory into which to write application logs for the S3 Service, Admin Service, HyperStore Service, Redis Monitor, Cloudian performance monitoring Agent, and Cloudian performance monitoring Data Collector.

Default = /var/log/cloudian

cleanup_directories_byage_withmatch

The *cleanup_directories_byage_** settings configure Puppet to automatically delete certain files from your HyperStore nodes after the files reach a certain age. The *cleanup_directories_byage_withmatch* setting is a comma-separated list of directories in which to look for such files.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = "/var/log/cloudian,/tmp,/var/log/puppetserver"

cleanup_directories_byage_withmatch_timelimit

The *cleanup_directories_byage_** settings configure Puppet to automatically delete certain files from your

HyperStore nodes after the files reach a certain age. The `cleanup_directories_byage_withmatch_timelimit` setting specifies the age at which such files will be deleted (based on time elapsed since file creation). The age can be specified as `<x>m` or `<x>h` or `<x>d` where `<x>` is a number of minutes, hours, or days.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = 15d

`cleanup_directories_byage_matches`

The `cleanup_directories_byage_*` settings configure Puppet to automatically delete certain files from your HyperStore nodes after the files reach a certain age. The `cleanup_directories_byage_matches` setting specifies the file types to delete.

This feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior), or if you regularly perform a Puppet push.

To apply change, do a Puppet push. No service restart is necessary.

Default = "diagnostics*.gz,diagnostics*.tgz,jna*.tmp,liblz4-java*.so,snappy-* .so,* .cloudian-bak,cloudian_system_info*.tar.gz,puppetserver*.log.zip"

`cleanup_sysinfo_logs_timelimit`

Retention period for Node Diagnostics packages. After a package reaches this age, Puppet will automatically delete the package.

A Node Diagnostics package is created under a `/var/log/cloudian/cloudian_sysinfo` directory on a node if you use the Collect Diagnostics feature for that node. For general information on Node Diagnostics see "**Smart Support and Diagnostics Feature Overview**" (page 137).

This cleanup feature works only if you leave the Puppet daemons running on your HyperStore nodes (which is the default behavior).

Specify this value as a number of days, hours, or minutes, with the value formatted as `<n>d` or `<n>h` or `<n>m` respectively (such as 15d or 12h or 30m).

Default = 15d

Note When you use the CMC to [collect diagnostics on a node](#), the UI gives you the option to have the diagnostics package automatically uploaded to Cloudian Support, and also the option to have the system delete the package immediately after it's been successfully uploaded to Cloudian Support. The retention period set by `cleanup_sysinfo_logs_timelimit` comes into play only if you do not use the "upload and then immediately delete" options.

`path_style_access`

Whether the CMC (and also the installer's basic validation test script) should use "path style" request formatting when submitting S3 requests to the HyperStore S3 Service. In path style S3 requests, the bucket name is part of the request URI rather than being part of the Host header value.

Options are:

- true — The CMC will use path style HTTP request formatting when submitting S3 requests to the HyperStore S3 Service. The bucket name associated with the request will be in the request URI. For example:

```
PUT /bucket1/objectname HTTP/1.1  
Host: s3-region1.mycompany.com
```

- false — The CMC will not use path style HTTP request formatting when submitting S3 requests to the HyperStore S3 Service. Instead it will use "virtual host" style access. The bucket name associated with the request will be in the HTTP Host header. For example:

```
PUT /objectname HTTP/1.1  
Host: bucket1.s3-region1.mycompany.com
```

If the CMC (or any other S3 client applications that you are using) uses virtual host style access to the HyperStore S3 Service, then your DNS environment must be configured to resolve this type of Host value. See [DNS Set-Up](#).

Note that this setting affects only the behavior of the CMC and the behavior of the installer's basic validation test script, in their role as S3 clients. Meanwhile the HyperStore S3 Service supports both path style access and virtual host style access -- regardless of which method is being used by S3 clients.

Default = true

To apply change, after Puppet propagation restart CMC

HyperStore SERVICE Section

hyperstore_data_directory

A quote-enclosed, comma-separated list of mount points to use for S3 object storage. In a production environment, use dedicated disks for S3 object storage. Do not use the same disk(s) that are storing the OS and Cassandra.

The system will automatically assign virtual nodes (vNodes) to each of your S3 data mount points, in a manner that allocates an approximately equal **total token range** to each mount point. For more information about HyperStore vNodes see ["How vNodes Work" \(page 32\)](#).

Do not change the *hyperstore_data_directory* setting once your cluster is operational.

Do not use symbolic links when specifying your mount points for the *hyperstore_data_directory* setting. The HyperStore system does not support symbolic links for these directories.

Example of a multiple mount point configuration = "/cloudian1,/cloudian2,/cloudian3,/cloudian4"

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cloudian

hyperstore_listen_ip

The IP interface on which each HyperStore Service node listens for data operations requests from clients. This setting must match the *cassandra_listen_address* setting.

Specify this as an IP address alias. Puppet will use the alias to determine the actual IP address for each node.

Options are %{:cloudian_ipaddress}, %{:ipaddress_eth#}, %{:ipaddress_lo}, or %{:ipaddress_bind#}

Default = %{:cloudian_ipaddress}

To apply change, after Puppet propagation restart HyperStore Service

hyperstore_timeout

For the S3 Service's connections to the HyperStore Service, the transaction completion timeout (session timeout) in milliseconds.

Default = 10000

To apply change, after Puppet propagation restart S3 Service

For a diagram showing the place of this timeout within the S3 request processing flow, see the description of *mts.properties.erb*: "**cassandra.cluster.CassandraThriftSocketTimeout**" (page 497).

hyperstore_connection_timeout

For the S3 Service's connections to the HyperStore Service, the connection establishment timeout in milliseconds.

Default = 10000

To apply change, after Puppet propagation restart S3 Service

For a diagram showing the place of this timeout within the S3 request processing flow, see the description of *mts.properties.erb*: "**cassandra.cluster.CassandraThriftSocketTimeout**" (page 497).

hyperstore.maxthreads.repair

Maximum number of simultaneous client threads for one S3 Service node to use on HyperStore File System data repairs automatically performed during read operations. For more information on the "repair on read" mechanism see "**Data Repair Feature Overview**" (page 75).

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore.maxthreads.write

Maximum number of simultaneous client threads for one S3 Service node to use on writes to the HyperStore File System.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore.maxthreads.read

Maximum number of simultaneous client threads for one S3 Service node to use on reads of the HyperStore File System.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_maxperrouteconnections

The maximum allowed number of concurrently open connections between each S3 Service node and each HyperStore Service node. This allows for limiting the traffic load between each front-end S3 Service node (as it processes incoming requests from S3 clients) and any single HyperStore Service node.

Note that each of your S3 Service nodes has its own pool of connections to the HyperStore storage layer, so the total possible connections from the S3 Service as a whole to a single HyperStore Service node would be the number of S3 Service nodes multiplied by the value of "Max Connections from One S3 Service Node to One HyperStore Service Node".

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_maxtotalconnections

The maximum allowed number of concurrently open connections between each S3 Service node and all HyperStore Service nodes, combined. This allows for limiting the traffic load between each front-end S3 Service node (as it processes incoming requests from S3 clients) and the whole back-end HyperStore storage layer.

Note that each of your S3 Service nodes has its own pool of connections to the HyperStore storage layer, so the total possible connections from the front-end S3 Service as a whole to the back-end HyperStore storage layer as a whole would be the number of S3 Service nodes multiplied by the value of "Max Connections from One S3 Service Node to All HyperStore Service Nodes".

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_jetty_maxThreads

Each HyperStore Service node maintains a thread pool to process incoming HTTP requests from clients (S3 Service nodes). When there is a request to be serviced, a free thread from the pool is used and then returned to the pool afterward. If a thread is needed for a job but no thread is free, a new thread is created and added to the pool — unless the maximum allowed number of threads in the pool has been reached, in which case queued jobs must wait for a thread to become free.

The *hyperstore_jetty_maxThreads* parameter sets the maximum number of threads to allow in the thread pool.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_jetty_minThreads

Each HyperStore Service node maintains a thread pool to process incoming HTTP requests from clients (S3 Service nodes). Idle threads are terminated if not used within a timeout period — unless the number of threads in the pool is down to the required minimum pool size, in which case the idle threads are kept.

The *hyperstore_jetty_minThreads* parameter sets the minimum number of threads to keep in the thread pool.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore.messaging_service_threadpool

Maximum size of the thread pool used by the HyperStore inter-node messaging service. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_repair_session_threadpool

Maximum size of the thread pool used for [hsstool repair](#) or [hsstool repairec](#) operations. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionThreadPoolCorePoolSize)

hyperstore_repair_digest_index_threadpool

Maximum size of the thread pool used for reading file digests on a node in order to build the index required by Merkle Tree based repair (the default [hsstool repair](#) type). When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → DigestIndexCorePoolSize)

hyperstore_rangerepair_threadpool

Maximum size of the thread pool used for running multiple range repair tasks in parallel during [hsstool repair](#). When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RangeRepairThreadPoolCorePoolSize)

hyperstore_stream_outbound_threadpool

Maximum size of the thread pool used for streaming files from one HyperStore node to another during Merkle Tree based [hsstool repair](#). When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileStreamingService → Attributes → StreamOutboundThreadPoolCorePoolSize)

hyperstore_downloadrange_session_threadpool

Maximum size of the thread pool used for range download sessions conducted by a HyperStore Service node. When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_uploadrange_session_threadpool

Maximum size of the thread pool used for range upload sessions conducted by a HyperStore Service node.

When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_decommission_threadpool

Maximum size of the thread pool used for uploading files away from a node that is being decommissioned.

When there is a new task and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → DecommissionThreadPoolCorePoolSize)

hyperstore_cleanup_session_threadpool

This thread pool size limit determines the maximum number of blobs (object replicas or erasure coded fragments) to process in parallel within each cleanup "job" taking place on a node. Processing a blob entails checking the blob's corresponding object metadata to determine whether the blob is supposed to be where it is or rather should be deleted.

The maximum number of "jobs" running in parallel is set by "**cleanupjobs.threadpool.corepoolsize**" (page 492) in *hyperstore-server.properties.erb*.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

hyperstore_auto_repair_threadpool

Maximum size of the thread pool used by the HyperStore auto-repair feature. When there is an auto-repair to kick off and there are no idle threads available in the thread pool:

- If fewer than this many threads are in the thread pool, the thread pool executor will create a new thread.
- If this many or more threads are in the thread pool, the thread pool executor will queue the new task.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → AutoRepairThreadpoolCorePoolSize)

Note For more information about the auto-repair feature see "["Data Repair Feature Overview"](#)" (page 75).

hyperstore_repairrec_sessionscan_threadpool

During an [**hsstool repairrec**](#) operation, the threads in this thread pool are tasked with identifying erasure coded objects in the system and batching them for evaluation. This parameter sets the maximum size of the thread pool per node.

Default = 50

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECSessionScanThreadPoolCorePoolSize)

hyperstore_repairrec_digestrequest_threadpool

During an [**hsstool repairrec**](#) operation, the threads in this thread pool are tasked with reading the digests associated with batches of erasure coded objects, to determine whether any of those objects need repair. This parameter sets the maximum size of the thread pool per node.

Default = 10

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECDigestRequestThreadPoolFixedPoolSize)

hyperstore_repairrec_task_threadpool

During an [**hsstool repairrec**](#) operation, the threads in this thread pool are tasked with repairing erasure coded objects that have been determined to need repair. This parameter sets the maximum size of the thread pool per node.

Default = 20

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECTaskThreadPoolCorePoolSize)

hyperstore_repairrec_rocksdbscan_threadpool

During an [**hsstool repairrec**](#) operation, the threads in this thread pool enable concurrent reads of digest data in each [**RocksDB**](#) instance, as digest read requests come in from multiple digest request threads on multiple nodes. This parameter sets the maximum size of the thread pool per RocksDB instance.

Default = 10

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairECRock-sDBScanThreadPoolFixedPoolSize)

hyperstore_disk_check_interval

The interval (in minutes) at which each HyperStore Service node will run a check to see if there is significant imbalance in disk usage on the node. If an imbalance is found in excess of that configured by *disk.balance.delta*, then one or more tokens are automatically moved from the over-used disk(s) to one or more less-used disk(s) on the same host. For more information see "["Disk Usage Balancing Feature Overview"](#)" (page 85).

Default = 4320 (72 hours)

To apply change, after Puppet propagation restart the HyperStore Service

Note This feature applies only to HyperStore data disks (on which are stored S3 object data). It does not apply to disks that are storing only the OS and Cassandra.

auto_repair_computedigest_run_number

This property configures the [scheduled auto-repair feature](#) such that every *N*th run of [**hsstool repair**](#) (for replicated object data) and [**hsstool repairec**](#) (for erasure coded object data) will use the "-computedigest" option in order to detect and repair any data corruption on disk ("bit rot"). For example, if this property is set to 3, then on each node every 3rd run of *repair* will use the "-computedigest" option and for each data center every 3rd run of *repairec* will use the "-computedigest" option.

By default the auto-repair interval for *repair* is 30 days, and each individual node has its own every-30-days repair schedule. So if for example you set *auto_repair_computedigest_run_number* to 3, then on a given node the automatically triggered *repair* runs would be implemented like this:

- Day 0: *repair* without "-computedigest"
- Day 30: *repair* without "-computedigest"
- Day 60: *repair* with "-computedigest"
- Day 90: *repair* without "-computedigest"
- Day 120: *repair* without "-computedigest"
- Day 150: *repair* with "-computedigest"
- etc

By default the auto-repair interval for *repairec* is 29 days. With erasure coded data repair, running *hsstool repairec* on any one node repairs all the erasure coded data in the local data center. Consequently the auto-repair feature runs the command on just one randomly selected node in each data center every 29 days.

So if for example you set *auto_repair_computedigest_run_number* to 3, then for a given data center the automatically triggered *repairec* runs would be implemented like this:

- Day 0: *repairec* without "-computedigest"
- Day 29: *repairec* without "-computedigest"
- Day 58: *repairec* with "-computedigest"
- Day 87: *repairec* without "-computedigest"
- Day 116: *repairec* without "-computedigest"
- Day 145: *repairec* with "-computedigest"
- etc

Setting *auto_repair_computedigest_run_number* to 1 would result in all auto-repair runs using "-computedigest".

By default *auto_repair_computedigest_run_number* is set to 0, which disables using "-computedigest" for auto-repair runs. **So by default no auto-repair runs will use the "-computedigest" option.**

Default = 0

Note Because it entails recalculating a fresh MD5 hash of each replica or erasure coded fragment on the target node, using "-computedigest" on repair runs is an expensive operation in terms of resource utilization.

Note The `auto_repair_computedigest_run_number` setting has no impact on `hsstool repair` or `hsstool repairrec` runs that you manually execute on a node. The setting only impacts the auto-repair feature.

S3/ADMIN SERVICES Section

`phonehome_proxy_host`

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the hostname or IP address of the proxy.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

Note By default any proxy settings that you configure for the daily Smart Support uploads will also apply to the sending of on-demand Node Diagnostics packages (triggered by your using the CMC's [Collect Diagnostics](#) function). If you want to use a different proxy for Node Diagnostics sending than you do for the daily Smart Support upload, use the [`sysinfo.proxy.*`](#) settings in `mts.properties.erb` to separately configure proxy information for Node Diagnostics sending.

For more background information on these features see ["Smart Support and Diagnostics Feature Overview"](#) (page 137).

`phonehome_proxy_port`

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the proxy's port number.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

`phonehome_proxy_username`

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the username that HyperStore should use when connecting to the proxy (if a username and password are required by the proxy).

Default = empty

`phonehome_proxy_password`

If you want HyperStore to use a local forward proxy when the Smart Support (or "Phone Home") feature sends daily system diagnostics packages to Cloudian Support, use this setting to specify the password that HyperStore should use when connecting to the proxy (if a username and password are required by the proxy).

Default = empty

phonehome_uri

S3 URI to which to upload system-wide diagnostics data each day. By default this is the S3 URI for Cloudian Support.

If you set this to a different S3 destination, include the HTTP or HTTPS protocol part of the URI (*http://* or *https://*).

Default = `https://s3-support.cloudian.com:443`

To apply change, after Puppet propagation restart the S3 Service

Note If you set *phonehome_uri* to a URI for your own HyperStore S3 storage system (rather than Cloudian Support), and if your S3 Service is using HTTPS, then your S3 Service's SSL certificate must be a CA-verified, trusted certificate — not a self-signed certificate. By default the phone home function cannot upload to an HTTPS URI that's using a self-signed certificate. If you require that the upload go to an HTTPS URI that's using a self-signed certificate, contact Cloudian Support for guidance on modifying the phone home launch script.

phonehome_bucket

- If you leave *phonehome_uri* at its default value -- which is Cloudian Support S3 URI -- you can leave the *phonehome_bucket*, *phonehome_access_key*, and *phonehome_secret_key* properties empty. The Smart Support feature will automatically extract the Cloudian Support S3 bucket name and security credentials from your encrypted HyperStore license file.
- If you set *phonehome_uri* to an S3 URI other than the Cloudian Support URI, set the *phonehome_bucket*, *phonehome_access_key*, and *phonehome_secret_key* properties to the destination bucket name and the applicable S3 access key and secret key.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

phonehome_access_key

See *phonehome_bucket*.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

phonehome_secret_key

See *phonehome_bucket*.

Default = empty

To apply change, after Puppet propagation restart the S3 Service

phonehome_gdpr

To have the Smart Support feature comply with the European Union's General Data Protection Regulation (GDPR) requirements, set this to true. The impact of this setting is as follows:

- With *phonehome_gdpr* set to true, the Smart Support feature will remove user IDs and client IP addresses from the **copies** of the S3 request log ([cloudian-request-info.log](#)) and S3 application log ([cloudian-s3.log](#)) that are uploaded to Cloudian Support each day. However the user IDs and client IP

addresses will remain in the **original** S3 request and application logs on your HyperStore nodes. In the log file copies that get sent to Cloudian the user ID and IP address fields will say "Not Available".

- With *phonehome_gdpr* set to false, user IDs and client IP addresses will not be removed from the S3 request and application log copies that are uploaded to Cloudian Support each day.

Default = false

To apply change, after Puppet propagation restart the S3 Service

Note If you set this to true, the system will also remove user IDs and client IP addresses from the copies of logs that get sent to Cloudian Support in on-demand Node Diagnostics packages (by way of the [Collect Diagnostics](#) feature), while leaving this information in the original logs on your nodes.

admin_auth_user

Default username that you will supply in HTTP requests to the Admin Service for purposes of HTTP Basic Authentication. Relevant only if *admin_auth_enabled* is set to *true*.

Default = sysadmin

To apply change, after Puppet propagation restart S3 Service and CMC

admin_auth_pass

Default password that you will supply in HTTP requests to the Admin Service for purposes of HTTP Basic Authentication. Specify as a comma-separated pair of "<Jetty_obfuscated_password>,<cleartext_version_of_password>". For information about creating a Jetty-obfuscated password, see "**HTTP/S Basic Authentication for Admin API Access**" (page 703).

Relevant only if *admin_auth_enabled* is set to *true*.

Default = "1uvg1x1n1tv91tvt1x0z1uuq,public"

To apply change, after Puppet propagation restart S3 Service and CMC

admin_auth_realm

Realm name used by the Admin Service for purposes of HTTP Basic Authentication. Relevant only if *admin_auth_enabled* is set to *true*.

Default = CloudianAdmin

To apply change, after Puppet propagation restart S3 Service and CMC

admin_auth_enabled

Whether to have the Admin Service use HTTP Basic Authentication and require clients to supply a username and password when making a request. Set to true to enable HTTP Basic Authentication or false to disable it.

Default = true

Note For systems upgraded to the current version from a version older than 6.0.2, the default for this setting is "false".

To apply change, after Puppet propagation restart S3 Service

admin_secure

If set to "true", the Admin Service will accept **only** HTTPS connections from clients (through port 19443 by default). If set to "false", the Admin Service will allow regular HTTP connections from clients (through port 18080) as well as HTTPS connections (through port 19443).

This setting also controls CMC client-side behavior when the CMC calls the Admin Service for tasks such as creating users, creating storage policies, and retrieving system monitoring data. If `admin_secure` is "true", the CMC will exclusively use HTTPS when making requests to the Admin Service. If `admin_secure` is "false", the CMC will exclusively use regular HTTP when making requests to the Admin Service.

Note however that even if you set `admin_secure` to "false" -- so that the Admin Service accepts HTTP requests as well as HTTPS requests; and so that the CMC sends only HTTP requests to the Admin Service -- the Admin Service's HTTPS port will still be accessed by other HyperStore system components. In particular, some of the **"System cron Jobs"** (page 420) use HTTPS to make calls to the Admin Service.

Default = true

Note If your original HyperStore install was **older than version 6.0.2** and you have upgraded to the current version, the `admin_secure` setting does not appear in the `common.csv` file and an internal default value of "false" is used. In such systems, if you want the Admin Service to accept only HTTPS connections from clients, add the line `admin_secure,true` to the `common.csv` file.

To apply change, after Puppet propagation restart S3 Service and CMC

cmc_admin_secure_port

If the CMC is using HTTPS to connect to the Admin Service (as it will if `admin_secure` is set to "true"), this is the Admin Service listening port number to which it will connect. Note that this setting controls CMC client-side configuration, not Admin Service configuration. (The Admin Service-side configuration for HTTPS listening port is in `adminssl/configs.csv`. If you edit `cmc_admin_secure_port`, be sure to edit the Admin Service HTTPS port in `adminssl/configs.csv` also.)

Default = 19443

To apply change, after Puppet propagation restart CMC. And if you've edited `adminssl/configs.csv`, restart the S3 service also.

iam_service_enabled

If this is set to "true" then HyperStore's IAM Service is enabled and IAM functionality will display in the CMC. For more information see **"Identity and Access Management (IAM) Feature Overview"** (page 91).

Default = true

To apply change, after Puppet propagation restart the S3 Service.

iam_port

Port on which the HyperStore IAM Service listens for regular HTTP connections.

Default = 16080

To apply change, after Puppet propagation restart the S3 Service.

iam_secure

If set to "true", the IAM Service will accept **only** HTTPS connections from clients (through port 16443 by default). If set to "false", the IAM Service will allow regular HTTP connections from clients (through port 16080) as well

as HTTPS connections (through port 16443).

This setting also controls CMC client-side behavior when the CMC calls the IAM Service for tasks such as creating IAM or creating IAM policies. If *iam_secure* is "true", the CMC will exclusively use HTTPS when making requests to the IAM Service. If *iam_secure* is "false", the CMC will exclusively use regular HTTP when making requests to the IAM Service.

Default = false

To apply change, after Puppet propagation restart the S3 Service.

iam_secure_port

Port on which the HyperStore IAM Service listens for HTTPS connections.

Default = 16443

To apply change, after Puppet propagation restart the S3 Service.

cloudian_s3admin_min_threads

Minimum number of threads to keep in each Admin Service node's HTTP request processing thread pool.

The Admin Service uses a thread pool to process incoming HTTP requests from clients. An initial pool of threads is created at server initialization time, and additional threads may be added if needed to process queued jobs. Idle threads are terminated if not used within a thread timeout period — unless the number of threads in the pool is down to *cloudian_s3admin_min_threads*. If only this many threads are in the pool, then threads are kept open even if they've been idle for longer than the thread timeout.

Default = 10

To apply change, after Puppet propagation restart S3 Service

cloudian_s3admin_max_threads

Maximum number of threads to allow in the Admin Service's HTTP request processing thread pool. If there are fewer than this many threads in the pool, new threads may be created as needed in order to handle queued HTTP request processing jobs. If the maximum thread pool size is reached, no more threads will be created — instead, queued HTTP request processing jobs must wait for an existing thread to become free.

Default = 50

To apply change, after Puppet propagation restart S3 Service

cloudian_s3admin_max_idletime

When the Admin Service processes HTTP requests from clients, the maximum allowed connection idle time in milliseconds. If this much idle time passes before a new request is received on an open connection with a client, or if this much idle time passes during the reading of headers and content for a request, or if this much idle time passes during the writing of headers and content of a response, the connection is closed.

Default = 60000

To apply change, after Puppet propagation restart S3 Service

cloudian_s3admin_lowres_maxidletime

Special, "low resource" maximum idle time to apply to Admin Service HTTP connections when the number of simultaneous connections to an Admin Service node exceeds *cloudian_s3admin_lowres_maxconnections*. Configured in milliseconds. With this setting, you can have the Admin Service be less tolerant of connection

idle time during times of high concurrent usage. (For general idle timer behavior, see the description of *cloudian_s3admin_max_idletime* above.)

Default = 5000

To apply change, after Puppet propagation restart S3 Service

cloudian_s3admin_lowres_maxconnections

If the number of simultaneous HTTP connections to an Admin Service node exceeds this value, the special idle timer configured by *cloudian_s3admin_lowres_maxidletime* is applied to that node rather than the usual *cloudian_s3admin_max_idletime*.

Default = 1000

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_max_threads

Maximum number of threads to allow in the S3 Service's HTTP request processing thread pool. If there are fewer than this many threads in the pool, new threads may be created as needed in order to handle queued HTTP request processing jobs. If the maximum thread pool size is reached, no more threads will be created — instead, queued HTTP request processing jobs must wait for an existing thread to become free.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

cloudian_s3_max_idletime

When the S3 Service processes HTTP requests from clients, the maximum allowed connection idle time in milliseconds. If this much idle time passes before a new request is received on an open connection with a client, or if this much idle time passes during the reading of headers and content for a request, or if this much idle time passes during the writing of headers and content of a response, the connection is closed.

Default = 60000

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_lowres_maxidletime

Special, *low resource* maximum idle timer to apply to S3 Service HTTP connections when the number of simultaneous connections to an S3 Service node exceeds *cloudian_s3_lowres_maxconnections*. Configured in milliseconds.

Default = 5000

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_lowres_maxconnections

If the number of simultaneous HTTP connections to an S3 Service node exceeds this value, the special idle timer configured by *cloudian_s3_lowres_maxidletime* is applied to that node.

Default = 2000

To apply change, after Puppet propagation restart S3 Service

cloudian_tiering_useragent

User agent string used by HyperStore when it acts as an S3 client for auto-tiering to an external S3 system.

Default = "APN/1.0 Cloudian/1.0 HyperStore/"

To apply change, after Puppet propagation restart the S3 Service

s3_proxy_protocol_enabled

If you are using [HAProxy](#) as the load balancer in front of your S3 Service -- or a different load balancer that supports the [PROXY Protocol](#) -- you can set *s3_proxy_protocol_enabled* to *true* if you want the S3 Service to support the PROXY Protocol. If you do so, the following will happen:

- The S3 Server will create dedicated PROXY Protocol connectors listening on port 81 (for regular PROXY Protocol) and port 4431 (for PROXY Protocol with SSL/TLS). These connectors will be enabled and configured in *s3.xml.erb*. By default these connectors are disabled that template.
- If you configure your load balancer to use the PROXY Protocol for communicating with the S3 Service, the load balancer when relaying each S3 request to the S3 Service will pass along the originating client's IP address.

Note: For guidance on load balancer configuration consult with your Cloudian Sales Engineering or Professional Services representative.

- In the [S3 request log](#), the S3 request entries will then show the true client IP address as the source address, rather than showing the loader balancer's IP address as the source. Also, the true client IP address for each request will be available to support using S3 bucket policies that filter based on source IP address; or billing rating plans that "whitelist" certain source IP addresses.

Setting *s3_proxy_protocol_enabled* to *true* is appropriate if you're using HAProxy for load balancing -- or a different load balancer that supports the PROXY Protocol -- and you want S3 Service request logging to show the true origin address associated with S3 requests; and/or you want to implement bucket policies or rating plans that are responsive to the origin address. If you're using a load balancer that supports PROXY Protocol, using this protocol is the preferred method for providing the originating client IP address to the S3 layer, rather than using the *X-Forwarded-for* header.

Note If you intend to use the PROXY Protocol with SSL/TLS (on S3 Service listening port 4431) you must set up SSL/TLS for the S3 Service, if you have not already done so. See "[TLS/SSL for S3 Service \(CA-Verified Certificate\)](#)" (page 547) or "[TLS/SSL for S3 Service \(Self-Signed Certificate\)](#)" (page 543).

If *s3_proxy_protocol_enabled* is set to *true* then when you configure SSL/TLS for the S3 Service your configuration information will be applied to PROXY Protocol port 4431 as well as to the regular S3 HTTPS port 443.

Default = false

To apply change, after Puppet propagation restart the S3 Service

cloudian_s3_heap_limit

Maximum heap size limit for the S3 Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_s3_max_heap_percent*.

Default = 8g

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_max_heap_percent

Maximum heap size for the S3 Service application, as a percentage of host system RAM. The JAVA_OPTS -

Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_s3_heap_limit*.

Default = 10

To apply change, after Puppet propagation restart S3 Service

cloudian_hss_heap_limit

Maximum heap size limit for the HyperStore Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_hss_max_heap_percent*.

Default = 16g

To apply change, after Puppet propagation restart HyperStore Service

cloudian_hss_max_heap_percent

Maximum heap size for the HyperStore Service application, as a percentage of host system RAM. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_hss_heap_limit*.

Default = 20

To apply change, after Puppet propagation restart HyperStore Service

cloudian_admin_heap_limit

Maximum heap size limit for the Admin Service application. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the percent-of-system-RAM value set by *cloudian_admin_max_heap_percent*.

Default = 8g

To apply change, after Puppet propagation restart S3 Service

cloudian_admin_max_heap_percent

Maximum heap size for the Admin Service application, as a percentage of host system RAM. The JAVA_OPTS -Xmx value passed to the JVM will be the **lower** of this value and the limiting value set by *cloudian_admin_heap_limit*.

Default = 5

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the S3 Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart S3 Service

cloudian_hss_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the HyperStore Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart HyperStore Service

cloudian_admin_init_heap_percent

Initial heap size (JAVA_OPTS -Xms value) for the Admin Service application, as a percentage of the -Xmx value.

Default = 25

To apply change, after Puppet propagation restart S3 Service

cloudian_s3_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the S3 Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart S3 Service

cloudian_hss_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the HyperStore Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart HyperStore Service

cloudian_admin_new_heap_percent

New heap size (JAVA_OPTS -Xmn value) for the Admin Service application, as a percentage of the -Xmx value. This is the heap size specifically for "young generation" objects.

Default = 25

To apply change, after Puppet propagation restart S3 Service

cassandra_per_s3node

Maximum number of Cassandra nodes to which an individual S3 Service node may keep simultaneous live connections.

Default = 9

To apply change, after Puppet propagation restart S3 Service

cassandra_max_active

The maximum allowed number of simultaneously active connections in a Cassandra connection pool. If this limit has been reached and a thread requires a new connection to Cassandra, the thread will wait for a period configured by *cassandra.cluster.MaxWaitTimeWhenExhausted* in *mts.properties.erb* (default 9 seconds) before returning an error to the client.

Set to a negative value (e.g. -1) to disable this limit on active connections.

Default = During HyperStore installation, a HyperStore performance optimization script automatically sets this parameter to a value appropriate to your environment.

AES-256 Encryption Section

cloudian_s3_aes256encryption_enabled

This setting is for enabling AES-256 in HyperStore, for use with server-side encryption. If AES-256 is not

enabled, AES-128 is used instead.

Default = false

REDIS Section

redis_credentials_master_port

Port on which the Redis Credentials master node listens for data storage requests from clients. The Redis Credentials slave nodes will listen on this port as well.

Default = 6379

To apply change, after Puppet propagation restart Redis Credentials, S3 Service, and HyperStore Service

redis_qos_master_port

Port on which the Redis QoS master node listens for data storage requests from clients. The Redis QoS slave nodes will listen on this port as well.

Default = 6380

To apply change, after Puppet propagation restart Redis QoS, S3 Service, and HyperStore Service

redis_monitor_listener_port

Port on which the Redis Monitor can be queried for information about the Redis cluster state, via the Redis Monitor CLI.

Default = 9078

To apply change, after Puppet propagation restart Redis Monitor Service

redis_lib_directory

Directory in which to store Redis data files.

Default = /var/lib/redis

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

redis_log_directory

Directory in which to store Redis log files.

Default = /var/log/redis

To apply change, after Puppet propagation restart Redis Credentials and Redis QoS

CASSANDRA Section

cassandra_max_heap_size

Max Heap size setting (memory allocation) for the Cassandra application. If this setting is assigned a value, this value will be used for *MAX_HEAP_SIZE* in *cassandra-env.sh*. By default the *cassandra_max_heap_size* setting is commented out and not used by the system. Instead, the default behavior for Cloudian HyperStore is for *MAX_HEAP_SIZE* in *cassandra-env.sh* to be automatically set based on the host's RAM size, and for *HEAP_NEWSIZE* in *cassandra-env.sh* to be set to one-half of *MAX_HEAP_SIZE*.

If you uncomment this setting and assign it a value you must also uncomment and set *cassandra_heap_news-size*.

Default = commented out and unused

To apply change, after Puppet propagation restart Cassandra Service

cassandra_heap_newsize

New Heap size setting for the Cassandra application. This is the heap size specifically for "young generation" objects. If this is set, the value will be used for *HEAP_NEWSIZE* in *cassandra-env.sh*.

If you uncomment this setting and assign it a value, Cloudian, Inc. recommends setting it to one-half of *cassandra_max_heap_size*.

Default = commented out and unused

To apply change, after Puppet propagation restart Cassandra Service

cassandra_enable_gc_logging

Enable Java garbage collection (GC) logging for Cassandra. The log is written to */var/log/cassandra/gc.log*.

Default = true

To apply change, after Puppet propagation restart Cassandra Service

Note GC logging is also enabled for the S3 Service, Admin Service, and HyperStore Service. These GC logs are under */var/log/cloudian* and are named *s3-gc.log*, *admin-gc.log*, and *hss-gc.log*, respectively.

cassandra_heapdump_on_oomemerr

Whether to enable Java heap dump on a Cassandra application out-of-memory error. Set to "false" to disable heap dumps on out-of-memory errors. This removes the *JVM_OPTS -XX:+HeapDumpOnOutOfMemoryError* option from *cassandra-env.sh*.

Default = false

To apply change, after Puppet propagation restart Cassandra Service

cassandra_lib_directory

Directory in which to store Cassandra application state data.

Default = */var/lib/cassandra*

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

cassandra_log_directory

Directory in which to store Cassandra application log files.

Default = */var/log/cassandra*

To apply change, after Puppet propagation restart Cassandra Service

cassandra_saved_cache_directory

Directory in which to store the Cassandra saved_caches file.

Default = */var/lib/cassandra*

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

cassandra_commit_log_directory

Directory in which to store the Cassandra commit log file.

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cassandra_commit

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

cassandra_data_directory

Directory in which to store Cassandra data files. By default, Cassandra is used only for storing S3 object metadata (metadata associated with individual objects) and service metadata such as account information, usage data, and system monitoring data.

Default = Set during installation based on operator input, if host has multiple disks. For hosts with only one disk, default is /var/lib/cassandra/data

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

IMPORTANT: The Cassandra data directory should **not** be mounted on a shared file system such as a NAS device.

cassandra_port

Port on which Cassandra listens for data operations requests from clients.

Default = 9160

To apply change, after Puppet propagation restart Cassandra Service

concurrent_compactors

Number of simultaneous Cassandra compactions to allow, not including validation "compactions" for anti-entropy repair. Simultaneous compactions can help preserve read performance in a mixed read/write workload, by mitigating the tendency of small sstables to accumulate during a single long running compaction.

Default = 2

cassandra_tombstone_warn_threshold

Cassandra logs a warning if a query scans more than this number of tombstones (objects marked for deletion).

Default = 1000

To apply change, after Puppet propagation restart Cassandra

cassandra_tombstone_failure_threshold

Cassandra aborts a query if it scans more than this number of tombstones (objects marked for deletion).

Default = 100000

To apply change, after Puppet propagation restart Cassandra

cassandra_tombstone_cleanup_threshold

If more than this many tombstones (objects marked for deletion) exist in a CLOUDIAN_METADATA or MPSession column family in Cassandra, a tombstone purge process is automatically triggered for that column family.

Default = 75000

To apply change, after Puppet propagation restart Cassandra

cassandra_tombstone_gcgrace

While doing a purge of tombstones in a CLOUDIAN_METADATA or MPSession column family, do not purge tombstones that are fewer than this many seconds old.

Default = 0

To apply change, after Puppet propagation restart Cassandra

cassandra_listen_address

For each Cassandra node, the IP interface on which the node will listen for cluster management communications from other Cassandra nodes in the cluster. **Specify this as an IP address alias**. Puppet will use the alias to determine the actual IP address for each node.

Options are %{:cloudian_ipaddress}, %{:ipaddress_eth#}, %{:ipaddress_lo}, or %{:ipaddress_bind#}

Default = %{:cloudian_ipaddress}

To apply change, after Puppet propagation restart Cassandra Service

cassandra_rpc_address

For each Cassandra node, the IP interface on which the node will listen for data operations requests from clients, via Thrift RPC. **Specify this as an IP address alias**. Puppet will use the alias to determine the actual IP address for each node.

Options are %{:cloudian_ipaddress}, %{:ipaddress_eth#}, %{:ipaddress_lo}, or %{:ipaddress_bind#}

If desired, this can be the same IP address alias as used for *cassandra_listen_address*.

Default = %{:cloudian_ipaddress}

If you want to change this for a HyperStore system that's already in operation, consult with Cloudian Support.

cassandra_default_node_datacenter

This instance of this setting is not used. Instead, the instance of *cassandra_default_node_datacenter* in *region.csv* is used. Typically you should have no need to edit that setting.

Default = commented out

CMC Section

admin_whitelist_enabled

Whether to enable the billing "whitelist" feature that allows favorable billing terms for a specified list of source IP addresses or subnets. If this feature is enabled, whitelist management functionality displays in the CMC. This functionality is available only to HyperStore system administrators, not to group admins or regular users.

Default = true

To apply change, after Puppet propagation restart S3 Service and CMC

cmc_log_directory

Directory in which to store CMC application logs.

Default = /var/log/cloudian

To apply change, after Puppet propagation restart CMC

cmc_admin_host_ip

Fully qualified domain name for the Admin Service. The CMC connects to this service.

If you have multiple service regions for your HyperStore system, this FQDN must be the one for the Admin Service in your default service region.

Default = Set during installation based on operator input.

To apply change, after Puppet propagation restart CMC.

cmc_clouidian_admin_user

For the CMC, the login user name of the default system admin user. The system admin will use this user name to log into the CMC.

Default = admin

This cannot be changed through configuration.

Note The default password for the default CMC system admin user is "public". You can change this password when you are logged into the CMC as the default admin user. Hold your cursor over the user name that displays in the upper right of the screen (by default, "Admin"), then select the **Security Credentials** option and change the password.

cmc_domain

Service endpoint (fully qualified domain name) for the CMC. This endpoint must be resolvable for CMC clients.

Default = Set during installation based on operator input.

To change this endpoint, use the "**Installer Advanced Configuration Options**" (page 449).

cmc_web_secure

Whether the CMC should require HTTPS for all incoming client connections, true or false.

Set this property to "true" to require HTTPS. In this mode of operation, requests incoming to the CMC's regular HTTP port will be redirected to the CMC's HTTPS port.

Set this property to "false" to allow clients to connect through regular HTTP. In this mode of operation, requests incoming to the CMC's HTTPS port will be redirected to the CMC's regular HTTP port.

Default = true

To apply change, after Puppet propagation restart CMC.

cmc_http_port

Port on which the CMC listens for regular HTTP requests.

Default = 8888

To apply change, after Puppet propagation restart CMC.

cmc_https_port

Port on which the CMC listens for HTTPS requests.

Default = 8443

To apply change, after Puppet propagation restart CMC.

cmc_admin_secure_ssl

If the CMC is using HTTPS to connect to the Admin Service (as it will if "**admin_secure**" (page 471) is set to "true"), this setting controls the CMC's requirements regarding the Admin Service's SSL certificate:

- If set to "true", then when the CMC makes HTTPS connections to the Admin Service the CMC's HTTPS client will require that the Admin Service's SSL certificate be **CA validated** (or else will drop the connection).
- If set to "false", then when the CMC makes HTTPS connections to the Admin Service the CMC's HTTPS client will allow the Admin Service's SSL certificate to be self-signed. Note that the SSL certificate that is used with the Admin Service by default is self-signed.

Default = false

To apply change, after Puppet propagation restart CMC.

cmc_application_name

Name of the CMC web application, to be displayed in the URL paths for the various CMC UI pages. The CMC's URL paths are in the form *https://<host>:<port>/<application_name>/<page>.htm*.

Use only alphanumeric characters. Do not use spaces, dashes, underscores, or other special characters..

Default = Cloudian (and so URL paths are in form *https://<host>:<port>/Cloudian/<page>.htm*. For example *https://enterprise2:8443/Cloudian/dashboard.htm*).

To apply change, after Puppet propagation restart CMC.

cmc_storageuri_ssl_enabled

If this is set to "true", the CMC uses HTTPS to connect to the HyperStore S3 Service (in implementing the CMC **Buckets & Objects** functionality). If "false", the CMC uses regular HTTP to connect to the HyperStore S3 Service.

Do not set this to "true" unless you have set up SSL for your HyperStore S3 Service (following the instructions in "**TLS/SSL for S3 Service (Self-Signed Certificate)**" (page 543) or "**TLS/SSL for S3 Service (CA-Verified Certificate)**" (page 547)). By default the HyperStore S3 Service does not use SSL/HTTPS.

Note There are additional steps for the CMC to be able to connect to the S3 Service via HTTPS, as described at the end of the above referenced procedures.

Default = false

cmc_grouplist_enabled

This setting controls whether the CMC will show a drop-down list of group names when selection of a group is necessary in interior parts of the CMC UI (parts other than the login page). This is relevant only for system administrators, since only system administrators have the opportunity to choose among groups for certain features (such as user management, group management, or usage reporting). Set this to "false" to have the UI instead present a text box in which the administrator can type the group name.

Default = true

To apply change, after Puppet propagation restart CMC.

Note If the number of groups in your system exceeds the value of the *cmc_grouplist_size_max* setting (100 by default), then group drop-down lists are not supported and the UI will display a text input box for group name regardless of how you've set *cmc_grouplist_enabled*.

cmc_login_languageselection_enabled

This setting controls whether to display at the top of the CMC's **Sign In** screen a selection of languages from which the user can choose, for rendering the CMC's text (such as screen names, button labels, and so on). The supported languages are English, Japanese, Spanish, German, and Portuguese. With this set to "true", the CMC language will initially be based on the user's browser language setting, but in the **Sign In** screen the user will be able to select a different supported language if they wish.

If you set this to "false", then the language selection will not display at the top of the CMC's **Sign In** screen, and instead the CMC text language will be exclusively based on the user's browser language setting. If the user's browser language setting matches one of the supported CMC languages, then that language will be used for the CMC text. If the user's browser language setting does not match any of the CMC's supported languages, the CMC text will display in English.

Default = true

To apply change, after Puppet propagation restart CMC.

cmc_login_grouplist_enabled

This setting controls whether to enable the **Group** drop-down list on the CMC's **Sign In** screen. The **Group** drop-down list lists all groups registered in the HyperStore system, and when users log in they can choose their group from the list. If disabled, the drop-down list will not display and instead users will need to enter their group name in a text input box when logging into the CMC.

Set this to "false" if you don't want users to see the names of other groups.

Default = true

To apply change, after Puppet propagation restart CMC.

Note If the number of groups in your system exceeds the value of the *cmc_grouplist_size_max* setting (100 by default), then group drop-down lists are not supported and the UI will display a text input box for group name regardless of how you've set *cmc_login_grouplist_enabled*.

cmc_grouplist_size_max

Maximize number of groups that can be displayed in a CMC drop-down list.

If you have more than this many groups in your HyperStore system, then in parts of the CMC interface that require the user to select a group the interface will display a text input box rather than a drop-down list of groups to select from. The CMC will do this regardless of your setting for *cmc_login_grouplist_enabled* and *cmc_grouplist_enabled*.

For example, if *cmc_login_grouplist_enabled* and *cmc_grouplist_enabled* are set to "true" and *cmc_grouplist_size_max* is set to 100 (the default values), then the CMC will display drop-down lists for group selection if you have up to 100 groups in your system, or text input boxes for group name entry if you have more than 100 groups in your system.

Default = 100

To apply change, after Puppet propagation restart CMC.

cmc_keystore_passwd

The password for the CMC's TLS/SSL keystore. This setting is managed automatically by the HyperStore installer's Advanced Configuration function "Import Java keystore to CMC". You should not need to manually edit this setting. For more information see "**TLS/SSL for the CMC (Self-Signed Certificate)**" (page 553) or "**TLS/SSL for the CMC (CA-Verified Certificate)**" (page 556).

Default = 123cloudian456

cmc_bucket_tiering_default_destination_list

The list of auto-tiering destinations to display in the CMC interface that bucket owners use to configure auto-tiering for a bucket (for more information on this interface see "**Configure a Bucket Lifecycle Policy for Object Auto-Tiering or Expiration**" (page 197)). Specify this as a quote-enclosed list, with comma-separation between destination attributes and vertical bar separation between destinations, like this:

```
"<name>,<endpoint>,<protocol>|<name>,<endpoint>,<protocol>|..."
```

This can be multiple destinations (as it is by default), or you can edit the setting to have just one destination in the "list" if you want your users to only use that one destination.

For multiple destinations you can have as many as you want, within reason (bear in mind that in the interface the dialog box will expand to accommodate the additional destinations).

The *<name>* will display in the CMC interface that bucket owners use to configure auto-tiering, as the auto-tiering destination name. The *<protocol>* must be one of the following:

- s3
- glacier
- azure
- spectra

If you wish you can include multiple destinations of the same type, if those destinations have different endpoints. For example, "Spectra 1,<endpoint1>,spectra|Spectra 2,<endpoint2>,spectra". Each such destination will then appear in the CMC interface for users configuring their buckets for auto-tiering.

Default = "AWS S3,https://s3.amazonaws.com,s3|AWS GLACIER,https://s3.amazonaws.com,glacier|Google,https://storage.googleapis.com,s3|Azure,https://blob.core.windows.net,azure"

To apply change, after Puppet propagation restart CMC.

Note If your original HyperStore version was older than 7.1.4, then after upgrade to 7.1.4 or later your default value here will also include a Spectra destination.

Note For more information about setting up auto-tiering, see "**Setting Up Auto-Tiering**" (page 56).

That section includes information about how to enable the auto-tiering feature (which is disabled by default in the CMC interface) and about the option of having all end users use the same system-configured security credentials for accessing the tiering destination (rather than supplying their own security credentials for tiering, which is the default behavior). Note that if you choose to have all users use the same tiering security credentials you will need to specify a single system default tiering destination -- using a setting in the CMC's **Configuration Settings** page, as described in "**Setting Up Auto-**

Tiering" (page 56) -- and that configuration setting will **override** the *cmc_bucket_tiering_default_destination_list* setting.

AWS MMS Proxy Section

awsmmsproxy_host

If you are using the AWS Marketplace Metering Service version of Cloudian HyperStore, you must set this property to the public IP address of your AWS Proxy Server running on an AWS EC2 instance. For more information about this version of HyperStore, contact your Cloudian representative.

Default = empty

To apply change, after Puppet propagation restart S3 Service.

Usage Section

bucketstats_enabled

Whether to enable usage statistics reporting on a per-bucket basis, true or false. If you set this to true, you can subsequently retrieve usage data for a specified bucket or buckets by using the Admin API's [GET /usage](#) and [POST /usage/bucket](#) methods. Per-bucket usage statistics will be available only dating back to the point in time that you enabled this feature. Per-bucket usage statistics are not tracked by default and will not be available for time periods prior to when you set this parameter to true.

Default = false

To apply change, after Puppet propagation restart S3 Service.

Elastic Search Section

cloudian_elasticsearch_hosts

For information about this setting, see "[Elasticsearch Integration for Object Metadata](#)" (page 106).

IMPORTANT: Do not manually edit settings that appear below this point in the *common.csv* file.

9.5.2. *hyperstore-server.properties.erb*

The *hyperstore-server.properties* file configures the HyperStore Service. On each of your HyperStore nodes, the file is located at the following path by default:

/opt/cloudian/conf/hyperstore-server.properties

Do not directly edit the *hyperstore-server.properties* file on individual HyperStore nodes. Instead, if you want to make changes to the settings in this file, edit the configuration template file *hyperstore-server.properties.erb* on the Puppet master node:

/etc/cloudian-<version>-puppet/modules/cloudians3/templates/hyperstore-server.properties.erb

Certain *hyperstore-server.properties.erb* properties take their values from settings in [common.csv](#) or from settings that you can control through the CMC's [Configuration Settings](#) page. In the *hyperstore-server.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like <%= ... %>. In

the property documentation below, the descriptions of such properties indicate "Takes its value from <location>: <setting>; use that setting instead." . The remaining properties in the *hyperstore-server.properties.erb* file -- those that are "hard-coded" with specific values -- are settings that in typical circumstances you should have no need to edit. Therefore **in typical circumstances you should not need to manually edit the *hyperstore-server.properties.erb* file.**

IMPORTANT: If you do make edits to *hyperstore-server.properties.erb*, be sure to push your edits to the cluster and restart the HyperStore Service to apply your changes. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

The *hyperstore-server.properties.erb* file has the following settings:

cloudian.storage.datadir

Takes its value from *common.csv*: "**hyperstore_data_directory**" (page 462); use that setting instead.

secure.delete

Set this to true if you want HyperStore to use "**secure delete**" methodology whenever implementing the deletion of an object from a bucket.

For background, note that object data is stored in an *ext4* file system on each HyperStore node, and the process of writing, reading, and deleting object data from the file system is managed by the HyperStore Service (see "**HyperStore Service and the HSFS**" (page 12)). Also note that, depending on your storage policies, each object is either replicated or erasure coded, and the replicas or erasure coded fragments are distributed across multiple nodes. Larger objects are broken into chunks first, before those chunks are then replicated or erasure coded.

When secure delete is used, HyperStore implements the deletion of an object by overwriting the object data three times and then deleting it. The three passes at overwriting each of the object's bytes are executed as:

- 1st pass: 00110101 (0x35)
- 2nd pass: 11001010 (0xCA)
- 3rd pass: 10010111 (0x97)

This overwriting occurs for every byte of every replica or fragment of the object, on every node on which the object's data resides.

The *secure.delete* setting is a **system-wide setting**. When *secure.delete=true* then **all deletes** -- by any user, for any bucket -- are implemented as secure deletes. You cannot, for instance, apply this feature only to some buckets and not to others.

Secure delete activity is logged in *cloudian-hyperstore-request-info.log* on each node. The activity is logged only after completion of the third and final overwrite pass. The log entry indicates SECURE-DELETE as the operation type, and a 200 status code in the log entry indicates that the secure delete was successful. The log entry also includes the object name and the corresponding *ext4* file name. For more information on this log see "**HyperStore Service request log (*cloudian-hyperstore-request-info.log*)**" (page 579).

Note In the case of buckets that use **versioning**, to delete all versions of an object the S3 client application must explicitly delete each version by specifying the version ID.

Note Secure delete does not apply to **object metadata** stored in Cassandra. When objects are deleted, the system deletes the corresponding object metadata in the normal way -- with no overwriting passes -- regardless of whether or not you have the secure delete feature turned on. Therefore you should limit any user-defined object metadata created by your S3 client application(s) to information that does not require secure delete.

IMPORTANT: Using secure delete has substantial impact on **system performance** for delete operations. Consult with your Cloudian representative if you are considering using secure delete.

Default = false

messaging.service.listen.address

Takes its value from *common.csv*: "**hyperstore_listen_ip**" (page 462); use that setting instead.

messaging.service.listen.port

The port on which a HyperStore Service node listens for messages from other HyperStore Service nodes. This internal cluster messaging service is used in support of cluster management operations such as node repair.

Default = 19050

messaging.service.read.buffer.size

When a HyperStore Service node reads data it has received over the network from other HyperStore Service nodes -- such as during repair operations -- this is the read buffer size.

Default = 65536

messaging.service.write.buffer.size

When a HyperStore Service node writes data to the network for transferring to other HyperStore Service nodes -- such as during repair operations -- this is the write buffer size.

Default = 1048576

messaging.service.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore.messaging_service_threadpool**" (page 464); use that setting instead.

messaging.service.maxconnections

Maximum simultaneous number of connections that the HyperStore messaging service will accept.

Default = 2000

messaging.service.repairfile.timeout

Maximum time in seconds to allow for repair of a single file on a HyperStore node. File repair entails checking other HyperStore nodes to find the most recent copy of the file and then downloading that copy.

Default = 120

messaging.service.connection.timeout

Maximum time in seconds that a HyperStore node will allow for establishing a connection to another

HyperStore node, for conducting inter-node operations.

Default = 300

`repair.session.threadpool.corepoolsize`

Takes its value from `common.csv:"hyperstore_repair_session_threadpool"` (page 464); use that setting instead.

`repair.session.rangeslice.maxrows`

During [hsstool repair](#) or [hsstool repairec](#) operations, the maximum number of row keys to retrieve per get_range_slice query performed on Cassandra <GROUPID>_METADATA column families.

Default = 2

Note Cloudian, Inc recommends that you leave this setting at its default value. Do **not** set it to a value lower than 2.

`repair.session.columnslice.maxcolumns`

During [hsstool repair](#) or [hsstool repairec](#) operations, the maximum number of columns to retrieve per get_slice or get_range_slice query performed on Cassandra <GROUPID>_METADATA column families.

Default = 1000

`repair.session.slicequery.maxretries`

During [hsstool repair](#) or [hsstool repairec](#) operations, the maximum number of times to retry get_slice or get_range_slice queries after encountering a timeout. The timeout interval is configured by `cassandra.cluster.CassandraThriftSocketTimeout` in `mts.properties`, and retries are attempted as soon as a timeout occurs.

Default = 3

`repair.session.updateobjs.queue maxlen`

During [hsstool repair](#) operations, the target maximum number of object update jobs to queue for processing. Object update jobs are placed in queue by a differencer mechanism that detects discrepancies between object metadata on remote replicas versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for `repair.session.updateobjs.queue.maxwaittime` (below).

Default = 1000

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionJobQueueMaxLength)

`repair.session.updateobjs.queue.waittime`

If during [hsstool repair](#) operations the differencer detects that the number of queued object update jobs is at or above the target maximum (as configured by `repair.session.updateobjs.queue maxlen`), the number of seconds to wait before checking the queue size again. During this interval the differencer adds no more object update jobs to the queue.

Default = 2

`repair.session.updateobjs.queue.maxwaittime`

During [hsstool repair](#) operations, the maximum total number of seconds for the differencer to wait for the object update job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of update requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object update requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

repair.session.object.download.maxretries

During [hsstool repair](#) or [hsstool repairec](#) operations, the maximum number of times to retry object download requests after encountering a timeout. The timeout interval is configured by *mts.properties: cassandra.cluster.CassandraThriftSocketTimeout*, and retries are attempted as soon as a timeout occurs.

Default = 3

repair.session.inmemory.fileindex

When performing Merkle Tree based [hsstool repair](#) (the default repair type) for files in the HyperStore File System, whether to hold the file indexes in memory rather than writing them to disk. Options are:

- true — For each vNode being repaired, a file index directory is created and is held in memory unless its size exceeds a threshold in which case it is written to disk (under the HyperStore data mount point that the vNode is associated with). For most vNodes it will not be necessary to write the file indexes to disk. If file indexes are written to disk, they are automatically deleted after the repair operation completes.
- false — For each vNode being repaired, a file index directory is created and written to disk (under the HyperStore data mount point that the vNode is associated with), regardless of size. The file indexes are automatically deleted after the repair operation completes.

Default = true

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionInMemoryFileIndex)

repair.digest.index.threadpool.corepoolsize

Takes its value from *common.csv: "hyperstore_repair_digest_index_threadpool"* (page 465); use that setting instead.

repair.merkletree.response.waittime

During *hsstool repair* operations, the repair coordinator node retrieves Merkle Trees from HyperStore endpoint nodes. When contacted by the coordinator node, the endpoint nodes first must construct the Merkle Trees before returning them to the coordinator node. Constructing the trees can take some time if a very large number of objects is involved.

The *repair.merkletree.response.waittime* property sets the maximum amount of time in minutes that the coordinator node will wait for Merkle Trees to be returned by all endpoint nodes. If not all Merkle Trees have been returned in this time, the repair operation will fail.

Default = 120

rangerepair.threadpool.corepoolsize

Takes its value from *common.csv: "hyperstore_rangerepair_threadpool"* (page 465); use that setting instead.

stream.outbound.threadpool.corepoolsize

Takes its value from *common.csv: "hyperstore_stream_outbound_threadpool"* (page 465); use that setting

instead.

repairec.sessionscan.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_repairrec_sessionscan_threadpool**" (page 467); use that setting instead.

repairec.digestrequest.threadpool.fixedpoolsize

Takes its value from *common.csv*: "**hyperstore_repairrec_digestrequest_threadpool**" (page 467); use that setting instead.

repairec.task.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_repairrec_task_threadpool**" (page 467); use that setting instead.

repairec.rocksdbscan.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_repairrec_rocksdbscan_threadpool**" (page 467); use that setting instead.

repairec.session.queue maxlen

During "**hsstool repairrec**" (page 658) operations, the target maximum number of object update jobs to queue for processing. Object update jobs are placed in queue by a differencer mechanism that detects discrepancies between object metadata on remote replicas versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for *repairec.session.queue.maxwaittime*(below).

Default = 1000

repairec.session.queue.waittime

If during **hsstool repairrec** operations the differencer detects that the number of queued object update jobs is at or above the target maximum (as configured by *repairec.session.updateobjs.queue maxlen*), the number of seconds to wait before checking the queue size again. During this interval the differencer adds no more object update jobs to the queue.

Default = 2

repairec.session.queue.maxwaittime

During **hsstool repairrec** operations, the maximum total number of seconds for the differencer to wait for the object update job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of update requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object update requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

downloadrange.session.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_downloadrange_session_threadpool**" (page 465); use that setting instead.

uploadrange.session.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_uploadrange_session_threadpool**" (page 466); use that

setting instead.

decommission.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_decommission_threadpool**" (page 466); use that setting instead.

cleanup.session.threadpool.corepoolsize

Takes its value from *common.csv*: "**hyperstore_cleanup_session_threadpool**" (page 466); use that setting instead.

cleanup.session.deleteobjs.queue maxlen

During [hsstool cleanup](#) or [hsstool cleanupec](#) operations, the target maximum number of object delete jobs to queue for processing. Object delete jobs are placed in queue by a cleanup job that detects discrepancies between object metadata in Cassandra versus object metadata on the local node.

This target maximum may be exceeded in certain circumstances as described for *cleanup.session.deleteobjs.queue.maxwaittime*.

Default = 1000

cleanup.session.deleteobjs.queue.waittime

If during [hsstool cleanup](#) or [hsstool cleanupec](#) operations a cleanup job detects that the number of queued object delete jobs is at or above the target maximum (as configured by *cleanup.session.deleteobjs.queue maxlen*), the number of seconds to wait before checking the queue size again. During this interval the cleanup job adds no more object delete jobs to the queue.

Default = 2

cleanup.session.deleteobjs.queue.maxwaittime

During [hsstool cleanup](#) or [hsstool cleanupec](#) operations, the maximum total number of seconds for the differencer to wait for the object delete job queue to fall below its target maximum size. After this interval, the differencer goes ahead and writes its current batch of delete requests to the queue. In this scenario, the queue can grow beyond the target maximum size. The next time that the differencer has object delete requests, it again checks the queue size, and if it's larger than the target maximum size, the wait time procedure starts over again.

Default = 120

cleanup.session.delete.graceperiod

During [hsstool cleanup](#) or [hsstool cleanupec](#) operations, only consider an object for deletion if at least this many seconds have passed since the object's Last Modified timestamp.

Default = 86400

cleanupjobs.threadpool.corepoolsize

During [hsstool cleanup](#) or [hsstool cleanupec](#) operations, this setting controls how many cleanup "jobs" can run in parallel on a single HyperStore node. For each HyperStore data mount point on a node, the object data directory structure is as follows:

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

Under the *hsfs* (for replica data) or *ec* (for erasure coded data) directory level, there are sub-directories for each of the mount point's vNodes (identified by token), and under those, sub-directories for each storage policy configured in your system (identified by system-generated policy ID). For more information on this directory structure, see "**HyperStore Service and the HSFS**" (page 12).

When a physical HyperStore node is cleaned, there is a separate cleanup "job" for each *<policyId>* sub-directory on the physical node. The *cleanupjobs.threadpool.corepoolsize* setting controls how many such jobs can run in parallel on a given physical node. Each concurrent job will run on a different HyperStore data disk on the node.

Within each job, the separate setting *common.csv:"hyperstore_cleanup_session_threadpool"* (page 466) controls how many blobs (object replicas or erasure coded fragments) can be processed in parallel. Processing a blob entails checking the blob's corresponding object metadata to determine whether the blob is supposed to be where it is or rather should be deleted.

So for example with *cleanupjobs.threadpool.corepoolsize = 4* and *hyperstore_cleanup_session_threadpool = 10*, then on a given physical node being cleaned a maximum of 4 cleanup jobs would run in parallel (with each job working on a different disk), with a maximum of 10 blobs being processed in parallel within each of the 4 jobs.

Default = 4

hyperstore.proactiverepair.poll_time

At this recurring interval each HyperStore node checks to see whether any proactive repair jobs are queued for itself, and executes those jobs if there are any. Configured as a number of minutes.

This check is also automatically performed when a HyperStore Service node starts up. Subsequently the recurring check occurs at this configured interval.

For more information about the proactive repair feature, see "**Data Repair Feature Overview**" (page 75).

Default = 60

Note If for some reason you want to trigger proactive repair on a particular node immediately, you can do so by running the "**hsstool proactiverepairq**" (page 637) command with the "-start" option.

Note For information about temporarily disabling the proactive repair feature, see "**Disabling or Stopping Data Repairs**" (page 79).

stream.throughput.outbound

During **hsstool cleanup** or **hsstool cleanupec** operations, HyperStore nodes may stream large amounts of data to other HyperStore nodes. This setting places an upper limit on outbound streaming throughput during repair operations, in megabits per second.

Default = 800

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileStreamingService → Attributes → MaxStreamThroughputOutbound)

auto.repair.threadpool.corepoolsize

Takes its value from *common.csv: "hyperstore_auto_repair_threadpool"* (page 466); use that setting instead.

auto.repair.scheduler.polltime

Interval (in minutes) at which each HyperStore node's auto-repair scheduler will check the auto-repair queues for HSFS repair, Cassandra repair, and erasure coded data repair to see whether it's time to initiate a repair on that node.

Default = 10

Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → AutoRepairSchedulerPollTime)

auto.repair.schedule.interval

Takes its value from the "**Replicas Repair Interval (Minutes)**" (page 305) setting in the CMC's [Configuration Settings](#) page; use that setting instead.

auto.repairrec.schedule.interval

Takes its value from the "**EC Repair Interval (Minutes)**" (page 306) setting in the CMC's [Configuration Settings](#) page; use that setting instead.

auto.repaircassandra.schedule.interval

Takes its value from the "**Cassandra Full Repair Interval (Minutes)**" (page 307) setting in the CMC's [Configuration Settings](#) page; use that setting instead.

cloudian.storage.jmx.port

The port on which the HyperStore Service listens for JMX requests.

Default = 19082 (set elsewhere in the manifest structure; do not edit this property)

disk.fail.action

Takes its value from the "**HyperStore Disk Failure Action**" (page 297) setting in the CMC's [Configuration Settings](#) page; use that setting instead.

disk.repair.rebuild

When HyperStore implements a [replaceDisk](#) operation it automatically executes *hsstool repair* and *hsstool repairrec* for the replacement disk. With *disk.repair.rebuild=true* (the default setting), the automatic executions of *hsstool repair* and *hsstool repairrec* will use the *-rebuild* option that those operations support. Using the *-rebuild* option is the most efficient way to rebuild data on to a replacement disk. Typically the only occasion you would have for setting *disk.repair.rebuild=false* -- so that the *-rebuild* option is not used -- is if you are instructed to do so by Cloudian Support in the context of troubleshooting a failed attempt to replace a disk.

Default = true

disk.fail.error.count.threshold

This setting in combination with the *disk.fail.error.time.threshold* setting provides you the option to specify a read/write error frequency threshold that must be met before the system takes the automated action that is specified by the "HyperStore Disk Failure Action" setting.

The threshold, if configured, is in the form of "If *disk.fail.error.count.threshold* number of HSDISKERROR messages are logged in *cloudian-hyperstore.log* in regard to the same disk within an interval of *disk.-fail.error.time.threshold* seconds, then take the automated action specified by the *disk.fail.action* setting."

For example, if *disk.fail.error.count.threshold* = 3, and *disk.fail.error.time.threshold* = 60, and *disk.fail.action* = "disable", then the system will disable any HyperStore data disk for which 3 or more HSDISKERROR messages appear in *cloudian-hyperstore.log* within a 60 second time span.

If you set the two threshold settings to "0", then no threshold behavior is implemented, and instead the automated action specified by the *disk.fail.action* setting is triggered by any single occurrence of an HSDISKERROR message in *cloudian-hyperstore.log*.

Default = 10

disk.fail.error.time.threshold

Disk error threshold time span in seconds. For more description see *disk.fail.error.count.threshold* above.

Default = 300

disk.check.interval

Takes its value from *common.csv*: "**hyperstore_disk_check_interval**" (page 467); use that setting instead.

disk.balance.delta

When the disk balance check is run (at the *disk.check.interval*), token migration is triggered if a disk's utilization percentage differs from the average disk utilization percentage on the node by more than the configured *disk.balance.delta*. If *disk.balance.delta* = 10 (the default), then, for example:

- If the average disk space utilization on a node is 35%, and the disk space utilization for Disk4 is 55%, then one or more tokens will be migrated away from Disk4 to other disks on the node (since the actual delta of 20% exceeds the maximum allowed delta of 10%).
- If the average disk utilization on a node is 40%, and the disk utilization for Disk7 is 25%, then one or more tokens will be migrated to Disk7 from the other disks on the node (since the actual delta of 15% exceeds the maximum allowed delta of 10%).

For more information on this feature see "**Disk Usage Balancing Feature Overview**" (page 85).

Default = 10

disk.audit.interval

At this configurable interval (in number of minutes), the system tries to write one byte of data to each HyperStore data disk. If any of these writes fail, */var/log/messages* is scanned for messages indicating that the file system associated with the disk drive in question is in a read-only condition (message containing the string "Remounting filesystem read-only"). If any such message is found, the disk is automatically disabled in accordance with your configured "**HyperStore Disk Failure Action**" (page 297).

The scan of */var/log/messages* will be limited to the time period since the the last time the disk audit was run.

This recurring audit of disk drive health is designed to proactively detect disk problems even during periods when there is no HyperStore Service read/write activity on a disk.

Default = 60

disk.error.check.fs

When scanning */var/log/messages* as part of the disk audit, the messages will be first filtered by this file system name string.

Default = "EXT4-fs"

max.diskusage.percentage

The [hsstool repair](#) operation will fail if any HyperStore data disk that stores token ranges impacted by the operation is more than this percent full.

Default = 95

auto.repair.computedigest.run.number

Takes its value from *common.csv*: "[auto_repair_computedigest_run_number](#)" (page 468); use that setting instead.

hss.errorlogger.appenders

Do not edit this setting.

hss.heallogger.appenders

Do not edit this setting.

retry.rebalance.ranges

If this is set to true, HyperStore will automatically retry any failed sub-tasks from an *hsstool rebalance* operation that has completed on a newly added node. Like other types of [proactive repair](#), this retry of any failed rebalance sub-tasks occurs once per hour.

Default = true

enable.cassandra.rangerepair

If this is set to true, HyperStore uses a "range repair" approach when executing Cassandra auto-repairs. Each impacted token range is repaired one range at a time, sequentially. This approach improves the performance for Cassandra auto-repairs.

For background information on the scheduled auto-repair feature see "[Data Repair Feature Overview](#)" (page 75).

Default = true

Note The *rocksdb.** properties at the bottom of the *hyperstore-server.properties.erb* file control the behavior and performance of the RockDB database in which object digests are stored. Do not edit these settings.

9.5.3. *mts.properties.erb*

The *mts.properties* file configures the S3 Service and the Admin Service. On each of your HyperStore nodes, the file is located at the following path by default:

/opt/cloudian/conf/mts.properties

Do not directly edit the *mts.properties* file on individual HyperStore nodes. Instead, if you want to make changes to the settings in this file, edit the configuration template file *mts.properties.erb* on the Puppet master node:

/etc/cloudian-<version>-puppet/modules/cloudians3/templates/mts.properties.erb

Certain *mts.properties.erb* properties take their values from settings in [common.csv](#) or from settings that you can control through the CMC's [Configuration Settings](#) page. In the *mts.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like <%= ... %>. In the property documentation below, the

descriptions of such properties indicate "Takes its value from <location>: <setting>; use that setting instead." The remaining properties in the *mts.properties.erb* file -- those that are "hard-coded" with specific values -- are settings that in typical circumstances you should have no need to edit. Therefore **in typical circumstances you should not need to manually edit the *mts.properties.erb* file.**

IMPORTANT: If you do make edits to *mts.properties.erb*, be sure to push your edits to the cluster and restart the S3 Service to apply your changes. Note that restarting the S3 Service automatically restarts the Admin Service as well. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

The *mts.properties.erb* file is divided into these sections:

Cassandra Interfaces

The S3 Service, Admin Service, and HyperStore Service each separately maintain their own pool of connections to the Cassandra data store. The configuration settings in this section are applied separately to each of the three connection pools. For example, if *MaxActive*=10, then the S3 Service to Cassandra, Admin Service to Cassandra, and HyperStore Service to Cassandra connection pools are **each** allowed a maximum of 10 simultaneously active connections.

cassandra.cluster.name

Takes its value from *region.csv*: *cassandra_cluster_name*. Typically you should have no need to edit that file.

cassandra.cluster.Hosts

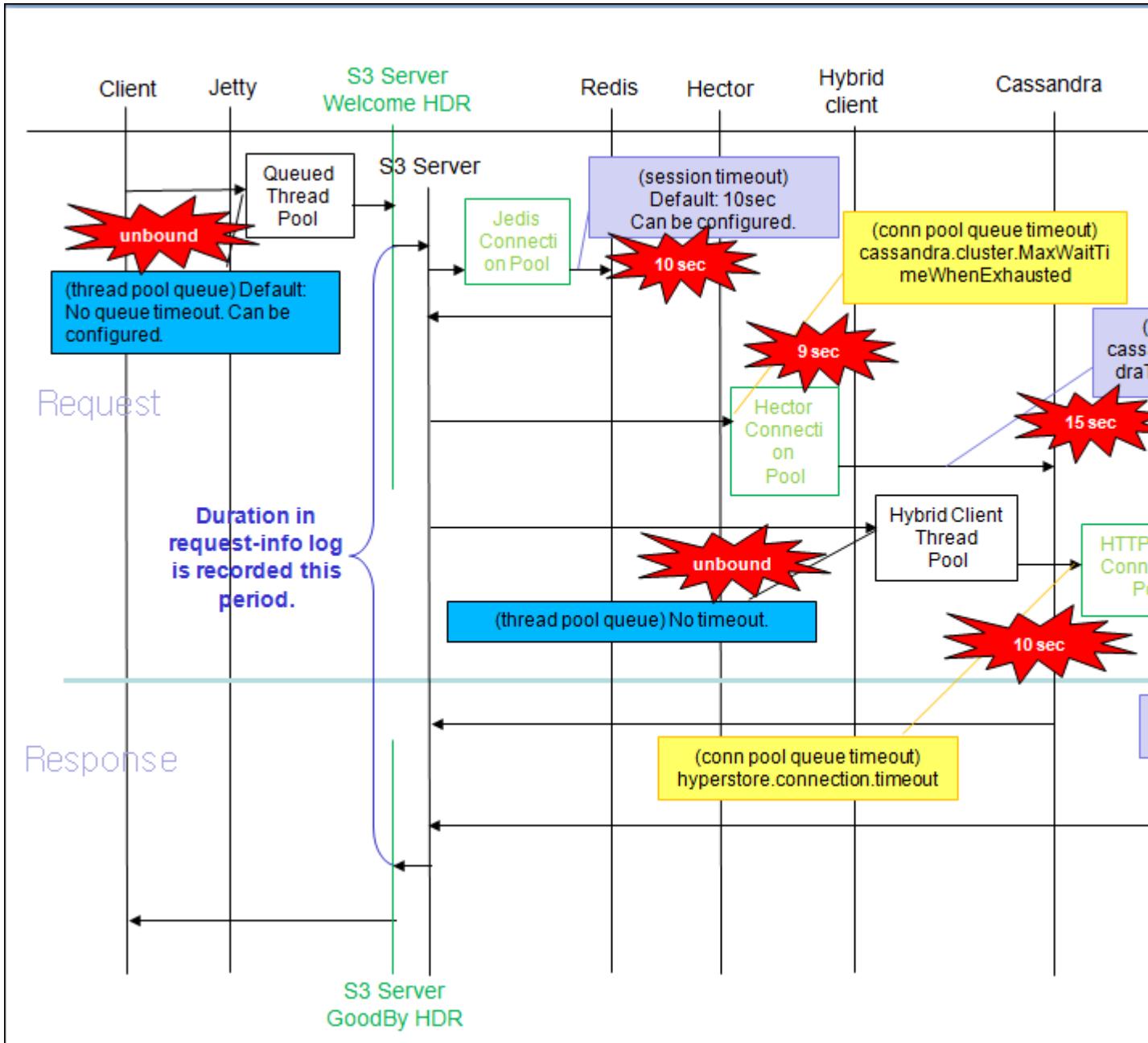
Takes its value from *topology.csv*. Typically you should have no need to edit that file.

cassandra.cluster.CassandraThriftSocketTimeout

After submitting a request to a Cassandra instance via its Thrift socket, the amount of time in milliseconds to wait for a response from Cassandra before failing the operation.

Default = 15000

The diagram below shows the place of *cassandra.cluster.CassandraThriftSocketTimeout* and other timeouts within the S3 Service's request processing flow.



`cassandra.cluster.MaxActive`

Takes its value from `common.csv`: "**cassandra_max_active**" (page 477); use that setting instead.

`cassandra.cluster.MaxIdle`

The maximum allowed number of idle connections in a Cassandra connection pool. Any idle connections in excess of this limit are subject to being closed. Set to a negative value to disable this limit. Note this control is applicable only if `TimeBetweenEvictionRunsMillis` is set to a positive value.

Default = 20

`cassandra.cluster.MaxWaitTimeWhenExhausted`

If `cassandra.cluster.MaxActive` has been reached for a target Cassandra host and a thread requires a new

connection to the host, the thread will wait this many milliseconds before returning an error to the client.

Default = 9000

For a diagram showing the place of this timeout within the S3 request processing flow, see *cassandra.cluster.CassandraThriftSocketTimeout* (above).

cassandra.cluster.RetryDownedHosts

Whether or not to periodically retry Cassandra hosts that have been detected as being down, using a background thread. If set to "true", the retry is attempted at configurable interval *cassandra.cluster.RetryDownedHostsDelayInSeconds*.

Default = true

cassandra.cluster.RetryDownedHostsQueueSize

Maximum number of downed Cassandra hosts to maintain in the downed host retry queue at the same time. If multiple Cassandra nodes are down, and if *cassandra.cluster.RetryDownedHosts=true*, then a queue is maintained for retrying downed nodes. The *cassandra.cluster.RetryDownedHostsQueueSize* sets a limit on the number of nodes that can be in the retry queue simultaneously.

Default = -1 (unlimited)

cassandra.cluster.RetryDownedHostsDelayInSeconds

The number of seconds to wait between retry attempts for downed hosts. Applicable only if

cassandra.cluster.RetryDownedHosts=true.

Default = 10

cassandra.cluster.Lifo

If true, use a "last in, first out" policy for retrieving idle connections from a pool (use the most recently used idle connection). If false, use "first in, first out" retrieval of idle connections (use the oldest idle connection).

Default = true

cassandra.cluster.MinEvictableIdleTimeMillis

The minimum time in milliseconds that a connection must be idle before it becomes eligible for closing due to maximum idle connection limits.

Default = 100000

cassandra.cluster.TimeBetweenEvictionRunsMillis

The interval in milliseconds at which to check for idle connections in a pool, for enforcement of maximum idle connection limits.

Default = 100000

cassandra.cluster.UseThriftFramedTransport

Whether to use framed transport for Thrift. Strongly recommended to leave as true. If set to false, then *thrift_framed_transport_size_in_mb* in *cassandra.yaml* must be set to 0.

Default = true

cassandra.cluster.AutoDiscoverHosts

Whether or not to periodically check for the presence of new Cassandra hosts within the cluster and to use these same settings to interface with those new hosts. If set to true, a check for new hosts is performed at configurable interval `cassandra.cluster.AutoDiscoveryDelayInSeconds`.

Default = true

`cassandra.cluster.AutoDiscoveryDelayInSeconds`

Number of seconds to wait between checks for new hosts. Applicable only if `cassandra.cluster.AutoDiscoverHosts=true`

Default = 60

`cassandra.cluster.RunAutoDiscoveryAtStartup`

Whether or not to perform auto-discovery at cluster start-up. See `cassandra.cluster.AutoDiscoverHosts`.

Default = false

`cassandra.cluster.HostTimeoutCounter`

If a Cassandra node returns more than this many host timeout exceptions within an interval of

`cassandra.cluster.HostTimeoutWindow`, mark the node as temporarily suspended. This setting and the other `HostTimeout*` settings below are applicable only if `cassandra.cluster.UseHostTimeoutTracker=true`.

Default = 3

`cassandra.cluster.HostTimeoutWindow`

If within an interval of this many milliseconds a Cassandra node returns more than `cassandra.cluster.HostTimeoutCounter` host timeout exceptions, mark the node as temporarily suspended.

Default = 1000

`cassandra.cluster.HostTimeoutUnsuspendCheckDelay`

How often to check suspended nodes list to see which nodes should be unsuspended, in seconds.

Default = 10

`cassandra.cluster.HostTimeoutSuspensionDurationInSeconds`

When the periodic check of the suspended nodes list is performed, if a node has been suspended for more than this many seconds, unsuspend the node and place it back in the available pool.

Default = 30

`cassandra.cluster.UseHostTimeoutTracker`

Whether to keep track of how often each Cassandra node replies with a host timeout exception, and to temporarily mark nodes as suspended if their timeout exceptions are too frequent. See the `HostTimeout*` setting descriptions above for details.

Default = true

`cassandra.cluster.UseSocketKeepalive`

Whether to periodically send keep-alive probes to test pooled connections to Cassandra hosts.

Default = true

cassandra.data.directories

Takes its value from *common.csv*: "**cassandra_data_directory**" (page 480); use that setting instead.

cassandra.fs.keyspace

Base name of the Cassandra keyspaces in which object metadata is stored. A storage policy ID is appended to this base name to create a keyspace for a particular storage policy (for example, *UserData_b06c5f9213ae396de1a80ee264092b56*). There will be one such keyspace for each storage policy that you have configured in your system.

Default = *UserData*

cassandra.jmx.port

Cassandra's JMX listening port.

Default = 7199

cassandra.tombstone_cleanup_threshold

Takes its value from *common.csv*: "**cassandra_tombstone_cleanup_threshold**" (page 480); use that setting instead.

cassandra.tombstone_gcgrace

Takes its value from *common.csv*: "**cassandra_tombstone_gcgrace**" (page 481); use that setting instead.

cloudian.repair.eventqueue.create.interval

When **proactive repair** is run on a node, it reads from a queue of node-specific object write failure events that is stored in Cassandra. The object write failure events are timestamped and are bundled together based on the time interval in which they occurred. The *cloudian.repair.eventqueue.create.interval* setting controls the time span (in minutes) that's used for the bundling. For example with the default of 60 a new bundle starts each 60 minutes, if a node is unavailable for longer than this and write failure events for the node are accumulating.

When the node comes back online, the first automatic run of proactive repair will repair the objects from all the interval-based bundles except for the most current one (the in-progress interval, such as the current hour if the default of 60 is being used). That bundle will be processed at the next automatic run of proactive repair. By default proactive repair runs (if needed) every 60 minutes — this frequency is configurable by "**hyper-store.proactiveremove.poll_time**" (page 493).

Default = 60

Consistency Levels for System Metadata

In a distributed storage ring in which data is replicated on more than one node, there is the question of consistency requirements for read and write operations. With the HyperStore system you can configure these requirements:

- Consistency requirements for S3 object data and object metadata are managed through the creation and application of storage policies. For information about creating storage policies, "**Add a Storage Policy**" (page 308).
- Consistency requirements for system metadata stored in Cassandra -- such as usage data and monitoring data -- are configurable in the "Consistency Levels" section of *mts.properties.erb*. For information about system metadata storage, see "**Storage of System Metadata**" (page 145).

cloudian.cassandra.default.ConsistencyLevel.Read

For the Reports keyspace (service usage data), AccountInfo keyspace (user account information), and Monitoring keyspace (system monitoring data), the consistency level to require for read operations. The reads are requested by other HyperStore components such as the S3 Service and the Admin Service.

For consistency level definitions, see "[Consistency Levels](#)" (page 321).

You can optionally use a comma-separated list to implement [Dynamic Consistency Levels](#).

Default = LOCAL_QUORUM,ONE

cloudian.cassandra.default.ConsistencyLevel.Write

For the Reports keyspace (service usage data), AccountInfo keyspace (user account information), and Monitoring keyspace (system monitoring data), the consistency level to require for write operations. The writes are requested by other HyperStore components such as the S3 Service and the Admin Service.

For consistency level definitions, see "[Consistency Levels](#)" (page 321).

You can optionally use a comma-separated list to implement [Dynamic Consistency Levels](#).

Default = QUORUM,LOCAL_QUORUM

IMPORTANT: Since this setting applies to writes, using "ONE" is not recommended.

cloudian.cassandra.UserData.ConsistencyLevel.HyperStore

The consistency level to require when [hsstool](#) is reading or writing to the *UserData_<policyid>* keyspaces in order to implement a repair or cleanup operation. If the configured consistency requirement can't be met, the operation fails and subsequently retries. The operation will abort if two retry attempts fail to achieve the required consistency level.

For consistency level definitions, see "[Consistency Levels](#)" (page 321).

You can optionally use a comma-separated list to implement [Dynamic Consistency Levels](#).

Default = QUORUM

IMPORTANT: Since this setting applies to writes as well as reads, using "ONE" is not recommended.

cloudian.cassandra.localdatacenter

Takes its value from *topology.csv*. Typically you should have no reason to edit that file.

Authorized Services and Namespaces

cloudian.s3.services

Comma-separated list of service names allowed in Authorization header of incoming S3 requests.

Default = AWS

cloudian.s3.default_service

Default service name for the S3 Service to use when incoming S3 requests do not specify a service name in an Authorization header. Must be one from the *cloudian.s3.services* list.

Default = AWS

cloudian.s3.xmlns.AWS

When the service name in the Authorization header is "AWS", the URI for the XML namespace.

Default = <http://s3.amazonaws.com/doc/2006-03-01/>

cloudian.s3.acp.acl.grant.group.alluser.uri.AWS

When the service name in the Authorization header is "AWS", the URI representing the "all users" access control grantee group.

Default = <http://acs.amazonaws.com/groups/global/AllUsers>

cloudian.s3.acp.acl.grant.group.authuser.uri.AWS

When the service name in the Authorization header is "AWS", the URI representing the "authenticated users" access control grantee group.

Default = <http://acs.amazonaws.com/groups/global/AuthenticatedUsers>

cloudian.s3.acp.acl.grant.group.logdelivery.uri.AWS

When the service name in the Authorization header is "AWS", the URI representing the "LogDelivery" access control grantee group. This canned ACL is associated with the S3 bucket logging feature and grants to the system the authority to write bucket logs into a target bucket.

Default = <http://acs.amazonaws.com/groups/s3/LogDelivery>

cloudian.s3.authorizationV4.singleregioncheck

In Cloudian deployments that consist of only one service region, this setting impacts the system's handling of incoming S3 requests that are using AWS Signature Version 4 Authentication:

- If this setting is set to "true", the system will validate that the region name that the client used when creating the request signature (as indicated in the scope information that the client specifies in the request) is in fact the region name for your single region. If not the request will be rejected.
- If set to "false" the system will not validate the region name. Instead, the system calculates and validates the signature using whatever region name is specified in the request.

Default = false

Note Do not set this property to "true" if the S3 service endpoint (URI) for your one region does not include a region name string. For example, you can set this property to "true" -- thereby enabling region name validation --if your S3 endpoint is *s3-tokyo.enterprise.com* (where *tokyo* is the region name), but not if your S3 endpoint is *s3.enterprise.com* (which lacks a region name string).

cloudian.s3.authorizationV4.multiregioncheck

In Cloudian deployments that consist of multiple service regions, this setting impacts the system's handling of incoming S3 requests that are using AWS Signature Version 4 Authentication:

- If this setting is set to "true", the system will validate that the region name that the client used when creating the request signature (as indicated in the scope information that the client specifies in the request) is in fact the region name for the region to which the request has been submitted. If not the request will be rejected.
- If set to "false" the system will not validate the region name. Instead, the system calculates and validates the signature using whatever region name is specified in the request.

Default = false

IMPORTANT: Do not set this property to "true" if the S3 service endpoints (URIs) for your regions do not include region name strings. For example, you can set this property to "true" -- thereby enabling region name validation --if your S3 endpoints are `s3-tokyo.enterprise.com` and `s3-osaka.enterprise.com` (where *tokyo* and *osaka* are region names), but not if your S3 endpoints lack region name strings.

`cloudian.s3.serverside.encryption.keylength`

Takes its value from `common.csv`: "**cloudian_s3_aes256encryption_enabled**" (page 477); use that setting instead.

QoS

The following settings pertain to the implementation of quality of service (QoS) limits for the S3 service. For more detail on the HyperStore QoS feature, see "**Quality of Service (QoS) Feature Overview**" (page 110).

`cloudian.s3.qos.enabled`

Takes its value from "**Enforce Configured QoS Limits for Storage Utilization**" (page 300) in the CMC's [Configuration Settings](#) page; use that setting instead.

`cloudian.s3.qos.rate.enabled`

Takes its value from "**Enforce Configured QoS Limits for Request Rates and Data Transfer Rates**" (page 301) in the CMC's [Configuration Settings](#) page; use that setting instead.

`cloudian.s3.qos.cache.enabled`

When QoS enforcement is enabled, for each S3 request the S3 Service checks current user and group QoS usage against configured QoS limits. The usage counters and configured limits are stored in the Redis QoS DB. This setting if set to "true" enables the S3 Service to cache QoS counter and limits data and to check that cached data first when performing its QoS enforcement role. If there is no cache hit then Redis is checked.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → `QosCacheEnabled`)

`cloudian.s3.qos.cache.expiryms`

For the S3 Service's QoS data cache (if enabled), the cache entry expiry time in milliseconds.

Default = 10000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → `QosCacheExpiryMs`)

`cloudian.s3.qos.storagewrite.batch.enabled`

If set to "true" then the AccountHandler's updates of storage object and storage bytes counters in Redis are batched together.

Default = false

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → `QosStorageWriteEnabled`)

*cloudian.s3.qos.storagewrite.intervalm*s

If batching of storage counter updates is enabled, the batch interval in milliseconds.

Default = 60000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → QosStorageWriteIntervalMs)

cloudian.s3.qos.maxdelay

If a PUT Object or PUT Object Copy operation that is overwriting an existing object takes more than this many milliseconds to complete, then before the system updates the user's Storage Bytes (SB) count it re-checks the existing object's metadata to ensure that the calculated size differential between the existing object and the new object is based on fresh metadata. This configurable behavior is intended to reduce the likelihood of erroneous SB counts resulting from race conditions wherein multiple client requests are overwriting the same object at the same time.

The context is that in the case of an existing object being overwritten by a new instance of the object, the system updates the user's SB count by calculating the delta between the original object's size (as indicated by its metadata) and the new object's size and then applying that difference to the user's SB count. It's important therefore that the calculated delta be based on up-to-date metadata for the object that's being overwritten, even in the case where the writing of the new object takes a long time to complete.

Default = 60000

S3 Service Caching (Bucket Metadata, etc)

cloudian.s3.metadata.cache.enabled

If set to *true*, bucket metadata is cached by the S3 Service. When S3 request processing requires bucket metadata, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → MDCCacheEnabled)

cloudian.s3.group.cache.enabled

If set to *true*, then user group metadata is cached by the S3 Service. When S3 request processing requires group status, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GroupCacheEnabled)

cloudian.s3.credentials.cache.enabled

If set to *true*, users active security credentials are cached by the S3 Service. When S3 request processing

requires a user's credentials, the cache is checked first. If there is no cache hit then the needed credentials retrieved from Redis, and are cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source credentials in Redis change and then automatically invalidates the cached credentials.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GroupCacheEnabled)

cloudian.s3.user.cache.enabled

If set to *true*, user metadata is cached by the S3 Service. When S3 request processing requires a user's status and/or display name, the cache is checked first. If there is no cache hit then the needed metadata is retrieved from Redis, and is cached for subsequent use.

On an ongoing basis, the system detects if the corresponding source metadata in Redis changes and then automatically invalidates the cached metadata.

Default = true

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → UserCacheEnabled)

Domains and Regions

The "region" settings in this section are applicable to multi-region HyperStore deployments. If your HyperStore system does not have multiple service regions, you can leave the settings in this section at their default values.

cloudian.s3.regions

Takes its value from *common.csv*: "**regions**" (page 459); use that setting instead.

cloudian.s3.home_region

Takes its value from *region.csv*: *region_name*; use that setting instead.

cloudian.s3.default_region

Takes its value from *common.csv*: "**default_region**" (page 458); use that setting instead.

cassandra.cluster.name.<region>

Takes its value from *region.csv*: *cassandra_cluster_name*. Typically you should have no reason to edit that setting.

cassandra.cluster.hosts.<region>

Takes its value from *region.csv*: *cassandra_cluster_hosts*. Typically you should have no reason to edit that setting.

cloudian.s3.domain.<region>

Takes its value from *region.csv*: *s3_domain_and_port*. Typically you should have no reason to edit that setting.

cloudian.s3.ssl_domain.<region>

Domain and SSL listening port from one of your regions, in format <*FQDN*>.⟨*port*⟩. This is used if your system has multiple service regions and you have configured your S3 service to use SSL. For example, if a GET

Object request comes in to the S3 service in region1, and the object is stored in region2, and region2 is configured to use SSL for its S3 service — then the S3 service in region1 needs to know the domain and SSL port for the S3 service in region2, so that it can specify a correct Location header in the Redirect message it returns to the requesting client.

If you have a multi-region HyperStore deployment, the installation script automatically configures the *cloudian.s3.ssl_domain.<region>* setting for **each** of your regions, including the local region. For example:

```
cloudian.s3.ssl_domain.region1 = s3.region1.mycompany.com:443
cloudian.s3.ssl_domain.region2 = s3.region2.mycompany.com:443
cloudian.s3.ssl_domain.region3 = s3.region3.mycompany.com:443
```

If you have only one region in your HyperStore deployment, then this setting is not relevant to your system.

Default = commented out

cloudian.s3.website_endpoint

Takes its value from *region.csv*: *cloudians3_website_endpoint*.

To change this, use the HyperStore "**Installer Advanced Configuration Options**" (page 449).

cloudian.util.dns.resolver

Method used to resolve foreign buckets to the correct region, in support of the S3 LocationConstraint feature. Currently only one option is supported:

- com.gemini.cloudian.util.dns.RedirectResolver — Never resolve foreign bucket domain names, always reply with a 307 redirect. The Location value to supply in the redirect response is obtained from the global Redis database.

Default = com.gemini.cloudian.util.dns.RedirectResolver

cloudian.publicurl.port

Takes its value from *common.csv*: *ld_cloudian_s3_port*, which is controlled by the installer. To change S3 listening ports use the installer's Advanced Configuration options.

cloudian.publicurl.sslport

Takes its value from *common.csv*: *ld_cloudian_s3_ssl_port*, which is controlled by the installer. To change S3 listening ports use the installer's Advanced Configuration options.

S3 Service

cloudian.s3.server

HTTP Server header value to return in responses to S3 requests. If you want no Server header value returned in responses to S3 requests, uncomment this setting and set it to empty.

Default = Commented out; uses internal default "CloudianS3"

cloudian.s3.tmpdir

The directory in which to temporarily store large files in order to reduce memory usage.

Default = Commented out; uses internal default *java.io.tmpdir*.

cloudian.fs.read_buffer_size

When interacting with the file storage system in Cassandra, the size of the S3 Service's and Admin Service's

read buffer, in bytes. A larger read buffer can enhance performance but will also consume more memory.

Default = 65536

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → ReadBufferSize)

cloudian.s3.putobject.max_size

Takes its value from the "**Put Object Maximum Size (Bytes)**" (page 304) setting on the CMC's [Configuration Settings](#) page; use that setting instead.

cloudian.s3.getbucket.max_num_of_keys

When performing an S3 GET Bucket operation (which returns meta-data about the objects in the bucket specified in the request), the maximum number of objects to list in the response. If the client request sets a "max-keys" parameter, then the lower of the client-specified value and the *cloudian.s3.getbucket.max_num_of_keys* value is used.

Default = 1000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → GetBucketMaxKeys)

cloudian.s3.max_user_buckets

Takes its value from the "**Maximum Buckets Per User**" (page 305) setting on the CMC's [Configuration Settings](#) page; use that setting instead.

cloudian.s3.delimiter_regex

Regular expression indicating allowed delimiters in getBucketList objects.

Default = .+

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → DelimiterRegex)

cloudian.s3.multipart.maxparts

Takes its value from the "**Multipart Upload Maximum Parts**" (page 305) setting on the CMC's [Configuration Settings](#) page; use that setting instead.

cloudian.s3.multipart.minpartsize

In an S3 Multipart Upload submitted to HyperStore, the required minimum size per part, excluding the last part. Expressed in number of bytes. The operation will fail if a part other than the last part is smaller than this many bytes. For example, if you set this to 5242880 (5MB, which is the Amazon S3 default for minimum part size) then in a Multipart Upload each part uploaded must be at least 5MB in size, except for the last part which is allowed to be as small as necessary to complete the object.

The HyperStore default of 1 byte essentially places no restriction on minimum part size.

Default = 1

cloudian.s3.unsupported

Comma-separated list of Amazon S3 URI parameters that the HyperStore system does not support. This list applies across HTTP methods and across S3 resource types. In response to requests that include an unsupported URI parameter, the HyperStore system will return 501, Not Implemented.

Default = accelerate,requestPayment,analytics,inventory,metrics,select,notification,object-lock

cloudian.util.ntp.Path

Path to *ntpstat*. Setting this value is required only if *ntpstat* is not in the environment PATH.

Default = commented out; internal default = ''

cloudian.util.ntp.MaximumSynchronizationDistance

Maximum system clock skew to allow when servicing S3 requests that require the S3 Service to generate an object versionId, in milliseconds.

- If this setting is set to a positive value, a S3 Service node that is processing S3 requests that require generating a versionId will perform an *ntpstat* check. If the node's system clock is skewed by more than *cloudian.util.ntp.MaximumSynchronizationDistance* milliseconds, the S3 Service rejects the S3 request with a "503 Service Unavailable" error response. The frequency of the *ntpstat* check can be limited by using the *cloudian.util.ntp.CheckIntervalMillis* setting.
- If this setting is set to 0, the *ntpstat* check is not performed when processing S3 requests that require generating a versionId.

Default = 0

cloudian.util.ntp.CheckIntervalMillis

Minimum time between *ntpstat* checks, in milliseconds.

- If this setting is set to a positive value, then when the S3 Service is processing S3 requests that require generating a versionId, an *ntpstat* check will be performed only if *cloudian.util.ntp.CheckIntervalMillis* milliseconds or more have passed since the last *ntpstat* check was performed.
- If this setting is set to 0, an *ntpstat* check is performed with every S3 request that requires generating a versionId.

Default = 60000

cloudian.s3.batch.delete.maxkeys

When HyperStore clients delete S3 objects, the HyperStore system deletes the object metadata immediately but does not delete the actual objects until the next run of the [.system/deleteObject cron job](#) which runs every hour (by default). The *cloudian.s3.batch.delete.maxkeys* setting can be used to set a maximum number of objects to delete in a single run of the [.system/deleteObject](#) cron job.

If you want each [.system/deleteObject](#) cron job run to delete all the objects that are queued for deletion, with no maximum, then set this setting to 0.

Default = 0

cloudian.s3.batch.delete.delay

When executing the the [.system/deleteObject cron job](#), the number of milliseconds to pause in between deletes of individual objects.

To have no delay between individual object deletes, set this setting to 0.

Default = 0

S3 Client

The settings in this section are used by an internal S3 Service client that sends requests to the HyperStore S3

Service in other regions of a [multi-region HyperStore deployment](#). These settings are relevant only if you have a multi-region system.

Note In the configuration file commenting in this section, the specified "defaults" are internal defaults that would be used if these settings were commented out in the file. In the documentation below, the specified "defaults" are the values that are assigned to the settings in the default version of the configuration file after install.

cloudian.s3.client.ConnectionTimeout

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the connection establishment timeout in milliseconds.

Default = 2000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientConnectionTimeout)

cloudian.s3.client.SocketTimeout

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the request processing timeout in milliseconds. When a request is sent over an open connection, if a complete response is not received within this interval, the request times out and the connection is closed.

Default = 50000

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientSocketTimeout)

cloudian.s3.client.MaxErrorRetry

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the maximum number of times to retry a retryable failed request (such as when a 5xx response is received). Set to 0 if you want no retries.

Default = 0

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientMaxErrorRetry)

cloudian.s3.client.UserAgent

When interfacing with the S3 Service in a different Cloudian HyperStore service region, the value of the first part of the HTTP User-Agent header, where the whole header is "<ConfigValue> <AWS SDK Version> <OS Version> <JDK Version>". For example, if you set this setting to "Agent99", then the resulting User-Agent header will be "Agent99, <AWS SDK Version> <OS Version> <JDK Version>", with the latter three values being populated automatically by the system.

Default = Commented out; uses internal default "Cloudian/<version> <default-region>"

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → S3ClientUserAgent)

Reports

cloudian.auditlog.enabled

Do not edit this setting.

reports.raw.ttl

Time-to-live for "raw" S3 usage data in the Reports keyspace in Cassandra, in seconds. Raw service usage data will be automatically deleted this many seconds after its creation. This is set by the system -- **do not manually edit** this setting.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default =

- 604800 (seven days), if the "**Track/Report Usage for Request Rates and Data Transfer Rates**" (page 299) setting in the CMC's [Configuration Settings](#) page is set to "false" (as it is by default)
- 86400 (one day) if the "**Track/Report Usage for Request Rates and Data Transfer Rates**" (page 299) setting is set to "true"

For an overview of how the HyperStore system tracks service usage by groups, users, and buckets, see "**Usage Reporting Feature Overview**" (page 153).

reports.rolluphour.ttl

Time-to-live for hourly roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Hourly roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 5616000 (65 days)

IMPORTANT: This hourly rollup data is the basis for generating billing reports. After hourly rollup data is deleted it is no longer available for generating billing reports.

reports.rollupday.ttl

Time-to-live for daily roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Daily roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 5616000 (65 days)

reports.rollupmonth.ttl

Time-to-live for monthly roll-up S3 usage data in the Reports keyspace in Cassandra, in seconds. Monthly roll-up data will be automatically deleted this many seconds after its creation.

This applies to per-bucket usage data (if you have enabled per-bucket usage tracking) as well as to per-user and per-group usage data.

Default = 15552000 (180 days)

reports.auditdata.ttl

Time-to-live for audit data in the Reports keyspace in Cassandra, in seconds. Audit data will be automatically deleted this many seconds after its creation.

Default = 5616000 (65 days)

events.acknowledged.ttl

Time-to-live for acknowledged system events in the Monitoring keyspace in Cassandra, in seconds. Acknowledged events will be automatically deleted this many seconds after an administrator acknowledges them.

Unacknowledged events are not subject to this timer. The time-to-live countdown on an event record does not begin until an administrator acknowledges the event through the CMC's [Alerts](#) page or [Node Status](#) page.

If you want alerts to be deleted immediately after they have been acknowledged, set this property to 1.

Note As soon an alert occurs a record of it is written to the Smart Support log (which by default is uploaded to Cloudian Support once a day). So the Smart Support record of alerts persists even after the alerts have been deleted from your system. For more information on Smart Support see "[Smart Support and Diagnostics Feature Overview](#)" (page 137).

Default = 86400 (one day)

monitoring.ophistory.ttl

Time-to-live for per-node data repair and cleanup operation status summaries in Cassandra's Monitoring keyspace, in seconds. Each repair and cleanup status summary will be automatically deleted after it has been stored for *monitoring.ophistory.ttl* seconds.

For as long as repair and cleanup status summaries are stored in the Monitoring keyspace, they can be retrieved on a per-node basis by using the [hsstool opstatus](#) command, with the "-q history" option. This returns the operation history of a specified node. The *monitoring.ophistory.ttl* property controls the maximum length of that retrievable operation history -- for example with *monitoring.ophistory.ttl* at its default value, for each HyperStore node you can retrieve the history of its repair and cleanup operations from the past 90 days. Repair and cleanup operation status information older than that will be deleted.

Default = 7776000 (90 days)

Usage

The "Usage" section has settings for tuning the performance of [usage data repair](#) operations.

usage.repair.row.size

When querying Cassandra for the object metadata associated with individual service users, the maximum number of users to retrieve per query.

Default = 1000

usage.repair.column.size

When querying Cassandra for the object metadata associated with individual service users, for each user the maximum number of objects for which to retrieve metadata per query.

Default = 1000

usage.repair.maxdirtyusers

Maximum number of "dirty" users for whom to verify (and if necessary repair) usage data during a single run of the [POST /usage/repair/dirtyusers](#) operation. This operation is triggered every 12 hours by cron job.

Default = 1000

usage.rollup.userchunk.size

When performing rollups of usage data, the maximum number of users to take into memory at a time.

Default = 1000

usage.rollup.usagechunk.size

When performing rollups of usage data, the maximum number of usage records to take into memory at a time.

Default = 1000

usage.rollup.hour.maxretry

Each time the system performs an hourly rollup of usage data for the hour that just ended, it will check whether the hourly rollup data from the **preceding** hours exists (as it should, unless relevant system services have been down or unreachable). If the hourly rollup data from preceding hours is missing, then the system retries processing the hourly rollups for those hours that are missing the hourly rollup data. The *usage.rollup.hour.maxretry* property sets a maximum on the number of preceding hours to check on and (if needed) perform a retry for.

For example, suppose *usage.rollup.hour.maxretry*=6. With this setting, if the system is for example about to perform the hourly rollup from the 10th hour of the day, it will first check that hourly rollup data exists for the 9th, 8th, 7th, 6th, 5th, and 4th hours of the day -- and if any of those hourly rollups are missing, the system will try again to execute those hourly rollups. After doing so the system will then perform the hourly rollup of the usage data from the 10th hour of the day.

Default = 24

cloudian.s3.usagerates.enabled

Takes its value from the **"Track/Report Usage for Request Rates and Data Transfer Rates"** (page 299) setting on the CMC's [Configuration Settings](#) page; use that setting instead.

bucketstats.enabled

Takes its value from *common.csv*: "**bucketstats_enabled**" (page 486); use that setting instead.

cloudian.s3.redis.retry

The interval in seconds at which the S3 Service will retry a Redis node that has been unresponsive. If the S3 Service finds a Redis node to be unresponsive the S3 Service will temporarily remove that node from the list of Redis nodes that are available to service requests. At an interval of *cloudian.s3.redis.retry* seconds the S3 Service will retry the Redis node. If it's found to be responsive, the node is added back to the S3 Service's list of available Redis nodes.

This setting is applicable to Redis Credentials nodes and Redis QoS nodes.

Default = 30

Redis Monitor

The HyperStore **"Redis Monitor Service"** (page 17) is installed automatically with the S3 Service and Admin Service. You can run the Redis Monitor in order to monitor Redis cluster health and implement automatic fail-over of the Redis master node role. For Redis Monitor redundancy, it runs on two of your S3 Service / Admin Service nodes, with the Monitor configured as primary on one node and as backup on the other node.

redis.monitor.primary.pollInterval

The interval at which the backup Redis Monitor instance should check on the health of the primary instance, in

seconds. If the primary Redis Monitor instance is unresponsive, the backup instance takes over the monitoring duties.

Default = 5

redis.credentials.cluster.pollInterval

Interval at which the Redis Monitor application should check the health of the Redis Credentials servers, in seconds. At this interval, the Redis Monitor also checks the S3 Service / Admin Service nodes via JMX to ensure that they are configured to point to the current Redis Credentials master, and updates their configuration if necessary.

Default = 5

redis.credentials.cluster.client.request.waittime

Maximum time for the Redis Monitor to wait for a JMX connection attempt to a S3 Service / Admin Service node to complete, in seconds. If the connection attempt doesn't complete (with a success or failure result) within this interval, the Redis Monitor marks the S3 Service / Admin Service node as DOWN and writes an INFO level message to *cloudian-redismon.log*. Meanwhile, the connection attempt will continue until completion, and subsequently polling of the S3 Service / Admin Service node will resume at the regular polling interval.

The *redis.credentials.cluster.client.request.waittime* value must be smaller than the *redis.credentials.cluster.pollInterval* value.

Default = 3

Credentials

credentials.user.max

Maximum allowed number of S3 credentials per HyperStore user. Each credential is a key pair consisting of a public key (access key) and a private key (secret key). These credentials enable a HyperStore user to access the HyperStore S3 storage system through either the CMC or a third party S3 client.

Inactive credentials count toward this maximum as well as active credentials. Credentials can be created, made active or inactive, and deleted, through either the CMC or the Admin API.

Note If a HyperStore user creates IAM users under their HyperStore account and creates S3 credentials for those IAM users, the IAM users' credentials do **not** count toward the HyperStore user's maximum allowed number of S3 credentials. IAM user credentials are limited separately, by the *credentials.iamuser.max* property.

Default = 5

credentials.iamuser.max

Maximum allowed number of S3 credentials per IAM user. Each credential is a key pair consisting of a public key (access key) and a private key (secret key). These credentials enable an IAM user to access the HyperStore S3 storage system through a third party S3 client. IAM users cannot access the HyperStore S3 storage system through the CMC.

Inactive credentials count toward this maximum as well as active credentials. Credentials can be created, made active or inactive, and deleted, through either the CMC's **IAM User** section (which is accessible only to HyperStore group administrators or regular users -- not system administrators) or the HyperStore implementation of the Amazon IAM API.

Default = 2

keystore.pass

Password for the Java keystore file `/opt/cloudian/conf/.keystore`. This keystore file stores the Admin Service's pre-generated, self-signed, RSA-based public and private keys for SSL.

Default = adminpass

secure.transact.alias

Alias identifying the Admin Service's certificate entry within the keystore.

Default = secure

secure.transact.pass

Password to access the certificate entry that's identified by *secure.transact.alias*.

Default = private

admin.auth.realm

Takes its value from *common.csv*: "**admin_auth_realm**" (page 471); use that setting instead.

admin.auth.enabled

Takes its value from *common.csv*: "**admin_auth_enabled**" (page 471); use that setting instead.

admin.secure

Takes its value from *common.csv*: "**admin_secure**" (page 471); use that setting instead.

admin.user.password.length

Maximum allowed character length for users' Cloudian Management Console login passwords.

Default = 64

awsmms.proxy.host

Takes its value from *common.csv*: "**awsmmsproxy_host**" (page 486); use that setting instead.

awsmms.proxy.port

If you are using the AWS Marketplace Metering Service version of Cloudian HyperStore, this is the port that HyperStore connects to on your AWS Proxy Server. HyperStore transmits usage data to the AWS Proxy Server, which in turn relays the data to the AWS Marketplace Metering Service for billing purposes.

Default = 17081

admin.whitelist.enabled

Takes its value from *common.csv*: "**admin_whitelist_enabled**" (page 481); use that setting instead.

HyperStore Service

The HyperStore Service section configures S3 Service behavior in its role as a client of the [HyperStore Service](#).

hyperstore.endport

HyperStore Service listening port to which the S3 Service will submit data operation requests.

Default = Takes its value from elsewhere within the Puppet manifest structure; default is 19090

hyperstore.maxthreads.read

Takes its value from *common.csv*: "**hyperstore.maxthreads.read**" (page 463); use that setting instead.

hyperstore.maxthreads.write

Takes its value from *common.csv*: "**hyperstore.maxthreads.write**" (page 463); use that setting instead.

hyperstore.maxthreads.repair

Takes its value from *common.csv*: "**hyperstore.maxthreads.repair**" (page 463); use that setting instead.

hyperstore.snd.buffer

Socket send buffer size from S3 nodes to HyperStore nodes.

Default = 0

hyperstore.rcv.buffer

Socket receive buffer size from S3 nodes to HyperStore nodes.

Default = 0

hyperstore.timeout

Takes its value from *common.csv*: "**hyperstore_timeout**" (page 463); use that setting instead.

hyperstore.connection.timeout

Takes its value from *common.csv*: "**hyperstore_connection_timeout**" (page 463); use that setting instead.

hyperstore.maxtotalconnections

Takes its value from *common.csv*: *hyperstore_maxtotalconnections*, which is controlled by a performance optimization script that runs automatically when you install your cluster or resize your cluster.

hyperstore.maxperrouteconnections

Takes its value from *common.csv*: *hyperstore_maxperrouteconnections*, which is controlled by a performance optimization script that runs automatically when you install your cluster or resize your cluster

Process Diagnostics (Phone Home / Smart Support)

phonehome.enabled

The HyperStore Data Collector collects and stores system-wide diagnostic data for your HyperStore system on an ongoing basis. By default this diagnostics data is automatically uploaded to Cloudian Support via the S3 protocol once a day, as part of the Smart Support feature. For more information about this feature -- including what data gets sent to Cloudian Support and how they use it for your benefit -- see "**Smart Support and Diagnostics Feature Overview**" (page 137).

- If you want diagnostics data automatically uploaded to Cloudian Support via S3 each day, there's nothing you need to do -- just leave *phonehome.enabled* set to "true" and leave the setting *common.csv: phonehome_uri* set to the Cloudian Support S3 URI. This is the default behavior and is the recommended behavior.
- If you want diagnostics data automatically uploaded each day to an S3 destination other than Cloudian Support, leave *phonehome.enabled* set to "true" and set the *common.csv: phonehome_uri*

setting to the desired S3 URI (and also set *common.csv*: *phonehome_{bucket, access_key, secret_key}*).

- If you do not want diagnostics data automatically uploaded to an S3 destination each day, set *phonehome.enabled* to "false". This is not recommended.

Even if you choose not to automatically upload the daily diagnostic data to an S3 destination -- that is, even if you set *phonehome.enabled* to "false" -- a diagnostics data file is still generated locally and stored under */var/log/cloudian* on the node on which the HyperStore Monitoring Data Collector runs. (To see which of your nodes is the Data Collector node, go to the CMC's [Cluster Information](#) page and check for the identity of the "System Monitoring/Cronjob Primary Host".) The "live" diagnostics log -- which is recording the current day's performance statistics -- is named *diagnostics.csv*. The rolled up daily diagnostic packages from previous days -- which include prior days' *diagnostics.csv* files and also various application and transaction logs -- are named *diagnostics_<date/time>_<version>_<region>.tgz*.

Note The deletion of old diagnostics packages is managed by Puppet, as configured by "**cleanup_directories_byage_withmatch_timelimit**" (page 460) in *common.csv*. By default Puppet deletes the diagnostics packages after they are 15 days old. This presumes that you have left the Puppet daemons running in your HyperStore cluster, which is the default behavior. If you do not leave the Puppet daemons running the diagnostics logs will not be automatically deleted. In that case you should delete the old packages manually, since otherwise they will eventually consume a good deal of storage space.

Default = true

phonehome.uri

Takes its value from *common.csv*: "**phonehome_uri**" (page 470); use that setting instead.

phonehome.{bucket, accessKey, secretKey}

Take their values from *common.csv*: "**phonehome_bucket**" (page 470); use those settings instead.

Default = empty

phonehome.proxy.host

Takes its value from *common.csv*: "**phonehome_proxy_host**" (page 469); use that setting instead.

phonehome.proxy.port

Takes its value from *common.csv*: "**phonehome_proxy_port**" (page 469); use that setting instead.

phonehome.proxy.username

Takes its value from *common.csv*: "**phonehome_proxy_username**" (page 469); use that setting instead.

phonehome.proxy.password

Takes its value from *common.csv*: "**phonehome_proxy_password**" (page 469); use that setting instead.

phonehome.gdpr

Takes its value from *common.csv*: "**phonehome_gdpr**" (page 470); use that setting instead.

System Info Log Upload

sysinfo.uri

S3 URI to which to upload on-demand Node Diagnostics packages, when you use the CMC's [Collect](#)

[Diagnostics](#) function. By default this is the S3 URI for Cloudian Support, but if you prefer you can set this to a different S3 URI. For an overview of this feature see "**Smart Support and Diagnostics Feature Overview**" (page 137).

Include the HTTP or HTTPS protocol part of the URI (*http://* or *https://*).

Default = `https://s3-support.cloudian.com:443`

Note If you set `sysinfo.uri` to a URI for your own HyperStore S3 storage system (rather than Cloudian Support), and if your S3 Service is using HTTPS, then your S3 Service's SSL certificate must be a CA-verified, trusted certificate — not a self-signed certificate. By default the Node Diagnostics upload function cannot upload to an HTTPS URI that's using a self-signed certificate. If you require that the upload go to an HTTPS URI that's using a self-signed certificate, contact Cloudian Support.

`sysinfo.{bucket, accessKey, secretKey}`

- If you are using the Node Diagnostics upload feature to upload node diagnostic data to **Cloudian Support** via S3, you can leave the `sysinfo.bucket`, `sysinfo.accessKey`, and `sysinfo.secretKey` properties empty. The Node Diagnostics feature will automatically extract the Cloudian Support S3 bucket name and security credentials from your encrypted HyperStore license file. You would only modify these configuration properties if instructed to do so by Cloudian Support.
- If you set `sysinfo.uri` to **your own S3 URI** (rather than the Cloudian Support URI), set the `sysinfo.bucket`, `sysinfo.accessKey`, and `sysinfo.secretKey` properties to the destination bucket name and the applicable S3 access key and secret key.

Default = empty

`sysinfo.proxy.host`

Takes its value from `common.csv`: "**phonehome_proxy_host**" (page 469); use that setting instead.

Note This property, together with the other `sysinfo.proxy.*` properties below, is for using a local forward proxy when sending Node Diagnostics packages to Cloudian Support (or another external S3 destination). By default these properties will inherit the same `common.csv` values that you set for proxying of the daily Smart Support upload (also known as "phone home"). If you want to use a different proxy for sending Node Diagnostics packages -- not the same proxy settings that you use for the phone home feature -- edit the `sysinfo.proxy.*` settings directly in `mts.properties.erb`. For example you could change `sysinfo.proxy.host=<%= @phonehome_proxy_host %>` to `sysinfo.proxy.host=proxy2.enterprise.com`.

`sysinfo.proxy.port`

Takes its value from `common.csv`: "**phonehome_proxy_port**" (page 469); use that setting instead.

`sysinfo.proxy.username`

Takes its value from `common.csv`: "**phonehome_proxy_username**" (page 469); use that setting instead.

`sysinfo.proxy.password`

Takes its value from `common.csv`: "**phonehome_proxy_password**" (page 469); use that setting instead.

`s3.client.timeout`

When uploading a Node Diagnostics package to an S3 destination such as Cloudian Support, the socket

timeout in milliseconds.

Default = 1800000

s3.upload.part.minsize

When HyperStore uses Multipart Upload to transmit a Node Diagnostics package to an S3 destination such as Cloudian Support, each of the parts will be this many bytes or larger — with the exception of the final part, which may be smaller. For example, if Multipart Upload is used for an 18MB object, and the configured minimum part size is 5MB, the object will be transmitted in four parts of size 5MB, 5MB, 5MB, and 3MB.

Default = 5242880 (5MB)

s3.upload.part.threshold

When HyperStore transmits a Node Diagnostics package to an S3 destination such as Cloudian Support, it uses Multipart Upload if the package is larger than this many bytes.

Default = 16777216 (16MB)

Protection Policies (Storage Policies)

cloudian.protection.policy.max

Maximum number of bucket protection policies (storage policies) that the system will support. Policies with status "Active", "Pending", or "Disabled" count toward this system limit.

If the policy maximum has been reached, you will not be able to create new policies until you either delete existing policies or increase the value of *cloudian.protection.policy.max*.

Default = 25

Reloadable via JMX (S3 Service's JMX port 19080; MBean attribute = com.gemini.cloudian.s3 → Configuring → Attributes → MaxProtectionPolicies)

For more information about storage policies, see "**Storage Policies Feature Overview**" (page 140).

User Agent for Tiering

cloudian.tiering.useragent

Takes its value from *common.csv*: "**cloudian_tiering_useragent**" (page 474); use that setting instead.

SSL for SSE-C

cloudian.s3.ssec.usessl

This setting controls whether the S3 servers will require that incoming S3 requests use HTTPS (rather than regular HTTP) connections when the request is using Server Side Encryption with Customer-provided encryption keys (SSE-C). Leaving this setting at its default of "true" -- so that the S3 servers require HTTPS connections for such requests -- is the recommended configuration. The only circumstance in which you might set this to "false" is if:

- You are using a load balancer in front of your S3 servers -- and the load balancer, when receiving an incoming HTTPS request from clients, terminates the SSL and uses regular HTTP to connect to an S3 server over your internal network.
- You trust your internal network to safely transport users' encryption keys from the load balancer to the S3 servers over regular HTTP.

For background information about HyperStore support for server-side encryption (SSE and SSE-C), see ["Server-Side Encryption Feature Overview" \(page 117\)](#).

Default = true

AWS Region for AWS-KMS

util.awskmsutil.region

Amazon Web Services (AWS) service region to use if you are configuring your HyperStore system to support AWS-KMS as a method of server-side encryption. For complete instructions see ["Using Server-Side Encryption with AWS KMS" \(page 123\)](#).

Default = us-east-1

Tracker for Torrent

cloudian.s3.torrent.tracker

If you want your service users to be able to use BitTorrent for object retrieval, use this property to specify the URL of a BitTorrent "tracker" (a server that keeps track of the clients that have retrieved a particular object and makes this information available to other clients retrieving the object). This can be a tracker that you implement yourself or one of the many public BitTorrent trackers. HyperStore itself does not provide a tracker.

This must be a single URL, not a list of URLs.

The tracker URL that you specify here will be included in the torrent file that the HyperStore S3 Server returns to clients when they submit a ["GET Object torrent" \(page 934\)](#) request.

Default = commented out

Elasticsearch Integration

*cloudian.elasticsearch.**

For information about the *cloudian.elasticsearch.** settings other than *cloudian.elasticsearch.process.type* and *cloudian.elasticsearch.http.url*, see ["Enabling Elasticsearch Integration for Object Metadata" \(page 108\)](#).

cloudian.elasticsearch.process.type

If you change *cloudian.elasticsearch.process.type=elasticsearch* (the default) to *cloudian.elasticsearch.process.type=http* -- and push the configuration out to the cluster and restart the S3 Service -- then instead of transmitting object metadata to an Elasticsearch cluster, HyperStore will submit that data to an HTTP server that you specify with the *cloudian.elasticsearch.http.url* setting. HyperStore will transmit the data in the form of HTTP POST requests, with JSON-formatted metadata in the request body.

Here is an example of a POST request. In this example the request body contains the metadata associated with a PUT Object operation that was processed in HyperStore.

```
POST / HTTP/1.1.  
Content-type: application/json.  
Content-Length: 209.  
Host: 10.20.2.64:9000.  
Connection: Keep-Alive.  
User-Agent: Apache-HttpClient/4.5.2 (Java/1.8.0_172).  
Accept-Encoding: gzip,deflate.  
.{"id":"buser1%2Fabc","bucketName":"buser1","objectName":"abc",
```

```
"lmt":"2019-06-26T08:23:16.224Z","objectSize":1,
"contentType":"application/octet-stream","esTime":"Wed Jun 26 16:23:16 CST 2019",
"userMetadata":{}}
```

Default = elasticsearch

cloudian.elasticsearch.http.url

For information about using this property, see the description of the *cloudian.elasticsearch.process.type* property (above).

Default = empty

Node Status Configuration

In your storage cluster the [S3 Service](#) runs on each node and so too does the [HyperStore Service](#). S3 requests incoming to your cluster are distributed among the S3 Service instances, and in processing an S3 request an S3 Service instance sends write or read requests to the HyperStore Service instances on multiple nodes (such as when storing a new object in accordance with a 3X replication storage policy, for example).

The settings in the "Node Status Configuration" section of *mts.properties.erb* configure a feature whereby the S3 Service on any node will mark the HyperStore Service on any node as being "Down" if recent requests to that HyperStore Service node have failed at a rate in excess of defined thresholds. When an S3 Service node marks a HyperStore Service node as Down, that S3 Service node will temporarily stop sending requests to that HyperStore Service node. This prevents a proliferation of log error messages that would likely have resulted if requests continued to be sent to that HyperStore Service node, and also allows for the implementation of fall-back consistency levels in the case of storage policies configured with "[Dynamic Consistency Levels](#)" (page 29).

The configurable thresholds in this section are applied by each individual S3 Service node -- so that each individual S3 Service node makes its own determination of when a problematic HyperStore Service node should be marked as Down.

An S3 Service node will mark a HyperStore Service node as Down in either of these conditions:

- The **number of timeout error responses** from a HyperStore Service node has exceeded *hss.timeout.count.threshold* (default = 10) over a period of *hss.timeout.time.threshold* number of seconds (default = 300) and also the percentage of error responses of any type from that HyperStore Service node has exceeded *hss.fail.percentage.threshold* (default = 50) over the past *hss.fail.percentage.period* number of seconds (default = 300).
- The **number of other types of error responses** from a HyperStore Service node has exceeded *hss.fail.count.threshold* (default = 10) over a period of *hss.fail.time.threshold* number of seconds (default = 300) and also the percentage of error responses of any type from that HyperStore Service node has exceeded *hss.fail.percentage.threshold* (default = 50) over the past *hss.fail.percentage.period* number of seconds (default = 300).

So by default an S3 Service node will mark a HyperStore Service node as Down if during a five minute period the HyperStore Service node has returned either more than 10 timeout responses or more than 10 error responses of other types, while during that same five minute period more than half the requests that the S3 Service node has sent to that HyperStore Service node have failed.

Note that these triggering conditions are based on a combination of number of error responses and percentage of error responses from the problematic HyperStore Service node. This approach avoids marking a HyperStore Service node as down in circumstances when a high percentage of a very small number of

requests fail, or when the number of failed requests is sizable but constitutes only a small percentage of the total requests.

Once an S3 Service node has marked a HyperStore Service node as Down, that Down status will persist for `hss.bring.back.wait` number of seconds (default = 300) before the Down status is cleared and that S3 Service node resumes sending requests to that HyperStore Service node.

Note If the S3 Service node is restarted during this interval, then the Down status for that HyperStore Service node will be lost and the S3 Service node upon restarting will resume sending requests to that HyperStore Service node.

Note There is **special handling** in the event that a HyperStore Service node returns a "Connection Refused" error to an S3 Service node (such as would happen if the HyperStore Service was stopped on the target node). In this case the S3 Service node immediately marks that HyperStore Service node as being down, and will then resume sending requests to that HyperStore Service node after a wait period of 15 seconds. This behavior is not configurable.

Proactive repair queue restriction

`hyperstore.proactiverepair.queue.max.time`

When eventual consistency for writes is used in the system -- that is, if you have storage policies for which you have configured the write consistency level to be something less strict than ALL -- S3 writes may succeed in the system even in circumstances when one or more write endpoints are unavailable. When this happens the system's proactive repair feature queues information about the failed endpoint writes, and automatically executes those writes later -- on an hourly interval (by default), without operation intervention. For more information about proactive repair see "**Proactive Repair**" (page 76).

The proactive repair feature's queueing mechanism entails writing metadata to Cassandra, which is subsequently removed when the endpoint writes are executed by proactive repair. To avoid over-burdening Cassandra with proactive queueing data it's best if a cap be placed on how long the queueing can go on for in a given instance of a write endpoint being unavailable. The `hyperstore.proactiverepair.queue.max.time` property sets this cap, in minutes.

If a node has been unavailable for more than `hyperstore.proactiverepair.queue.max.time` minutes, the system stops writing to the proactive repair queue for that node, an error is logged in the S3 application log, and an alert is generated in the CMC. As indicated by the alert, in this circumstance after the node comes back online you need to wait for proactive to complete on the node (you can monitor this in the CMC's [Repair Status](#) page) and then you must manually initiate a full repair on the node (see [hsstool repair](#) and [hsstool repairec](#)).

Note that once the node is back up, the timer is reset to 0 in terms of counting against the `hyperstore.proactiverepair.queue.max.time` limit. So if that subsequently node goes down again, proactive repair queueing would again occur for that node for up to `hyperstore.proactiverepair.queue.max.time` minutes.

Default = 240

Note To disable this limit -- so that there is no limit on the time for which proactive repair queueing metadata can build up for a node that's unavailable -- set `hyperstore.proactiverepair.queue.max.time` to 0.

Bucket Share API

`cloudian.s3.enablesharedbucket`

To enable the HyperStore S3 API extension that allows an S3 user to list all the buckets that have been shared with him or her, set this property to *true*.

Default = false

Note For information about the relevant S3 API call and how to use the extension, see "**GET Service**" (page 892).

9.5.4. *mts-ui.properties.erb*

The *mts-ui.properties* file configures the Cloudian Management Console server (CMC). On each of your HyperStore nodes, the file is located at the following path by default:

`/opt/tomcat/webapps/Cloudian/WEB-INF/classes/mts-ui.properties`

Do not directly edit the *mts-ui.properties* file on individual HyperStore nodes. Instead, if you want to make changes to the settings in this file, edit the configuration template file *mts-ui.properties.erb* on the Puppet master node:

`/etc/cloudian-<version>-puppet/modules/cmc/templates/mts-ui.properties.erb`

Certain *mts-ui.properties.erb* properties take their values from settings in [common.csv](#) or from settings that you can control through the CMC's [Configuration Settings](#) page. In the *mts-ui.properties.erb* file these properties' values are formatted as bracket-enclosed variables, like `<%= ... %>`. In the property documentation below, the descriptions of such properties indicate "Takes its value from <location>: <setting>; use that setting instead." The remaining properties in the *mts-ui.properties.erb* file -- those that are "hard-coded" with specific values -- are settings that in typical circumstances you should have no need to edit. Therefore **in typical circumstances you should not need to manually edit the *mts-ui.properties.erb* file**

IMPORTANT: If you do make edits to *mts-ui.properties.erb*, be sure to push your edits to the cluster and restart the CMC to apply your changes. For instructions see "**Pushing Configuration File Edits to the Cluster and Restarting Services**" (page 454).

The *mts-ui.properties.erb* file has the settings below.

`admin.host`

Takes its value from *common.csv*: "**cmc_admin_host_ip**" (page 481); use that setting instead.

`admin.port`

Takes its value from *common.csv*: `id_cloudian_s3_admin_port`, which is controlled by the installer.

`admin.secure`

Takes its value from *common.csv*: "**admin_secure**" (page 471); use that setting instead.

`admin.secure.port`

Takes its value from *common.csv*: "**cmc_admin_secure_port**" (page 472); use that setting instead.

admin.secure.ssl

Takes its value from *common.csv*: "**cmc_admin_secure_ssl**" (page 483); use that setting instead.

iam.enabled

Takes its value from *common.csv*: "**iam_service_enabled**" (page 472); use that setting instead.

iam.port

Takes its value from *common.csv*: "**iam_port**" (page 472); use that setting instead.

iam.secure.port

Takes its value from *common.csv*: "**iam_secure_port**" (page 473); use that setting instead.

iam.secure

Takes its value from *common.csv*: "**iam_secure**" (page 472); use that setting instead.

iam.socket.timeout

If during an HTTP/S connection with the IAM Service this many milliseconds pass without any data being passed back by the IAM Service, the CMC will drop the connection.

Default = 30000

iam.max.retry

If when trying to initiate an IAM request the CMC fails in an attempt to connect to the IAM Service, the CMC will retry this many times before giving up.

Default = 3

web.secure

Takes its value from *common.csv*: "**cmc_web_secure**" (page 482); use that setting instead.

web.secure.port

Takes its value from *common.csv*: "**cmc_https_port**" (page 482); use that setting instead.

web.nonsecure.port

Takes its value from *common.csv*: "**cmc_http_port**" (page 482); use that setting instead.

storageuri.ssl.enabled

Takes its value from *common.csv*: "**cmc_storageuri_ssl_enabled**" (page 483); use that setting instead.

path.style.access

Takes its value from *common.csv*: "**path_style_access**" (page 461); use that setting instead.

application.name

Takes its value from *common.csv*: "**cmc_application_name**" (page 483); use that setting instead.

s3.client.timeout

Socket timeout on requests from the CMC to the S3 Service, in milliseconds.

Default = 1800000

s3.upload.part.minsize

When the CMC uses Multipart Upload to transmit an object to the S3 Service, each of the parts will be this many bytes or larger — with the exception of the final part, which may be smaller. For example, if Multipart Upload is used for an 18MB object, and the configured minimum part size is 5MB, the object will be transmitted in four parts of size 5MB, 5MB, 5MB, and 3MB.

Default = 5242880 (5MB)

s3.upload.part.threshold

When a CMC user uploads an object larger than this many bytes, the CMC uses Multipart Upload to transmit the object to the S3 Service, rather than PUT Object.

Default = 16777216 (16MB)

query.maxrows

For the CMC's **Usage By Users & Groups** page, the maximum number of data rows to retrieve when processing a usage report request.

Default = 100000

page.size.default

For the CMC's **Usage By Users & Groups** page, for usage report pagination, the default number of table rows to display on each page of a tabular report.

Default = 10

page.size.max

For the CMC's **Usage By Users & Groups** page, for usage report pagination, the maximum number of table rows that users can select to display on each page of a tabular report.

Default = 100

list.multipart.upload.max

In the CMC's **Objects** page, when a user is uploading multiple objects each of which is large enough to trigger the use of the S3 multipart upload method, the maximum number of multipart upload objects for which to simultaneously display upload progress.

For example, with the default value of 1000, if a user is concurrently uploading 1005 objects that require the use of the multipart upload method, the CMC's **Objects** page will display uploading progress for 1000 of those objects.

Default = 1000

graph.datapoints.max

For the CMC's **Usage By Users & Groups** page, the maximum number of datapoints to include within a graphical report.

Default = 1000

csv.rows.max

For the CMC's **Usage By Users & Groups** page, the maximum number of rows to include within a comma-separated value report.

Default = 1000

fileupload.abort.max.hours

When a very large file is being uploaded through the CMC, the maximum number of hours for the CMC to wait for the S3 file upload operation to complete. If this maximum is reached, the upload operation is aborted.

Default = 3

license.request.email

Email address to which to send Cloudian license requests. This address is used in a request license information link in the CMC interface.

Default = cloudian-license@cloudian.com

admin.auth.user

Takes its value from *common.csv*: "**admin_auth_user**" (page 471); use that setting instead.

admin.auth.pass

Takes its value from *common.csv*: "**admin_auth_pass**" (page 471); use that setting instead.

admin.auth.realm

Takes its value from *common.csv*: "**admin_auth_realm**" (page 471); use that setting instead.

acl.grantee.public

In the CMC UI dialogs that let CMC users specify permissions on S3 buckets, folders, or files, the label to use for the ACL grantee "public". If the *acl.grantee.public* property is not set in *mts-ui.properties*, then the system instead uses the *acl.grantee.public* value from your *resources_xx_XX.properties* files (for example, for the U.S. English version of the UI, the value is in *resources_en_US.properties*).

Default = commented out (value is taken from resource file[s])

acl.grantee.cloudianUser

In the CMC UI dialogs that let CMC users specify permissions on S3 buckets, folders, or files, the label to use for the ACL grantee "all Cloudian HyperStore service users". If the *acl.grantee.cloudianUser* property is not set in *mts-ui.properties*, then the system instead uses the *acl.grantee.cloudianUser* value from your *resources_xx_XX.properties* files (for example, for the U.S. English version of the UI, the value is in *resources_en_US.properties*).

Default = commented out (value is taken from resource file[s])

session.timedout.url

URL of page to display if a CMC user's login session times out.

If this value is not set in *mts-ui.properties*, the behavior defaults to displaying the CMC Login screen if the user's session times out.

Default = commented out (CMC Login screen displays)

admin.manage_users.enabled

This setting controls whether the **Manage Users** function will be enabled in the CMC GUI. For a screen shot and description of this function, see "**Manage Users**" (page 229).

Options are:

- true — This function will display for users logged in as a system administrator or group administrator. For group admins this function is restricted to their own group.
- false — This function will not display for any users. If you set this to "false" then the **Manage Users** functionality as a whole is disabled and the more granular *admin.manage_users.*.enabled* properties below are ignored.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note If you want to enable some aspects of the **Manage Users** function and not others, you can have *admin.manage_users.enabled* set so that the function is enabled for your desired user types, and then use the granular *admin.manage_users.*.enabled* properties below to enable/disable specific capabilities.

admin.manage_users.create.enabled

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to create new users.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

admin.manage_users.edit.enabled

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to edit existing users' profiles and service attributes.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note Regardless of how you configure the *admin.manage_users.edit.enabled* setting:

- * Nobody but a system administrator can ever change a user's rating plan assignment.
- * Regular users can edit their own profile information, in the **Profile** page of the CMC.

`admin.manage_users.delete.enabled`

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to delete users.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`admin.manage_users.viewuserdata.enabled`

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability of admin users to access regular users' storage buckets. When allowed this access, admin users can view regular users' data, add data to users' buckets, and delete users' data. Also when allowed this access, admin users can change the properties of regular users' buckets and objects.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any admin users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note If you set `fips_enabled` in `common.csv` to `true`, then the system automatically will set `admin.-manage_users.viewuserdata.enabled` to `false` so that no admin users have access to regular users' data through the CMC. If `fips_enabled` in `common.csv` is set to `false` -- as it is by default -- then it doesn't affect the `admin.manage_users.viewuserdata.enabled` setting.

Note Regardless of how the `admin.manage_users.viewuserdata.enabled` setting is configured, regular users can view and manage their own object data through the **Buckets & Objects** section of the CMC.

`admin.manage_users.edit.user_credentials.enabled`

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to change users' CMC login passwords and to view and manage user's S3 access credentials.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.

- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note Regardless of how you configure the `admin.manage_users.edit.user_credentials.enabled` setting, regular users can manage their own CMC login password and S3 access credentials, in the **Security Credentials** page of the CMC.

`admin.manage_users.edit.user_qos.enabled`

Within the **Manage Users** function in the CMC GUI, this setting enables or disables the capability to set Quality of Service (QoS) controls for specific users.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note This setting is relevant only if the `admin.manage_users.enabled` and `admin.manage_users.edit.enabled` settings are enabled.

`admin.manage_groups.enabled`

This setting controls whether the **Manage Groups** function will be enabled in the CMC GUI. For a screen shot and description of this function, see "**Manage Groups**" (page 237).

Options are:

- true — This function will display for users logged in as a system administrator or group administrator. For group admins this function is restricted to their own group.
- false — This function will not display for any users. If you set this to "false" then the **Manage Groups** functionality as a whole is disabled and the more granular `admin.manage_groups.*.enabled` properties below are ignored.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note If you want to enable some aspects of the **Manage Groups** function and not others, you can have `admin.manage_groups.enabled` set so that the function is enabled for your desired user types, and

then use the granular `admin.manage_groups.*.enabled` properties below to enable/disable specific capabilities.

`admin.manage_groups.create.enabled`

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to create new groups.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`admin.manage_groups.edit.enabled`

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to edit an existing group's profile and service attributes.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.

Note: Even when this capability is enabled for group admins, they will not be able to perform certain group-related actions that are reserved for system admins, such as setting QoS controls for the group as a whole or assigning a default rating plan for the group. Group admins' privileges will be limited to changing their group description and changing the default user QoS settings for the group. The latter capability is controlled by a more granular configuration property `admin.manage_groups.user_qos_groups_default.enabled` (below), which defaults to "true".

- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`admin.manage_groups.delete.enabled`

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to delete a group.

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`admin.manage_groups.user_qos_groups_default.enabled`

Within the **Manage Groups** function in the CMC GUI, this setting enables or disables the capability to set

default Quality of Service (QoS) controls for users within a specific group.

Options are:

- true — This capability will display for users logged in as a system administrator or group administrator. For group admins this capability is restricted to their own group.
- false — This capability will not display for any users.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note This setting is relevant only if the `admin.manage_groups.enabled` and `admin.manage_groups.edit.enabled` settings are enabled.

`account.profile.writeable.enabled`

This setting controls whether a user can edit his or her own account profile information in the **Account** section of the CMC GUI. For user types for which this editing capability is not enabled, account profile information will be read-only. For a screen shot and description of this function, see "**Profile**" (page 352).

Options are:

- true — This capability will be enabled for all user types (system administrator, group administrator, and regular user).
- false — This capability will be disabled for all user types.
- SystemAdmin — This capability will be enabled only for users logged in as a system administrator.
- GroupAdmin — This capability will be enabled only for users logged in as a group administrator.
- User — This capability will be enabled only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`account.credentials.enabled`

This setting controls whether the **Security Credentials** function will be enabled in the CMC GUI. For a screen shot and description of this function, see Security Credentials in the CMC Help document.

Options are:

- true — This function will display for all user types (system administrator, group administrator, and regular user).
- false — This function will not display for any user types. If you set this to "false" then the Security Credentials functionality as a whole is disabled and the more granular `account.credentials.*.enabled` properties below are ignored.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.
- User — This function will display only for users logged in as a regular user.

- You can also specify a comma-separated list of multiple user types for which to enable this function — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note If you want to enable some aspects of the **Security Credentials** function and not others, you can have `account.credentials.enabled` set so that the function is enabled for your desired user types, and then use the granular `account.credentials.*.enabled` properties below to enable/disable specific capabilities.

`account.credentials.access.enabled`

Within the **Security Credentials** function in the CMC GUI, this setting enables or disables the capability of CMC users to view and change their own S3 storage access keys.

Options are:

- true — This capability will display for all user types (system administrator, group administrator, and regular user).
- false — This capability will not display for any user types.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.
- User — This capability will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this capability — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`account.credentials.signin.enabled`

Within the **Security Credentials** function in the CMC GUI, this setting enables or disables the capability CMC users to change their own CMC login password.

Options are:

- true — This capability will display for all user types (system administrator, group administrator, and regular user).
- false — This capability will not display for any user types.
- SystemAdmin — This capability will display only for users logged in as a system administrator.
- GroupAdmin — This capability will display only for users logged in as a group administrator.
- User — This capability will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this capability — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

`account.activity.enabled`

This setting controls whether the **Account Activity** function will be enabled in the CMC GUI. For a screen shot

and description of this function, see "**Account Activity**" (page 247).

Options are:

- true — This capability will display for users logged in as a system administrator.
- false — This capability will not display for any users.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

usage.enabled

This setting controls whether the **Usage By Users and Groups** function will be enabled in the CMC GUI. For a screen shot and description of this function, see "**Usage By Users & Groups**" (page 185).

Options are:

- true — This function will display for all user types (system administrator, group administrator, and regular user).
- false — This function will not display for any user types.
- SystemAdmin — This function will display only for users logged in as a system administrator.
- GroupAdmin — This function will display only for users logged in as a group administrator.
- User — This function will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this function — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

security.serviceinfo.enabled

This setting controls whether HyperStore's S3 service endpoints will display for group admins and regular users in the CMC's **Security Credentials** page. For a screen shot and description of this function, see "**Security Credentials**" (page 353).

Options are:

- true — S3 service endpoints will display for group admins and regular users in the CMC's **Security Credentials** page.
- false — S3 service endpoints will not display for group admins or regular users in the CMC's **Security Credentials** page.

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

Note For HyperStore system admins, the S3 service endpoint information displays in the [Cluster Information](#) page. The *security.serviceinfo.enabled* property has no effect on this display.

login.languageselection.enabled

Takes its value from *common.csv*: "**cmc_login_languageselection_enabled**" (page 484); use that setting instead.

login.grouplist.enabled

Takes its value from *common.csv*: "**cmc_login_grouplist_enabled**" (page 484); use that setting instead.

grouplist.enabled

Takes its value from *common.csv*: "**cmc_grouplist_enabled**" (page 483); use that setting instead.

grouplist.size.max

Takes its value from *common.csv*: "**cmc_grouplist_size_max**" (page 484); use that setting instead.

error.stacktrace.enabled

Throughout the CMC UI, when an exception occurs and an error page is displayed to the user, include on the error page a link to a stack trace.

Options are:

- true — Stack trace links will display for all user types (system administrator, group administrator, and regular user).
- false — Stack trace links will not display for any user types.
- SystemAdmin — Stack trace links will display only for users logged in as a system administrator.
- GroupAdmin — Stack trace links will display only for users logged in as a group administrator.
- User — Stack trace links will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to enable this feature — for example, "SystemAdmin,GroupAdmin".

Default = false

admin.whitelist.enabled

Takes its value from *common.csv*: "**admin_whitelist_enabled**" (page 481); use that setting instead.

sso.enabled

Whether to enable CMC support for single sign-on functionality, true or false. When this is set to false, if a user attempts to access the CMC via SSO, the access will be denied and an error will be written to *cloudian-ui.log*.

Default = false

Note For more information on CMC SSO, see "**Implementing Single Sign-On for the CMC**" (page 358).

sso.shared.key

Shared security key used for hash creation, when using the "auto-login with one way hash" method of single sign-on access to the CMC.

Default = ss0sh5r3dk3y

IMPORTANT: If you enable CMC SSO functionality (using the *sso.enabled* setting), then for security reasons you should **set a custom value for *sso.shared.key***. Do not leave it at its default value.

sso.tolerance.millis

Maximum allowed variance between the CMC server time and the timestamp submitted in a client request invoking the "auto-login with one way hash" method of single sign-on access to the CMC, in milliseconds. If the

variance is greater than this, the request is rejected. This effectively serves as a request expiry mechanism.

Default = 3600000

sso.cookie.cipher.key

Triple DES key used for cookie encryption.

Default = 123456789012345678901234

IMPORTANT: If you enable CMC SSO functionality (using the *sso.enabled* setting), then for security reasons you should **set a custom value for *sso.cookie.cipher.key***. Do not leave it at its default value.

Note If you change this value after CMC SSO has already been in service, end users who had used SSO previously will have on their browser a Cloudian SSO cookie that is no longer valid. If such users access the CMC after your *sso.cookie.cipher.key* change, the CMC detects the invalid cookie, deletes it, and drops a new, valid one.

bucket.storagepolicy.showdetail.enabled

Within the **Bucket Properties** function in the CMC GUI, this setting enables or disables the display of a "Storage Policy" tab that provides information about the storage policy being used by the bucket. This information includes the storage policies replication factor or erasure coding k+m configuration..

Options are:

- true — This tab will display for all user types (system administrator, group administrator, and regular user).
- false — This tab will not display for any user types.
- SystemAdmin — This tab will display only for users logged in as a system administrator.
- GroupAdmin — This tab will display only for users logged in as a group administrator.
- User — This tab will display only for users logged in as a regular user.
- You can also specify a comma-separated list of multiple user types for which to display this tab — for example, "SystemAdmin,GroupAdmin".

Default = Commented out and uses internal default of "true". To assign a different value, uncomment the setting and edit its value.

keysecure.encryption.enabled

Within the **Storage Policies** function in the CMC GUI, when system administrators are adding or editing a storage policy, this setting controls whether "KeySecure" will display as one of the options for Server-Side Encryption. For more information on using Gemalto KeySecure with HyperStore -- including the requirement to configure a connection from HyperStore to the Gemalto KeySecure system -- see "**Using Server-Side Encryption with Gemalto KeySecure KMS**" (page 125).

Options are:

- true — "KeySecure" will display as one of the options for Server-Side Encryption.
- false — "KeySecure" will not display as one of the options for Server-Side Encryption.

Default = false

bucket.tiering.enabled

Takes its value from "**Enable Auto-Tiering**" (page 302) in the CMC's [Configuration Settings](#) page; use that setting instead.

tiering.perbucketcredentials.enabled

Takes its value from "**Enable Per Bucket Credentials**" (page 302) in the CMC's [Configuration Settings](#) page; use that setting instead.

tiering.customendpoint.enabled

Takes its value from "**Enable Custom Endpoint**" (page 303) in the CMC's [Configuration Settings](#) page; use that setting instead.

bucket.tiering.default.destination.list

Takes its value from *common.csv*: "**cmc_bucket_tiering_default_destination_list**" (page 485); use that setting instead.

bucket.tiering.custom.url

Takes its value from [Default Tiering URL](#) in the CMC's [Configuration Settings](#) page; use that setting instead.

puppet.master.licensefile

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

puppet.master.fileupdate.location

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

local.ssh.privateKey

Private key to use when the CMC connects via SSH to the Puppet master node or HyperStore nodes when implementing node management functions. The Cloudian installation script automatically populates this setting.

local.ssh.passphrase

Pass phrase to use when the CMC connects via SSH to the Puppet master node or HyperStore nodes when implementing node management functions. The Cloudian installation script automatically populates this setting.

local.temp.dir

This setting supports the feature whereby an updated HyperStore license file can be uploaded via the CMC. Do not edit.

remote.ssh.user

The user as which to connect via SSH to the Puppet master node or HyperStore nodes.

Default = root

remote.ssh.port

The port to which to connect via SSH to the Puppet master node or HyperStore nodes.

Default = 22

offload.services.node.options

Used internally by the CMC when invoking the installer script for certain operations. Do not edit.

Default = -m

uninstall.node.options

Used internally by the CMC when invoking the installer script for certain operations. Do not edit.

Default = -u -r

elasticsearch.enabled

For information about this setting, see "**Elasticsearch Integration for Object Metadata**" (page 106).

proactive.repair.queue.warning.enabled

The CMC [Dashboard](#) has a feature whereby it displays a "Long proactive repair queue" warning if the proactive repair queue for a particular node has more than 10,000 objects in it. This feature requires the Dashboard to retrieve proactive repair queue length data from Cassandra, each time the Dashboard page is loaded in your browser.

The *proactive.repair.queue.warning.enabled* property enables and disables this Dashboard feature. If this property is set to "false", then the Dashboard does not retrieve proactive repair queue length data from Cassandra and does not display any warnings in regard to proactive repair queue length.

Default = true

9.5.5. Other Configuration Files

The tables below briefly describe additional HyperStore configuration files that reside in the configuration directories on your Puppet master node. Under typical circumstances you should not need to manually edit these files.

All of these files are in sub-directories under the */etc/cloudian-<version>-puppet* directory. For brevity, in the section headings that follow */etc/cloudian-<version>-puppet* is replaced with "..." and only the sub-directory is specified.

- ["Files under .../manifests/extdata/" \(page 537\)](#)
- ["Files under .../modules/cloudians3/templates/" \(page 538\)](#)
- ["Files under .../modules/cmc/templates/" \(page 539\)](#)
- ["Files under .../modules/salt" \(page 539\)](#)

9.5.5.1. Files under .../manifests/extdata/

File	Purpose
<i>adminsslconfigs.csv</i>	Configures TLS/SSL implementation for the Admin Service's HTTPS listener.
<i>dynsettings.csv</i>	Used by the CMC's Configuration Settings page. Do not edit this file.
<i>fileupdates.csv</i>	There may be rare circumstances in which Cloudian Support asks you to use this file, in which case you will be provided instructions for using it.

File	Purpose
<i>iamsslconfigs.csv</i>	Configures TLS/SSL implementation for the IAM Service, if you want the IAM Service to support an HTTPS listener. This file is updated automatically if you use the "Installer Advanced Configuration Options" (page 449) to set up TLS/SSL for your IAM Service.
<i><node>.csv</i>	These files are for settings that are tailored to individual nodes, as identified by the <code><node></code> segment of the file names (for example, <code>host1.csv</code> , <code>host2.csv</code> , and so on). There will be one such file for each node in your system. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation. Settings in a <code><node>.csv</code> file override default settings from <code>common.csv</code> , for the specified node.
<i><regionName>_region.csv</i>	Configures settings that are particular to a service region. Settings that would have different values in different regions are in this file. If you have multiple HyperStore service regions, you will have multiple instances of this file. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation.
<i>s3sslconfigs.csv</i>	Configures TLS/SSL implementation for the S3 Service, if you want the S3 Service to support an HTTPS listener. This file is updated automatically if you use the "Installer Advanced Configuration Options" (page 449) to set up TLS/SSL for your S3 Service.
<i><regionName>_topology.csv</i>	Specifies a topology of nodes, racks, and data centers in a service region. If you have multiple HyperStore service regions, you will have multiple instances of this file. These files are created and pre-configured by the HyperStore install script, based on information that you provided during installation.

9.5.5.2. Files under .../modules/cloudians3/templates/

File	Purpose
<i>admin.xml.erb</i>	This file configures the Admin Service's underlying Jetty server functionality, for processing incoming HTTP requests.
<i>admin_realm.properties.erb</i>	This file configures HTTP Basic Authentication for the Admin Service.
<i>cloudian-cron.tab.erb</i>	This file configures HyperStore system maintenance cron jobs.
<i>log4j-* .xml.erb</i>	These files configure logging behavior for various services such as the S3 Service, Admin Service, HyperStore Service, and so on. For information about settings in these files, see "Log Configuration Settings" (page 593).
<i>s3.xml.erb</i>	This file configures the S3 Service's underlying Jetty server functionality, for processing incoming HTTP requests.

File	Purpose
<i>storage.xml.erb</i>	This file configures the HyperStore Service's underlying Jetty server functionality, for processing incoming HTTP requests.
<i>tiering-map.txt.erb</i>	This file is not used currently.
<i>tiering-regions.xml.erb</i>	In support of the HyperStore auto-tiering feature, this file lists S3 endpoints that objects can be auto-tiered to. By default this file is pre-configured with all the Amazon S3 regional service endpoints. If you have a multi-region system, the file is also pre-configured with the S3 endpoints for each of your service regions, and the file is used in support of the cross-region replication feature.

9.5.5.3. Files under .../modules/cmc/templates/

File	Purpose
<i>server.xml.erb</i>	This file configures the HyperStore Service's underlying Tomcat server functionality, for processing incoming HTTP requests.

9.5.5.4. Files under .../modules/salt

Starting with HyperStore version 7.2, HyperStore uses the open source version of [Salt](#) for certain configuration management functions (such as configuration management of the HyperStore firewall -- which from a user perspective is controlled through the installer's Advanced Configuration Options menu). **Do not edit any of the configuration files under /etc/cloudian-<version>-puppet/modules/salt.**

9.5.6. Using JMX to Dynamically Change Configuration Settings

Certain HyperStore configuration settings (a minority of them) can be dynamically reloaded via JMX. You can use a JMX client such as [JConsole](#) or [Jmxterm](#) to connect to the JMX listening port of the particular service that you're configuring (S3 Service JMX port = 19080; Admin Service JMX port = 19081; HyperStore Service JMX port = 19082; Redis Monitor JMX port = 19083). [JConsole](#) is a graphical user interface that's part of the HyperStore system's Java installation and can be found under `JAVA_HOME/bin`. [Jmxterm](#) is a command line tool that comes bundled with the HyperStore system and can be found under `opt/cloudian/tools`.

When you use JMX to make a configuration setting change, the change is applied to the service dynamically — you do not need to restart the service for the change to take effect. However, the **setting change persists only for the current running session of the affected service**. If you restart the service it will use whatever setting is in the configuration file. Consequently, if you want to change a setting dynamically and also have your change persist through a service restart, you should change the setting in the configuration file as well as changing it via JMX.

In the documentation of HyperStore configuration files, if a setting supports being dynamically changed via JMX, the setting description indicates "Reloadable via JMX". It also indicates the name of the dynamically changeable MBean attribute that corresponds to the configuration file setting. For example, the documentation of the HyperStore Service configuration property `repair.session.threadpool.corepoolsize` includes a note that says:

"Reloadable via JMX (HyperStore Service's JMX port 19082; MBean attribute = com.gemini.cloudian.hybrid.server → FileRepairService → Attributes → RepairSessionThreadPoolCorePoolSize)"

Note Some settings in HyperStore configuration files can be set through the CMC's [Configuration Settings](#) page. The CMC uses JMX to apply setting changes dynamically, and the CMC also automatically makes the corresponding configuration file change and triggers a Puppet push so that your changes will persist across service restarts. For these settings it's therefore best to use the CMC to make any desired edits rather than directly using JMX. For such configuration file settings, the descriptions in the HyperStore configuration file documentation indicate that you should use the CMC to edit the setting. Consequently these settings are not flagged in the documentation as JMX reloadable.

9.6. Configuration Special Topics

9.6.1. Anti-Virus Software

If you are considering using anti-virus software on your HyperStore nodes, be aware that the HyperStore system is provisioned and uses server resources according to the server specifications and use case. Any use of third party software must ensure that it does not interfere with HyperStore's use of these resources (such as disk space, disk I/O, RAM, CPU, network, and ports). Unavailability or sharing of these resources can cause the HyperStore system to not function and/or have reduced performance.

If you want to use anti-virus software to monitor OS files **other than** HyperStore-related files, configure the anti-virus software to exclude these directories from monitoring:

- All HyperStore data directories (as specified by the configuration setting *hyperstore_data_directory* in *common.csv*)
- */var/lib/{cassandra,cassandra_commit,redis}*
- */var/log/{cloudian,cassandra,redis}*
- */opt/{cassandra,cloudian,cloudian-packages,cloudianagent,dnsmasq,redis,tomcat}*
- */etc/cloudian-<version>-puppet**

9.6.2. NTP Automatic Set-Up

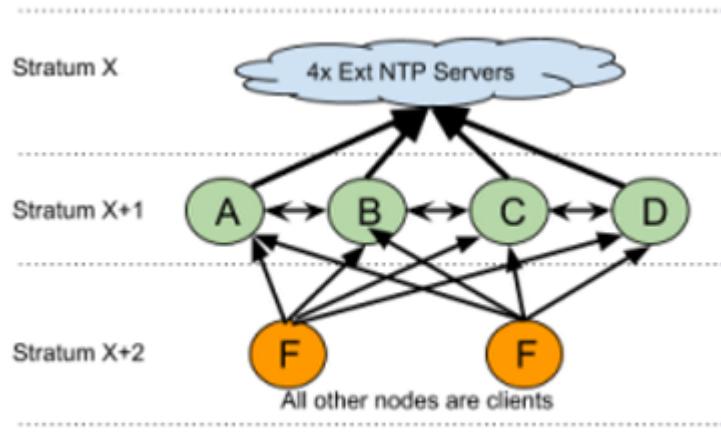
Note This topic describes the NTP configuration that HyperStore automatically implements. If instead you are using your own custom NTP set-up, Cloudian recommends using **at least 3 root clock sources**.

Accurate, synchronized time across the cluster is vital to HyperStore service. For example, object versioning relies on it, and so does S3 authorization. It's important to have a robust NTP set-up.

When you install your HyperStore cluster, the installation script **automatically** configures a robust NTP set-up using *ntpd*, as follows:

- In each of your HyperStore data centers, four HyperStore nodes are configured as internal NTP servers. These internal NTP server will synchronize with external NTP servers -- from the *pool.ntp.org* project by default -- and are also configured as peers of each other. (If a HyperStore data center has only four or fewer nodes, then all the nodes in the data center are configured as internal NTP servers.)

- All other nodes in the data center are configured as clients of the four internal NTP servers.



- In the event that all four internal NTP servers in a DC are unable to reach any of the external NTP servers, the four internal NTP servers will use "orphan mode" -- which entails the nodes choosing one of themselves to be the "leader" to which the others will sync -- until such time as one or more of the external NTP servers are reachable again.

Each HyperStore data center is independently configured, using this same approach.

To see which of your HyperStore hosts are configured as internal NTP servers, go to the CMC's [Cluster Information](#) page. On that page you can also view the list of external NTP servers to which the internal NTP servers will synchronize.

IMPORTANT: In order to connect to the external NTP servers the internal NTP servers must be allowed outbound internet access.

Note *ntpd* is configured to automatically start on host boot-up. However, it's recommended that after booting a HyperStore host, you verify that *ntpd* is running (which you can do with the *ntpq -p* command — if *ntpd* is running this command will return a list of connected time servers).

9.6.2.1. Changing the List of External NTP Servers or Internal NTP Servers

You can use the HyperStore installer's "Advanced Configuration Options" to change either the list of external NTP servers or the list of internal NTP servers. For more information see "[Change Internal NTP Servers or External NTP Servers](#)" (page 418).

9.6.3. Changing S3, Admin, or CMC Listening Ports

You can use the HyperStore installer's "Advanced Configuration Options" to change listening ports for the S3, Admin, and CMC services.

1. On the Puppet master node, change to the installation staging directory. Then launch the installer.

```
[7.2]# ./cloudianInstall.sh
```

2. From the installer's menu select "Advanced Configuration Options" and then select "Change S3, Admin or CMC ports".
3. Follow the prompts to specify your desired port numbers. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value. The final prompt will ask whether you want to save your changes -- type yes to do so.
4. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
5. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the affected services:
 - For changed S3 or Admin ports, restart the S3 Service and the CMC. Note that the Admin service is restarted automatically when you restart the S3 Service.
 - For changed CMC port, restart the CMC

9.6.4. Changing S3, Admin, or CMC Service Endpoints

You can use the HyperStore installer's "Advanced Configuration Options" to change the S3 service endpoint, S3 static website endpoint, Admin service endpoint, or CMC endpoint. (For more information on these HyperStore service endpoints, their default values, and how the endpoints are used, see [DNS Set-Up](#).)

1. On the Puppet master node, change to the installation staging directory. Then launch the installer.

```
[7.2]# ./cloudianInstall.sh
```

2. From the installer's menu select "Advanced Configuration Options" and then select "Change S3, S3-Website, Admin, or CMC endpoints".
3. Follow the prompts to specify your desired endpoints. The prompts indicate your current settings. At each prompt press Enter to keep the current setting value, or type in a new value.

For S3 service endpoint, the typical configuration is one endpoint per service region but you also have the option of specifying multiple endpoints per region (if for example you want to have different S3 service endpoints for different data centers within the same region). To do so simply enter a comma-separated list of endpoints at the prompt for the region's S3 service domain URL. Do not enclose the comma-separated list in quotes. If you want to have different S3 endpoints for different data centers within the same service region, the recommended S3 endpoint syntax is *s3-<region>.<d-cname>.<domain>*. For example if you have data centers named *chicago* and *cleveland* both within the *midwest* service region, the S3 endpoints would be *s3-midwest.chicago.enterprise.com* and *s3-midwest.cleveland.enterprise.com*. (Make sure that your DNS set-up resolves the different endpoints in the way that you want -- for example, with one S3 service endpoint resolving to the virtual IP address of a load balancer in your Chicago data center and one S3 service endpoint resolving to the virtual IP address of a load balancer in your Cleveland data center).

For S3 static website endpoint you can only have one endpoint per service region. For the Admin service you can only have one endpoint for your whole HyperStore system, and same for the CMC service -- only one CMC endpoint is allowed for the whole system.

The final prompt will ask whether you want to save your changes -- type yes to do so.

4. Go to the main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
5. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service and the CMC. If you are using DNSMASQ for HyperStore service endpoint resolution, then also restart DNSMASQ. (If instead you are using your DNS environment for HyperStore service endpoint resolution,

update your DNS entries to match your custom endpoints if you have not already done so. For guidance see [DNS Set-Up](#).)

9.6.5. TLS/SSL for S3 Service (Self-Signed Certificate)

Optionally, you can configure HyperStore's S3 Service to support HTTPS connections from S3 client applications. The procedure below describes how to set up TLS/SSL for your S3 Service using a **self-signed TLS/SSL certificate**. The procedure describes how to create the certificate as well as how to have the S3 Service use the certificate. A self-signed certificate may be your preferred approach for an evaluation or testing environment. (Alternatively, for instructions on using a CA-verified certificate -- which may be more appropriate in a production environment -- see "[TLS/SSL for S3 Service \(CA-Verified Certificate\)](#)" (page 547)).

Follow the procedure below if you want to use a self-signed TLS/SSL certificate to set up HTTPS support for your HyperStore S3 Service.

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu select "**Generate a self-signed certificate in a JKS keystore**".
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Please enter key store name [cloudian.jks]

Name of the Keystore file which will be created, and in which your TLS/SSL certificate for S3 will be stored. Make a note of this name as you will need it again later in this procedure. **Do not name the key-store "keystore"**.

Example: cloudian.jks

Please enter alias name []

An alias for identifying the TLS/SSL key pair that will be created and stored in the Keystore (and which will be used by the S3 Service). Make a note of this alias as you will need it again later in this procedure.

Example: cloudians3

Please enter the password for cloudian.jks [testpass]

The password to set for the Keystore file. Retain this password in a safe location — you will need it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: MyPassword

Please enter the key store manager password for cloudian.jks [testpass]

Enter the same password that you entered at the previous prompt.

Example: MyPassword

Please enter comma-separated domain names [.[s3-region1.mycloudianhyperstore.com](#)]*

Enter the wildcard domain of your S3 service (the service endpoint preceded by "*."). This wildcard domain should match the wildcard domain that was specified at the time your HyperStore cluster was installed and configured. If you have multiple domains, enter a comma-separated list of wildcard domains.

Example: *.s3-region1.mycompany.com

Enter your organizational unit name []

Name of your department or work group

Example: Engineering

Enter the name of your organization []

Organization name

Example: MyCompany

Enter the name of your City or Locality []

City or locality in which your organization is located

Example: San Francisco

Enter the name of your State or Province []

Full name of the state or province in which your organization is located

Example: California

Enter the two-letter country code for this unit []

Two-letter country code for the country in which your organization is located

Example: US

Note When you complete the above prompts, the installer creates a new Keystore file (with your specified Keystore name) under `/etc/cloudian-<version>-puppet/modules/baselayout/files/`. It also creates a Certificate Signing Request (CSR) in the same directory. If your Keystore file is named `cloudian.jks` for example, then the CSR file will be named `cloudian.csr`. However, since you are using a self-signed certificate in this procedure (not a CA-verified certificate), you do not need to do anything with the CSR file.

After completing all the prompts for generating the certificate and keystore, press any key to return to the installer's Advanced Configuration Options menu.

6. From the Advanced Configuration Options menu, select "**Enable and configure HTTPS access on S3 server**".
7. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Do you wish to enable HTTPS access on S3 server? [no]

Enter yes to enable HTTPS access on the S3 server.

Please enter name of your key store [cloudian.jks]

Name of the Keystore file that contains the key pair that you are using for TLS/SSL on the S3 Service.

This must be the same Keystore file name that you specified in Step 5 of this procedure.

Example: cloudian.jks

Certificate alias to use []

Alias identifying the key pair that you are using for TLS/SSL on the S3 Service. This must be the same alias that you specified in Step 5 of this procedure.

Example: cloudians3

Please enter trust keystore [cloudian.jks]

Keystore file that contains client certificates for TLS/SSL mutual authentication (if you are using mutual authentication). Enter your regular Keystore file name, unless you have put (or plan to put) client certificates for TLS/SSL mutual authentication in a separate Keystore file. Note that HyperStore does not provide automated support for setting up client certificates for TLS/SSL mutual authentication. If you want to use TLS/SSL mutual authentication for your S3 Service, consult with Cloudian Support.

Example: cloudian.jks

Please enter password for the keystore cloudian.jks [testpass]

Password for the S3 Service to use when accessing your Keystore file. This must be the same Keystore password that you set in Step 5 of this procedure.

Note that before storing Keystore-related passwords in HyperStore configuration files, the installer automatically obfuscates the passwords using a Jetty password obfuscation utility. For example, if your plain text password is "MyPassword", it will be entered in HyperStore configuration files as "OBF:1o4o1zly1qw01vu11ym71ym71vv91qxq1zlk1o5".

Example: MyPassword

Please enter password for the trust keystore cloudian.jks [testpass]

Enter your regular Keystore password (unless you have a separate trust Keystore and want to use a different password for that).

Example: MyPassword

Please enter keystore manager password [testpass]

Enter your regular Keystore password.

Example: MyPassword

Please enter path name in which to store keystore file [/opt/cloudian/conf]

Puppet will propagate the Keystore file to this location on your S3 Service nodes. Accept the default value.

Example: /opt/cloudian/conf

Please enter path name in which to store trust keystore file [/opt/cloudian/conf]

Puppet will propagate the trust Keystore file to this location on your S3 Service nodes. Accept the default value.

Example: /opt/cloudian/conf

Please enter connection maximum idle time in (ms) [60000]

When the S3 Service is servicing an HTTPS request from a client, the maximum allowed connection idle time in milliseconds. After this many idle milliseconds the connection will be dropped.

Example: 60000

Please enter connections at which system is considered to have low resource [1000]

If the number of concurrent connections to the S3 Service's HTTPS listener exceeds this value, use a special low resources idle timer to time out and drop idle connections.

Example: 1000

Please enter low resource connection idle time in (ms) [5000]

Special, "low resource" maximum idle time (in milliseconds) to apply to S3 Service HTTPS connections when the number of concurrent connections exceeds the number specified at the preceding prompt.

Example: 5000

8. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
9. From the installer's main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
10. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the S3 Service. After this is done you can exit the installer.

Once the S3 Service has been restarted on all nodes in the cluster, issue the following command on each node in the cluster to verify that the S3 Service is listening on TCP port 443:

```
[root]# netstat -tln | grep :443
```

You should see output similar to the following:

```
tcp 0 0 ::ffff:10.50.200.51: 443 ::*: LISTEN
```

In a browser, if you navigate to <https://<your-S3-domain>:443> you should see a warning that the domain is using a self-signed certificate.

You have now completed the procedure for creating and using a self-signed certificate for S3 TLS/SSL.

Note When you set up your S3 Service to support TLS/SSL, it will support HTTPS connections from client applications (on port 443 by default) but it **will not require clients to use HTTPS**. HTTP connections (on port 80 by default) will continue to be supported as well. If you want clients to communicate with the S3 Service only through HTTPS you can block port 80 at your load balancer.

Note The S3 Service **does not support client connections that use TLS/SSL versions 1.0 or 1.1**. Clients connecting by TLS/SSL must use TLS/SSL version 1.2 or later.

Note If you are using HAProxy for load balancing in front of the S3 Service and you've enabled PROXY Protocol (see "**s3_proxy_protocol_enabled**" (page 475) in [common.csv](#)), the procedure

described above sets up TLS/SSL for PROXY Protocol port 4431 as well as for regular HTTPS port 443.

9.6.5.1. To Have the CMC Connect to the S3 Service Using HTTPS

For the CMC to be able to connect to the S3 Service using HTTPS, you must manually import the S3 certificate into the default JVM keystore on each node on which the CMC runs. Follow these steps on each node:

1. Retrieve the certificate content and write it to a certificate named *s3.crt*. Enter this command as one line with no breaks. In the command, replace *s3-region1.localdomain* with your S3 service endpoint.

```
echo -n | openssl s_client -connect s3-region1.localdomain:443 | sed -ne
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > s3.crt
```

2. Import *s3.crt* into the default JVM keystore on the node. Enter this command as one line with no breaks.

```
keytool -import -trustcacerts -keystore
/usr/java/jdk1.8.0_172-amd64/jre/lib/security/cacerts -storepass changeit -
noprompt
-alias s3cert -file s3.crt
```

After executing the above steps on each node, return to your Puppet master node and do the following:

1. In [common.csv](#), set *cmc_storageuri_ssl_enabled* to "true".
2. Do a Puppet push to the cluster and then restart the cluster's CMC service.

9.6.6. TLS/SSL for S3 Service (CA-Verified Certificate)

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Generate a Certificate and a Certificate Signing Request" (page 548)**
- **"Import the CA-Signed Certificates into the Keystore" (page 549)**
- **"Configure the HyperStore S3 Service to Support HTTPS" (page 550)**

Optionally, you can configure HyperStore's S3 Service to support HTTPS connections from S3 client applications. The procedure below describes how to set up TLS/SSL for your S3 Service using a **CA-verified TLS/SSL certificate** (a certificate verified by a third party Certificate Authority organization). The procedure describes how to create the certificate and have it verified, as well as how to have the S3 Service use the certificate. A CA-verified certificate may be your preferred approach for a production environment. (Alternatively, for instructions on using a self-signed certificate -- which may be adequate in a testing environment -- see **"TLS/SSL for S3 Service (Self-Signed Certificate)" (page 543)**).

To use a CA-verified TLS/SSL certificate for your HyperStore S3 Service you will first generate a certificate and a corresponding Certificate Signing Request (CSR) that you can submit to your preferred CA. Then after the CA has verified and returned the certificate to you (typically along with a root certificate and intermediate certificate), you will import the certificates into your Keystore. As the third and final stage of this procedure you will configure the HyperStore S3 Service to support HTTPS.

9.6.6.1. Generate a Certificate and a Certificate Signing Request

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu select "**Generate a self-signed certificate in a JKS keystore**".
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Please enter key store name [cloudian.jks]

Name of the Keystore file which will be created, and in which your TLS/SSL certificate for S3 will be stored. Make a note of this name as you will need it again later in this procedure. **Do not name the key-store "keystore"**.

Example: cloudian.jks

Please enter alias name []

An alias for identifying the TLS/SSL key pair that will be created and stored in the Keystore (and which will be used by the S3 Service). Make a note of this alias as you will need it again later in this procedure.

Example: cloudians3selfsigned

Please enter the password for cloudian.jks [testpass]

The password to set for the Keystore file. Retain this password in a safe location — you will need it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: MyPassword

Please enter the key store manager password for cloudian.jks [testpass]

Enter the same password that you entered at the previous prompt.

Example: MyPassword

Please enter comma-separated domain names [.*s3-region1.mycloudianhyperstore.com*]*

Enter the wildcard domain of your S3 service (the service endpoint preceded by "*."). This wildcard domain should match the wildcard domain that was specified at the time your HyperStore cluster was installed and configured. If you have multiple domains, enter a comma-separated list of wildcard domains.

Example: *.s3-region1.mycompany.com

Enter your organizational unit name []

Name of your department or work group

Example: Engineering

Enter the name of your organization []

Organization name

Example: MyCompany

Enter the name of your City or Locality []

City or locality in which your organization is located

Example: San Francisco

Enter the name of your State or Province []

Full name of the state or province in which your organization is located

Example: California

Enter the two-letter country code for this unit []

Two-letter country code for the country in which your organization is located

Example: US

When you complete the above prompts, the installer creates a new Keystore file (with your specified Keystore name) under `/etc/cloudian-<version>-puppet/modules/baselayout/files/`. It also creates a Certificate Signing Request (CSR) in the same directory — if your Keystore file is named `cloudian.jks` for example, then the CSR file will be named `cloudian.csr`.

Submit this CSR file to your preferred Certificate Authority for signing, using the instructions from the CA.

Note In case you need this information during CSR submission: The HyperStore S3 Service is built on Jetty open source technology. Jetty is dual-licensed under the Apache and Eclipse projects.

9.6.6.2. Import the CA-Signed Certificates into the Keystore

Once your CA has signed your TLS/SSL Certificate, they should send it to you along with their Root and Intermediate CA certificates. Follow the steps below to import all these certificates to your Keystore.

Note If your CA is GoDaddy.com, then they should provide you a copy of the following three certificate files with your signed TLS/SSL Certificate:

`gdroot-g2_cross.crt` (GoDaddy G1 to G2 Cross Root Certificate)
`gd_cross_intermediate.crt` (GoDaddy Cross Intermediate CA Certificate)
`gdig2.crt` (GoDaddy Intermediate G2 CA Certificate)

You will need GoDaddy's Cross Root, Cross Intermediate CA and Intermediate G2 certificate files in order to be able to import your signed TLS/SSL Certificate into your Keystore file. If GoDaddy did not provide you a copy of their Cross Root, Cross Intermediate CA or Intermediate G2 Certificate file, you can download them from the following URLs:

GoDaddy G1 to G2 Cross Root Certificate:
https://certs.godaddy.com/repository/gdroot-g2_cross.crt

GoDaddy Cross Intermediate CA Certificate:
https://certs.godaddy.com/repository/gd_cross_intermediate.crt

GoDaddy Intermediate G2 CA Certificate:
<https://certs.godaddy.com/repository/gdig2.crt>

1. Log in to the Puppet master node as user *root*.
2. Copy all the certificates that you received from the CA into the */etc/cloudian-<version>-puppet/modules/baselayout/files* directory (where your Keystore file is).
3. Go to the */etc/cloudian-<version>-puppet/modules/baselayout/files* directory. For example:

```
[root]# cd /etc/cloudian-7.2-puppet/modules/baselayout/files
```

4. Issue the following commands to import the Root CA Certificate and Intermediate CA Certificate into your Keystore file. For example, if your Keystore is named *cloudian.jks*:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias root
-file <Root CA Certificate File> -keystore cloudian.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias intermediate
-file <Intermediate CA Certificate File> -keystore cloudian.jks
```

Example for GoDaddy as the CA:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyRoot
-file gdrootg2_cross.crt -keystore cloudian.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias
GoDaddyCrossCA
-file gd_cross_intermediate.crt -keystore cloudian.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyG2CA
-file gdig2.crt -keystore cloudian.jks
```

5. For the sake of clarity, rename or copy the CA-signed TLS/SSL Certificate file to *cloudianS3.crt*.

```
[files]# cp -p <signed certificate file> cloudianS3.crt
```

Example:

```
[files]# cp -p 769830d8b471822b.crt cloudianS3.crt
```

6. Issue the following command to import your CA-signed TLS/SSL Certificate into your Keystore file:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias cloudians3
-file cloudianS3.crt -keystore cloudian.jks
```

Now your Keystore is all set and all that remains is to enable and configure HTTPS on the S3 Service.

9.6.6.3. Configure the HyperStore S3 Service to Support HTTPS

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu, select "Enable and configure HTTPS access on S3 server".
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Please enter name of your key store [cloudian.jks]

Enter yes to enable HTTPS access on the S3 server.

Please enter password for the keystore [testpass]

Name of the Keystore file that contains the key pair that you are using for TLS/SSL on the S3 Service. This must be the same Keystore file name that you specified earlier in this procedure in Step 5 under "**Generate a Certificate and a Certificate Signing Request**" (page 548).

Example: cloudian.jks

Please enter certificate alias to use []

Alias identifying the key pair that you are using for TLS/SSL on the S3 Service. This must be the same alias that you specified in Step 6 under "**Import the CA-Signed Certificates into the Keystore**" (page 549).

Example: cloudians3

Please enter trust keystore [cloudian.jks]

Keystore file that contains client certificates for TLS/SSL mutual authentication (if you are using mutual authentication). Enter your regular Keystore file name, unless you have put (or plan to put) client certificates for TLS/SSL mutual authentication in a separate Keystore file. Note that HyperStore does not provide automated support for setting up client certificates for TLS/SSL mutual authentication. If you want to use TLS/SSL mutual authentication for your S3 Service, consult with Cloudian Support.

Example: cloudian.jks

Please enter password for the keystore cloudian.jks [testpass]

Password for the S3 Service to use to access your Keystore file. This must be the same Keystore password that you set in Step 5 under "**Generate a Certificate and a Certificate Signing Request**" (page 548).

Note that before storing Keystore-related passwords in HyperStore configuration files, the installer automatically obfuscates the passwords using a Jetty password obfuscation utility. For example, if your plain text password is "MyPassword", it will be entered in HyperStore configuration files as "OBF:1o4o1zly1qw01vu11ym71ym71vv91qxq1zlk1o5y".

Example: MyPassword

Please enter password for the trust keystore cloudian.jks [testpass]

Enter your regular Keystore password (unless you have a separate trust Keystore and want to use a different password for that).

Example: MyPassword

Please enter keystore manager password [testpass]

Enter your regular Keystore password.

Example: MyPassword

Please enter path name in which to store keystore file [/opt/cloudian/conf]

Puppet will propagate the Keystore file to this location on your S3 Service nodes. Accept the default value.

Example: /opt/cloudian/conf

Please enter path name in which to store trust keystore file [/opt/cloudian/conf]

Puppet will propagate the trust Keystore file to this location on your S3 Service nodes. Accept the default value.

Example: /opt/cloudian/conf

Please enter connection maximum idle time in (ms) [60000]

When the S3 Service is servicing an HTTPS request from a client, the maximum allowed connection idle time in milliseconds. After this many idle milliseconds the connection will be dropped.

Example: 60000

Please enter connections at which system is considered to have low resource [1000]

If the number of concurrent connections to the S3 Service's HTTPS listener exceeds this value, use a special low resources idle timer to time out and drop idle connections.

Example: 1000

Please enter low resource connection idle time in (ms) [5000]

Special, "low resource" maximum idle time (in milliseconds) to apply to S3 Service HTTPS connections when the number of concurrent connections exceeds the number specified at the preceding prompt.

Example: 5000

6. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
7. From the main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
8. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the S3 Service. After this is done you can exit the installer.

Once the S3 Service has been restarted on all nodes in the cluster, issue the following command on each node in the cluster to verify that the S3 Service is listening on TCP port 443:

```
[root]# netstat -tln | grep :443
```

You should see output similar to the following:

```
tcp 0 0 ::ffff:10.50.200.51: 443 ::*: LISTEN
```

Next, in a browser navigate to <https://<your-S3-domain>:443/>.

You can safely ignore the "Access Denied" error. However, check the certificate presented to your browser by the S3 Service and confirm that it is signed by your CA and that its Common Name (i.e. CN) value matches the

wildcard domain that you specified when you created your Cloudian Keystore file, e.g. *.s3-region1.-mycompany.com.

You have now completed the procedure for acquiring and using a CA-verified certificate for S3 TLS/SSL.

Note When you set up your S3 Service to support TLS/SSL, it will support HTTPS connections from client applications (on port 443 by default) but it **will not require clients to use HTTPS**. HTTP connections (on port 80 by default) will continue to be supported as well. If you want clients to communicate with the S3 Service only through HTTPS you can block port 80 at your load balancer.

Note The S3 Service **does not support client connections that use TLS/SSL versions 1.0 or 1.1**. Clients connecting by TLS/SSL must use TLS/SSL version 1.2 or later.

Note If you are using HAProxy for load balancing in front of the S3 Service and you've enabled PROXY Protocol (see "["s3_proxy_protocol_enabled"](#) (page 475) in [common.csv](#))[\),](#) the procedure described above sets up TLS/SSL for PROXY Protocol port 4431 as well as for regular HTTPS port 443.

9.6.6.4. To Have the CMC Connect to the S3 Service Using HTTPS

For the CMC to be able to connect to the S3 Service using HTTPS, you must manually import the S3 certificate into the default JVM keystore on each node on which the CMC runs. Follow these steps on each node:

1. Retrieve the certificate content and write it to a certificate named *s3.crt*. Enter this command as one line with no breaks. In the command, replace *s3-region1.localdomain* with your S3 service endpoint.

```
echo -n | openssl s_client -connect s3-region1.localdomain:443 | sed -ne
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > s3.crt
```

2. Import *s3.crt* into the default JVM keystore on the node. Enter this command as one line with no breaks.

```
keytool -import -trustcacerts -keystore
/usr/java/jdk1.8.0_172-amd64/jre/lib/security/cacerts -storepass changeit -
noprompt
-alias s3cert -file s3.crt
```

After executing the above steps on each node, return to your Puppet master node and do the following:

1. In [common.csv](#), set *cmc_storageuri_ssl_enabled* to "true".
2. Do a Puppet push to the cluster and then restart the cluster's CMC service.

9.6.7. TLS/SSL for the CMC (Self-Signed Certificate)

By default the CMC already supports TLS/SSL, and CMC clients must use HTTPS to communicate with the CMC. The default CMC TLS/SSL implementation uses a self-signed certificate -- unique to your system -- that is generated automatically during HyperStore installation. Consequently, no configuration steps are required in regard to the CMC's use of HTTPS -- it works out-of-the-box.

The only optional CMC TLS/SSL set-up actions you may want to consider are 1) creating your own self-signed certificate and Keystore for the CMC to use rather than the default Keystore; or 2) acquiring a CA-verified

certificate and having the CMC use that certificate in its TLS/SSL implementation. Using your own self-signed certificate is described below. For using a CA-verified certificate see "**TLS/SSL for the CMC (CA-Verified Certificate)**" (page 556).

Follow the procedure below if you want to **generate and use your own self-signed TLS/SSL certificate** to implement TLS/SSL for your CMC Service. This approach may be appropriate in a HyperStore testing or evaluation environment where you don't want to use the default Keystore and certificate that come bundled with the CMC, but you also don't want to go through the process of acquiring a CA-verified certificate.

1. Log into the Puppet master node as *root*.
2. Go to the */etc/cloudian-<version>-puppet/modules/cmc/files* directory for your current HyperStore version, for example:

```
[root]# cd /etc/cloudian-7.2-puppet/modules/cmc/files
```

3. Issue the following command to create a new TLS/SSL key pair and store them in a Java Keystore file called *cmc.jks*:

```
[root]# /usr/java/default/bin/keytool -genkey -alias cloudian -keyalg RSA -keysize 2048  
-validity 365 -keystore cmc.jks
```

Note: Do not name the Keystore file ".keystore_cloudian". That is the name of the default Keystore that comes bundled with the CMC.

4. When prompted by the *keytool* utility, provide the following information:

Enter keystore password

The password to use for the Keystore file. Retain this password in a safe location — you will need to provide it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: Password

Re-enter new password

Re-enter the same password.

Example: Password

What is your first and last name? (CMC FQDN!)

Enter the CMC's fully qualified domain name, **not your first and last name**. This domain should match the CMC domain that was specified at the time your HyperStore cluster was installed and configured.

Do not use *cloudian.com* in the CMC's FQDN, as this domain belongs to Cloudian, Inc. and your CMC service will not work properly if you do.

Example: cmc.mycompany.com

What is the name of your organizational unit?

Name of your department or work group.

Example: Engineering

What is the name of your organization?

Name of your organization.

Example: MyCompany

What is the name of your City or Locality?

City or locality in which your organization is located.

Example: San Francisco

What is the name of your State or Province?

Full name of the state or province in which your organization is located.

Example: California

What is the two-letter country code for this unit?

Two-letter country code of the country in which your organization is located.

Example: US

- After providing all the information above you will see a confirmation prompt similar to the following:

```
Is CN=cmc.mycompany.com, OU=Engineering, O=MyCompany, L=San Francisco, ST=California,
C=US correct?
[no] :
```

Check the information listed and if correct, type yes and press Enter.

- You will see the following prompt:

```
Enter key password for <cloudian>
(RTURN if same as keystore password) :
```

Press Enter to set the key password for the *cloudian* alias to the same value as your Keystore's password. The *keytool* then generates the Keystore file under the */etc/cloudian-<version>-puppet/module/cmc/files* directory (the directory that you went to in Step 2).

- Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

- From the installer's main menu select "Advanced Configuration Options".
- From the Advanced Configuration Options menu select "Import Java keystore to CMC".
- Provide the following information at the prompts.

Please enter name of your key store for CMC [.keystore_cloudian]:

Name of the new Keystore file that you created earlier in this procedure. This should be different than ".keystore_cloudian", which is the default Keystore that comes bundled with the CMC.

Example: cmc.jks

Please enter the password for the key store [123cloudian456]:

The password for the Keystore file. This must be the same Keystore password that you set earlier in this procedure (in Step 4).

Use alphanumeric characters only.

Example: Password

11. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
12. From the main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
13. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the CMC service. After this is done you can exit the installer.

Once the CMC service has been restarted on all nodes in the cluster, issue the following command on each node in the cluster to verify that the CMC is listening on TCP port 8443:

```
[root]# netstat -tln | grep :8443
```

You should see output similar to the following:

```
tcp 0 0 ::ffff:10.50.200.51: 8443 ::*: LISTEN
```

In a browser, if you navigate to <https://<your-cmc-domain>:8443> you should see a warning that the domain is using a self-signed certificate.

You have now completed the procedure for creating and using your own self-signed certificate for CMC TLS/SSL.

9.6.8. TLS/SSL for the CMC (CA-Verified Certificate)

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Generate a Certificate and a Certificate Signing Request" (page 557)**
- **"Import the CA-Signed Certificates into the Keystore" (page 558)**
- **"Configure the CMC to Use the New Keystore" (page 560)**

By default the CMC already supports TLS/SSL, and CMC clients must use HTTPS to communicate with the CMC. The default CMC TLS/SSL implementation uses a self-signed certificate -- unique to your system -- that is generated automatically during HyperStore installation. Consequently, no configuration steps are required in regard to the CMC's use of HTTPS -- it works out-of-the-box.

The only optional CMC TLS/SSL set-up actions you may want to consider are 1) creating your own self-signed certificate and Keystore for the CMC to use rather than the default Keystore; or 2) acquiring a CA-verified certificate and having the CMC use that certificate in its TLS/SSL implementation. Using a CA-verified certificate is described below. For using your own self-signed certificate see **"TLS/SSL for the CMC (Self-Signed Certificate)" (page 553)**.

To use a **CA-verified TLS/SSL certificate** for your CMC service you will first generate a certificate and a corresponding Certificate Signing Request (CSR) that you can submit to your preferred CA. Then after the CA has verified and returned the certificate to you (typically along with a root certificate and intermediate certificate), you will import the certificates into your Keystore and then configure the CMC to use the new Keystore.

9.6.8.1. Generate a Certificate and a Certificate Signing Request

1. Log into the Puppet master node as *root*.
2. Go to the */etc/cloudian-<version>-puppet/modules/cmc/files* directory for your current HyperStore version, for example:

```
[root]# cd /etc/cloudian-7.2-puppet/modules/cmc/files
```

3. Issue the following command to create a new TLS/SSL key pair and store them in a Java Keystore file called *cmc.jks*:

```
[root]# /usr/java/default/bin/keytool -genkey -alias cloudian -keyalg RSA -keysize 2048 -validity 365 -keystore cmc.jks
```

Note: Do not name the Keystore file ".keystore_cloudian". That is the name of the default Keystore that comes bundled with the CMC.

4. When prompted by the keytool utility, provide the following information:

Enter keystore password

The password to use for the Keystore file. Retain this password in a safe location — you will need to provide it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: Password

Re-enter new password

Re-enter the same password.

Example: Password

What is your first and last name? (CMC FQDN!)

Enter the CMC's fully qualified domain name, **not your first and last name**. This domain should match the CMC domain that was specified at the time your HyperStore cluster was installed and configured.

Do not use *cloudian.com* in the CMC's FQDN, as this domain belongs to Cloudian, Inc. and your CMC service will not work properly if you do.

Example: cmc.mycompany.com

What is the name of your organizational unit?

Name of your department or work group.

Example: Engineering

What is the name of your organization?

Name of your organization.

Example: MyCompany

What is the name of your City or Locality?

City or locality in which your organization is located.

Example: San Francisco

What is the name of your State or Province?

Full name of the state or province in which your organization is located.

Example: California

What is the two-letter country code for this unit?

Two-letter country code of the country in which your organization is located.

Example: US

- After providing all the information above you will see a confirmation prompt similar to the following:

```
Is CN=cmc.mycompany.com, OU=Engineering, O=MyCompany, L=San Francisco, ST=California,  
C=US correct?  
[no] :
```

Check the information listed and if correct, type *yes* and press Enter.

- You will see the following prompt:

```
Enter key password for <cloudian>  
(RETURN if same as keystore password) :
```

Press Enter to set the key password for the *cloudian* alias to the same value as your Keystore's password. The *keytool* then generates the Keystore file under the */etc/cloudian-<version>-puppet/module/cmc/files* directory (the directory that you went to in Step 2).

- Issue the following command to generate a Certificate Signing Request (CSR) for your TLS/SSL Certificate:

```
[root]# /usr/java/default/bin/keytool -certreq -alias cloudian -file cmc.csr -  
keystore cmc.jks
```

You will be prompted for your Keystore password. Type your Keystore password and press Enter.

The command above will generate a file called *cmc.csr* (in the same directory as your Keystore file) which will contain your Certificate Signing Request. Submit this CSR file to your preferred Certificate Authority (CA) for signing, using the instructions from the CA. When submitting the request be sure to select the TLS/SSL Certificate as the type compatible with the **Tomcat Server**. If Tomcat is not available as a Certificate Type, then select Apache compatible instead.

9.6.8.2. Import the CA-Signed Certificates into the Keystore

Once your CA has signed your TLS/SSL Certificate, they should send it to you along with their Root and Intermediate CA certificates. Follow the steps below to import all the certificates to your CMC Keystore.

Note If your CA is GoDaddy.com, then they should provide you a copy of the following three certificate files with your signed TLS/SSL Certificate:

- * gdroot-g2_cross.crt (GoDaddy G1 to G2 Cross Root Certificate)
- * gd_cross_intermediate.crt (GoDaddy Cross Intermediate CA Certificate)

* gdig2.crt (GoDaddy Intermediate G2 CA Certificate)

You will need GoDaddy's Cross Root, Cross Intermediate CA and Intermediate G2 certificate files in order to be able to import your signed TLS/SSL Certificate into your Keystore file. If GoDaddy did not provide you a copy of their Cross Root, Cross Intermediate CA or Intermediate G2 Certificate file, you can download them from the following URLs:

- * GoDaddy G1 to G2 Cross Root Certificate: https://certs.godaddy.com/repository/gdroot-g2_cross.crt
- * GoDaddy Cross Intermediate CA Certificate: https://certs.godaddy.com/repository/gd_cross_intermediate.crt
- * GoDaddy Intermediate G2 CA Certificate: <https://certs.godaddy.com/repository/gdig2.crt>

1. Log in to the Puppet master node as user *root*.
2. Copy all the certificates that you received from the CA into the */etc/cloudian-<version>-puppet/modules/cmc/files* directory (where your CMC Keystore file is).
3. Go to directory where your Keystore and certificates are. For example:

```
[root]# cd /etc/cloudian-7.2-puppet/modules/cmc/files
```

4. Issue the following commands to import the Root CA Certificate and Intermediate CA Certificate into your Keystore file (assuming you used *cmc.jks* as the Keystore name when you created the Keystore earlier in this procedure):

```
[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias root -file <Root CA Certificate File> -keystore cmc.jks
```

```
[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias intermediate -file <Intermediate CA Certificate File> -keystore cmc.jks
```

Example for GoDaddy as the CA:

```
[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyRoot -file gdrootg2_cross.crt -keystore cmc.jks

[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyCrossCA -file gd_cross_intermediate.crt -keystore cmc.jks

[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyG2CA -file gdig2.crt -keystore cmc.jks
```

5. For the sake of clarity, rename or copy the Signed TLS/SSL Certificate file that was provided by your CA to *cmc.crt*.

```
[root]# cp -p <signed certificate file> cmc.crt
```

Example:

```
[root]# cp -p 749830d8b451822b.crt cmc.crt
```

6. Issue the following command to import your signed TLS/SSL Certificate into your Keystore file:

```
[root]# /usr/java/default/bin/keytool -import -trustcacerts -alias cloudian -
```

```
file  
cmc.crt -keystore cmc.jks
```

Now your CMC Keystore is all set and all that remains is to configure the CMC to use this new Keystore rather than the default Keystore that comes bundled with the CMC.

9.6.8.3. Configure the CMC to Use the New Keystore

1. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

2. From the installer's main menu select "Advanced Configuration Options".
3. From the Advanced Configuration Options menu select "Import Java keystore to CMC".
4. Provide the following information at the prompts.

Please enter name of your key store for CMC [keystore_cloudian]:

Name of the new Keystore file that you created earlier in this procedure. This should be different than ".keystore_cloudian", which is the default Keystore that comes bundled with the CMC.

Example: cmc.jks

Please enter the password for the key store [123cloudian456]:

The password for the Keystore file. This must be the same Keystore password that you set earlier in this procedure (under "**Generate a Certificate and a Certificate Signing Request**" (page 557), Step 4).

Use alphanumeric characters only.

Example: Password

5. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
6. From the main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
7. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the CMC service. After this is done you can exit the installer.

Once the CMC service has been restarted on all nodes in the cluster, issue the following command on each node in the cluster to verify that the CMC is listening on TCP port 8443:

```
[root]# netstat -tln | grep :8443
```

You should see output similar to the following:

```
tcp 0 0 ::ffff:10.50.200.51: 8443 ::*: LISTEN
```

Next, in a browser navigate to <https://<your-cmc-domain>:8443/>

You can safely ignore the "Access Denied" error. However, check the certificate presented to your browser by the CMC service and confirm that it is signed by your CA and that its Common Name (i.e. CN) value matches the domain that you specified when you created your CMC Keystore file, e.g. *cmc.mycompany.com*.

You have now completed the procedure for acquiring and using a CA-verified certificate for CMC TLS/SSL.

9.6.9. TLS/SSL for IAM Service (Self-Signed Certificate)

Optionally, you can configure HyperStore's IAM Service to support HTTPS connections from client applications. The procedure below describes how to set up TLS/SSL for your IAM Service using a **self-signed TLS/SSL certificate**. The procedure describes how to create the certificate as well as how to have the IAM Service use the certificate. A self-signed certificate may be your preferred approach for an evaluation or testing environment. (Alternatively, for instructions on using a CA-verified certificate -- which may be more appropriate in a production environment -- see "**TLS/SSL for IAM Service (CA-Verified Certificate)**" (page 563)).

Follow the procedure below if you want to use a self-signed TLS/SSL certificate to set up HTTPS support for your HyperStore IAM Service.

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu select option **m**, "**Generate a self-signed certificate for IAM in a JKS keystore**".
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Please enter key store name [region1cloudian-iam.jks]

Name of the Keystore file which will be created, and in which your TLS/SSL certificate for IAM will be stored. Make a note of this name as you will need it again later in this procedure. **Do not name the key-store "keystore"**.

Example: region1cloudian-iam.jks

Please enter alias name []

An alias for identifying the TLS/SSL key pair that will be created and stored in the Keystore (and which will be used by the IAM Service). Make a note of this alias as you will need it again later in this procedure.

Example: cloudianiam

Please enter the password for region1cloudian-iam.jks [testpass]

The password to set for the Keystore file. Retain this password in a safe location — you will need it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: MyPassword

Please enter the key store manager password for region1cloudian-iam.jks [testpass]

Enter the same password that you entered at the previous prompt.

Example: MyPassword

Enter your organizational unit name []

Name of your department or work group

Example: Engineering

Enter the name of your organization []

Organization name

Example: MyCompany

Enter the name of your City or Locality []

City or locality in which your organization is located

Example: San Francisco

Enter the name of your State or Province []

Full name of the state or province in which your organization is located

Example: California

Enter the two-letter country code for this unit []

Two-letter country code for the country in which your organization is located

Example: US

Note When you complete the above prompts, the installer creates a new Keystore file (with your specified Keystore name) under `/etc/cloudian-<version>-puppet/modules/baselayout/files/`. It also creates a Certificate Signing Request (CSR) in the same directory. If your Keystore file is named `region1cloudian-iam.jks` for example, then the CSR file will be named `region1cloudian-iam.csr`. However, since you are using a self-signed certificate in this procedure (not a CA-verified certificate), you do not need to do anything with the CSR file.

After completing all the prompts for generating the certificate and keystore, press any key to return to the installer's Advanced Configuration Options menu.

6. From the Advanced Configuration Options menu, select option **n, "Enable and configure HTTPS access for IAM"**.
7. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Do you wish to enable HTTPS access for IAM service? [no]

Enter yes to enable HTTPS access on the IAM Service.

Please enter name of your key store [region1cloudian-iam.jks]

Name of the Keystore file that contains the key pair that you are using for TLS/SSL on the IAM Service. This must be the same Keystore file name that you specified in Step 5 of this procedure.

Example: region1cloudian-iam.jks

Certificate alias to use []

Alias identifying the key pair that you are using for TLS/SSL on the IAM Service. This must be the same alias that you specified in Step 5 of this procedure.

Example: cloudianiam

Please enter password for the keystore region1cloudian-iam.jks [testpass]

Password for the IAM Service to use when accessing your Keystore file. This must be the same

Keystore password that you set in Step 5 of this procedure.

Note that before storing Keystore-related passwords in HyperStore configuration files, the installer automatically obfuscates the passwords using a Jetty password obfuscation utility. For example, if your plain text password is "MyPassword", it will be entered in HyperStore configuration files as "OBF:1o4o1zly1qw01vu11ym71ym71vv91qxq1zlk1o5y".

Example: MyPassword

Please enter keystore manager password [testpass]

Enter your regular Keystore password.

Example: MyPassword

Please enter path name in which to store keystore file [/opt/cloudian/conf]

Puppet will propagate the Keystore file to this location on your IAM Service nodes. Accept the default value.

Example: /opt/cloudian/conf

8. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
9. From the installer's main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
10. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the S3 Service and the CMC. After this is done you can exit the installer.

You have now completed the procedure for creating and using a self-signed certificate for enabling HTTPS on the IAM Service.

9.6.9.1. To Have the CMC Connect to the IAM Service Using HTTPS

For the CMC to be able to connect to the IAM Service using HTTPS, you must manually import the IAM certificate into the default JVM keystore on each node on which the CMC runs. Follow these steps on each node:

1. Retrieve the certificate content and write it to a certificate named *iam.crt*. Enter this command as one line with no breaks. In the command, replace *localdomain* with your domain.

```
echo -n | openssl s_client -connect s3-admin.localdomain:16443 | sed -ne
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > iam.crt
```

2. Import *iam.crt* into the default JVM keystore on the node. Enter this command as one line with no breaks.

```
keytool -import -trustcacerts -keystore
/usr/java/jdk1.8.0_172-amd64/jre/lib/security/cacerts -storepass changeit -
noprompt
-alias iamcert -file iam.crt
```

9.6.10. TLS/SSL for IAM Service (CA-Verified Certificate)

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Generate a Certificate and a Certificate Signing Request" (page 564)**
- **"Import the CA-Signed Certificates into the Keystore" (page 565)**
- **"Configure the HyperStore S3 Service to Support HTTPS" (page 567)**

Optionally, you can configure your S3 Service to support HTTPS connections from S3 client applications. The procedure below describes how to set up TLS/SSL for your S3 Service using a **CA-verified TLS/SSL certificate** (a certificate verified by a third party Certificate Authority organization). The procedure describes how to create the certificate and have it verified, as well as how to have the S3 Service use the certificate. A CA-verified certificate may be your preferred approach for a production environment. (Alternatively, for instructions on using a self-signed certificate -- which may be adequate in a testing environment -- see **"TLS/SSL for S3 Service (Self-Signed Certificate)" (page 543)**).

To use a CA-verified TLS/SSL certificate for your HyperStore S3 Service you will first generate a certificate and a corresponding Certificate Signing Request (CSR) that you can submit to your preferred CA. Then after the CA has verified and returned the certificate to you (typically along with a root certificate and intermediate certificate), you will import the certificates into your Keystore. As the third and final stage of this procedure you will configure the HyperStore S3 Service to support HTTPS.

9.6.10.1. Generate a Certificate and a Certificate Signing Request

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu select option **m, "Generate a self-signed certificate in a JKS keystore"**.
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Please enter key store name [region1cloudian-iam.jks]

Name of the Keystore file which will be created, and in which your TLS/SSL certificate for IAM will be stored. Make a note of this name as you will need it again later in this procedure. **Do not name the key-store "keystore".**

Example: region1cloudian-iam.jks

Please enter alias name []

An alias for identifying the TLS/SSL key pair that will be created and stored in the Keystore.

Example: cloudianiamselfsigned

Please enter the password for region1cloudian-iam.jks [testpass]

The password to set for the Keystore file. Retain this password in a safe location — you will need it again later in this procedure. You will also need it anytime you want to access the Keystore file.

Use alphanumeric characters only.

Example: MyPassword

Please enter the key store manager password for region1cloudian-iam.jks [testpass]

Enter the same password that you entered at the previous prompt.

Example: MyPassword

Enter your organizational unit name []

Name of your department or work group

Example: Engineering

Enter the name of your organization []

Organization name

Example: MyCompany

Enter the name of your City or Locality []

City or locality in which your organization is located

Example: San Francisco

Enter the name of your State or Province []

Full name of the state or province in which your organization is located

Example: California

Enter the two-letter country code for this unit []

Two-letter country code for the country in which your organization is located

Example: US

When you complete the above prompts, the installer creates a new Keystore file (with your specified Keystore name) under `/etc/cloudian-<version>-puppet/modules/baselayout/files/`. It also creates a Certificate Signing Request (CSR) in the same directory. If your Keystore file is named `region1cloudian-iam.jks` for example, then the CSR file will be named `region1cloudian-iam.csr`.

Submit this CSR file to your preferred Certificate Authority for signing, using the instructions from the CA.

9.6.10.2. Import the CA-Signed Certificates into the Keystore

Once your CA has signed your TLS/SSL Certificate, they should send it to you along with their Root and Intermediate CA certificates. Follow the steps below to import all these certificates to your Keystore.

Note If your CA is GoDaddy.com, then they should provide you a copy of the following three certificate files with your signed TLS/SSL Certificate:

`gdroot-g2_cross.crt` (GoDaddy G1 to G2 Cross Root Certificate)
`gd_cross_intermediate.crt` (GoDaddy Cross Intermediate CA Certificate)
`gdig2.crt` (GoDaddy Intermediate G2 CA Certificate)

You will need GoDaddy's Cross Root, Cross Intermediate CA and Intermediate G2 certificate files in order to be able to import your signed TLS/SSL Certificate into your Keystore file. If GoDaddy did not provide you a copy of their Cross Root, Cross Intermediate CA or Intermediate G2 Certificate file, you can download them from the following URLs:

GoDaddy G1 to G2 Cross Root Certificate:

https://certs.godaddy.com/repository/gdroot-g2_cross.crt

GoDaddy Cross Intermediate CA Certificate:

https://certs.godaddy.com/repository/gd_cross_intermediate.crt

GoDaddy Intermediate G2 CA Certificate:

<https://certs.godaddy.com/repository/gdig2.crt>

1. Log in to the Puppet master node as user *root*.
2. Copy all the certificates that you received from the CA into the */etc/cloudian-<version>-puppet/modules/baseLayout/files* directory (where your Keystore file is).
3. Go to the */etc/cloudian-<version>-puppet/modules/baseLayout/files* directory. For example:

```
[root]# cd /etc/cloudian-7.2-puppet/modules/baseLayout/files
```

4. Issue the following commands to import the Root CA Certificate and Intermediate CA Certificate into your Keystore file. For example, if your Keystore is named *region1cloudian-iam.jks*:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias root
-file <Root CA Certificate File> -keystore region1cloudian-iam.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias intermediate
-file <Intermediate CA Certificate File> -keystore region1cloudian-iam.jks
```

Example for GoDaddy as the CA:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyRoot
-file gdrootg2_cross.crt -keystore region1cloudian-iam.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias
GoDaddyCrossCA
-file gd_cross_intermediate.crt -keystore region1cloudian-iam.jks
```

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias GoDaddyG2CA
-file gdig2.crt -keystore region1cloudian-iam.jks
```

5. For the sake of clarity, rename or copy the CA-signed TLS/SSL Certificate file to *cloudianIam.crt*.

```
[files]# cp -p <signed certificate file> cloudianIam.crt
```

Example:

```
[files]# cp -p 769830d8b471822b.crt cloudianIam.crt
```

6. Issue the following command to import your CA-signed TLS/SSL Certificate into your Keystore file:

```
[files]# /usr/java/default/bin/keytool -import -trustcacerts -alias cloudianiam
-file cloudianIam.crt -keystore region1cloudian-iam.jks
```

Now your Keystore is all set and all that remains is to enable and configure HTTPS on the IAM Service.

9.6.10.3. Configure the HyperStore S3 Service to Support HTTPS

1. Log into the Puppet master node as *root*.
2. Go to your HyperStore installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```

3. From the installer's main menu select "Advanced Configuration Options".
4. From the Advanced Configuration Options menu, select option **n**, "Enable and configure HTTPS access for IAM".
5. Provide the following information at the prompts. To accept a default (as indicated in the bracketed values in the prompt text) you can simply press Enter.

Do you wish to enable HTTPS access for IAM service? [no]

Enter yes to enable HTTPS access on the IAM Service.

Please enter name of your key store [region1cloudian-iam.jks]

Name of the Keystore file that contains the key pair that you are using for TLS/SSL on the IAM Service. This must be the same Keystore file name that you specified in Step 5 under "**Generate a Certificate and a Certificate Signing Request**" (page 564).

Example: region1cloudian-iam.jks

Certificate alias to use []

Alias identifying the key pair that you are using for TLS/SSL on the IAM Service. This must be the same alias that you specified in Step 6 under "**Import the CA-Signed Certificates into the Keystore**" (page 565).

Example: cloudianiam

Please enter password for the keystore region1cloudian-iam.jks [testpass]

Password for the IAM Service to use when accessing your Keystore file. This must be the same Keystore password that you set in Step 5 under "**Generate a Certificate and a Certificate Signing Request**" (page 564).

Note that before storing Keystore-related passwords in HyperStore configuration files, the installer automatically obfuscates the passwords using a Jetty password obfuscation utility. For example, if your plain text password is "MyPassword", it will be entered in HyperStore configuration files as "OBF:1o4o1zly1qw01vu11ym71ym71vv91qxq1zlk1o5y".

Example: MyPassword

Please enter keystore manager password [testpass]

Enter your regular Keystore password.

Example: MyPassword

Please enter path name in which to store keystore file [/opt/cloudian/conf]

Puppet will propagate the Keystore file to this location on your IAM Service nodes. Accept the default value.

Example: /opt/cloudian/conf

6. After completing the above prompts you are returned to the Advanced Configuration Options menu. From here select "Return to Main Menu".
7. From the main menu select "Cluster Management", then select "Push Configuration Settings to Cluster". Follow the prompts to push the configuration update out to all the nodes in your cluster. The push of configuration settings may take a few minutes. After it completes press any key to return to the Cluster Management menu.
8. From the Cluster Management menu select "Manage Services", then use the Service Management menu to restart the S3 Service and the CMC. After this is done you can exit the installer.

You have now completed the procedure for acquiring and using a CA-verified certificate for S3 TLS/SSL.

9.6.10.4. To Have the CMC Connect to the IAM Service Using HTTPS

For the CMC to be able to connect to the IAM Service using HTTPS, you must manually import the IAM certificate into the default JVM keystore on each node on which the CMC runs. Follow these steps on each node:

1. Retrieve the certificate content and write it to a certificate named *iam.crt*. Enter this command as one line with no breaks. In the command, replace *localdomain* with your domain.

```
echo -n | openssl s_client -connect s3-admin.localdomain:16443 | sed -ne  
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > iam.crt
```

2. Import *iam.crt* into the default JVM keystore on the node. Enter this command as one line with no breaks.

```
keytool -import -trustcacerts -keystore  
/usr/java/jdk1.8.0_172-amd64/jre/lib/security/cacerts -storepass changeit -  
noprompt  
-alias iamcert -file iam.crt
```

9.6.11. Tuning HyperStore Performance Parameters

The HyperStore system includes a performance configuration optimization script that is automatically run on each node when you install HyperStore; and that also automatically runs on any new nodes that you subsequently add to your cluster. The script adjusts OS configuration settings on each node, and certain HyperStore system configuration settings, for optimal performance based on your particular environment (taking into account factors such as RAM and CPU specs).

Since the script runs automatically during installation and during cluster expansion, under normal circumstances you should not need to run the script yourself. However, you can run the performance configuration optimization script indirectly through the installer's Advanced Configuration Options menu, if for example you have made configuration changes on your own and your system is now under-performing as a result. Running the script in this way will return the configuration settings to the optimized values as determined by the script.

To run the script from the Advanced Configuration Options menu:

1. On the Puppet master node, change to the installation staging directory and then launch the installer:

```
[7.2]# ./cloudianInstall.sh
```
2. From the installer's menu select "Advanced Configuration Options" and then select "Configure Performance Parameters on Nodes".

3. At the prompt, specify a node for which to run the performance configuration optimization; or specify a comma-separated list of nodes; or leave the prompt blank and press enter if you want to run the optimization for all nodes in your cluster. When the script run is done the installer interface will prompt you to continue to the next steps.
4. Go to the installer's main menu again and choose "Cluster Management" → "Push Configuration Settings to Cluster" and follow the prompts.
5. Go to the "Cluster Management" menu again, choose "Manage Services", and restart the S3 Service, the HyperStore Service, and the Cassandra Service.

9.6.12. Vanity Domains for S3 Buckets

The HyperStore system supports the use of "vanity domains", whereby an S3 bucket can be accessed using just the bucket name as the URL, without the service provider's domain name. To use this feature:

- The bucket name must exactly equal the desired vanity domain name. The bucket name must be DNS-compatible, and must use only lower case letters. For example:

Desired Vanity Domain Name	Required Bucket Name
baseball-stats.com	baseball-stats.com
japan.travel.org	japan.travel.org
never-ending-profits.biz	never-ending-profits.biz

IMPORTANT: The desired vanity domain must not already exist in the global DNS system or this feature will not work. It's up to end users to ensure that their proposed vanity domains do not yet exist on the web.

- In your DNS set-up, use CNAME to map the vanity domain to the bucket's fully qualified name. For example, if your HyperStore S3 service domain is `s3.enterprise.com` then you would map vanity domain `baseball-stats.com` to CNAME `baseball-stats.com.s3.enterprise.com`.
- In your DNS set-up, use CNAME to map a wildcard of sub-domains of your S3 service domain to the S3 service domain itself. For example, if your HyperStore S3 service domain is `s3.enterprise.com` then you would map wildcard `*.s3.enterprise.com` to CNAME `s3.enterprise.com`.

9.6.13. Configuring the HyperStore Firewall

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Enabling or Disabling the HyperStore Firewall"** (page 570)
- **"Default Behavior of the HyperStore Firewall When Enabled"** (page 571)
- **"Customizing the HyperStore Firewall"** (page 572)

Each HyperStore node includes a built-in firewall that is pre-configured with settings appropriate for a typical HyperStore deployment. The HyperFile firewall is either enabled or disabled by default depending on whether your original HyperStore installation was older than version 7.2:

- In systems originally installed as version 7.2 or newer, the HyperStore firewall is enabled by default.
- In systems originally installed as a version older than 7.2 and then later upgraded to 7.2 or newer, the HyperStore firewall is available but is disabled by default.

You can enable or disable the HyperStore firewall using the installer's Advanced Configuration Options, as described below in **"Enabling or Disabling the HyperStore Firewall"** (page 570). When the firewall is enabled, you can optionally customize certain aspects of the firewall's behavior, as described further below in **"Customizing the HyperStore Firewall"** (page 572).

Cloudian, Inc. strongly recommends using a firewall to protect sensitive internal services such as Cassandra, Redis, and so on, while allowing access to public services. The pre-configured HyperStore firewall serves this purpose, if enabled. Alternatively, if you have upgraded to HyperStore 7.2 from an older version and you already have a custom firewall in place that you have been successfully using with HyperStore, you may prefer to keep using that firewall -- since the HyperStore firewall is limited as to how much it can be customized.

If you have upgraded to HyperStore 7.2 from an older version and you do wish to enable the HyperStore firewall rather than continuing to use your own custom firewall, then before enabling the HyperStore firewall do the following:

- If you have created custom *firewalld* Zone and Service configuration files, make a backup copy of those files for your own retention needs. When you enable the HyperStore firewall **your existing Zone and Service configuration files will be deleted from the /etc/firewalld directory**.
- Disable your existing firewall service on each node. For example, to disable *firewalld* do the following on each node:

```
[root]# systemctl stop firewalld  
[root]# systemctl disable firewalld
```

The HyperStore installer will not allow you to enable the HyperStore firewall on your nodes if an existing firewall service is active.

Note The HyperStore firewall service is implemented as a custom version of the *firewalld* service, named *cloudian-firewalld*.

9.6.13.1. Enabling or Disabling the HyperStore Firewall

To enable or disable the HyperStore firewall on all your HyperStore nodes:

1. On the Puppet master node, change into your current HyperStore version installation staging directory. Then launch the installer.

```
[7.2]# ./cloudianInstall.sh
```
2. At the installer main menu enter **4** for Advanced Configuration Options.
3. At the Advanced Configuration Options menu enter **s** for Configure Firewall.
4. At the Cloudian HyperStore Firewall Configuration menu enter **a** for Enable/Disable Cloudian HyperStore Firewall.
5. In the Enable/Disable Cloudian HyperStore Firewall interface, the firewall's current status is displayed. At the prompt enter **enable** to enable the firewall or **disable** to disable the firewall. After the interface indicates that the firewall is set as you specified, press any key to return to the Firewall Configuration menu.
6. At the Firewall Configuration menu enter **x** to apply your configuration changes. Then at the next prompt that displays enter **yes** to confirm that you want to apply your configuration changes. This will

apply your change to all nodes in your HyperStore system (all nodes in all of your data centers and service regions).

The installer interface should then display a status message "changes: True, result: **OK**" for each node, to indicate the successful applying of your configuration change to each node.

Note If the installer displays a Warning about one or more nodes not responding you can try the Firewall Configuration menu's **x** option again, and this retry may work for the node(s) if the problem the first time was a temporary network issue. If one of your nodes is **down**, the node will automatically be updated with your configuration change when the node comes back online.

You are now done with enabling or disabling the HyperStore firewall. **You do not need to do a Puppet push or restart any services.**

9.6.13.2. Default Behavior of the HyperStore Firewall When Enabled

When the HyperStore firewall is enabled, the default behavior on each HyperStore node is as follows:

- On all IP interfaces, all TCP ports will allow inbound traffic originating from other HyperStore nodes. In a multi-DC or multi-region system, this includes inbound traffic originating from HyperStore nodes in other DCs or regions.
- On all IP interfaces, only the following TCP ports will allow inbound traffic that originates from sources outside your HyperStore cluster:
 - S3 HTTP service port (80 by default)
 - S3 HTTPS service port (443 by default)
 - S3 PROXY HTTP service port (81)
 - S3 PROXY HTTPS service port (4431)
 - IAM HTTP service port (16080 by default)
 - IAM HTTPS service port (16443 by default)
 - SQS HTTP service port (18090 by default)
 - SQS HTTPS service port (18443 by default)
 - CMC HTTP service port (8888 by default)
 - CMC HTTPS service port (8443 by default)
 - Admin HTTPS service port (19443 by default)
 - SSH service port (22)

Traffic originating from outside your HyperStore cluster will be blocked (DROP'd) on all TCP ports other than those listed above.

Note The firewall also allows incoming ICMP traffic originating from outside your HyperStore cluster.

- On all IP interfaces, outbound traffic is allowed on all ports.

When the firewall is enabled, the firewall configuration will automatically adjust to system changes in the following ways:

- Because the firewall is configured by reference to services rather than hard-coded port numbers, if you [change the port that a particular HyperStore public service uses](#), the firewall accommodates this change automatically.
- If you resize your cluster by adding or removing nodes, or by adding or removing a data center or service region, the firewall accommodates this change automatically. In the case of expanding your cluster, the firewall will be automatically enabled on new nodes, and the firewall on the existing nodes will allow inbound traffic from the new nodes, on any port. In the case of removing nodes, the firewall on the existing nodes will be updated such that the removed nodes are no longer part of the cluster and can only access the cluster's public services.

Note: If the firewall is disabled when you add new nodes to your cluster, the firewall will also be disabled on the new nodes.

On each node, requests blocked by the HyperStore firewall are logged to `/var/log/firewall.log`. For more information about this log including its rotation and retention behavior, see "[HyperStore Firewall Log](#)" (page 592).

For details about HyperStore port usage "HyperStore Listening Ports" in the Reference section of the *HyperStore Installation Guide*.

9.6.13.3. Customizing the HyperStore Firewall

The default behavior of the HyperStore firewall (when enabled) is as described above. If you wish you can customize the behavior by denying access to one of the services that the firewall allows access to by default. For example, you could deny access to the S3 HTTP service so that the S3 HTTPS service is used exclusively. Subsequently, if there is a change in your preferences or circumstances, you could customize the firewall to once again allow access to that service.

To customize the HyperStore firewall on all your HyperStore nodes:

1. On the Puppet master node, change into your current HyperStore version installation staging directory. Then launch the installer.

```
[7.2]# ./cloudianInstall.sh
```

2. At the installer main menu enter **4** for Advanced Configuration Options.
3. At the Advanced Configuration Options menu enter **s** for Configure Firewall.
4. At the Cloudian HyperStore Firewall Configuration menu enter the menu letter corresponding to the service for which you want to deny or allow access (for example **d** for S3 HTTP).

```

Cloudian HyperStore(R) Firewall Configuration
-----
When enabling the Cloudian HyperStore(R) Firewall, all internal services
like Cassandra and Hyperstore are automatically protected to outside access
and by default, connections to all public-facing services like S3 and IAM,
are all allowed. External access to each service can be further managed by
enabling (Allow) or disabling (Deny) traffic to each public Service.

a ) Enable/Disable Cloudian HyperStore(R) Firewall

b ) Configure access to CMC HTTP
c ) Configure access to CMC HTTPS
d ) Configure access to S3 HTTP
e ) Configure access to S3 HTTPS
f ) Configure access to IAM HTTP
g ) Configure access to IAM HTTPS
h ) Configure access to SQS HTTP
i ) Configure access to SQS HTTPS
j ) Configure access to Admin API HTTPS

x ) Return to previous menu

Choice: [ ]

```

5. In the Allow/Deny Access to Service <Service Type> interface, the service's current setting is displayed ("Allow" or "Deny"). At the prompt enter **deny** to deny access to the service or **allow** to allow access to the service. After the interface indicates that the service is set as you specified, press any key to return to the Firewall Configuration menu.
6. At the Firewall Configuration menu enter **x** to apply your configuration changes. Then at the next prompt that displays enter **yes** to confirm that you want to apply your configuration changes. This will apply your change to all nodes in your HyperStore system (all nodes in all of your data centers and service regions).

The installer interface should then display a status message "changes: True, result: **OK**" for each node, to indicate the successful applying of your configuration change to each node.

Note If the installer displays a Warning about one or more nodes not responding you can try the Firewall Configuration menu's **x** option again, and this retry may work for the node(s) if the problem the first time was a temporary network issue. If one of your nodes is **down**, the node will automatically be updated with your configuration change when the node comes back online.

You are now done with customizing the HyperStore firewall. **You do not need to do a Puppet push or restart any services.**

This page left intentionally blank

Chapter 10. Logging

10.1. HyperStore Logs

The major HyperStore services each generate their own application log. The S3 Service, Admin Service, and HyperStore Service, in addition to generating application logs, also generate transaction (request) logs.

The log descriptions below indicate each log's default location, logging level, rotation and retention policy, log entry format, and where to modify the log's configuration.

Note With the exception of Cassandra and Redis logs, all HyperStore logs are located in `/var/log/cloudian`.

10.1.1. Admin Service Logs

Admin Service application log (`cloudian-admin.log`)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-admin.log</code>
Log Entry Format	<code>YYYY-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE</code> The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help.
Log Entry Example	<code>2017-05-04 15:48:03,158 ERROR[main]HS081003 Cas-sandraProtectionPolicy:Caught: me.prettyprint.hector.api.exceptions.HectorException: [10.50.10.21 (10.50.10.21):9160] All host pools marked down. Retry burden pushed out to client.</code>
Default Logging Level	INFO
Default Rotation and Retention Policy	Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size. Rotated files are named as <code>cloudian-admin.log.YYYY-MM-DD.i.gz</code> , where <i>i</i> is a rotation counter that resets back to 1 after each day. Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Configuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-admin.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFileName="ADMINAPP"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

Admin Service request log (cloudian-admin-request-info.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-admin-request-info.log</code>
Log Entry Format	<code>YYYY-mm-dd HH:m-m:ss,SSS ClientIpAddress HttpMethod Uri QueryParams DurationMicrosecs HttpStatus</code> <div style="background-color: #e0f2e0; padding: 5px; margin-top: 10px;"> Note Query parameters are not logged for requests that involve user credentials. </div>
Log Entry Example	<code>2016-10-27 14:54:01,170 10.20.2.57 GET /group/list limit:100 188212 200</code>
Logging Level	Not applicable
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <code>cloudian-admin-request-info.log.YYYY-MM-DD.i.gz</code>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 2GB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-admin.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="ADMINREQ"></code> . For setting descriptions see " Log Configuration Settings " (page 593).

10.1.2. Cassandra Logs

Cassandra application log (system.log)

Name and Location	On every node, <code>var/log/cassandra/system.log</code>
Log Entry Format	<code>PriorityLevel [ThreadId] Date(ISO8601) CallerFile:Line# - MESSAGE</code>
Log Entry Example	<code>INFO [FlushWriter:3] 2016-11-10 21:09:59,487 Memtable.java:237 - Writing Memtable-Migrations @445036697(12771/15963 serialized/live bytes, 1 ops)</code>
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 20MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <code>system.log.YYYY-MM-DD.i.gz</code>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p>

	Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 200MB or if oldest rotated file age reaches 30 days.
Con- figuration	Configuration: <code>/etc/cloudian-<version>-puppet/modules/cassandra/templates/logback.xml.erb</code> . For setting descriptions see the online documentation for Logback: <ul style="list-style-type: none">• FixedWindowRollingPolicy• SizeBasedTriggeringPolicy

Cassandra request log (cassandra-s3-tx.log)

Name and Location	On every node, <code>/var/log/cloudian/cassandra-s3-tx.log</code> . Note that these request logs are written on the client side as the S3 Service sends requests to Cassandra.
Log Entry Format	<code>YYYY-mm-dd HH:mm:ss,SSS LogEntryType S3RequestId MESSAGE</code> The LogEntryType is one of NORMAL, ERROR, or SLOW.
Log Entry Example	2016-11-22 13:06:27,192 c.d.d.c.Q.SLOW [cluster1] [/10.20.2.146:9042] Query too slow, took 7674 ms: alter table "UserData_20d784f480b0374559b- dd71054bbb8c1"."CLOUDIAN_METADATA" with gc_grace_seconds=864000 and bloom_filter_fp_chance=0.1 and com- paction = <code>{'class': 'LeveledCompactionStrategy'};</code>
Default Logging Level	DEBUG Note In <code>log4j-s3.xml.erb</code> on your Puppet master node there are three different <code>AsyncLogger</code> instances for Cassandra request logging. The ERROR logger logs entries when a Cassandra request results in an error; the SLOW logger logs entries when a Cassandra request takes more than 5 seconds to process; and the NORMAL logger logs all Cassandra requests. The three loggers all write to <code>/var/log/cloudian/cassandra-s3-tx.log</code> , and the implementation prevents duplicate entries across the three loggers. All three loggers are set to DEBUG level by default; and each logger works only if set to DEBUG or TRACE. To disable a logger, set its level to INFO or higher. For example to disable the NORMAL logger so that only error and slow requests are recorded, set the NORMAL logger's level to INFO. Then do a Puppet push and restart the S3 Service.
Default Rotation and Retention Policy	Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size. Rotated files are named as <code>cassandra-s3-tx.log.YYYY-MM-DD.i.gz</code> , where <i>i</i> is a rotation counter that resets back to 1 after each day. Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Con- figuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="CASSANDRATX"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

Note Currently only a small fraction of request types from the S3 Service to Cassandra support this request logging feature. These are request types that use a new DataStax Java driver (which supports the request logging) rather than the older Hector driver (which does not). Currently the only types of requests that use the DataStax driver are requests to clean up tombstones. As new Cassandra request types are implemented in HyperStore they will use the DataStax driver, and thus over time a growing portion of Cassandra requests will support request logging.

10.1.3. CMC Log

CMC application log (*cloudian-ui.log*)

Name and Location	On every node, <i>/var/log/cloudian/cloudian-ui.log</i>
Log Entry Format	<p>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId] ClassName:MESSAGE</p> <p>In the case of log entries for user logins to the CMC, the MESSAGE value will be formatted as follows:</p> <p>Normal login</p> <pre>Login <groupId> <userId> from: <ipAddress> Success</pre> <pre>Login <groupId> <userId> from: <ipAddress> Failed [<reason>]</pre> <p>SSO login</p> <pre>SSOLogin <groupId> <userId> from: <ipAddress> Success</pre> <pre>SSOLogin <groupId> <userId> from: <ipAddress> Failed [<reason>]</pre>
Log Entry Example	2017-05-04 11:56:48,475 INFO [localhost-startStop-1] ServiceMapUtil:Loading service map info from: <i>cloudianservicemap.json</i>
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-ui.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cmc/templates/log4j.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="APP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

10.1.4. HyperStore Service Logs

HyperStore Service application log (cloudian-hyperstore.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-hyperstore.log</code>
Log Entry Format	<p><code>YYYY-mm-dd HH:mm:ss,SSS PriorityLevel[S3RequestId][ThreadId]MessageCode ClassName:MESSAGE</code></p> <ul style="list-style-type: none"> The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help. The S3RequestId value is present only in messages associated with implementing S3 requests.
Log Entry Example	<code>2017-05-04 23:58:34,634 ERROR[] [main]HS220008 Cloud-ianAbstractServer:Unable to load configuration file: storage.xml</code>
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <code>cloudian-hyperstore.log.YYYY-MM-DD.i.gz</code>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="APP"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

HyperStore Service request log (cloudian-hyperstore-request-info.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-hyperstore-request-info.log</code>
Log Entry Format	<p><code>YYYY-mm-dd HH:m- m:ss,SSS IpAddressOfClientS3Server S3RequestId HttpStatus HttpOperation OriginalUri HyperStoreFilePath ContentLength DurationMicrosecs Etag</code></p> <ul style="list-style-type: none"> The IpAddressOfClientS3Server is the IP address of the S3 Server node that submits the request to the HyperStore Service. The HttpOperation is the HyperStore Service HTTP API operation that the S3 Service invokes and will be a simple operation like "PUT" or "GET". In the case of a "secure delete", the operation will be "SECURE-DELETE" and this request won't be logged until the third and final data overwriting pass for the object is completed. By contrast a regular delete will be logged as operation "DELETE". The secure delete feature is disabled by default. For more information on the secure delete feature see the description of the con-

	<p>figuration property "secure.delete" (page 487).</p> <ul style="list-style-type: none"> The OriginalUri field shows the group ID, bucket name, and object name from the originating S3 API request, in URI-encoded form ("CloudianTest1", "buser1", and "514kbtes", respectively, in the example above). The Etag field will be "0" for operations other than PUT.
Log Entry Example	2016-10-27 15:18:18,031 10.20.2.52 6e4c6884-a4a2-1238-a908-525400c5e557 200 PUT /file/CloudianTest1%2Fbuser1%2Ftest100b /cloudian2/hsfs/1IjBeBudSCVmsYbKdPV8Ns/4a3ce-b36ee344e1eb-d43ed413b310bc8/046/075/56017837606746367338485930470043970723.1477552697400 100 854411 7b2a7abdfdaa1a01c33432b5c41e0939
Logging Level	Not applicable
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 300MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-hyperstore-request-info.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 3GB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="REQ"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

HyperStore Service cleanup log (cloudian-hyperstore-cleanup.log)

Name and Location	On every node, <i>/var/log/cloudian/cloudian-hyperstore-cleanup.log</i>
Log Entry Format	<pre>yyyy-mm-dd HH:mm:ss,SSS Command# ObjectKey ObjectFilePath</pre> <p>This log has entries when an hsstool cleanup or hsstool cleanupec operation results in files being deleted from the node. A cleanup operation that determines that no files need to be deleted from the node will not cause any entries to this log.</p>
Log Entry Example	2018-02-28 05:57:25,743 1 buser-1/obj1 /var/lib/cloudian/hsfs/Sa1A11OVu6oCThSSRafvH/7cf10597b0360421d7564e7c248b2445/165/206/16398559635448146388914806157301-167971.1476861996241
Default Logging Level	INFO
Default Rotation and Retention	Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.

tion Policy	Rotated files are named as <i>cloudian-hyperstore-cleanup.log.YYYY-MM-DD.i.gz</i> , where <i>i</i> is a rotation counter that resets back to 1 after each day. Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Con- figuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="CLEANUP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

HyperStore Service repair log (cloudian-hyperstore-repair.log)

Name and Location	On every node, <i>/var/log/cloudian/cloudian-hyperstore-repair.log</i>
Log Entry Format	<pre>YYYY-mm-dd HH:m- m:ss,SSS Type Com- mand# Coordinator RepairEndpoint StreamFromEndpoint ObjectKey ObjectFilePath ObjectSize Md5Hash RepairLatencyMillisecs Status</pre> <p>This log has entries when a repair operation results in an attempt to repair files on the node. A repair operation that determines that no repairs are needed on the node will not cause any entries to this log.</p> <p>The Type field is one of {RR (regular repair for replica), PRR (proactive repair for replicas), REC (regular repair for erasure coded data), PREC (proactive repair for erasure coded data), or UECD (update of EC digest fields)}. The Status field indicates the status of the repair attempt for the object and is one of {OK (success), PRQUEUE (added to proactive repair queue, in the case of a failed regular repair of replicated object data), or ERROR / STREAMERROR / STREAMTIMEOUT (failed to repair and did not add to proactive repair queue)}.</p>
Log Entry Example	<pre>2018-02-13 00:58:55,816 RR 2 10.20.2.34 10.20.2.35 10.20.2.34 new- b1/efile586 /home/disk2/hsfs/ eFq94PDeBSa5RV1sZMbh2/2479a0125408240ff77bd1b0a8ea28b0/082/083/ 137628540267427877754176464820359359646.1484021921915 10000 fe912baa5d49737c70624b5b82328838 22 OK</pre>
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-hyperstore-repair.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Con- figuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hyperstore.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="REPAIR"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

HyperStore Service repair error and heal logs (hss-error.log and hss-heal.log)

The *hss-error.log* and *hss-heal.log* are for use by Cloudian Support if they are helping you to troubleshoot repair failures.

HyperStore Service whereis log (whereis.log)

This log is generated if and only if you execute the *hsstool whereis -a* command, to output location detail for every S3 object in the system. For information about this log see [hsstool whereis](#).

10.1.5. Monitoring Service Logs

Monitoring Agent application log (cloudian-agent.log)

Name and Location	On every node, <i>/var/log/cloudian/cloudian-agent.log</i>
Log Entry Format	<i>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId] ClassName:MESSAGE</i>
Log Entry Example	2017-05-04 11:57:03,698 WARN [pool-2-thread-7] LogFileTailerListener:Log file not found for service:cron
Default Logging Level	WARN
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-agent.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudianagent/templates/log4j-agent.xml.erb</i> , this log is configurable in the block that starts with <i>RollingRandomAccessFile name="APP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

Monitoring Data Collector application log (cloudian-datacollector.log)

Name and Location	On Monitoring Data Collector node, <i>/var/log/cloudian/cloudian-datacollector.log</i>
Log Entry Format	<i>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId]MessageCode</i> <i>ClassName:MESSAGE</i> <p>The MessageCode uniquely identifies the log message, for messages of level WARN or</p>

	higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help.
Log Entry Example	2017-05-04 00:00:05,898 WARN[main]DC040073 SmtpNotification:Failed to send due to messaging error: Couldn't connect to host, port: smtp.notification.configure.me, 465; timeout 5000
Default Logging Level	WARN
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-datacollector.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-datacollector.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="APP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

10.1.6. Phone Home (Smart Support) Log

Phone Home application log (cloudian-phonehome.log)

Name and Location	On Monitoring Data Collector node, <i>/var/log/cloudian/cloudian-phonehome.log</i>
Log Entry Format	<p>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE</p> <p>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help..</p>
Log Entry Example	2017-05-04 19:13:03,384 ERROR[main]HS200043 S3:All Redis read connection pools are unavailable. Redis HGETALL of key BPP_MAP fails.
Default Logging Level	WARN
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-phonehome.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>

Con- figuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-phonehome.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="APP"</code> . For setting descriptions see " Log Configuration Settings " (page 593).
--------------------	--

10.1.7. Redis and Redis Monitor Logs

Redis Credentials application log (redis-credentials.log)

Name and Location	On Redis Credentials nodes, <code>/var/log/redis/redis-credentials.log</code>
Log Entry Format	<code>[PID] dd MMM hh:mm:ss # * MESSAGE</code>
Log Entry Example	<code>[17861] 18 Oct 11:55:45.657 # Server started, Redis version 2.8.23</code>
Default Logging Level	NOTICE
Default Rotation Policy	Not rotated by default. You can set up rotation by using logrotate .
Configuration	<p>Redis Credentials application logging is configured in the main Redis configuration file. The file name depends on the Redis node type -- master or slave. These templates are on the Puppet master node, under <code>/etc/cloudian-<version>-puppet/modules/redis/templates/</code>:</p> <ul style="list-style-type: none"> Redis Credentials master node: <code>redis-credentials.conf.erb</code> Redis Credentials slave node: <code>redis-credentials-slave.conf.erb</code> <p>The only configurable logging settings are the log file name and the logging level. See the commenting in the configuration file for more detail.</p>

Redis QoS application log (redis-qos.log)

Name and Location	On Redis QoS nodes, <code>/var/log/redis/redis-qos.log</code>
Log Entry Format	<code>[PID] dd MMM hh:mm:ss # * MESSAGE</code>
Log Entry Example	<code>[18135] 18 Oct 11:55:52.571 * The server is now ready to accept connections on port 6380</code>
Default Logging Level	NOTICE
Default Rotation Policy	Not rotated by default. You can set up rotation by using logrotate .

Con- figuration	<p>Redis QoS application logging is configured in the main Redis configuration file. The file name depends on the Redis node type -- master or slave. These templates are on the Puppet master node, under <code>/etc/cloudian-<version>-puppet/modules/redis/templates/</code>:</p> <ul style="list-style-type: none"> • Redis QoS master node: <code>redis-qos.conf.erb</code> • Redis QoS slave node: <code>redis-qos-slave.conf.erb</code> <p>The only configurable logging settings are the log file name and the logging level. See the commenting in the configuration file for more detail.</p>
--------------------	---

Redis request logs (redis-{s3,admin,hss}-tx.log)

Name and Location	<p>If enabled, these logs are written to <code>/var/log/cloudian/redis-{s3,admin,hss}-tx.log</code> on the S3 Service, Admin Service, and/or HyperStore nodes (that is, these request logs are written on the client side as S3, Admin, and HyperStore service instances send requests to Redis).</p>
Log Entry Format	<code>YYYY-mm-dd HH:mm:ss,SSS S3RequestId MESSAGE</code>
Log Entry Example	<code>2016-11-17 23:54:01,695 CLIENT setname ACCOUNT_GROUPS_M</code>
Default Logging Level	<p>INFO</p> <p>Note The default logging level of INFO disables these logs. If you want these logs to be written, you must edit the Puppet template files <code>log4j-s3.xml.erb</code> (for logging S3 Service access to Redis), <code>log4j-admin.xml.erb</code> (for logging Admin Service access to Redis), and/or <code>log4j-hyperstore.xml.erb</code> (for logging HyperStore Service access to Redis). Find the <code>AsyncLogger name="redis.clients.jedis"</code> block and change the <code>level</code> from "INFO" to "TRACE". Then do a Puppet push, and then restart the relevant services (S3-Admin and/or HyperStore).</p>
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <code>redis-{s3,admin,hss}-tx.log.YYYY-MM-DD.i.gz</code>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	<p>In the Puppet template files <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-{s3,admin,hss}.xml.erb</code>, this log is configurable in the block that starts with <code><RollingRandomAccessFile name="REDISTX"</code>. For setting descriptions see "Log Configuration Settings" (page 593).</p>

Redis Monitor application log (cloudian-redismon.log)

Name and Location	<p>On Redis Monitor nodes, <code>/var/log/cloudian/cloudian-redismon.log</code></p>
-------------------	---

Log Entry Format	<pre>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [ThreadId]MessageCode ClassName:MESSAGE</pre> <p>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help.</p>
Log Entry Example	2017-05-04 00:01:13,590 INFO [pool-23532-thread-3] RedisCluster:Failed to connect to cloudian jmx service [mothra:19082]
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-redismon.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-redismon.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="APP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

10.1.8. S3 Service Logs

S3 Service application log (*cloudian-s3.log*)

Name and Location	On every node, <i>/var/log/cloudian/cloudian-s3.log</i>
Log Entry Format	<pre>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel [S3RequestId] [ThreadId]MessageCode ClassName:MESSAGE</pre> <p>The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help.</p>
Log Entry Example	2017-05-04 00:33:39,435 ERROR [bc6f3fca-9037-135f-a964-0026b95cedde] [qtp1718906711-94]HS204017 XmlSaxParser:Rule doesn't have AllowedMethods/AllowedOrigins.
Default Logging Level	INFO
Default Rotation and Retention	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-s3.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p>

Policy	Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Configuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="S3APP"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

S3 Service request log (cloudian-request-info.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-request-info.log</code>
Log Entry Format	<p>YYYY-mm-dd HH:m- m:ss,SSS ClientIpAddress BucketOwnerUserId Operation BucketName ContentAccessUserId RequestHeaderSize RequestBodySize ResponseHeaderSize ResponseBodySize TotalRequestResponseSize DurationMicrosecs UrlEncodedObjectName HttpStatus S3RequestId Etag ErrorCode SourceBucketName/UrlEncodedSourceObjectName</p> <ul style="list-style-type: none"> The Operation field indicates the S3 API operation. Note that "getBucket" indicates GET Bucket (List Objects) Version 1 whereas "getBucketV2" indicates GET Bucket (List Objects) Version 2. The Etag field is the Etag value from the response, if applicable to the request type. For information about Etag see for example Common Response Headers from the Amazon S3 REST API spec. This field's value will be 0 for request/response types that do not use an Etag value. The ErrorCode field is the Error Code in the response body, applicable only for potentially long-running requests like PUT Object. If there is no Error Code in the response body this field's value will be 0. For possible Error Code values see Error Responses from the Amazon S3 REST API spec. <p>Note: In the case where the Operation field value is <code>deleteObjects</code>, the ErrorCode field will be formatted as <code>objectname1-errorcode1,objectname2-errorcode2,objectname3-errorcode3...</code>, and the object names will be URL-encoded. If there are no errors the field is formatted as <code>objectname1-0,objectname2-0,objectname3-0....</code></p> <ul style="list-style-type: none"> The last field (SourceBucketName/UrlEncodedSourceObjectName) is populated only for copyObject and uploadPartCopy operations and is empty for other operation types. If you want the ClientIpAddress, BucketOwnerUserId, and ContentAccessUserId to be removed from the copies of the S3 request log that get uploaded to Cloudian Support as part of the Smart Support feature and on-demand Node Diagnostics feature, set "phone-home_gdpr" (page 470) to true in common.csv. Doing so will not remove these fields from the original S3 request logs on your HyperStore nodes -- just from the log file copies that get sent to Cloudian. For more information on these support feature see "Smart Support and Diagnostics Feature Overview" (page 137).

Log Entry Example	2017-07-19 10:19:28,970 10.50.10.21 Pub-sUser1 getBucket bucket1 PubsUser1 696 0 187 535 1418 8117 200 928c8fd2-0b63-1a2a-be6e-0026b95cedde 0 0
Logging Level	Not applicable
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-request-info.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 2GB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFile name="S3REQ"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

Note For information on setting up an Elastic Stack node and streaming *cloudian-request-info.log* data to that node to support integrated analysis of your S3 request traffic, see "**Setting Up Elastic Stack for S3 Request Traffic Analysis**" (page 598).

Logging the True Originating Client IP Address

If you use a load balancer in front of your S3 Service (as would typically be the case in a production environment), then the *ClientIpAddress* in your S3 request logs will by default be the IP address of a load balancer rather than that of the end client. If you want the S3 request logs to instead show the end client IP address, your options depend on what load balancer you're using.

If your load balancer is HAProxy or a different load balancer that supports the PROXY Protocol, enable S3 support for the PROXY Protocol (see "**s3_proxy_protocol_enabled**" (page 475) in *common.csv*) and configure your load balancer to use the PROXY Protocol for relaying S3 requests to the S3 Service. Consult with your Cloudian Sales Engineering or Support representative for guidance on load balancer configuration.

If your load balancer does not support the PROXY Protocol:

1. Configure your load balancers so that they pass the HTTP *X-Forwarded-For* header to the S3 Service. This is an option only if you run your load balancers run in "HTTP mode" rather than "TCP mode". Consult with your Cloudian Sales Engineering or Support representative for guidance on load balancer configuration.
2. Configure your S3 Service to support the *X-Forwarded-For* header. You can enable S3 Service support for this header by editing the configuration file *s3.xml.erb* on your Puppet master node. The needed configuration lines are already in that file; you only need to uncomment them.

Before uncommenting:

```
<!-- Uncomment the block below to enable handling of X-Forwarded- style headers
-->
<!--
<Call name="addCustomizer">
  <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/></Arg>
```

```
</Call>
-->
```

After uncommenting:

```
<!-- Uncomment the block below to enable handling of X-Forwarded- style headers
-->
<Call name="addCustomizer">
<Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/></Arg>
</Call>
```

After making this configuration edit, [do a Puppet push and restart the S3 Service](#) to apply your change.

Auto-Tiering request log (cloudian-tiering-request-info.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-tiering-request-info.log</code>
	<p>Note For regular auto-tiering that occurs on a defined schedule this request logging occurs on the same node that is running the HyperStore system cron jobs. In the special case of "bridge mode" auto-tiering, the request logging is distributed across all your S3 nodes -- whichever S3 node processes the upload of a given object into the source bucket also initiates the immediate auto-tiering of the object to the destination system, and the tiering request log entry for that is written locally on that node.</p>
Log Entry Format	<code>YYYY-mm-dd HH:m-</code> <code>m:ss,SSS Command Protocol SourceBucket/Object SourceObjectVersion </code> <code>Tar-</code> <code>getBuck-</code> <code>et TargetObjectVersion ObjectSize TotalRequestSize Status LatencyMicrosecs</code>
Log Entry Example	<code>2017-10-31 20:39:37,282 MOVEOBJECT AZURE cbuck-</code> <code>et/a.c null testbucket null 112 115 </code> <code>COMPLETED 83000</code>
Logging Level	Not applicable
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <code>cloudian-tiering-request-info.log.YYYY-MM-DD.i.gz</code>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Con-figuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="TIER"></code> . For setting descriptions see " Log Configuration Settings " (page 593).

*Cross-Region Replication request log (*cloudian-crr-request-info.log*)*

Name and Location	<p>On every node, <code>/var/log/cloudian/cloudian-crr-request-info.log</code></p> <p>Note Whichever S3 Service node processes a PUT of an object into a source bucket configured for CRR will be the node that initiates the replication of the object to the destination bucket. This node will have an entry for that object in its CRR request log. In the case of retries of replication attempts that failed with a temporary error the first time, the retries will be logged in the CRR request log on the cron job node. For general information on the cross-region replication feature, see "Cross-Region Replication" (page 132).</p>
Log Entry Format	<p><code>yyyy-mm-dd HH:mm:ss,SSS INFO[S3RequestId] [ThreadId] ClassName:yyyy-mm-dd HH:mm:ss,SSS SourceBucket/Object ObjectVersionId DestinationBucket Status</code></p> <p>The Status will be one of:</p> <ul style="list-style-type: none"> • COMPLETED -- The object was successfully replicated to the destination bucket. • FAILED -- The object replication attempt resulted in an HTTP 403 or 404 response from the destination system. This is treated as a permanent error and no retry attempt will occur for replicating this object. This type of error could occur if for example the destination bucket has been deleted or if versioning has been disabled for the destination bucket. • PENDING -- The object replication attempt encountered an error other than an HTTP 403 or 404 response. All other errors -- such as a connection error or an HTTP 5xx response from the destination system -- are treated as temporary errors. The object replication will be retried again once every four hours until either the object is successfully replicated to the destination bucket or a permanent error is encountered. Each retry attempt results in a new log entry in <i>cloudian-crr-request-info.log</i>. Note that in log entries for retry attempts the S3RequestId field will be empty. <p>Note If an attempt to replicate an object to the destination region -- either the original attempt or a retry attempt -- results in a FAILED status in the Cross-Region Replication request log, so that there will be no further retries, this triggers an Alert in the CMC's Alerts page. This type of alert falls within the alert rules category for S3 service errors (there is not a separate alert rule category for CRR replication failures).</p>
Log Entry Example	<pre>2017-10-31 12:06:21,139 INFO[bf9d590c-56c2-1cb3-95b6-0a1dff8728b4] [crr-executor4] CRRLogger:2017-10-31 12:06:21,139 bktregion1/OBJ1 fe184217-8780-b4ff-95b6-0a1dff8728b4 bktregion2 COMPLETED</pre>
Logging Level	Not applicable
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-crr-request-info.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p>

	Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Con- figuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="CRR"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

WORM request log (s3-worm.log)

Name and Location	On every node, <code>/var/log/cloudian/s3-worm.log</code> Note For information about the WORM feature see " WORM (Bucket Lock) Feature Overview " (page 159).
Log Entry Format	<code>YYYY-mm-dd HH:m-m:ss,SSS Host Operation Bucket CanonicalUserId StatusCode StatusMessage</code> The Operation will be one of s3:PostLockPolicy, s3:PostLockId, s3:DeleteLockPolicy, s3:GetLockPolicy, s3:PrivilegedDelete (for the operation that establishes a privileged delete user for a bucket), s3:DeleteObject, s3:DeleteObjectVersion (these last two operations are for object deletes performed by a privileged user).
Log Entry Example	<code>2018-02-28 14:06:46,399 cld12 s3:PostLockPolicy bn_1519798006 d786e4f700b5afec152d83c47bdced4d 201 SC_CREATED</code>
Default Logging Level	INFO
Default Rotation and Retention Policy	Rotation occurs if live file size reaches 100MB. Rotation also occurs at end of each day, regardless of live file size. Rotated files are named as <code>s3-worm.log.YYYY-MM-DD.i.gz</code> , where <i>i</i> is a rotation counter that resets back to 1 after each day. Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.
Con- figuration	In the Puppet template file <code>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-s3.xml.erb</code> , this log is configurable in the block that starts with <code><RollingRandomAccessFile name="S3WORM"</code> . For setting descriptions see " Log Configuration Settings " (page 593).

IAM application log (cloudian-iam.log)

Name and Location	On every node, <code>/var/log/cloudian/cloudian-iam.log</code> Note This log is present only if you have enabled IAM in your system. See " Identity and Access Management (IAM) Feature Overview " (page 91).
-------------------	--

Log Entry Format	<pre>yyyy-mm-dd HH:mm:ss,SSS PriorityLevel[ThreadId]MessageCode ClassName:MESSAGE</pre> <ul style="list-style-type: none"> The MessageCode uniquely identifies the log message, for messages of level WARN or higher. For documentation of specific message codes including recommended operator action, see the HyperStore online Help.
Log Entry Example	2018-02-16 10:16:36,246 INFO[qtp214187874-41] CloudianHFactory:Creating DynamicKS for: AccountInfo
Default Logging Level	INFO
Default Rotation and Retention Policy	<p>Rotation occurs if live file size reaches 10MB. Rotation also occurs at end of each day, regardless of live file size.</p> <p>Rotated files are named as <i>cloudian-iam.log.YYYY-MM-DD.i.gz</i>, where <i>i</i> is a rotation counter that resets back to 1 after each day.</p> <p>Deletion of oldest rotated log file occurs if aggregate rotated files size (after compression) reaches 100MB or if oldest rotated file age reaches 180 days.</p>
Configuration	In the Puppet template file <i>/etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-iam.xml.erb</i> , this log is configurable in the block that starts with <i><RollingRandomAccessFileName="IAMAPP"</i> . For setting descriptions see " Log Configuration Settings " (page 593).

10.1.9. HyperStore Firewall Log

HyperStore firewall log (firewall.log)

If the HyperStore firewall is enabled in your system then DROP'd connections are logged in the HyperStore firewall log. For information about the firewall see "**Configuring the HyperStore Firewall**" (page 569).

Name and Location	On every node, <i>/var/log/cloudian/firewall.log</i>
Log Entry Format	The log records information about dropped packets, including the timestamp, host, firewall zone (<i>cloudian-backend</i> [for the designated internal interface] or <i>cloudian-frontend</i> [for all other interfaces]), interface name and MAC address, source and destination address, protocol, TCP flags, and so on.
Log Entry Example	<pre>Apr 18 11:03:19 demo4-node1 kernel: IN_cloudian-frontend_DROP: IN=eth0 OUT= MAC=52:54:00:e3:82:d7:52:54:00:11:dd:79:08:00 SRC=10.254.254.103 DST=10.254.254.118 LEN=44 TOS=0x00 PREC=0x00 TTL=57 ID=43247 PROTO=TCP SPT=52377 DPT=74 WINDOW=1024 RES=0x00 SYN URGP=0</pre>
Default Rotation and Retention Policy	<p>Rotation occurs daily.</p> <p>Rotated files are named as <i>firewall.log-YYYYMMDD</i>. The most recent rotated file is not compressed; all other rotated files are compressed and the file names will have a <i>.gz</i> suffix.</p> <p>14 rotated files are retained. Older files are automatically deleted.</p>
Con-	Rotation of this log is managed by the Linux <i>logrotate</i> utility. In the current version of Hyper-

figuration	Store, the rotation settings for the HyperStore firewall log are not configurable.
------------	--

10.2. Log Configuration Settings

The S3 Service, HyperStore Service, Redis Monitor, Admin Service, Monitoring Data Collector, Monitoring Agent, and CMC each have their own XML-formatted *log4j-* .xml.erb* configuration template in which you can adjust logging settings. Within a *log4j-* .xml.erb* file, specific logs -- such as the S3 application log and the S3 request log -- are configured by named instances of *RollingRandomAccessFile*. The "**HyperStore Logs**" (page 575) overview topic indicates the specific *log4j-* .xml.erb* file and the specific *RollingRandomAccessFile* name by which each log is configured (for example the S3 application log is configured by the *RollingRandomAccessFile* instance named "S3APP" in the *log4j-s3.xml.erb* file).

Note After making any configuration file edits, be sure to [trigger a Puppet sync-up and then restart the affected service](#) (for example, the S3 Service if you've edited the *log4j-s3.xml.erb* file).

Within a particular log's *RollingRandomAccessFile* instance there are these editable settings:

- **PatternLayout pattern="*<pattern>*"** — The log entry format. This flexible formatting configuration is similar to the *printf* function in C. For detail see [PatternLayout](#) from the online Apache Log4j2 documentation.
- **TimeBasedTriggeringPolicy interval="*<integer>*"** — Roll the log after this many days pass. (More precisely, the log rolls after *interval* number of time units pass, where the time unit is the most granular unit of the date pattern specified within the *filePattern* element — which in the case of all HyperStore logs' configuration is a day). Defaults to rolling once a day if *interval* is not specified. All HyperStore logs use the default of one day.
- **SizeBasedTriggeringPolicy size="*<size>*"** — Roll the log when it reaches this size (for example "10 MB"). Note that this trigger and the *TimeBasedTriggeringPolicy* operate together: the log will be rolled if either the time based trigger **or** the size based trigger occur.
- **IfLastModified age="*<interval>*"** — When a rolled log file reaches this age the system automatically deletes it (for example "180d").
- **IfAccumulatedFileSize exceeds="*<size>*"** — When the aggregate size (after compression) of rolled log files for this log reaches this size, the system automatically deletes the oldest rolled log file (for example "100 MB"). Note that this setting works together with the *IfLastModified* setting -- old rolled log files will be deleted if either the age based trigger **or** the aggregate size based trigger occur.

Note: Each *RollingRandomAccessFile* instance also includes a *DefaultRolloverStrategy max-*
x="<integer>" parameter which specifies the maximum number of rolled files to retain from a
single day's logging. However, by default this parameter is not relevant for HyperStore because
HyperStore logs are configured such that the *IfAccumulatedFileSize* trigger will be reached
before the *DefaultRolloverStrategy* trigger.

For each log's default value for the settings above, see the "**HyperStore Logs**" (page 575) overview topic.

In the *log4j-* .xml.erb* files, in addition to *RollingRandomAccessFile* instances there are also **Logger** instances. Each **Logger** instance contains an *AppenderRef* element that indicates which log that **Logger** instance applies to, by referencing the log's *RollingRandomAccessFile* name (for example *AppenderRef ref="S3APP"* means

that the *Logger* instance is associated with the S3 application log). Note that multiple *Logger* instances may be associated with the same log — this just means that multiple core components of a service (for example, multiple components within the S3 Service) have separately configurable loggers. If you’re uncertain about which *Logger* instance to edit to achieve your objectives, consult with Cloudian Support.

The *Logger* instances are where you can configure a logging level, using the *level* attribute:

- **level="<level>"** — Logging level. The following levels are supported (only events at the configured level and above will be logged):
 - OFF = Turn logging off.
 - ERROR = Typically a fail-safe for programming errors or a server running outside the normal operating conditions. Any error which is fatal to the service or application. These errors will cause administrator alerts and typically force administrator intervention.
 - WARN = Anything that can potentially cause application oddities, but where the server can continue to operate or recover automatically. Exceptions caught in “catch” blocks are commonly at this level.
 - INFO = Generally useful information to log (service start/stop, configuration assumptions, etc). Info to always have available. Normal error handling, like a user not existing, is an example.
 - DEBUG = Information that is diagnostically helpful.
 - TRACE = Very detailed information to “trace” the execution of a request or process through the code.
 - ALL = Log all levels.

For each log’s default log level, see the ["HyperStore Logs"](#) (page 575) overview topic.

10.3. Aggregating Logs to a Central Server

If you wish you can have application logs and request logs from the S3 Service, Admin Service, and HyperStore Service — as well as system logs from the host machines in your cluster — aggregated to a central logging server using *rsyslog*. This is not enabled by default, but you can enable it by editing configuration files. This procedure sets up your HyperStore logging so that logs are written to a central logging server **in addition to** being written on each HyperStore node locally.

The procedure presumes that:

- You have a central logging server running *rsyslog*.
- Each of your HyperStore nodes has *rsyslog* installed.

IMPORTANT: The central logging server must **not** be one of your HyperStore nodes.

Note *rsyslog* is included in the HyperStore Appliance and also in standard RHEL/CentOS distributions. This procedure has been tested using *rsyslog* v5.8.10.

Note For information on setting up an [Elastic Stack](#) node and streaming S3 request log data to that node to support analysis and visualization of your S3 request traffic, see ["Setting Up Elastic Stack for S3 Request Traffic Analysis"](#) (page 598).

To aggregate HyperStore application logs, request logs, and system logs to a central logging server follow the instructions below.

1. **On the central logging server**, do the following:

- a. In the configuration file `/etc/rsyslog.conf`, enable UDP by uncommenting these lines:

```
# BEFORE

#$ModLoad imudp
#$UDPServerRun 514

# AFTER

$ModLoad imudp
$UDPServerRun 514
```

- b. Create a file `/etc/rsyslog.d/cloudian.conf` and enter these Cloudian-specific configuration lines in the file:

```
$template CloudianTmpl,"%HOSTNAME% %TIMESTAMP:::date-rfc3339% %sys-
logseverity-
text:::uppercase% %msg%\n"
$template CloudianReqTmpl,"%HOSTNAME% %msg%\n"
:programname, isEqual, "ADMINAPP" /var/log/cloudian-admin.log;CloudianTmpl
:programname, isEqual, "HSAPP"    /var/log/cloudian-hyper-
store.log;CloudianTmpl
:programname, isEqual, "HSREQ"   /var/log/cloudian-hyperstore-request-
info.log;CloudianReqTmpl
:programname, isEqual, "S3APP"    /var/log/cloudian-s3.log;CloudianTmpl
:programname, isEqual, "S3REQ"   /var/log/cloudian-request-info.-
log;CloudianReqTmpl
:programname, isEqual, "S3WORM"   /var/log/s3-worm.log;CloudianReqTmpl
```

- c. Restart `rsyslog` by "service rsyslog restart".
- d. Enable rotation on the centralized HyperStore logs. For example, to use `logrotate` for rotating the HyperStore logs, create a file `/etc/logrotate.d/cloudian` and enter the following configuration lines in the file. (Optionally adjust the rotated file retention scheme — 14 rotations before deletion in the example below — to match your retention policy.)

```
/var/log/cloudian-*.*.log
{
    daily
    rotate 14
    create
    missingok
    compress
    delaycompress
    sharedscripts
    postrotate
        /bin/kill -HUP `cat /var/run/syslogd.pid 2>/dev/null` 2>/dev/null
    || true
    endscript
}
```

2. **On your HyperStore Puppet Master node**, do the following:

- a. Go to `/etc/cloudian-<version>-puppet/modules/cloudians3/templates/`. Here you will edit three configuration files:

- `log4j-admin.xml.erb`
- `log4j-hyperstore.xml.erb`
- `log4j-s3.xml.erb`

In each of these three files uncomment the `syslog` sections. In each file there are a few `syslog` sections and each such section is marked with a `#syslog` tag. When you are done uncommenting, there should be no remaining `#syslog` tags.

Also, in the first `syslog` section at the top of each file — nested within the "Properties" block — in addition to uncommenting the `syslog` section set the `sysloghost` property to the hostname or IP address or your central syslog server and set the `syslogport` property to the central syslog server's UDP port (which by default is port number is 514).

Here is a before and after example for uncommenting and editing the `syslog` section within a "Properties" block. In this example the hostname of the central logging server is "regulus".

```
# BEFORE

<Properties>
    <!-- #syslog
    <Property name="sysloghost">localhost</Property>
    <Property name="syslogport">514</Property>
    -->
</Properties>

# AFTER

<Properties>
    <Property name="sysloghost">regulus</Property>
    <Property name="syslogport">514</Property>
</Properties>
```

Be sure to uncomment **all** the `#syslog` tagged sections in each file. Only the `syslog` section within the "Properties" block requires editing an attribute value; the rest of the `syslog` sections only require uncommenting.

Here is a before and after example of uncommenting a `syslog` section (in `log4j-s3.xml.erb`):

```
# BEFORE

<!-- #syslog
<Syslog name="SYSLOG-S3APP" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="S3APP" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>
<Syslog name="SYSLOG-S3REQ" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="S3REQ" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>
```

```

<Syslog name="SYSLOG-S3WORM" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="s3-worm" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>
-->

# AFTER

<Syslog name="SYSLOG-S3APP" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="S3APP" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>
<Syslog name="SYSLOG-S3REQ" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="S3REQ" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>
<Syslog name="SYSLOG-S3WORM" format="RFC5424" host="${sysloghost}" port-
t="${syslogport}"
    protocol="UDP" appName="s3-worm" mdcId="mdc" includeMDC="true"
facility="USER" newLine="true">
</Syslog>

```

- b. Go to `/etc/cloudian-<version>-puppet/modules/rsyslog/templates/`. Then edit `loghost.conf.erb` to uncomment the following line and replace "<loghost>" with the hostname (or IP address) of your central syslog server host:

```

# BEFORE

#*. * @<loghost>:514

# AFTER

*.* @regulus:514

```

IMPORTANT: The central log host must **not** be one of your HyperStore hosts. (The reason is that if you have one of your HyperStore hosts acting as the central log host, then that host is sending logs to itself which results in a loop and rapid proliferation of log messages.)

- c. Go to your installation staging directory, then launch the HyperStore installer:

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

- d. From the installer's main menu, select "Cluster Management". Then from the "Cluster Management" menu select "Push Configuration Settings to Cluster" and follow the prompts.
- e. From the "Cluster Management" menu, select "Manage Services". Then from the "Service Management" menu, restart the HyperStore Service and the S3 Service. (Restarting the S3 Service also automatically restarts the Admin Service.)

3. Back on the central logging server:

- a. Confirm that logs are being written in `/var/log/cloudian-*` files.
- b. Confirm that system messages from HyperStore nodes appear on the log host (for example in `/var/log/messages`). If you want you can proactively test this by running the command "logger test 1" on any HyperStore node.

10.4. Setting Up Elastic Stack for S3 Request Traffic Analysis

Subjects covered in this section:

- *Introduction (immediately below)*
- **"Installing Elasticsearch, Kibana, and Logstash"** (page 598)
- **"Installing Filebeat"** (page 601)
- **"Configuring Kibana for Custom Metrics Visualizations"** (page 602)

These instructions describe how to install an Elasticsearch-Logstash-Kibana (ELK) stack on a **single node** that will process S3 request logs (`cloudian-request-info.log` files) from HyperStore nodes.

Note These instructions involve acquiring and using technologies from the open source [Elastic Stack](#).

Note If you also want to integrate with Elasticsearch for searching HyperStore object metadata (as described in **"Elasticsearch Integration for Object Metadata"** (page 106)), contact Cloudian Sales Engineering or Support to discuss using the same Elastic Stack cluster for both object metadata search and S3 request log analysis. Note that for object metadata search, a minimum of three Elasticsearch nodes are recommended and so the single-node set-up instructions below are not sufficient to that use case.

ELK Cluster Components:

- Elasticsearch: Stores the S3 request log files from HyperStore nodes in the filtered form specified in Logstash
- Logstash: Filters S3 request logs into a format that is searchable by Elasticsearch
- Kibana: Provides a web interface to access Elasticsearch data

HyperStore Cluster Components:

- Filebeat: Forwards S3 request logs from HyperStore nodes to ELK cluster

Prerequisites:

- An existing HyperStore cluster
- A CentOS 7 machine that has Java 8 already installed and Internet access

Note To work with HyperStore your **Elasticsearch version must be a 6.x version, 6.6 or newer**. In the installation instructions that follow the version is 6.6.0.

10.4.1. Installing Elasticsearch, Kibana, and Logstash

On the host that you want to set up as an ELK node, take the following steps.

10.4.1.1. Installing Elasticsearch

1. Run the following command to import the ES public GPG key into RPM:

```
sudo rpm --import http://packages.elastic.co/GPG-KEY-elasticsearch
```

2. Run the following command to download the ES RPM file:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-  
6.6.0.rpm
```

3. Install the RPM by running the following command:

```
sudo rpm --install elasticsearch-6.6.0.rpm
```

4. Open */etc/elasticsearch/elasticsearch.yml* in a text editor and uncomment the "network.host" entry and set it to:

```
network.host: "localhost"
```

5. Also uncomment "cluster.name" and set it to the desired name for the ES cluster so that a cluster can be formed.

6. Save the configuration changes that you made in *elasticsearch.yml*.

To start Elasticsearch, run the following command:

```
sudo systemctl start elasticsearch
```

To have Elasticsearch start automatically on boot:

```
sudo systemctl enable elasticsearch
```

10.4.1.2. Installing Kibana

1. Run the following command to download the Kibana RPM:

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-6.6.0-x86_64.rpm
```

2. Install the RPM by running the following command:

```
sudo rpm --install kibana-6.6.0-x86_64.rpm
```

3. If you want to be able to access Kibana from other machines besides the ELK node itself, open */etc/kibana/kibana.yml*, find and uncomment the "server.host: " line, and set it to the IP address of the ELK node:

```
server.host: <ELK_NODE_IP_ADDR>
```

To start Kibana, run the following command:

```
sudo systemctl start kibana
```

To have Kibana start automatically on boot:

```
sudo systemctl enable kibana
```

10.4.1.3. Installing Logstash

1. Run the following command to download the Logstash RPM:

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-6.6.0.rpm
```

2. Install the RPM by running the following command:

```
sudo rpm --install logstash-6.6.0.rpm
```

10.4.1.4. Configuring Logstash

Under the `/etc/logstash/conf.d` directory create a file `logstash.conf` and paste the following lines into it:

```
input {
  beats {
    port => 5044
  }
}

filter {
  if ([document_type] == "cloudian-request-info") {
    urldecode {
      field => "message"
    }
    csv {
      id => "cloudian-request-info"
      autogenerate_column_names => false
      separator => "|"
      columns => [
        "timestamp",
        "ipAddress",
        "bucketOwnerUserId",
        "operation",
        "bucketName",
        "contentAccessorUserID",
        "requestHeaderSize",
        "requestBodySize",
        "responseHeaderSize",
        "responseBodySize",
        "totalRequestResponseSize",
        "durationMsec",
        "objectName",
        "httpStatus",
        "s3RequestID",
        "eTag",
        "errorCode",
        "copySource"
      ]
      convert => {"requestHeaderSize" => "integer"}
      convert => {"requestBodySize" => "integer"}
      convert => {"responseHeaderSize" => "integer"}
      convert => {"responseBodySize" => "integer"}
      convert => {"totalRequestResponseSize" => "integer"}
      convert => {"durationMsec" => "integer"}
      remove_field => "message"
    }
    date {
      match => ["timestamp", "ISO8601"]
    }
  }
}
```

```

        remove_field => [ "timestamp" ]
    }
    geoip {
        source => "ipAddress"
    }
}

output {
    if "_csvparsefailure" not in [tags] and "_dateparsefailure" not in [tags] {
        elasticsearch {
            hosts => [ "localhost:9200" ]
            document_type => "%{[document_type]}"
            index => "logstash-%{+YYYY.MM.DD}"
        }
    }
}

```

Save the file and exit the text editor.

To start Logstash, run the following command:

```
sudo systemctl start logstash
```

To have Logstash start automatically on boot:

```
sudo systemctl enable logstash
```

10.4.2. Installing Filebeat

Take the following steps on **each of your HyperStore nodes**.

1. Get the Elastic GPG key:

```
sudo rpm --import http://packages.elastic.co/GPG-KEY-elasticsearch
```

2. Run the following command to download the Filebeat RPM:

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.6.0-x86_64.rpm
```

3. Install the RPM by running the following command:

```
sudo rpm --install filebeat-6.6.0-x86_64.rpm
```

4. Open `/etc/filebeat/filebeat.yml` and find the “paths:” section. Replace the “- /var/log/*.log” line with “- /var/log/cloudian/cloudian-request-info.log”, while maintaining the level of indentation.
5. Find the “fields:” section, uncomment it, and under it add the indented line “document_type: cloudian-request-info”, so that the section looks like this

```
fields:
  document_type: cloudian-request-info
```

6. Just above the “fields:” section that you uncommented add the line “fields_under_root: true”, so that the result looks like this:

```
fields_under_root: true
fields:
```

- ```
document_type: cloudian-request-info
```
7. Find the “output.elasticsearch:” section and comment it out along with any entries in that section. You don’t need this section since you will have the log files sent to Logstash instead.
  8. Find the “output.logstash:” section and uncomment it. Also uncomment the line “#hosts: [“localhost:5044”]” under the “output.logstash:” section and replace localhost with the IP address of the ELK node:

```
hosts: [<ELK_NODE_IP_ADDR>:5044]
```
  9. Save the configuration changes that you made in *filebeat.yml*.

To start Filebeat, run the following command:

```
sudo systemctl start filebeat
```

To have Filebeat start automatically on boot:

```
sudo systemctl enable filebeat
```

#### 10.4.3. Configuring Kibana for Custom Metrics Visualizations

**Note** The following instructions are intended only as an example, and the screen images that you see here may not exactly match what you see in your version of Kibana.

1. Open up a browser that can access the ELK node’s IP address and enter the following:

```
http://<ELK_NODE_IP_ADDR>:5601/
```

You should see a page similar to this:

Management / Kibana

Index Patterns Saved Objects Advanced Settings

**Warning**  
No default index pattern.  
You must select or create one to continue.

**Configure an index pattern**

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify search and analytics against. They are also used to configure fields.

**Index pattern** advanced options  
logstash-\*

**Time Filter field name**  @timestamp

Expand index pattern when searching [DEPRECATED]  
With this option selected, searches against any time-based index pattern that contains a wildcard will automatically be expanded to contain data within the currently selected time range.  
Searching against the index pattern logstash-\* will actually query Elasticsearch for the specific matching indices (e.g., logs-\*). This is useful for current time range.  
With recent changes to Elasticsearch, this option should no longer be necessary and will likely be removed in future versions.

Use event times to create index names [DEPRECATED]

**Create**

If you don't see a page like this, then Logstash has not yet pushed an index to Elasticsearch, either because of an installation error or because no requests have been made to your cluster yet.

- Under the "Time Filter field name" drop-down select "@timestamp" and click **Create**.

#### 10.4.3.1. Top 5 users in # of requests the last hour

- To create custom metrics, go to "Visualize" in the sidebar, click **Create a visualization**, select "Data Table", and then select the "logstash-\*" index. You should then be at a screen like this:

2. At the top right of the screen, make sure the time setting is set to “Last 1 hour”. If not, click on it, then select “Quick”, and then select “Last 1 hour” from the list of options.
3. Click **Add a filter** under the search bar, select the “@timestamp” field, select the “is between” operator, enter “now-1h” in the “from” field, enter “now” in the “to” field, and then click **Save**:

4. Next, in the “buckets” section, click **Split Rows**, select “Terms” from the “Aggregation” drop-down, select “contentAccessoruserID.keyword” from the “Field” drop-down, select “metric: Count” from the “Order By” drop-down, select “Descending” from the “Order” drop-down, and finally set “Size” to “5”. Click the play button at the top of the options and you should see a screen like this:

The screenshot shows the Kibana Visualize interface. The sidebar on the left has options: Discover, Visualize (selected), Dashboard, Timelion, Dev Tools, Management, and Collapse. The main area is titled "Visualize / New Visualization (unsaved)". A search bar at the top says "Search... (e.g. status:200 AND extension:PHP)". Below it is a "Add a filter +" button. The visualization configuration is for the "logstash-\*" index. It's set to "Metric" with "Count" as the metric. Under "Aggregation", "Terms" is selected with "contentAccessorUserID.keyword" as the field. In "Order By", "metric: Count" is selected and ordered "Descending" with a size of 5. On the right, the results table shows one row: "test1" with "contentAccessorUserID.keyword: Descending". At the bottom right, there are "Export" buttons for "Raw" and "Formatted".

- Click **Save** in the top right of the page to save the visualization.

#### 10.4.3.2. Top 5 client IPs in # of requests the last hour

- Similarly to the last metric, go to “Visualize” in the sidebar, click **Create a visualization**, select “Data Table”, and then select the “logstash-\*” index. Again confirm the time setting is “Last 1 hour”, and again create the “@timestamp” filter as explained in the previous section.
- Next, in the “buckets” section, click “Split Rows”, select “Terms” from the “Aggregation” drop-down, select “ipAddress.keyword” from the “Field” drop-down, select “metric: Count” from the “Order By” drop-down, select “Descending” from the “Order” drop-down, and finally set “Size” to 5. Click the play button at the top of the options and you should see a screen like this:

The screenshot shows the Kibana interface with the sidebar menu open. The 'Visualize' option is selected. A visualization titled 'logstash-\*' is displayed. The configuration shows a 'Metric' type with 'Count' as the metric. The 'Aggregation' dropdown is set to 'Terms' and the 'Field' is 'ipAddress.keyword'. The 'Order By' section shows 'metric: Count' with 'Order' set to 'Descending' and 'Size' set to 5. The results pane displays two items:

| ipAddress.keyword |
|-------------------|
| 10.10.0.140       |
| 10.10.0.139       |

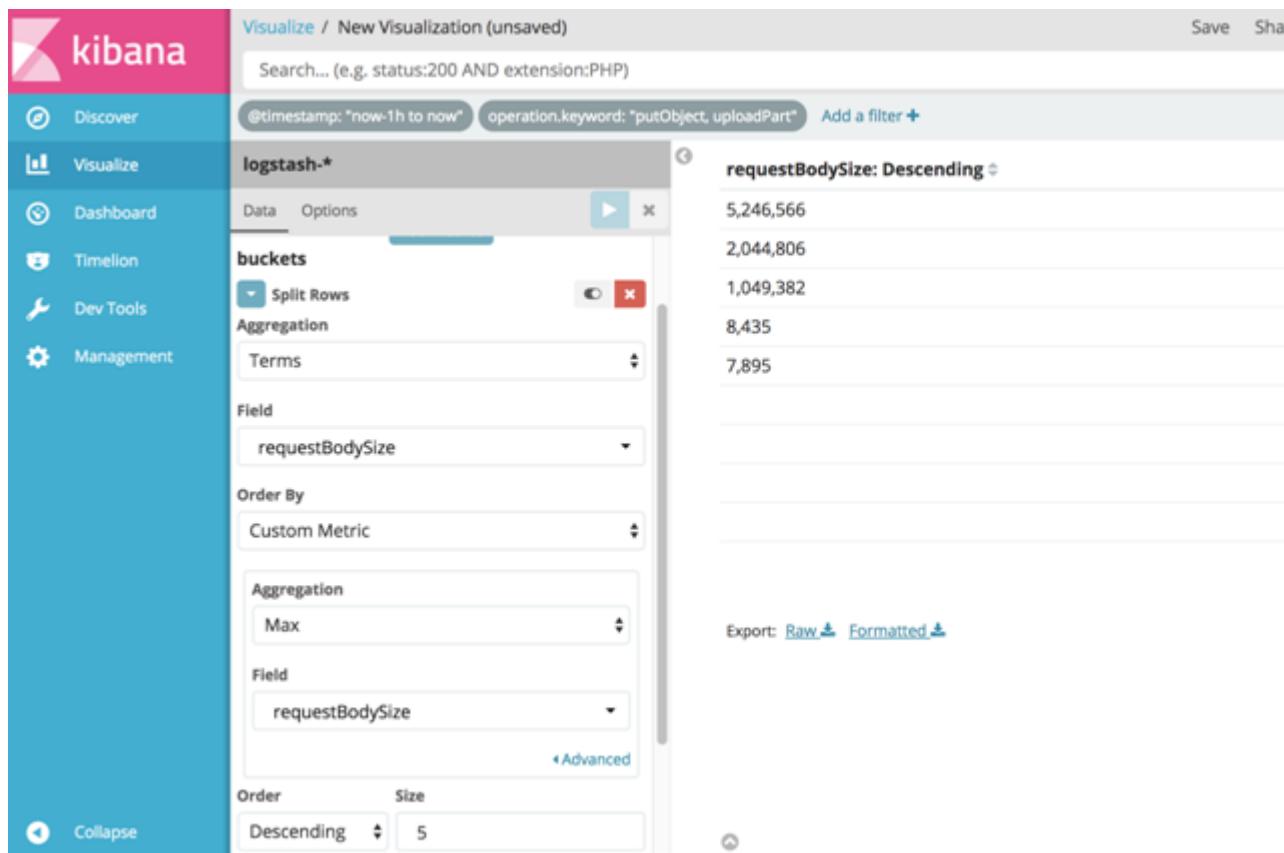
- Click **Save** in the top right of the page to save the visualization.

#### 10.4.3.3. Top 5 PUT object sizes in the last hour

- Similarly to the last metric, go to “Visualize” in the sidebar, click **Create a visualization**, select “Data Table”, and then select the “logstash-\*” index. Again confirm the time setting is “Last 1 hour”, and again create the “@timestamp” filter as explained in the previous section.
- We also have to add another filter for this visualization. Click **Add a filter** under the search bar, select the “operation.keyword” field, select the “is one of” operator, enter “putObject” and “uploadPart” in “Values” and then click Save:



3. Next, in the “buckets” section, click “Split Rows”, select “Terms” from the “Aggregation” drop-down, select “requestBodySize” from the “Field” drop-down, select “Custom Metric” from the “Order By” drop-down, select “Max” from the new “Aggregation” drop-down, select “requestBodySize” from the new “Field” drop-down, select “Descending” from the “Order” drop-down, and finally set “Size” to 5. Click the play button at the top of the options and you should see a screen like this:



4. Click **Save** in the top right of the page to save the visualization.

#### 10.4.3.4. Dashboard

To view all of your saved visualizations in one place go to “Dashboard” and click **Create a Dashboard**. Click the **Add** button at the top of the page, then click on each visualization you just created to add it to the dashboard. Click **Save** at the top of the page. You should now see a screen like this:

The screenshot shows the Kibana interface with a pink header bar containing the Kibana logo and the title "Dashboard / Top 5 Dashboard". On the far right of the header are "Share", "Clone", and "Edit" buttons, and a note "Uses lucene". The left sidebar has a teal background with icons for "Discover", "Visualize", "Dashboard" (which is selected), "Timelion", "Dev Tools", and "Management". A "Collapse" button is at the bottom of the sidebar.

**Top 5 Client IPs in # of requests**

| ipAddress.keyword: Descending | Count |
|-------------------------------|-------|
| 10.10.0.139                   | 170   |
| 10.10.0.140                   | 53    |

**Top 5 PUT object requests**

| requestBodySize: Descending | Count     |
|-----------------------------|-----------|
| 5,246,566                   | 5,246,566 |
| 2,044,806                   | 2,044,806 |
| 1,049,382                   | 1,049,382 |
| 8,435                       | 8,435     |
| 7,895                       | 7,895     |

**Top 5 Users Requesting**

| contentAccessorUserId.keyword: Descending | Count |
|-------------------------------------------|-------|
| test1                                     | 218   |
| bttestuser20171115145614                  | 5     |

# Chapter 11. Commands

## 11.1. hsstool

The HyperStore system includes its own cluster management utility called *hsstool*. This tool has functionality that in many respects parallels the Cassandra utility *nodetool*, with the important distinction that *hsstool* applies its operations to the [HyperStore File System \(HSFS\)](#) as well as to the Cassandra storage layer.

The *hsstool* utility is in the */opt/cloudian/bin* directory of each HyperStore node. Because the HyperStore installation adds */opt/cloudian/bin* to each host's \$PATH environment variable, you can run the *hsstool* utility directly from any directory location on any HyperStore host.

The tool has the following basic syntax:

```
[root]# hsstool -h <host> [-p <port>] <command> [<command-options>]
```

- The *<host>* is the hostname or IP address of the HyperStore node on which to perform the operation. Although some *hsstool* commands allow you to omit the *-h <host>* option (commands that only retrieve data and don't modify anything in the system), for consistency's sake it's best to just always specify the *-h <host>*. Consequently, the syntax summaries and examples in the documentation of individual commands always indicates to specify the *-h <host>*.
- The *<port>* is the HyperStore Service's JMX listening port. If you do not supply the port number when using *hsstool*, it defaults to 19082. There is no need to supply the port when using *hsstool* unless you've configured your system to use a non-default port for the HyperStore Service's JMX listener. The syntax summaries and examples in the documentation of individual commands omit the *-p <port>* option.
- The *hsstool* options *-h <host>* and (if you use it) *-p <port>* must precede the *<command>* and the *<command-options>* on the command line. For example, do not have *-h <host>* come after the *<command>*.
- For best results when running *hsstool* on the command line, run the commands as *root*. While some commands may work when run as a non-root user, others will return error responses.

All *hsstool* operations activity is logged in the *cloudian-hyperstore.log* file on the node to which you sent the *hsstool* command.

For usage information type ***hsstool help*** or ***hsstool help <command>***. The usage information that this returns is not nearly as detailed as what's provided in this documentation, but it does provide basic information about syntax and command options.

The table below lists the commands that *hsstool* supports. For command options and usage information, click on a command name.

**Note** All *hsstool* commands can be executed either on the command line or through the CMC's [Node Advanced](#) page. The documentation in this section shows the CMC interface for each command as well as the command line syntax.

| Command Type         | Command                          | Purpose                                                                                     |
|----------------------|----------------------------------|---------------------------------------------------------------------------------------------|
| Information Commands | <a href="#">ring</a>             | View vNode info for whole cluster                                                           |
|                      | <a href="#">info</a>             | View vNode and data load info for a physical node                                           |
|                      | <a href="#">status</a>           | View summary status for whole cluster                                                       |
|                      | <a href="#">opstatus</a>         | View status of operations such as cleanup, repair, and rebalance                            |
|                      | <a href="#">proactiverepairq</a> | View status of proactive repair queues                                                      |
|                      | <a href="#">repairqueue</a>      | View auto-repair schedule or enable/disable auto-repair                                     |
|                      | <a href="#">ls</a>               | View vNode and data load info per mount point                                               |
|                      | <a href="#">whereis</a>          | View storage location information for an S3 object                                          |
|                      | <a href="#">metadata</a>         | View metadata for an S3 object                                                              |
|                      | <a href="#">trmap</a>            | View token range snapshot IDs or snapshot content                                           |
| Maintenance Commands | madd                             | This is for internal use by the install script, or for use as directed by Cloudian Support. |
|                      | <a href="#">cleanup</a>          | Clean a node of replicated data that doesn't belong to it                                   |
|                      | <a href="#">cleanupec</a>        | Clean a node of erasure coded data that doesn't belong to it                                |
|                      | autorepair                       | In the CMC interface this invokes the <a href="#">repairqueue</a> command                   |
|                      | <a href="#">repair</a>           | Repair the replicated data on a node                                                        |
|                      | <a href="#">repairec</a>         | Repair the erasure coded data in a data center                                              |
|                      | <a href="#">repaircassandra</a>  | Repair only the Cassandra metadata on a node, not the object data                           |
|                      | <a href="#">rebalance</a>        | Shift data load from your existing nodes to a newly added node                              |
|                      | <a href="#">opctl</a>            | List or stop all repair and cleanup operations currently running in the region              |

### 11.1.1. *hsstool cleanup*

Use this [\*\*hsstool\*\*](#) command on a node when you want to identify and delete replica data that does not belong on the node. Broadly, *hsstool cleanup* removes two classes of "garbage" data from a target node:

- Data that belongs to a token range that the target node is no longer responsible for. For example if you add a new node to a cluster, that node will be assigned certain token range responsibilities that previously belonged to the existing nodes in the cluster. After data associated with those token ranges is populated on to the new node (by applying the *hsstool rebalance* command), you use *hsstool cleanup* to remove from the other nodes the replica data that they are no longer responsible for.
- Data that should not be on the node even though the data falls within the token ranges that the node is responsible for. This can occur, for example, if an object delete request from an S3 client succeeds for some but not all of the object's replicas — leaving a garbage replica on one of the nodes.

By default *hsstool cleanup* performs both types of cleanups, but the command supports an *-x* option to perform only the first type so that you can efficiently clean the pre-existing nodes after adding a new node to a cluster.

**Note** Normally you should only run *hsstool cleanup* on one node at a time, per data center. If you have circumstances where you need to clean up multiple nodes within a data center, you can try running it on two or three nodes concurrently, but pay close attention to cluster performance and also contact Cloudian Support for further guidance.

**Note** The system will not allow you to run cleanup operation on a node on which a repair operation is currently running.

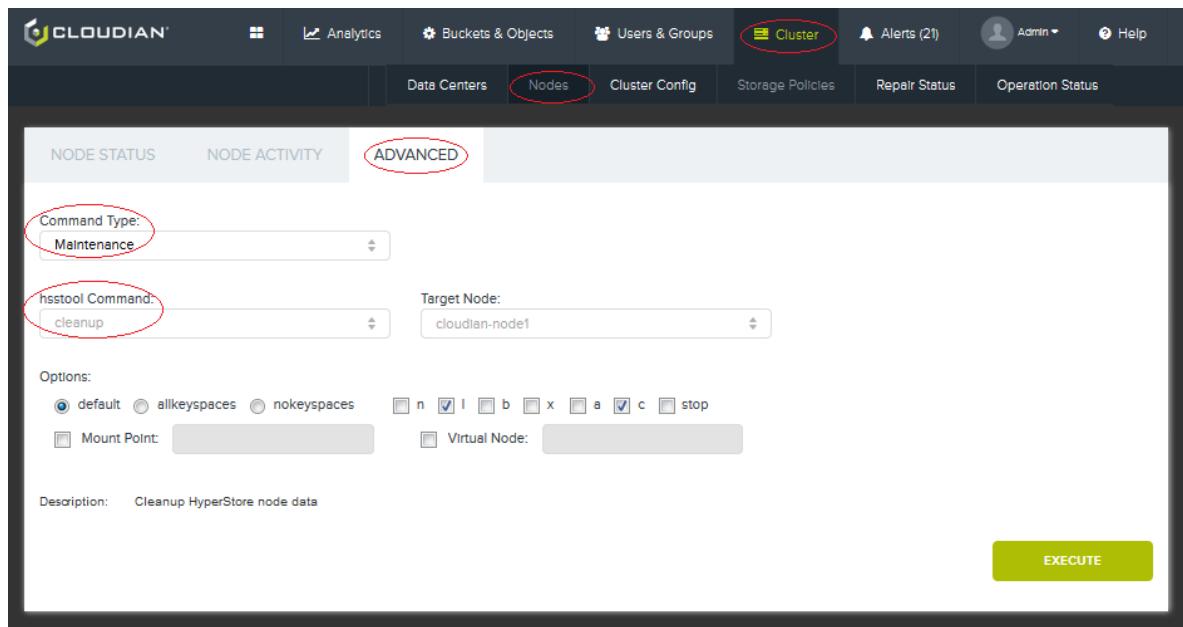
**Note** The *cleanup* operation will only clean objects whose Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *cleanup*.

### 11.1.1.1. Command Format

The *hsstool cleanup* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool cleanup Parameters**" (page 612).

```
[root]# hsstool -h <host> cleanup [allkeyspaces|nokeyspaces] [-n] [-l <true|false>]
[-b] [-x] [-a] [-c <true|false>] [-d <mountpoint>] [-vnode <token>] [-policy] [-stop]
```

You can also run the *hsstool cleanup* command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's [Operation Status](#) page. This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the [hsstool opstatus](#) command.

### 11.1.1.2. Command/Response Example

The example below shows a default run of *cleanup*, using no options. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**"hsstool cleanup and hsstool opstatus cleanup Response Items "** (page 615).

```
[root]# hsstool -h cloudian-node1 cleanup
Executing cleanup. keyspaces=UserData deletedata=true logging=true deleteobject-
without-bucketinfo=false
check-protection=true deleteobject-without-bucketinfo=false
CLEANUP cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true delete-
data=true
deleteobject-without-policy=false keyspaces=UserData logging=true delete-only-outo-
frange-objects=false
start: Thu May 18 05:13:50 PDT 2017
end: Thu May 18 05:13:50 PDT 2017
duration: 0.145 sec
progress percentage: 100%
cassandra cleanup time: 0.015 sec
task count: 2
completed count: 2
Number of files deleted count: 0
failed count: 0
skipped count: 2
```

**Note** In this example there is very little data in the system and so the operation completes almost instantly. In a real-world environment this is a long-running operation and the command response will not return until the operation completes or fails. In the meanwhile you can track operation progress as described in the Command Format section above.

**Note** If you use the *-n* option when you run the *cleanup* command, one of the response items will be "Number of files to be deleted count" rather than "Number of files deleted count".

### 11.1.1.3. Using *hsstool cleanup*

The operational procedure during which you should use *hsstool cleanup* is:

- **"Adding Nodes"** (page 369)

You may also want to use *hsstool cleanup* as part of:

- **"Restoring a Node That Has Been Offline"** (page 400)
- **"Delete a Storage Policy"** (page 332)

Please refer to those procedures for step-by-step instructions, including the proper use of *hsstool cleanup* within the context of the procedure.

### 11.1.1.4. *hsstool cleanup* Parameters

*-h <host>*

---

(Mandatory) Hostname or IP address of the node to clean.

**Note** In the CMC UI for this command this parameter is called "Target Node".

*allkeyspaces | nokeyspaces*

(Optional) You have three alternatives for choosing which Cassandra metadata keyspaces to clean up, while also cleaning replicated S3 object data in the HyperStore File System (HSFS):

- Use *allkeyspaces* to clean up replicated S3 objects in the HSFS and also clean up all the Cassandra keyspaces. Cassandra cleanup will be completed first, then HSFS replica cleanup. The Cassandra keyspaces that will be cleaned are: *UserData\_<storage-policy-ID>* keyspaces; *AccountInfo*; *Reports*; *Monitoring*; and *ECKeyspace*. (For more information see the overview of [Cassandra keyspaces for HyperStore](#))
- Use *nokeyspaces* to clean up only replicated objects in the HSFS, and not any Cassandra keyspaces
- If you specify neither *allkeyspaces* nor *nokeyspaces* then the default behavior is to clean up replicated objects in the HSFS and also to clean the *Cassandra UserData\_<storage-policy-ID>* keyspaces (which store object metadata). Cassandra cleanup will be completed first, then HSFS replica cleanup.

*-n*

(Optional) Don't actually delete anything. Instead, just do a dry run that identifies the replica data that doesn't belong to the node. If you use the *-n* option you must also:

- Use the *nokeyspaces* option. The ability to do a dry run without actually deleting data is supported only for HSFS replica data. Therefore you must select the *nokeyspaces* option or else the cleanup run will actually clean Cassandra keyspace data.
- Have cleanup operation logging turned on (as it is by default). See the description of the *-l* option below.

**Note** If you use the *-n* option when you run the *cleanup* command, one of the response items will be "Number of files to be deleted count". The specific objects identified for deletion will be listed in the cleanup log file as in the *-l* option description below.

*-l <true|false>*

(Optional, defaults to true) Write to a log file a list of all the objects that were identified as not belonging to the node. Defaults to true, so you only need to specify the *-l* option if you do **not** want cleanup object logging (in which case you'd specify *-l false*).

If you use logging without using the *-n* option, then the list in the log file is a list of all objects that were deleted by the *cleanup* operation.

If you use logging in combination with the *-n* option, then the list in the log file is a list of HSFS replica objects that will be deleted if you run *cleanup* again without the *-n* option.

The log is named *cloudian-hyperstore-cleanup.log* and is written into the Cloudian HyperStore log directory of the target host. Activity associated with a particular instance of a *cleanup* command run is marked with a unique command number.

*-b*

(Optional) Delete any objects that are not associated with a valid S3 bucket. If you use this option the cleanup

operation will check each object to verify that it is part of a bucket, and delete any objects that are not part of a bucket. In typical cleanup scenarios it's not necessary to use this option.

`-X`

(Optional) Remove only data that belongs to token ranges that the target node is not responsible for. This is the recommended option to use when you are cleaning the other nodes after adding a new node to a cluster. In this circumstance, using the `-x` option makes the cleanup more efficient and faster.

`-a`

(Optional) Clean up garbage replica data and then also clean up garbage erasure coded data. When you use the `-a` option, as soon as the `hsstool cleanup` operation completes on a node an `hsstool cleanupec` operation is automatically run on the node as well. This is a convenient option if you are cleaning a node that is storing both replica data and erasure coded data.

Do **not** use this option if you are using either the `-d <mount-point>` option or the `-vnode <token>` option.

`-c <true|false>`

(Optional, defaults to true) If true, then before removing an object replica because it does not belong to the token ranges that the target node is responsible for, the system will first check to make sure that at least one replica of the object exists on the correct endpoint nodes within in the cluster. If no other replicas exist, then the out-of-range replica will be left in place on the node that's being cleaned and an ERROR level message will be written to the HyperStore Service application log (which will also result in the triggering of an Alert in the CMC, if you are using the default alert rules).

This safety feature guards against the possibility of a cleanup operation deleting an incorrectly placed replica when no other replicas of the object exist in any of the correct locations within the cluster. The trade-off is that the cleanup operation will take longer if this approach is used.

This safety feature defaults to true, so there's no need to use the `-c` option unless you want to specify `-c false` in order to skip this safety check.

`-d <mount-point>`

(Optional) Clean only the specified HyperStore data mount point (for example `/cloudian1`). This option may be useful if you want to delete garbage data from a particular disk, in an effort to free up space on the disk.

`-vnode <token>`

(Optional) Clean only those objects mapped to the specified vNode (identified by its token such as 18315119863185730105557340630830311535). This option may be useful if during a full node cleanup (or a disk-specific cleanup), the operation failed for a particular vNode. In that case you can then use the `-vnode <token>` option to retry cleaning just that one vNode.

`-policy`

(Optional) By default `cleanup` will evaluate and clean only data associated with storage policies that currently exist in your system. It will not evaluate or delete data associated with storage policies that you've deleted from your system. If you want `cleanup` to also delete from the target node **all data associated with storage policies that you've deleted from the system**, use the `-policy` option.

**Note** Before deleting a storage policy you are required to delete any buckets and objects that are stored under that policy. However, the way that object deletion works in HyperStore is that the system

deletes the object metadata immediately but does not delete the actual object data until the next hourly run of the [object deletion cron job](#). Meanwhile, for storage policy deletion, the full deletion of the metadata associated with the policy is implemented by a [daily cron job](#). Depending on when you delete buckets and objects associated with a storage policy and when you delete the policy itself, the timing may be such that the daily storage policy deletion cron job executes before some of the object data associated with the policy gets deleted by the hourly object deletion cron job. It's this residual "garbage data" that will be detected and removed if you use the *-policy* option when running *cleanup* on a node.

**Note** The CMC interface does not support the *-policy* option. This option is supported only on the command line.

#### *-stop*

(Optional) Use *hsstool -h <host> cleanup -stop* to terminate an in-progress cleanup operation on the specified node.

You can subsequently use the "**hsstool opstatus**" (page 630) command to confirm that the cleanup has been stopped (status = TERMINATED) and to see how much cleanup progress had been made before the stop.

If you terminate an in-progress cleanup operation you will not subsequently be able to resume that operation from the point at which it stopped. Instead, when you want to clean the node you can run a regular full cleanup operation.

#### 11.1.1.5. *hsstool cleanup* and *hsstool opstatus cleanup* Response Items

##### *cmdno#*

Command number of the run. Each run of a command is assigned a number.

##### *status*

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For high-level information about object cleanup successes and failures (if any), see the other fields in the *cleanup* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *cleanup* response. For details on any FAILED operation you can also scan *cloudian-hyper-store.log* for error messages from the period during which the operation was running.

A TERMINATED status means that the cleanup run was terminated by an operator, using *cleanup -stop*.

##### *arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear.

##### *start*

Start time of the operation.

*end, duration*

End time and duration of a completed operation.

*estimated completion time, time remaining*

Estimated completion time and estimated time remaining for an in-progress operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*cassandra cleanup time*

The time spent cleaning metadata in Cassandra.

*task count*

The number of object replicas on the node, which the cleanup operation must evaluate to determine whether they correctly belong on the node.

*completed count*

The number of object replicas that the cleanup operation evaluated to determine whether they correctly belong on the node.

*Number of files deleted count*

The number of object replicas that the cleanup operation successfully deleted from the node because they don't belong on the node.

**Note** If you use the `-n` option when you run the `cleanup` command, this response item will be "Number of files to be deleted count" rather than "Number of files deleted count".

*failed count*

The number of object replicas that the cleanup operation tried to delete (because they don't belong on the node), but failed.

*skipped count*

The number of object replicas that the cleanup operation left on the node because they belong on the node.

### 11.1.2. `hsstool cleanupec`

**Note** If you want to clean **replica data and also erasure coded data** on a node, use [`hsstool cleanup`](#) with the `-a` option. If you want to clean **only erasure coded data** on a node, use `hsstool cleanupec` as described below.

Use this [`hsstool`](#) command on a node when you want to identify and delete erasure coded data that does not belong on the node. Broadly, `hsstool cleanupec` removes two classes of "garbage" data from a target node:

- Data that belongs to a token range that the target node is no longer responsible for. For example if you add a new node to a cluster, that node will be assigned certain token range responsibilities that

previously belonged to the existing nodes in the cluster. After data associated with those token ranges is populated on to the new node (by applying the *hsstool rebalance* command), you use *hsstool cleanupec* to remove from the other nodes the erasure coded data fragments that they are no longer responsible for.

- Data that should not be on the node even though the data falls within the token ranges that the node is responsible for. This can occur, for example, if an object delete request from an S3 client succeeds for some but not all of the object's fragments — leaving a garbage fragment on one of the nodes.

By default *hsstool cleanupec* performs both types of cleanups, but the command supports an *-x* option to perform only the first type so that you can efficiently clean the pre-existing nodes after adding a new node to a cluster.

**Note** Normally you should only run *hsstool cleanupec* on one node at a time, per data center. If you have circumstances where you need to clean up multiple nodes within a data center, you can try running it on two or three nodes concurrently, but pay close attention to cluster performance and also contact Cloudian Support for further guidance.

**Note** The system will not allow you to run cleanup operation on a node on which a repair operation is currently running.

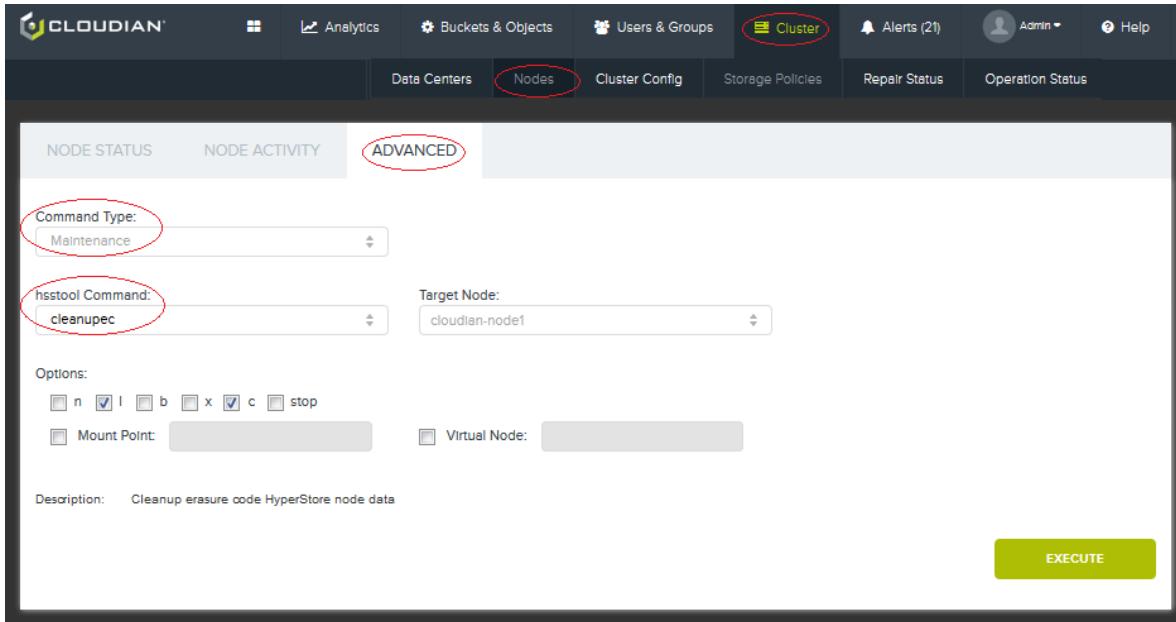
**Note** The *hsstool cleanupec* operation will only clean objects whose Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *hsstool cleanupec*.

### 11.1.2.1. Command Format

The *hsstool cleanupec* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool cleanupec Parameters**" (page 619).

```
[root]# hsstool -h <host> cleanupec [-n] [-l <true|false>] [-b] [-x]
[-c <true|false>] [-d <mountpoint>] [-vnode <token>] [-policy] [-stop]
```

You can also run the *hsstool cleanupec* command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's [Operation Status](#) page. This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the [hsstool opstatus](#) command.

### 11.1.2.2. Command/Response Example

The example below shows a default run of *cleanupec*, using no options. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "["hsstool cleanupec and hsstool opstatus cleanupec Response Items"](#) (page 621).

```
[root]# hsstool -h cloudian-node1 cleanupec
Executing cleanupec. deleteobject-without-bucketinfo=false check-protection=true
deletedata=true logging=true
delete-only-outofrange-objects=false
CLEANUPEC cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true delete-
data=true logging=true
delete-only-outofrange-objects=false
start: Thu May 18 05:14:18 PDT 2017
end: Thu May 18 05:14:18 PDT 2017
duration: 0.011 sec
progress percentage: 100%
task count: 0
completed count: 0
Number of files deleted count: 0
failed count: 0
skipped count: 0
```

**Note** In this example there is very little data in the system and so the operation completes almost instantly. In a real-world environment this is a long-running operation and the command response will not return until the operation completes or fails. In the meanwhile you can track operation progress as described in the Command Format section above.

**Note** If you use the *-n* option when you run the *cleanupec* command, one of the response items will be "Number of files to be deleted count" rather than "Number of files deleted count".

### 11.1.2.3. Using *hsstool cleanupec*

If you have erasure coded data in your HyperStore system, the operational procedure during which you should use *hsstool cleanupec* is:

- **"Adding Nodes"** (page 369)

You may also want to use *hsstool cleanupec* as part of:

- **"Restoring a Node That Has Been Offline"** (page 400)
- **"Delete a Storage Policy"** (page 332)

Please refer to those procedures for step-by-step instructions, including the proper use of *hsstool cleanupec* within the context of the procedure.

**Note** It's OK for *hsstool cleanupec* to be running on multiple nodes in parallel. To do so, you need to initiate the cleanups one node at a time, but you don't need to wait for *hsstool cleanupec* to complete on one node before starting it on another node.

**Note** The *hsstool cleanupec* operation will only clean objects whose Last Modified timestamp is older than the interval set by the system configuration property *hyperstore-server.properties: cleanup.session.delete.graceperiod*. By default this interval is one day. So by default no objects with Last Modified timestamps within the past 24 hours will be deleted by *hsstool cleanupec*.

### 11.1.2.4. *hsstool cleanupec* Parameters

*-h <host>*

(Mandatory) Hostname or IP address of the node to clean.

**Note** In the CMC UI for this command this parameter is called "Target Node".

*-n*

(Optional) Don't actually delete anything. Instead, just do a dry run that identifies the erasure coded data that doesn't belong to the node. If you use the *-n* option you must also have cleanup operation logging turned on (as it is by default). See the description of the *-l* option below.

**Note** If you use the `-n` option when you run the `cleanupec` command, one of the response items will be "Number of files to be deleted count". The specific objects identified for deletion will be listed in the cleanup log file as in the `-l` option description below.

`-l <true|false>`

(Optional, defaults to true) Write to a log file a list of all the objects that were identified as not belonging to the node. Defaults to true, so you only need to specify the `-l` option if you do **not** want cleanup object logging (in which case you'd specify `-l false`).

If you use logging without using the `-n` option, then the list in the log file is a list of all objects that were deleted by the `cleanupec` operation.

If you use logging in combination with the `-n` option, then the list in the log file is a list of objects that will be deleted if you run `cleanupec` again without the `-n` option.

The log is named `cloudian-hyperstore-cleanup.log` and is written into the Cloudian HyperStore log directory of the target host. Activity associated with a particular instance of a `cleanupec` command run is marked with a unique command number.

`-b`

(Optional) Delete any objects that are not associated with a valid S3 bucket. If you use this option the cleanup operation will check each object to verify that it is part of a bucket, and delete any objects that are not part of a bucket. In typical cleanup scenarios it's not necessary to use this option.

`-x`

(Optional) Remove only data that belongs to token ranges that the target node is not responsible for. This is the recommended option to use when you are cleaning the other nodes after adding a new node to a cluster. In this circumstance, using the `-x` option makes the cleanup more efficient and faster.

`-c <true|false>`

(Optional, defaults to true) If true, then before removing an object fragment because it does not belong to the token ranges that the target node is responsible for, the system will first check to make sure that all  $k+m$  fragments of the object exist on the correct endpoint nodes within in the cluster. If fewer than  $k+m$  fragments exist, then the out-of-range fragment will be left in place on the node that's being cleaned and an ERROR level message will be written to the HyperStore Service application log (which will also result in the triggering of an Alert in the CMC, if you are using the default alert rules).

This safety feature guards against the possibility of a cleanup operation deleting an incorrectly placed fragment when fewer than  $k+m$  fragments of the object exist within the cluster. The trade-off is that the cleanup operation will take longer if this approach is used.

This safety feature defaults to true, so there's no need to use the `-c` option unless you want to specify `-c false` in order to skip this safety check.

`-d <mount-point>`

(Optional) Clean only the specified HyperStore data mount point (for example `/cloudian1`). This option may be useful if you want to delete garbage data from a particular disk, in an effort to free up space on the disk.

`-vnode <token>`

(Optional) Clean only those objects mapped to the specified vNode (identified by its token such as

18315119863185730105557340630830311535). This option may be useful if during a full node cleanup (or a disk-specific cleanup), the operation failed for a particular vNode. In that case you can then use the `-vnode <token>` option to retry cleaning just that one vNode.

#### `-policy`

(Optional) By default `cleanupec` will evaluate and clean only data associated with storage policies that currently exist in your system. It will not evaluate or delete data associated with storage policies that you've deleted from your system. If you want `cleanupec` to also delete from the target node **all data associated with storage policies that you've deleted from the system**, use the `-policy` option.

**Note** Before deleting a storage policy you are required to delete any buckets and objects that are stored under that policy. However, the way that object deletion works in HyperStore is that the system deletes the object metadata immediately but does not delete the actual object data until the next hourly run of the [object deletion cron job](#). Meanwhile, for storage policy deletion, the full deletion of the metadata associated with the policy is implemented by a [daily cron job](#). Depending on when you delete buckets and objects associated with a storage policy and when you delete the policy itself, the timing may be such that the daily storage policy deletion cron job executes before some of the object data associated with the policy gets deleted by the hourly object deletion cron job. It's this residual "garbage data" that will be detected and removed if you use the `-policy` option when running `cleanup` on a node.

**Note** The CMC interface does not support the `-policy` option. This option is supported only on the command line.

#### `-stop`

(Optional) Use `hsstool -h <host> cleanupec -stop` to terminate an in-progress `cleanupec` operation on the specified node.

You can subsequently use the [\*\*"hsstool opstatus"\*\*](#) (page 630) command to confirm that the `cleanupec` operation has been stopped (status = TERMINATED) and to see how much progress had been made before the stop.

If you terminate an in-progress `cleanupec` operation you will not subsequently be able to resume that operation from the point at which it stopped. Instead, when you want to clean the node you can run a regular full `cleanupec` operation.

### 11.1.2.5. `hsstool cleanupec` and `hsstool opstatus cleanupec` Response Items

#### `cmdno#`

Command number of the run. Each run of a command is assigned a number.

#### `status`

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For high-level information about object cleanup successes and failures (if any), see the other fields in the `cleanup` response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *cleanup* response. For details on any FAILED operation you can also scan *cloudian-hyper-store.log* for error messages from the period during which the operation was running.

A TERMINATED status means that the clean up run was terminated by an operator, using *cleanupec -stop*.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear.

*start*

Start time of the operation.

*end, duration*

End time and duration of a completed operation.

*estimated completion time, time remaining*

For an INPROGRESS operation: the estimated completion time of the operation, and estimated time remaining to complete the operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*cassandra cleanup time*

The time spent cleaning metadata in Cassandra.

*task count*

The number of erasure coded fragments on the node, which the cleanup operation must evaluate to determine whether they correctly belong on the node.

*completed count*

The number of erasure coded fragments that the cleanup operation evaluated to determine whether they correctly belong on the node.

*Number of files deleted count*

The number of erasure coded fragments that the cleanup operation successfully deleted from the node because they don't belong on the node.

**Note** If you use the *-n* option when you run the *cleanupec* command, this response item will be "Number of files to be deleted count" rather than "Number of files deleted count".

*failed count*

The number of erasure coded fragments that the cleanup operation tried to delete (because they don't belong on the node), but failed.

*skipped count*

The number of erasure coded fragments that the cleanup operation left on the node because they belong on the node.

### 11.1.3. hsstool info

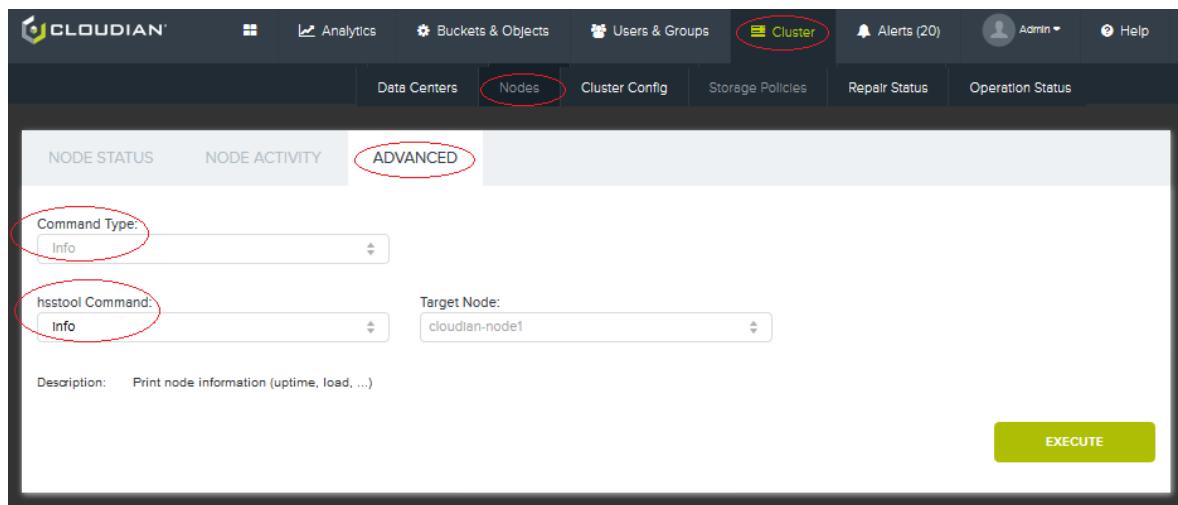
This [hsstool](#) command returns [virtual node](#) (vNode) information and data load information for a specified physical node within a storage cluster. The return includes a list of virtual nodes assigned to the physical node.

#### 11.1.3.1. Command Format

The *hsstool info* command line syntax is as follows.

```
[root]# hsstool -h <host> info
```

You can also run the *hsstool info* command through the CMC UI:



#### 11.1.3.2. Command/Response Example

The example below shows an excerpt from a response to the *info* command. The command returns information about a specific node, "cloudian-node1". The node's token (vNode) list is sorted in ascending order. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "["hsstool info Response Items"](#) (page 624).

```
[root]# hsstool -h cloudian-node1 info
Cloudian : 6.1
Cloudian Load : 1.12 TB / 42.3 TB (2.6%)
Uptime (seconds) : 230681
Token : 18315119863185730105557340630830311535
Token : 21637484670727122681279388562251225465
Token : 23572420191725176386844252744138398599
Token : 25984083804572739863688602357781003456
Token : 32049925251885239737462386844023262134
Token : 34776961444994655981702644433932691872
Token : 39011904510130716900391258282509705889
...
...
Token : 141833557733030600282377220263378688424
Cassandra : 2.0.11
```

|                       |   |        |
|-----------------------|---|--------|
| <u>Cassandra Load</u> | : | 1.9 MB |
| <u>Data Center</u>    | : | DC1    |
| <u>Rack</u>           | : | RAC1   |

**Note** To see which tokens are on which disks on a node, use [hsstool ls](#).

### 11.1.3.3. *hsstool info* Parameters

*-h <host>*

(Mandatory) Hostname or IP address of the node for which to retrieve token and load information.

**Note** In the CMC UI for this command this parameter is called "Target Node".

### 11.1.3.4. *hsstool info* Response Items

#### *Cloudian*

Cloudian HyperStore software version installed on the node.

#### *Cloudian Load*

The total volume of S3 object data (replicas and/or erasure coded fragments) stored in the HyperStore File System on the node, across all HyperStore data disks combined.

This field also shows the total volume of disk space allocated for S3 object storage on the node (the total capacity of HyperStore data disks combined); and the percentage of used volume over total capacity.

#### *Uptime*

The number of seconds that the HyperStore Service has been running since its last start.

#### *Token*

A storage token assigned to the node. Tokens are randomly generated from an integer token space ranging 0 to  $2^{127} - 1$ , and distributed around the cluster. Each token is the top of a token range that constitutes a virtual node (vNode). Each vNode's token range spans from the next-lower token (exclusive) in the cluster up to its own token (inclusive). A physical node's set of tokens/vNodes determines which S3 object data will be stored on the physical node.

For more background information see "**How vNodes Work**" (page 32).

#### *Cassandra*

Cassandra software version installed on the node.

#### *Cassandra Load*

Cassandra storage load (quantity of data stored in Cassandra) on the node. There will be some Cassandra load even if all S3 object data is stored in the HyperStore File System. For example, Cassandra is used for storage of object metadata and service usage data, among other things.

#### *Data Center*

Data center in which the node resides.

**Rack**

Rack in which the node resides.

### 11.1.4. hsstool ls

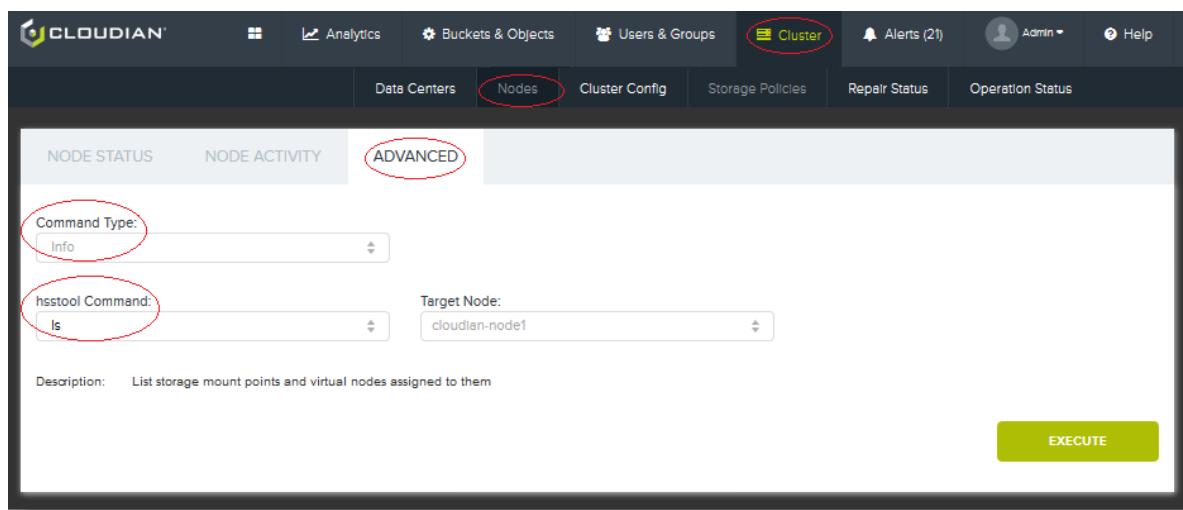
This [hsstool](#) command returns a node's list of HyperStore data mount points, the list of storage tokens currently assigned to each mount point on the node, and the current disk usage per mount point.

#### 11.1.4.1. Command Format

The *hsstool ls* command line syntax is as follows.

```
[root]# hsstool -h <host> ls
```

You can also run the *hsstool ls* command through the CMC UI:



#### 11.1.4.2. Command/Response Example

The example below shows a response to an *hsstool ls* command. The command response snippet below is truncated; the actual response would list all the tokens assigned to each HyperStore data directory mount point on the node. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "["hsstool ls Response Items"](#)" (page 626).

```
[root]# hsstool -h cloudian-node1 ls

<u>device</u>	<u>mount-point</u>	<u>total (KB)</u>	<u>available (KB)</u>	<u>status</u>
/dev/sdf1	/cassandra	226955412	225396540	OK
/dev/sda1	/cloudian1	3784528904	2992343200	OK
/dev/sdb1	/cloudian2	3784528904	2864827364	OK
/dev/sdc1	/cloudian3	3784528904	2932836832	OK
/dev/sdd1	/cloudian4	3784528904	3244435388	OK
 /cloudian1/hsfs: 119513460404589000549564154532759249912 2jf00jEMI1ffVOT7No4LU0				
166983878496718254895534451073709499214 3p37rkVSXWvrnGkGrwNh5a				


```

```
14669198764159070178516665850483502746 Kp70oWCXwTDVyCuWhy18U
...
```

### 11.1.4.3. *hsstool ls* Parameters

*-h <host>*

(Mandatory) Hostname or IP address of the node for which to retrieve each mount point's vNode list and disk usage information.

**Note** In the CMC UI for this command this parameter is called "Target Node".

### 11.1.4.4. *hsstool ls* Response Items

*device*

Device name of the disk drive

*mount-point*

Mount point of the device

*total (KB)*

Total capacity of the disk, in KBs

*available (KB)*

Remaining available capacity of the disk, in KBs

*status*

Disk status: either OK or ERROR or DISABLED. For more information on the disk's status see the CMC's **Node Status** page, [Disk Detail Info](#) section.

*token list*

For each HyperStore data mount point, the lower section of the */s* command response lists all the storage tokens currently assigned to that mount point. Displayed alongside the decimal version of each storage token is the base62 encoding of the token. In the HyperStore File System, base62 encoded tokens will be part of the directory structure for stored S3 object data. For example, under directory *<mount-point>/hsfs/<base62-encoded-tokenX>/...* would be the S3 object replica data associated with the token range for which *tokenX* is the upper bound. For more information on S3 storage directory structure see ["HyperStore Service and the HSFS"](#) (page 12).

## 11.1.5. *hsstool metadata*

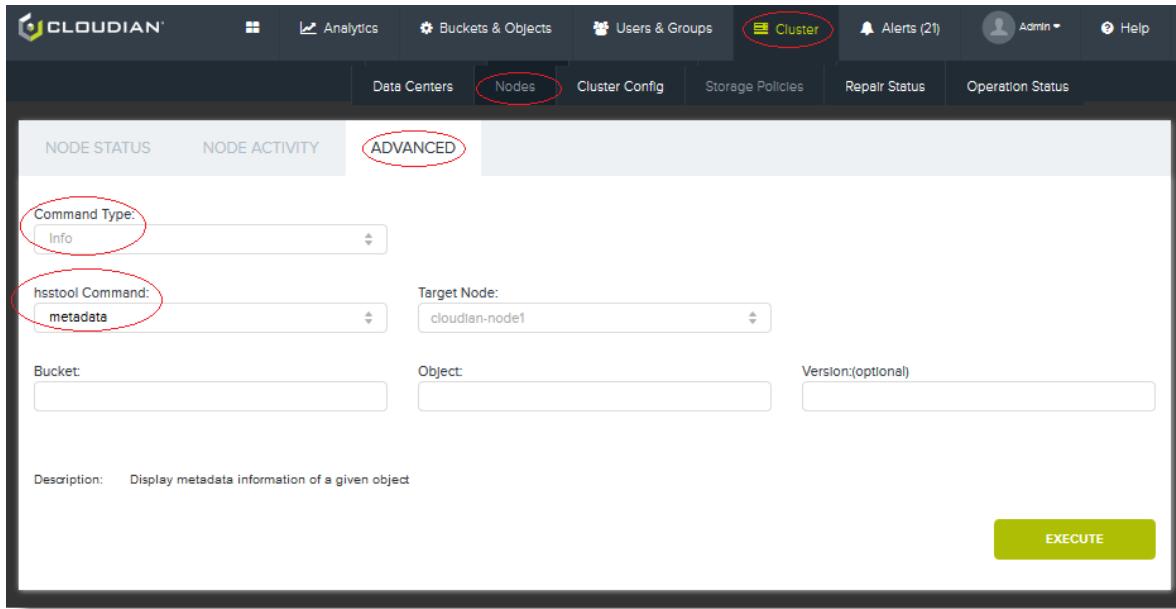
This [\*\*hsstool\*\*](#) command returns metadata for a specified S3 object.

### 11.1.5.1. Command Format

The *hsstool metadata* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see [\*\*"hsstool metadata Parameters"\*\*](#) (page 627).

```
[root]# hsstool -h <host> metadata <bucket>/<object> [-v <version>]
```

You can also run the *hsstool metadata* command through the CMC UI:



### 11.1.5.2. Command/Response Example

The *metadata* command example below returns the metadata for the specified object. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**"hsstool metadata Response Items"** (page 628).

```
[root]# hsstool -h cloudian-node1 metadata pubs.bucket/object1.txt
Key: pubs.bucket/object1.txt
Policy ID: d8d492e9482859017701fee7ff94b649
Uri: file://pubs.bucket/object1.txt
Version: null
Create Time: 2015-12-09T16:37:49.345Z
Last Modified: 2015-12-09T16:37:49.345Z
Last Access Time: 2015-12-09T16:37:49.742Z
Digest: f0875d672f76c3c7f8626874000691ce
Size: 556
Region: region1
```

### 11.1.5.3. *hsstool metadata* Parameters

*-h <host>*

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**Note** In the CMC UI for this command this parameter is called "Target Node".

*<bucket>/<object>*

(Mandatory) Bucket name, followed by a forward slash, followed by the full object name (including "folder

path", if any). For example, *mybucket/file1.txt* or *mybucket/Videos/Vacation/Italy\_2016-06-27.mpg*.

If the object name has spaces in it, enclose the *bucket/object* name pair in quotes. For example, "*mybucket/big document.doc*".

The *bucket/object* name is case-sensitive.

**Note** In the CMC UI implementation of this command, you enter the bucket name and the full object name (including folder path) in separate fields. For example, bucket name *mybucket* and full object name *Videos/Vacation/Italy\_2016-06-27.mpg*.

*-v <version>*

(Optional) Version ID of the object, if versioning has been used for the object. Versions are identified by *timeuuid* values in hexadecimal format (for example, "fe1be647-5f3b-e87f-b433-180373cf31f5"). If versioning has been used for the object but you do not specify a version number in this field, the operation returns metadata for the most recent version of the object.

#### 11.1.5.4. *hsstool metadata* Response Items

##### *Key*

Key that uniquely identifies the S3 object, in format *<bucketname>/<objectname>*. For example, *bucket1/Documents/Meetings\_2016-06-27.docx*.

##### *PolicyID*

System-generated identifier of the storage policy that applies to the bucket in which this object is stored.

##### *URI*

This value is used internally by the system to locate the object. It takes the form of *<location-string>://<key>*. The *<location-string>* string is one of "cassandra" (for objects stored in Cassandra), "file" (for objects stored in the HyperStore File System), "ec" (for objects stored in the erasure coding file system), or "http" (for objects that have been auto-tiered to Amazon storage).

##### *Version*

Object version, if versioning has been used for the object. Versions are identified by *timeuuid* values in hexadecimal format. If versioning has not been used for the object, the Version field displays "Null".

##### *Create Time*

Timestamp for the original creation of the object. Format is ISO 8601 and the time is in Coordinated Universal Time (UTC).

##### *Last Modified*

Timestamp for last modification of the object. Format is ISO 8601 and time is in UTC.

##### *Last Access Time*

Timestamp for last access of the object. An object's Last Access Time is updated if the object is accessed either for retrieval (GET or HEAD) or modification (PUT/POST/Copy). Format is ISO 8601 and time is in UTC.

##### *Digest*

MD5 digest of the object. This will be a 32 digit hexadecimal number. This digest is used in a variety of

operations including data repair.

#### *Size*

The object's size in bytes.

#### *Region*

The HyperStore service region in which the object is stored.

### 11.1.6. hsstool opctl

This [hsstool](#) command returns a list of all [hsstool repair](#), [hsstool repairec](#), [hsstool cleanup](#), and [hsstool cleanupec](#) operations **currently running** in a service region. You can also use the command to **stop** all those in-progress operations.

#### 11.1.6.1. Command Format

The *hsstool opctl* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "[hsstool opctl Parameters](#)" (page 629).

```
[root]# hsstool -h <host> opctl [-l] [-stop]
```

With this command you must use either the *-l* option or the *-stop* option ("*hsstool opctl*" by itself doesn't do anything).

**Note** The *hsstool opctl* command is not supported in the CMC UI.

#### 11.1.6.2. Command/Response Examples

The example below shows the responses to *hsstool opctl* commands. The first command lists the in-progress repair and cleanup operations in the cluster (in this example only a replica repair operation is in progress). The second command stops the in-progress operation(s).

```
[root]# hsstool -h cloudian-node1 opctl -l
10.10.0.184:
REPAIR: rebuild=false, keyspaces=nokeyspaces, max-modified-ts=1526942350092, primary-
range=false,
min-modified-ts=0, logging=true, full-repair=true, check-metadata-
a=true, cmdno=1, merkletree=true,
computedigest=false

[root]# hsstool -h cloudian-node1 opctl -stop
Aborted REPAIR on 10.10.0.184
```

#### 11.1.6.3. *hsstool opctl* Parameters

*-h <host>*

(Mandatory) Hostname or IP address of the node to which to submit the command. This can be any node in the service region. The command will apply to all nodes in the service region.

`-/`

(Optional) List all in-progress *repair*, *repairec*, *cleanup*, and *cleanupec* operations in the service region.

`-stop`

(Optional) Terminate all in-progress *repair*, *repairec*, *cleanup*, and *cleanupec* operations in the service region.

### 11.1.7. hsstool opstatus

This [hsstool](#) command returns the status of the most recent runs of repair, cleanup, rebalance, or decommission operations that have been performed on a specified node. For each operation type:

- If a run of the operation is **in progress** on the node, then that's the run for which status is returned.
- If the operation is not currently in progress on the node, then status is returned for the **most recent** run of that operation on the node, in the time since the last restart of the node.

For checking the status of repair runs other than the most recent run, *opstatus* also supports a command line option to return a **90-day history** of repairs performed on the target node.

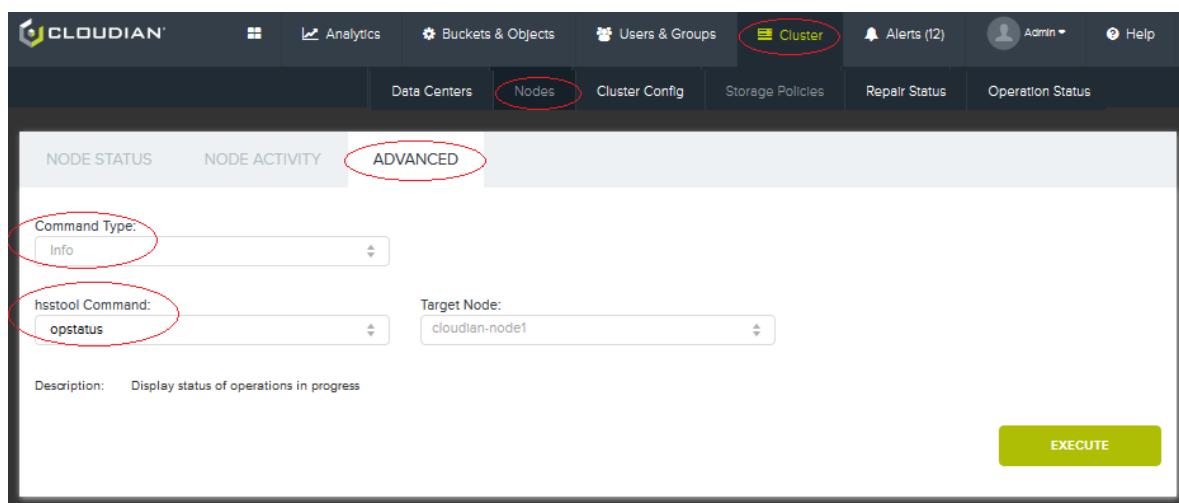
**Note** For operations that you've launched through the CMC UI, a convenient way to check operation status is through the CMC's [Operation Status](#) page. This page does not report on operations that you've launched on the command line -- for such operations *hsstool opstatus* is your only option for checking status.

#### 11.1.7.1. Command Format

The *hsstool opstatus* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "[hsstool opstatus Parameters](#)" (page 634).

```
[root]# hsstool -h <host> opstatus [<op-type>] [-a] [-q history]
```

You can also run the *hsstool opstatus* command through the CMC UI:



### 11.1.7.2. Command/Response Examples

*opstatus for cleanup and cleanupec*

For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool cleanup and hsstool opstatus cleanup Response Items**" (page 615) and "**hsstool cleanupec and hsstool opstatus cleanupec Response Items**" (page 621).

```
[root]# hsstool -h cloudian-node1 opstatus cleanup

CLEANUP cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true delete-
data=true
deleteobject-without-policy=false keyspaces=UserData logging=true delete-only-outo-
frange-objects=false
start: Thu May 18 05:13:50 PDT 2017
end: Thu May 18 05:13:50 PDT 2017
duration: 0.145 sec
progress percentage: 100%
cassandra cleanup time: 0.015 sec
task count: 2
completed count: 2
deleted count: 0
failed count: 0
skipped count: 2
```

```
[root]# hsstool -h cloudian-node1 opstatus cleanupec

CLEANUPEC cmdno#: 1 status: COMPLETED
arguments: deleteobject-without-bucketinfo=false check-protection=true delete-
data=true logging=true
delete-only-outofrange-objects=false
start: Thu May 18 05:14:18 PDT 2017
end: Thu May 18 05:14:18 PDT 2017
duration: 0.011 sec
progress percentage: 100%
task count: 0
completed count: 0
deleted count: 0
failed count: 0
skipped count: 0
```

**Note** For in-progress operations, rather than an end time and duration the *opstatus* response will show an estimated completion time and estimated time remaining.

*opstatus for repair and repairec*

For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool repair and hsstool opstatus repair Response Items**" (page 653) and "**hsstool repairec and hsstool opstatus repairec Response Items**" (page 665).

```
[root]# hsstool -h cloudian-node1 opstatus repair
```

```

REPAIR cmdno#: 1 status: COMPLETED
arguments: keyspaces=UserData max-modified-ts=1495109681947 primary-range=false min-
modified-ts=0
logging=true check-metadata=true merkletree=true computedigest=false
operation ID: ddec91f8-0b56-1149-bd85-5254005d772d
start: Fri Dec 08 05:14:42 PDT 2017
end: Fri Dec 08 05:14:50 PDT 2017
duration: 8.107 sec
progress percentage: 100%
total range count: 3
executed range count: 3
keyspace count: 1
repair file count: 0
failed count: 0
repaired count: 0
pr queued count: 0
completed count: 3
total bytes streamed: 0
scan time: 0.3685
stream time: 0.0

```

```
[root]# hsstool -h cloudian-node1 opstatus repairrec
```

```

REPAIRREC cmdno#: 1 status: COMPLETED
arguments: logging=true check-metadata=true computedigest=false
operation ID: ddec9200-0b56-1149-bd85-5254005d772d
start: Fri Dec 08 17:53:05 PDT 2017
end: Fri Dec 08 17:53:05 PDT 2017
duration: 0.416 sec
progress percentage: 100%
task count: 1064
completed count: 1064
repaired count: 0
failed count: 0
skipped count: 1064

```

**Note** For in-progress operations, rather than an end time and duration the *opstatus* response will show an estimated completion time and estimated time remaining.

**Note** "REPAIRCASSANDRA" status metrics will appear in *opstatus* results for the Cassandra key-space repair part of the auto-repair feature; or for when you manually run *hsstool repair* either with its default behavior (which includes a repair of user data keyspaces in Cassandra) or with the "allkey-spaces" option (which includes a repair of user data keyspaces and service metadata keyspaces in Cassandra).

#### *opstatus for rebalance and rebalanceec*

For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool rebalance and hsstool opstatus rebalance/rebalanceec Response Items**" (page 645).

```
[root]# hsstool -h cloudian-node1 opstatus rebalance

REBALANCE cmdno#: 1 status: COMPLETED
arguments: logging=true
operation ID: 2edcb684-94f0-198d-91da-5254007cc5f2
start: Thu May 04 16:51:03 CST 2017
end: Thu May 04 16:51:04 CST 2017
duration: 0.324 sec
progress percentage: 100%
task count: 1
completed count: 1
streamed count: 1
failed count: 0
skipped count: 0
stream jobs total: 5
stream jobs completed: 5
streamed bytes: 500
```

```
[root]# hsstool -h cloudian-node1 opstatus rebalanceec
```

```
REBALANCEEC cmdno#: 1 status: COMPLETED
arguments: logging=true
operation ID: 2edcb684-94f0-198d-91da-5254007cc5f2
start: Thu May 04 16:51:04 CST 2017
end: Thu May 04 16:51:04 CST 2017
duration: 0.248 sec
progress percentage: 100%
task count: 3
completed count: 3
streamed count: 3
failed count: 0
skipped count: 0
stream jobs total: 6
stream jobs completed: 6
streamed bytes: 1024
```

**Note** For in-progress operations, rather than an end time and duration the *opstatus* response will show an estimated completion time and estimated time remaining.

*opstatus for decommissionreplicas and decommissionec*

**Note** The *decommission* operation is automatically invoked by the CMC's "Uninstall" feature, if you use the Uninstall feature to remove a node that's "live" in the Cassandra ring. Status reporting on a decommission operation is broken out to *decommissionreplicas* (for replicated data) and *decommissionec* (for erasure coded data).

For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**Response Items for hsstool opstatus decommissionreplicas or decommissionec**" (page 635).

```
[root]# hsstool -h cloudian-node1 opstatus decommissionreplicas
```

```
DECOMMISSIONREPLICAS cmdno#: 1 status: INPROGRESS
operation ID: d697d9c4-2fc5-124b-ab0c-525400137a9c
start: Tue May 23 20:31:24 PDT 2017
cassandra decommission time: 61.072 sec
task count: 2410
completed count: 2139
streamed count: 2139
failed count: 0
skipped count: 0
stream jobs total: 40
stream jobs completed: 31
streamed bytes: 4403497430
```

```
[root]# hsstool -h cloudian-node1 opstatus decommissionec
```

```
DECOMMISSIONEC cmdno#: 1 status: INPROGRESS
operation ID: d697d9c4-2fc5-124b-ab0c-525400137a9c
start: Tue May 23 20:32:38 PDT 2017
task count: 757
completed count: 743
streamed count: 743
failed count: 0
skipped count: 0
stream jobs total: 2
stream jobs completed: 1
streamed bytes: 0
```

### 11.1.7.3. *hsstool opstatus* Parameters

**-h <host>**

(Mandatory) Hostname or IP address of the node for which to retrieve operations status.

**Note** In the CMC UI for this command this parameter is called "Target Node".

**<op-type>**

(Optional) Type of operation for which to retrieve status. Valid types are listed below. If you do not specify a type, status is returned for all supported operation types.

- cleanup
- cleanupec
- decommissionreplicas
- decommissionec

**Note:** The *decommission* operation is automatically invoked by the CMC's "Uninstall" feature, if you use the Uninstall feature to remove a node that's "live" in the Cassandra ring. Status reporting on a decommission operation is broken out to *decommissionreplicas* (for replicated data) and *decommissionec* (for erasure coded data).

- proactiverebalance
- proactiverepair
- proactivereparec
- rebalance
- rebalanceec
- repair
- repairec

**Note** The CMC does not support the "<op-type>" option.

*-a*

(Optional) Verbose status output for a repair or rebalance operation. This provides repair or rebalance status details per token range. This detailed data can be helpful if you are working with Cloudian Support to troubleshoot a repair or rebalance problem. This option is supported only for the "repair", "rebalance", and "rebalanceec" operation types — not for the other types.

For example:

```
[root]# /opt/cloudian/bin/hsstool -h <host> opstatus repair -a
```

**Note** The CMC does not support the "-a" option.

*-q history*

(Optional) Use *hsstool -h <host> opstatus -q history* to retrieve a history of rebalance, repair, and cleanup options performed on the target node. By default this history spans the past 90 days. This period is configurable by *mts.properties.erb*: "**monitoring.ophistory.ttl**" (page 512).

If you want just a history of a particular repair or cleanup operation type, use:

```
hsstool -h <host> opstatus -q history <op-type>
```

where <op-type> is *rebalance*, *rebalanceec*, *repair*, *repairec*, *cleanup*, or *cleanupec*. For example:

```
hsstool -h <host> opstatus -q history repairec
```

**Note** The CMC does not support the "-q history" option.

#### 11.1.7.4. Response Items for *hsstool opstatus decommissionreplicas* or *decommissionec*

You can check on the status of an in-progress decommission operation by using the "**hsstool opstatus**" (page 630) command.

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, or FAILED

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For high-level information about decommission operation successes and failures (if any), see the other fields in the response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the response. For details on any FAILED operation you can also scan *cloudian-hyperstore.log* for error messages from the period during which the operation was running.

**Note** Decommission processes replica data first and then erasure coded data. After decommission finishes for erasure coded data the HyperStore Service on the node immediately shuts down and can no longer respond to hsstool commands including hsstool opstatus. Therefore the *decommissionec* operation will never show a status of COMPLETED (since the HyperStore Service shuts down upon *decommissionec* completion).

*operation ID*

Globally unique identifier of the *decommission* run. This may be useful if Cloudian Support is helping you troubleshoot a decommission failure. Note that when *decommission* is run, the DECOMMISSIONREPLICAS part of the response (for replica data) and DECOMMISSIONEC part of the response (for erasure coded data) will both have the same operation ID.

**Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

*start*

Start time of the operation.

*progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

*cassandra decommission time*

The time spent decommissioning the node from the Cassandra ring. (Applicable to the *decommissionreplicas* task only -- not *decommissionec*).

*task count*

The total number of files that are evaluated for possible streaming from the node to be decommissioned to other nodes in the cluster. Each such file constitutes a "task".

*completed count*

From the total task count, the number of tasks that have been completed so far (that is, the number of files for which processing has been completed). Each completed task results in the incrementing of either the "streamed count", the "failed count", or the "skipped count".

*stream jobs total*

The system breaks the decommission streaming operation down into a number of small "jobs", with the number of jobs reflecting factors such as the particular storage policies in the cluster and the token ranges in which

data associated with those storage policies is stored. This metric shows the total number of such jobs.

*stream jobs completed*

The number of stream jobs completed so far.

*streamed count*

The number of object replica files successfully streamed (copied) from the decommissioned node to other nodes in the cluster.

*streamed bytes*

The total number of object replica file bytes successfully streamed (copied) from the decommissioned node to other nodes in the cluster.

*failed count*

The number of files for which the attempt to stream the file to a different node failed as a result of an error. For information about such failures, on the decommissioned node you can scan */var/log/cloudian/cloudian-hyperstore.log* for error messages from the time period during which the decommission operation was running.

*skipped count*

The number of files for which the streaming operation is skipped because a file that was going to be streamed from the decommissioned node to a different target node in the cluster is found to already exist on the target node.

The "streamed count" plus the "failed count" plus the "skipped count" will equal the "completed count".

**Note** For "decommission" operations, *opstatus* can only report in-progress status, not final, completed status. This is because the HyperStore service on the decommissioned node is immediately stopped at the completion of the *decommission* operation. For final decommission operation status you can review */var/log/cloudian/cloudian-hyperstore.log* on the decommissioned node.

## 11.1.8. hsstool proactiverepairq

This **hsstool** command returns information about nodes that are in need of automated proactive repair. This includes nodes for which automated proactive repair is in progress as well as nodes for which automated proactive repair will begin shortly. You can also use the command to immediately start proactive repair (rather than waiting for the automatic hourly run); or to stop in-progress proactive repairs; or to temporarily disable the proactive repair feature (and to re-enable it after having disabled it).

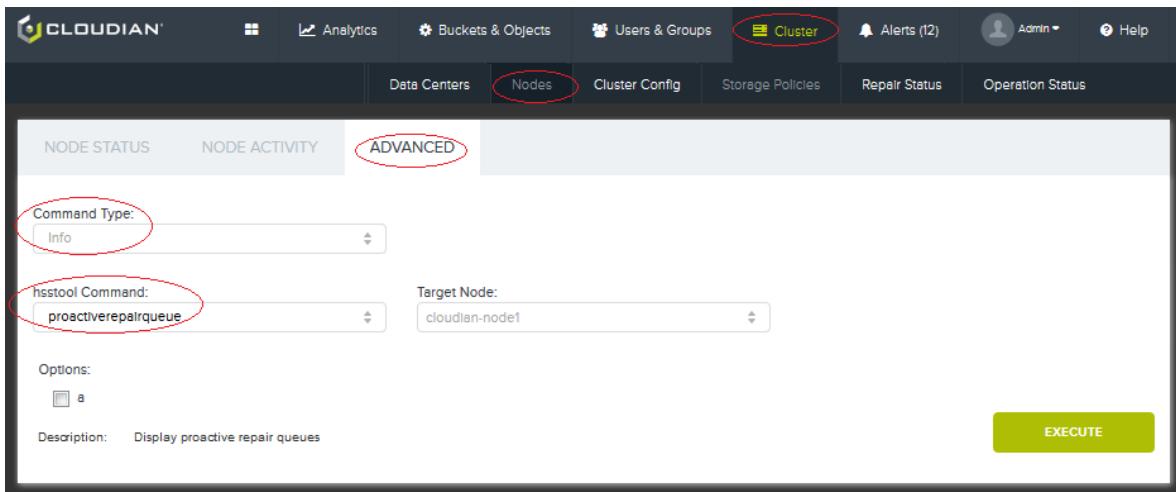
For more information about the HyperStore proactive repair feature see "**Data Repair Feature Overview**" (page 75).

### 11.1.8.1. Command Format

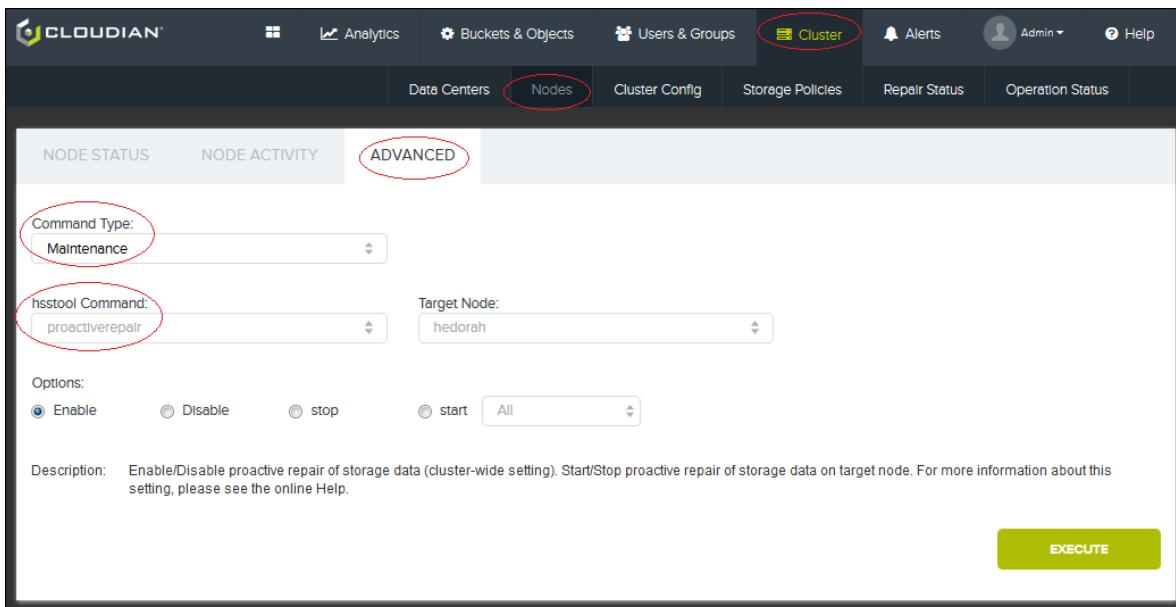
The *hsstool proactiverepairq* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool proactiverepairq Parameters**" (page 639).

```
[root]# hsstool -h <host> proactiverepairq [-a] [-delete <host>]
[-start [-type replicas|ec|rebalance]] [-stop] [-enable true|false]
```

In the CMC UI you can run the *hsstool proactiverepairq* command's proactive repair queue status reporting function through this interface:



The functions for disabling or re-enabling proactive repair, immediately starting a proactive repair, or stopping an in-progress proactive repair have their own separate CMC interface and the command is there renamed as "proactiverepair" (although *hsstool proactiverepairq* is being invoked behind the scenes):



### 11.1.8.2. Command/Response Examples

The *proactiverepairq* command example below shows that one node in the cluster is in need of proactive repair, and that an estimated 100 objects are queued for proactive repair on that node. This proactive repair occurs automatically; no operator action is required. The repair is either already underway or will be triggered at the next interval (default is hourly). For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**"hsstool proactiverepairq Response Items"** (page 641).

```
[root]# hsstool -h localhost proactiverepairq
Proactive repair: true
Number of nodes to repair: 1
```

**Estimated number of PR events per node:**  
cloudian7(10.20.2.57) 100

The *proactiverepairq -a* command example below shows that one node in the cluster is in need of proactive repair, and that the needed repair involves eight object replicas totaling about 1.6 MBs. This proactive repair occurs automatically; no operator action is required. The repair is either already underway or will be triggered at the next interval (default is hourly). For description of a particular response item, click on the response item; or for the full list of response item descriptions see "["hsstool proactiverepairq Response Items"](#) (page 641).

```
[root]# hsstool -h cloudian-node1 proactiverepairq -a
Proactive repair: true
Number of nodes to repair: 1
cld01(10.20.2.123): REPLICAS [count = 8 bytes = 1602141] EC [count = 0 bytes = 0]
REBALANCE [count =
0 bytes = 0]
```

**Note** The *proactiverepairq -a* option provides more precisely accurate information about the number of queued objects than does the *proactiverepairq* command without the *-a* option -- but using the *-a* option is resource intensive.

### 11.1.8.3. *hsstool proactiverepairq* Parameters

**-h <host>**

(Mandatory) Hostname or IP address of the target node:

- For retrieving proactive repair queue information (*hsstool -h <host> proactiverepairq*) or for disabling or re-enabling the proactive repair feature (*hsstool -h <host> proactiverepairq -enable true|false*), this can be any host in your cluster. The command applies to all nodes in the service region of the host that you specify.
- For starting or stopping a proactive repair (*hsstool -h <host> proactiverepairq -start* or *hsstool -h <host> proactiverepairq -stop*), this is the host on which you want to start or stop a proactive repair.

**Note** In the CMC UI for this command this parameter is called "Target Node".

**-a**

(Optional) If you use *hsstool -h <host> proactiverepairq -- without the -a flag* or other options -- the command will return the number of nodes that are in need of proactive repair, the IP addresses of those nodes, and an **estimate** of the number of objects in need of proactive repair on those nodes.

If you use *hsstool -h <host> proactiverepairq -a*, the command will return the number of nodes that are in need of proactive repair and the IP addresses of those nodes, and precise information about the count and total size of objects that are in need of proactive repair on each node. When you use the *-a* option, a scan of Cassandra metadata is done. This is a much more resource intensive operation than if you omit the *-a* option.

**Note** To see how much proactive repair work **has already been completed** on a given node, use the "["hsstool opstatus"](#) (page 630) command on that node.

**-delete <host>**

(Optional) Use this option to completely clear out a specified node's current proactive repair queue. If you do

so, the objects that had been in the proactive repair queue will not be fixed by proactive repair, and instead you will need to fix them by running "**hsstool repair**" (page 647) and "**hsstool repairrec**" (page 658) on the node.

Under normal circumstances you should not need to use the **-delete <host>** option. You might however use this option if you are working with Cloudian Support to troubleshoot problems on a node.

**Note** The CMC interface does not support the **-delete <host>** option. This option is supported only on the command line.

**-start [-type replicas|ec|rebalance]**

(Optional) Use *hsstool -h <host> proactiverpairq -start* to immediately initiate proactive repair on the specified host. This applies **only to the specified host**.

Optionally you can restrict the immediate proactive repair to a particular category of proactive repair: proactive repair of replica data for an existing node; proactive repair of erasure coded data for an existing node; or proactive repair of object data streaming failures from a recently completed rebalance operation for a newly added node. For example, use *hsstool -h <host> proactiverpairq -start -type rebalance* to immediately initiate proactive repair in the wake of a rebalance operation that reported failures for some objects.

If you do not include the **-type** option, then using **-start** will immediately initiate proactive repair for all types of proactive repairs that are currently needed the target node.

**Note** Proactive repair is triggered automatically every hour (by default configuration; see *hyperstore-server.properties.erb: "hyperstore.proactiverpair.poll\_time"* (page 493)), on all nodes that are in need of proactive repair, for all proactive repair types. No operator action is required. So there is no need to use the **-start** option unless for some reason you want proactive repair to begin immediately on a particular node rather than waiting for the next automatic hourly run of proactive repair.

**-stop**

(Optional) Use *hsstool -h <host> proactiverpairq -stop* to immediately stop any in-progress proactive repair on the specified host. This applies **only to the specified host**. Remaining repair tasks that are still in the proactive repair queue for that host will be processed the next time that proactive repair is run.

If proactive repair is in progress on multiple hosts and you want to stop all the proactive repairs, you must submit a *hsstool -h <host> proactiverpairq -stop* command separately for each of those hosts.

**Note** Unlike the **-start** option, the **-stop** option does not support specification of a proactive repair type. Instead the **-stop** option stops all types of in-progress proactive repair on the target host.

**-enable true|false**

(Optional) Enable or disable the proactive repair feature. By default the feature is enabled.

If you use *hsstool -h <host> proactiverpairq -enable false* to disable the proactive repair feature, this applies to **all nodes in the service region of the specified host**. So, it doesn't matter which host you specify in the command as long as it's in the right service region. Likewise *hsstool -h <host> proactiverpairq -enable true* re-enables the proactive repair feature for all nodes in the service region.

Disabling the proactive repair feature **does not abort in-progress proactive repairs**. Rather, it prevents any additional proactive repairs from launching. To stop in-progress proactive repairs on a particular node use the `-stop` option.

**IMPORTANT:** The proactive repair feature is important for maintaining data integrity in your system. Do not leave it disabled permanently.

**Note** The proactive repair feature can also be disabled and re-enabled by using `hsstool repairqueue -h <host> -enable true|false`. The difference is that the "**hsstool repairqueue**" (page 667) approach disables and re-enables scheduled auto-repairs and also proactive repairs, whereas the `hsstool proactiverepairq` approach disables and re-enables only proactive repair.

If you use both types of commands, then whichever command you used most recently will be operative in regard to the proactive repair feature. For example if you run `hsstool repairqueue -h <host> -enable false` and then subsequently you run `hsstool -h <host> proactiverepairq -enable true`, then proactive repair will be enabled in the cluster (while the scheduled auto-repair feature will remain disabled).

#### 11.1.8.4. *hsstool proactiverepairq* Response Items

##### *Proactive repair*

This indicates whether the proactive repair feature is enabled, *true* or *false*. By default proactive repair is enabled.

##### *Number of nodes to repair: <#>*

Number of nodes that are in need of proactive repair.

##### *Estimated number of PR events per node*

This is an **estimate** of the number of objects queued for proactive repair, on each node that currently has objects in its proactive repair queue. This gives you a general idea of the number of queued objects but it is not an exact count.

An exact count is available if you use the `proactiverepairq -a` option, but using that option is resource intensive.

##### *<hostname>(<IPAddress>)*

Hostname and IP address of a node that is in need of proactive repair. The results will show one line for each node that needs proactive repair, with each such line starting with the node's IP address.

##### *REPLICAS[*count = <#> bytes = <#>*]*

For the node identified by <IP Address>, the number of object replicas that need to be written to the node by proactive repair, and the total aggregate size of those replicas in bytes.

If the proactive repair is in progress, these numbers indicate how much is left to do.

##### *EC[*count = <#> bytes = <#>*]*

For the node identified by <IP Address>, the number of erasure coded object fragments that need to be written to the node by proactive repair, and the total aggregate size of those replicas in bytes.

If the proactive repair is in progress, this number indicates how much is left to do.

`REBALANCE[count = <#> bytes = <#>]`

This is applicable only to new nodes that you've recently added to your system. It indicates the number of objects for which streaming to the new node failed during the rebalance operation, and which have consequently been added to the proactive repair queue for the new node. The aggregate number of bytes of such objects is indicated also.

If the proactive repair is in progress, this number indicates how much is left to do.

## 11.1.9. hsstool rebalance

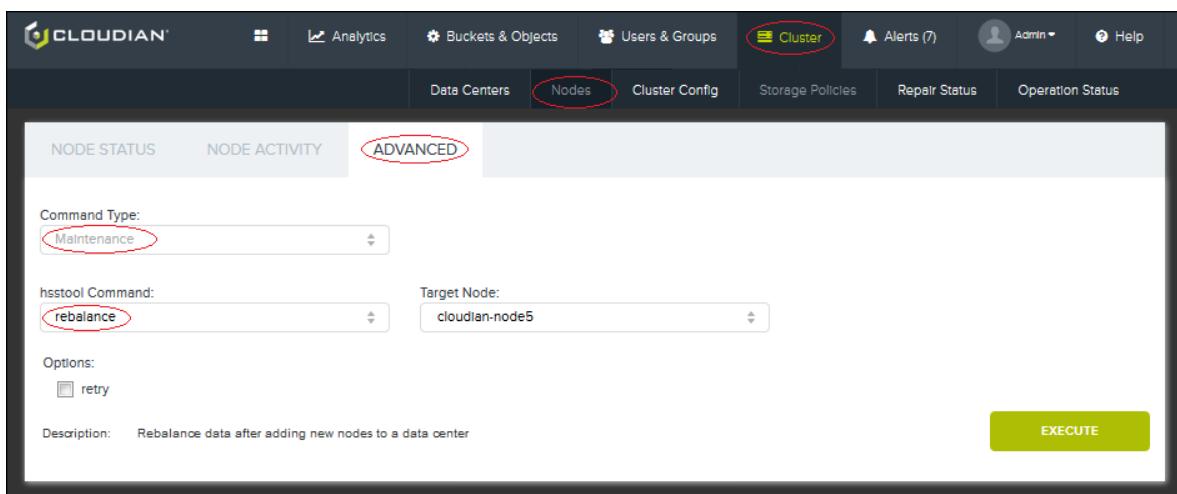
This [hsstool](#) command copies S3 object data from your existing nodes to a specified new node that you have added to your HyperStore cluster. The *rebalance* operation populates the new node with its share of S3 object replica data and erasure coded data, based on the token ranges that the system automatically assigned the new node when you added it to the cluster.

### 11.1.9.1. Command Format

The *hsstool rebalance* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "[hsstool rebalance Parameters](#)" (page 644).

```
[root]# hsstool -h <host> rebalance [-list] [-l <true|false>] [-retry]
```

You can also run this command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's [Operation Status](#) page. This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the [hsstool opstatus](#) command.

### 11.1.9.2. Command/Response Example

The example below shows a *rebalance* operation being executed on a newly added node (cloudian-node5), and the command response. Note that rebalance is implemented separately for replica data ("REBALANCE cmdno #1") and erasure coded data ("REBALANCEEC cmdno #1"). For description of a particular response

item, click on the response item; or for the full list of response item descriptions see "**hsstool rebalance and hsstool opstatus rebalance/rebalanceec Response Items**" (page 645).

**Note** For more detailed status information on a rebalance operation -- including a break-down of status by token range -- you can run `hsstool -h <host> opstatus rebalance -a` (or `opstatus rebalanceec -a`).

```
[root]# hsstool -h cloudian-node5 rebalance
Executing rebalance.
REBALANCE cmdno#: 1 status: COMPLETED
arguments: logging=true
operation ID: 2edcb684-94f0-198d-91da-5254007cc5f2
start: Thu May 04 16:51:03 CST 2017
end: Thu May 04 16:51:04 CST 2017
duration: 0.324 sec
progress percentage: 100%
task count: 1
completed count: 1
streamed count: 1
failed count: 0
skipped count: 0
stream jobs total: 5
stream jobs completed: 5
streamed bytes: 500
REBALANCEEC cmdno#: 1 status: COMPLETED
arguments: logging=true
operation ID: 2edcb684-94f0-198d-91da-5254007cc5f2
start: Thu May 04 16:51:04 CST 2017
end: Thu May 04 16:51:04 CST 2017
duration: 0.248 sec
progress percentage: 100%
task count: 3
completed count: 3
streamed count: 3
failed count: 0
skipped count: 0
prqueued count: 0
stream jobs total: 6
stream jobs completed: 6
streamed bytes: 1024
```

**Note** In this example there is very little data in the system and so the operation completes almost instantly. In a real-world environment this is a long-running operation and the command response will not return until the operation completes or fails. In the meanwhile you can track operation progress as described in the Command Format section above.

### 11.1.9.3. Using `hsstool rebalance`

The only time to use this command is when you have added a new node or nodes to an existing data center.

For complete instructions on adding new nodes including the proper use of the *rebalance* command within the context of the procedure, see:

- **"Adding Nodes"** (page 369)

The rebalance is a background operation that may take many hours or even days, depending on your Hyper-Store cluster size and stored data volume. If you are adding multiple nodes to your cluster, it's OK to have rebalance operations running on multiple new nodes concurrently. See the **"Adding Nodes"** (page 369) procedure for detail.

In the event that the rebalance operation fails for some objects that are supposed to be copied to the new node, these failures will subsequently be corrected automatically by the **"Proactive Repair"** (page 76) feature.

**Note** The **"Adding Nodes"** (page 369) procedure includes as a final step doing a cleanup of your previously existing nodes. This is because the *rebalance* operation **copies** certain object replicas and erasure coded fragments to the new node, from other nodes in the cluster. When you subsequently do the cleanups of other nodes, those replicas and fragments will be deleted from nodes where they no longer belong (as a result of some portions of the token space having been taken over by the new node).

#### 11.1.9.4. *hsstool rebalance* Parameters

**-h <host>**

(Mandatory) Hostname or IP address of the newly added node for which to perform the rebalance operation. In the CMC, only your newly added node(s) will appear in the drop-down node selection list. It is not appropriate to run the *rebalance* command on any node other than a newly added node.

**Note** In the CMC UI for this command this parameter is called "Target Node".

**Note** If you are using the *rebalance* command's "-list" option then the target host can be any host in the region. The returned list will be the list for the region, regardless of which host processes the command.

**-list**

(Optional) If you use this option the command returns a list of newly added nodes in the service region and their status in regard to the rebalance operation.

- A node status of "REQUIRED" means that the node has been added to the cluster (through the CMC's **Add Node** operation) but that you have not yet run *hsstool rebalance* on the new node.
- A node status of "DONE" means that rebalance has been successfully completed for the node.
- A node status of "FAILED" means that the rebalance operation failed for one or more token ranges.

**Note** This parameter is supported only on the command line -- not in the CMC UI.

Sample command and response, where one new node has been added to the cluster and rebalance has been successfully completed on the node:

```
[root@vm151 bin]# ./hsstool -h localhost rebalance -list
vm124(10.50.41.49) :DC1:ca-sm3 DONE
```

The response format is *host(IPaddress):datacenter:region status*

*-l <true|false>*

(Optional, defaults to true) If this option is true, entries for each object rebalanced by this operation will be written to the *cloudian-hyperstore-repair.log* file on the newly added node. Object replicas will be logged with the string "REBR" and object erasure coded fragments will be logged with the string "REBEC".

This option defaults to true, so you only need to specify the *-l* option if you do **not** want rebalanced object logging (in which case you'd specify *-l false*).

**Note** This parameter is supported only on the command line -- not in the CMC UI.

*-retry*

(Optional) If an initial run of *hsstool rebalance* on a new node results in a failure for one or more of the token ranges that the system has assigned to the new node, the next automatic hourly run of proactive repair will try again to stream to the new node the data from the failed token range(s). Alternatively, if you do not want to wait until the next hourly run of proactive repair you can run *hsstool rebalance* with the *-retry* option. The system will then immediately retry the data streaming for the failed token range(s).

**Note** For detailed status information on a rebalance operation that you have already run -- including a break-down of status by token range -- you can run *hsstool -h <host> opstatus rebalance -a* (or *opstatus rebalanceec -a*) on the node.

### 11.1.9.5. *hsstool rebalance* and *hsstool opstatus rebalance/rebalanceec* Response Items

*cmdno#*

Command number of the run. Each run of a command is assigned a number.

*status*

Status of the command run: INPROGRESS, COMPLETED, or FAILED

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For example in the case of repair, a COMPLETED status means that all objects in the scope of the operation were checked to see if they needed repair. It does not mean that all objects determined to need repair were successfully repaired. For high-level information about object repair successes and failures (if any), see the other fields in the *repairrec* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *repair* response. For details on any FAILED operation you can also scan *cloudian-hyperstore.log* for error messages from the period during which the operation was running.

*arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax,

but the relationships should be clear.

#### *operation ID*

Globally unique identifier of the *rebalance* run. This may be useful if Cloudian Support is helping you troubleshoot a rebalance failure. Note that when *rebalance* is run, the REBALANCE part of the response (for replica data) and REBALANCEEC part of the response (for erasure coded data) will both have the same operation ID.

**Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

#### *start*

Start time of the operation.

#### *end, duration*

End time and duration of a completed operation.

#### *progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

#### *task count*

From the existing cluster, the total number of files that are evaluated for possible streaming (copying) to the newly added node, based on the newly added node's assigned tokens. Each such file constitutes a "task".

#### *completed count*

From the total task count, the number of tasks that have been completed so far -- that is, the number of files that the system has evaluated and if appropriate has attempted to stream to the new node. Each "completed" task results in the incrementing of either the "streamed count", the "failed count", the "skipped count", or the "prqueued count".

At the end of the operation the "completed count" should equal the "task count".

#### *streamed count*

The number of files successfully streamed (copied) from the existing nodes to the new node.

#### *failed count*

The number of files for which the attempt to stream the file to the new node fails, and the resulting attempt to insert the file stream job into the proactive repairqueue fails also.

**Note** If the stream attempt for a file fails, but the stream job for that file is successfully added to the proactive repair queue, that file is counted toward the "prqueued count" -- not the "failed count".

For detail about rebalance streaming failures, on the target node for the rebalance operation (the new node) you can scan `/var/log/cloudian/cloudian-hyperstore.log` for error messages from the time period during which the rebalance operation was running.

*skipped count*

For rebalancing of replicated object data: Replicas of each object to be streamed (copied) to a newly added node will typically exist on multiple existing nodes. For example, in a 3X replication environment, for a given object that should be streamed to the new node (based on the new node's token ranges), typically a replica file will reside on three of the existing nodes. The evaluation and processing of each such replica file counts towards the "task count" (so, 3 toward the task count in our example). But once a replica is streamed from one existing node to the new node, it doesn't need to be streamed from the other two existing nodes to the new node. On those other two existing nodes the replica file is "skipped".

*prqueued count*

The number of files for which the attempt to stream the file to the new node fails (even after automatic retries), but the stream job for that file is successfully added to the proactive repair queue. These stream jobs will then be automatically executed by the next run of the hourly proactive repair process.

*stream jobs total*

The system breaks the rebalance streaming operation down into a number of small "jobs", with the number of jobs reflecting factors such as the particular storage policies in the cluster and the token ranges in which data associated with those storage policies is stored. This metric shows the total number of such jobs.

**Note** For rebalance status detail for each individual job (also known as a "session"), run `hsstool -h <host> opstatus rebalance -a` for replicated object data or `hsstool -h <host> opstatus rebalanceec -a` for erasure coded object data. This detailed information can be helpful if you are working with Cloudian Support to troubleshoot rebalance problems.

*stream jobs completed*

The number of stream jobs completed so far. At the end of the operation the "stream jobs completed" should equal the "stream jobs total".

*streamed bytes*

The number of bytes of object data streamed to the new node.

### 11.1.10. hsstool repair

Use this [hsstool](#) command to check whether a physical node has all of the replicated data that it is supposed to have (based on the node's assigned tokens and on replication settings for the system); and to replace or update any data that is missing or out-of-date. Replacement or update of data is implemented by retrieving correct and current replica data from other nodes in the system.

**Note** Within a service region do not run `hsstool repair` on more than one node at a time. If `hsstool repair` is already running on one node (initiated either by an administrator or by the system's [auto-repair](#) feature), and you try to execute `hsstool repair` on a second node in the same service region, the repair operation on the second node will immediately fail. To check to see whether a repair is currently running you can use the CMC's [Operation Status](#) page.

(The one **exception** to this rule is if you use the `-pr` option -- you can run `hsstool repair -pr` on multiple nodes concurrently.)

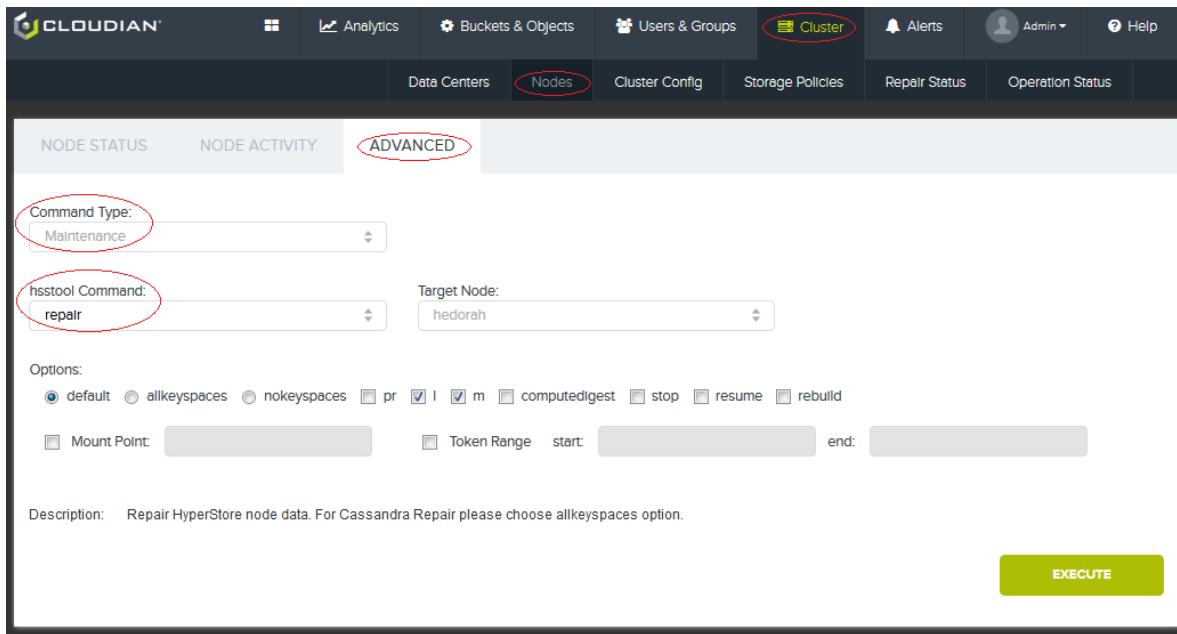
**Note** The system will not allow you to run a repair operation on a node on which a cleanup operation is currently running.

### 11.1.10.1. Command Format

The *hsstool repair* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool repair Parameters**" (page 650).

```
[root]# hsstool -h <host> repair [allkeyspaces|nokeyspaces] [-pr] [-l <true|false>]
[-m <true|false>] [-computedigest] [-stop] [-resume] [-d <mount-point>] [-rebuild]
[-range <start-token,end-token>] [-t <min-timestamp,max-timestamp>]
```

You can also run the *hsstool repair* command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's [Operation Status](#) page. This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the [hsstool opstatus](#) command.

### 11.1.10.2. Command/Response Example

The example below shows a default run of *hsstool repair*, using no options. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool repair and hsstool opstatus repair Response Items**" (page 653).

```
[root]# hsstool -h cloudian-node1 repair
Executing repair. keyspace=UserData max-modified-ts=1495109681947 primary-range-
e=false min-modified-ts=0
logging=true check-metadata=true merkletree=true computedigest=false
REPAIR cmdno#: 1 status: COMPLETED
arguments: keyspace=UserData max-modified-ts=1495109681947 primary-range=false min-
```

```

modified-ts=0 logging=true
check-metadata=true merkletree=true computedigest=false
operation ID: 085cab00-a069-1f51-92c5-525400814603
start: Fri Dec 08 01:03:53 PDT 2017
end: Fri Dec 08 01:03:58 PDT 2017
duration: 4.629 sec
progress percentage: 100%
total range count: 5
executed range count: 5
failed range count: 0
keyspace count: 1
repair file count: 195
failed count: 0
repaired count: 195
pr queued count: 0
completed count: 195
total bytes streamed: 504196225
scan time: 0.37325
stream time: 3.74097663698E8

```

**Note** In this example there is very little data in the system and so the operation completes almost instantly. In a real-world environment this is a long-running operation and the command response will not return until the operation completes or fails. In the meanwhile you can track operation progress as described in the Command Format section above.

#### 11.1.10.3. Using *hsstool repair*

The HyperStore system automatically uses a combination of [read repair](#), [proactive repair](#), and [scheduled auto-repair](#) to keep the replica data on each node complete and current. Consequently, you should rarely need to manually initiate a *repair* operation.

However, there are these uncommon circumstances when you should manually initiate *repair* on a specific node:

- If you are removing a "dead" node from your cluster. In this circumstance, after removing the dead node you will run repair on each of the remaining nodes, one node at a time. See "["Removing a Node"](#)" (page 390) for details.
- If a node is unavailable for longer than the configurable "[hyper-store.proactiverepair.queue.max.time](#)" (page 522) (default = 4 hours). In this case then the metadata required for implementing [proactive repair](#) on the node will stop being written to Cassandra, and an alert will display in the CMC's [Alerts](#) page. When the node comes back online you will need to:
  1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until it completes. You can check the CMC's [Repair Status](#) page periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the objects from during the period when proactive repair metadata was still being written to the Cassandra for the node.
  2. After proactive repair on the node completes, manually initiate a full repair of the node (using *hsstool repair* and, if appropriate for your environment, [hsstool repairec](#)). This will repair objects that were written after the proactive repair queueing time maximum was reached.

**Note** The *repair* operation will fail if the HyperStore service is down on any of the nodes affected by the operation (the nodes storing the affected token ranges). The operation will also fail if any disk storing affected token ranges is disabled or [more than 95% full](#).

#### 11.1.10.4. Problems Remedied by *repair* and *repair computedigest*

The table below lists data problem cases and shows whether or not they are remedied by regular *repair* and by *repair* that uses the *computedigest* option. Although regular *repair* can handle some cases of corruption, if corruption is suspected on a node and you're not certain exactly which data is corrupted, it's best to use *repair computedigest*.

| Case                           | Will repair fix it? | Will repair computedigest fix it? |
|--------------------------------|---------------------|-----------------------------------|
| Missing blob file              | yes                 | yes                               |
| Missing digest                 | yes                 | yes                               |
| Missing blob file and digest   | yes                 | yes                               |
| Corrupted blob file            | no                  | yes                               |
| Corrupted digest               | yes                 | yes                               |
| Corrupted blob file and digest | yes                 | yes                               |

#### 11.1.10.5. *hsstool repair* Parameters

`-h <host>`

(Mandatory) Hostname or IP address of the node to repair.

**Note** In the CMC UI for this command this parameter is called "Target Node".

`allkeyspaces | nokeyspaces`

(Optional) You have three alternatives for choosing which Cassandra metadata keyspaces to repair, while also cleaning replicated S3 object data in the HyperStore File System (HSFS):

- Use `allkeyspaces` to repair replicated objects in the HSFS and also repair all the Cassandra keyspaces. Cassandra repair will be completed first, then HSFS replica repair. The Cassandra keyspaces that will be repaired are: `UserData_<storage-policy-ID>` keyspaces; `AccountInfo`; `Reports`; `Monitoring`; and `ECKeyspace`. (For more information see the overview of [Cassandra keyspaces for HyperStore](#)).
- Use `nokeyspaces` to repair only replicated objects in the HyperStore File System, and not any Cassandra keyspaces.
- If you specify neither `allkeyspaces` nor `nokeyspaces` then the default behavior is to repair replicated objects in the HSFS and also to repair the Cassandra `UserData_<storage-policy-ID>` keyspaces (which store object metadata). Cassandra repair will be completed first, then HSFS replica repair.

`-pr`

(Optional) Only repair objects that fall within the target node's primary ranges (objects for which the hash of the object name falls into one of the token ranges assigned to the node). Do not repair objects that fall into other nodes' primary ranges and that are replicated on to the target node. For background information on replication

in HyperStore, see "**How vNodes Work**" (page 32).

If each node in the cluster is being repaired in succession, using this option makes the successive repair operations less duplicative and more efficient.

**Note** The HyperStore auto-repair feature -- which automatically runs node repairs on a schedule -- uses the *-pr* option when it initiates the *hsstool repair* operation. For more on this feature see "**Data Repair Feature Overview**" (page 75).

**-l <true|false>**

(Optional, defaults to true) If this option is true, write to a log file a list of all the objects that were repaired. Defaults to true, so you only need to specify an *-l* option if you do **not** want repair object logging (in which case you'd specify *-l false*).

The log is named *cloudian-hyperstore-repair.log* and is written into the Cloudian HyperStore log directory of the target node. Activity associated with a particular instance of a command run is marked with a unique command number.

**-m <true|false>**

(Optional, defaults to true) If this option is true, then before repairing a given object the Merkle Tree based repair process will verify that metadata for the object exists in Cassandra. This prevents attempts to repair objects that had been intentionally deleted by users, as indicated by the absence of corresponding object metadata in Cassandra. Defaults to true, so you only need to specify an *-m* option if you do **not** want the object metadata check to be performed (in which case you'd specify *-m false*).

**-computedigest**

(Optional) Use this option if you want to check for and repair not only missing replicas but also any replicas that are present but corrupted. When doing the repair with the *-computedigest* option, the system will compute a fresh digest for each replica rather than using cached digests. If the re-computed digest of a given replica does not match the original digest (stored in a file alongside the object data) or does not match the re-computed digests of the other replicas of that same object (on other nodes), the replica is considered to be corrupted and is replaced by a copy of a correct replica streamed in from a different node.

This way of running *repair* is resource-intensive.

**Note** If you wish, you can have some or all of the [scheduled auto-repairs](#) of replica data use the "-computedigest" option to combat bit rot. This aspect of auto-repair is controlled by the "[auto\\_repair\\_computedigest\\_run\\_number](#)" (page 468) setting in *common.csv*. By default "-computedigest" is not used in auto-repair runs.

**-stop**

(Optional) Use *hsstool -h <host> repair -stop* to terminate an in-progress repair on the specified node.

**If Cassandra repair is in-progress** — that is, if *repair* was launched with the default behavior or the *allkeyspaces* option, and the Cassandra part of the repair is still in-progress — the Cassandra repair is terminated. The HSFS replica repair — which would normally launch after the Cassandra repair — is canceled.

**If HSFS replica repair is in-progress** — that is, if *repair* was launched with the *nokeyspaces* option, or if it was launched with the default behavior or the *allkeyspaces* option and the Cassandra part of the repair has already completed and HSFS replica repair is underway — the HSFS replica repair is terminated.

You can subsequently use the "**hsstool opstatus**" (page 630) command to confirm that the repair has been stopped (status = TERMINATED) and to see how much repair progress had been made before the stop.

If you subsequently want to resume a repair that you stopped, you can use the *repair -resume* option.

**Note** The *-stop* option stops a single in-progress repair on a single node. It does **not** disable the HyperStore [scheduled auto-repair](#) feature.

*-resume*

(Optional) Use this option if you want to resume a previous repair operation that did not complete. This will repair token ranges that were not repaired by the previous repair.

You may want to resume an incomplete repair in any of these circumstances:

- The repair was interrupted by the *repair -stop* command.
- The repair was interrupted by a restart of the target node or a restart of the HyperStore Service on the target node
- The repair reported failed token ranges

**Note** If during the incomplete repair run you used the *-pr* option or the *-d <mount-point>* option, you do not need to re-specify the option when running *repair -resume*. The system will automatically detect that one of those options was used in the previous repair and will use the same option when resuming the repair.

If during the incomplete repair run you used the *-range* option, then resuming the repair is not supported. Repair resumption works by starting from whichever token ranges were not repaired by the previous repair. Since a repair that uses the *-range* option targets just one token range, resuming such a repair would not be any different than running that same repair over again. If you want to run the repair over again, do *repair -range <start-token,end-token>* again, not *repair -resume*.

**Note** If you run *repair -resume*, then in the command results the Arguments section will include "resuming-cmd=<n>", where <n> is the number of times that *repair -resume* has been run on the node since the last restart of the HyperStore Service. This Argument string -- which serves to distinguish *repair -resume* result output from regular *repair* result output -- also appears in "**hsstool opstatus**" (page 630) results for *repair -resume* operations.

*-d <mount-point>*

(Optional) If you use this option the repair is performed only for objects mapped to the vNodes that are assigned to the specified HyperStore data mount point (for example */cloudian1*), either as primary replicas or as secondary or tertiary replicas. This option may be useful in circumstances where data is known or suspected to be missing or incorrect on a particular disk.

**Note** For best repair efficiency in repairing a mount point, use the *-rebuild* option together with the *-d <mount-point>* option -- for example *hsstool repair -d /cloudian1 -rebuild*.

**Note** If you use the `-d <mount-point>` option do not use the `-pr` option or the `-range <start-token,end-token>` option. The system does not support using the `-d` option together with the `-pr` option, or the `-d` option together with the `-range` option.

#### `-rebuild`

(Optional) Use this option together with the `-d <mount-point>` option -- for example `hsstool repair -d /cloudian1 -rebuild`. Using the `-rebuild` option makes the mount point repair process more efficient and less time-consuming than it would be without using the `-rebuild` option.

#### `-range <start-token,end-token>`

(Optional) If you use this option the repair is performed only for objects mapped to your specified token range. You specify the range by indicating the start token and end token (an example of a token is 18315119863185730105557340630830311535). The repair operation will repair all objects that fall within the token range bounded by the start and end tokens.

The **end-token must be a token that is assigned to the target host**. To see what tokens are assigned to each node you can use the "**hsstool ring**" (page 672) command. The **start-token must be the next-lower token in the ring** from the end-token. Put differently, the start-token is the token that forms the lower boundary of the vNode that's identified by the end-token.

This option may be useful if a previous full node repair failed for particular ranges. You can obtain information about range repair failures (including the start and end tokens that bound any failed ranges) by running `hsstool -h <host> opstatus repair -a` on the command line.

**Note** If you use the `-range <start-token,end-token>` option do not use the `-pr` option or the `-d <mount-point>` option. The system does not support using the `-range` option together with the `-pr` option, or the `-range` option together with the `-d` option.

#### `-t <min-timestamp,max-timestamp>`

(Optional) If you use this option the repair is performed only for objects that have a last-modified timestamp equal to or greater than `min-timestamp` and less than `max-timestamp`. When using this option, use Unix milliseconds as the timestamp format.

**Note** This option is not supported in the CMC interface -- only on the command line.

### 11.1.10.6. *hsstool repair* and *hsstool opstatus repair* Response Items

#### `cmdno#`

Command number of the run. Each run of a command is assigned a number.

#### `status`

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED.

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For example in the case of repair, a COMPLETED status means that all objects in the scope of the operation were checked to see if they

needed repair. It does not mean that all objects determined to need repair were successfully repaired. For high-level information about object repair successes and failures (if any), see the other fields in the *repair* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *repair* response. For details on any FAILED operation you can also scan *cloudian-hyper-store.log* for error messages from the period during which the operation was running. More details can also be had by running *hsstool -h <host> opstatus repair -a* on the command line.

A TERMINATED status means that the repair run was terminated by an operator, using *repair -stop*.

#### *arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear. For example, *hsstool repair* command-line syntax supports a "-pr" option, and within "arguments" response item the use or non-use of this option is indicated as "primary-range=true" or "primary-range=false".

#### *operation ID*

Globally unique identifier of the *repair* run. This may be useful if Cloudian Support is helping you troubleshoot a repair failure.

**Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

#### *start*

Start time of the operation.

#### *end, duration*

End time and duration of a completed operation.

#### *progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

#### *total range count*

The total number of token ranges that will be repaired during this repair operation. The repair operation will encompass not only the token ranges assigned to the target repair node but also certain token ranges on other nodes in the cluster. These are token ranges in which are stored replicas of the same objects that are on the target repair node. As a simplified example, if objects residing in token range "c to d" on the target repair node are also replicated in ranges "d to e" and "e to f" on other nodes in the cluster, then ranges "d to e" and "e to f" will be among the ranges repaired by the repair operation (as well as "c to d"). Consequently the total number of ranges to be repaired will be larger than the number of tokens assigned to the node that is the target of the repair.

The exception is if the "-pr" option was used when the *hsstool repair* operation was executed, in which case the repair operation addresses only the target node's "primary ranges". In this case the "total range count" value will equal the number of tokens assigned to the node.

**Note** For repair status detail for each token range, run `hsstool -h <host> opstatus repair -a`. This detailed, per-range status information can be helpful if you are working with Cloudian Support to troubleshoot repair problems.

#### *executed range count*

For each of the token ranges in the "total range count" metric, the repair thread pool schedules a sub-job. The "executed range count" metric indicates the number of sub-jobs scheduled. This does not necessarily mean that all these ranges were successfully repaired -- only that the per-range sub-jobs were scheduled by the thread pool.

#### *failed range count*

The number of token ranges for which the range repair sub-job did not run successfully. For example if range repair sub-jobs are scheduled by the thread pool but then you subsequently terminate the *repair* operation (using the *-stop* option) or reboot the node, this will result in some failed ranges. Problems with the endpoint nodes or the disks impacted by repair of particular token ranges are other factors that may result in failed ranges.

For information about such failures, on the target node for the repair you can scan `/var/log/cloudian/cloudian-hyperstore-repair.log` for the time period during which the repair operation was running. Running `hsstool -h <host> opstatus repair -a` on the command line will also provide useful details about repair failures.

#### *keyspace count*

Number of Cassandra keyspaces repaired. With the default repair behavior this will equal the number of storage policies that are in your HyperStore system. There is one Cassandra `UserData_<policyid>` keyspace for each storage policy. This is where object metadata is stored.

#### *repair file count*

Of all the replica files evaluated by the repair operation, this is the number of files that were determined to be in need of repair. This figure may include files on other nodes as well as files on the target repair node. For example, if an object is correct on the target node but one of the object's replicas on a different node is missing and needs repair, then that counts as one toward the repair file count. For a second example, if two of an object's three replicas are found to be out-dated, that counts as two toward the "repair file count".

#### *failed count*

Of the files that were found to be in need of repair, the number of files for which the attempted repair failed. For information about such failures, on the target node for the repair you can scan `/var/log/cloudian/cloudian-hyperstore-repair.log` for the time period during which the repair operation was running. Running `hsstool -h <host> opstatus repair -a` on the command line will also provide useful details about repair failures.

If possible, files for which the repair attempt fails are added to the proactive repair queue (see "pr queued count" below).

The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

**Note** One thing that can increment the "failed count" is if the operation entails writing data to a disk that is in a [stop-write](#) condition (which by default occurs when a disk is 90% full). Such write attempts will fail.

#### *repaired count*

Of the files that were found to be in need of repair, the number of files for which the repair succeeded. The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

*pr queued count*

The number of files that the *hsstool repair* operation adds to the proactive repair queue, to be fixed by the next proactive repair run. If the *hsstool repair* operation fails to repair a file that requires repair, it adds the file to the proactive repair queue. Proactive repair is a different type of repair operation and may succeed in cases where regular *hsstool repair* failed. By default proactive repair runs every 60 minutes.

The "repaired count" plus the "failed count" plus the "pr queued count" should equal the "repair file count".

*completed count*

The total number of files that were assessed to see if they were in need of repair. This number reflects replication across the cluster — for example, if an object is supposed to be replicated three times (with one replica on the target repair node and two replicas on other nodes), then repair assessment of that object counts as three files toward the "completed count".

Note that "completed" here does not necessarily mean that all object repair attempts succeeded. For more information on success or failure of object repair attempts, see the other status metrics.

*total bytes streamed*

Total number of bytes streamed to the target repair node or other nodes in order to implement repairs. For example, if a 50000 byte object on the target repair node is found to be out-dated, and an up-to-date replica of the object is streamed in from a different node, that counts as 50000 toward "total bytes streamed". As a second example, if a 60000 byte object exists on the target node, and replicas of that object are supposed to exist on two other nodes but are found to be missing, and the repair streams the good object replica from the target repair node to the two other nodes — that counts as 120000 toward "total bytes streamed".

*scan time*

Total time in seconds that it took to scan the file systems and build the Merkle Tree that is used to detect discrepancies in object replicas across nodes.

*stream time*

Total time in seconds that was spent streaming replicas across nodes, in order to implement needed repairs.

### 11.1.11. *hsstool repaircassandra*

Use this [\*\*hsstool\*\*](#) command to repair only the Cassandra data on a node (the system metadata and object metadata stored in Cassandra) and **not** the object data on the node. Under normal circumstances you should not need to use this command, but you might use it when in a troubleshooting or recovery situation.

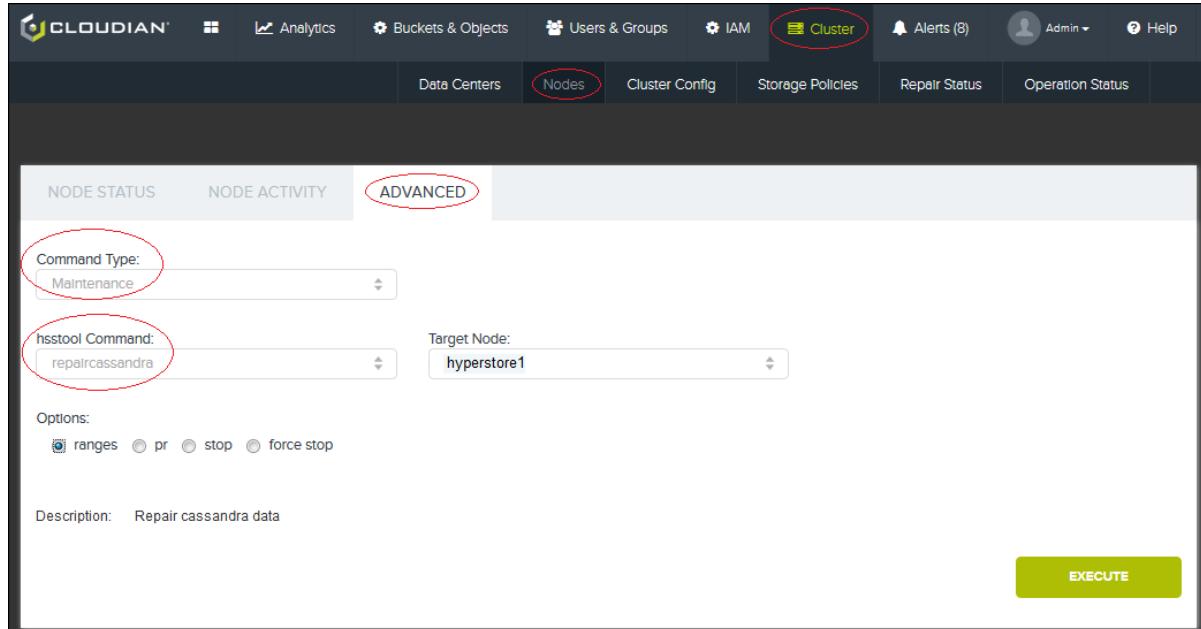
**IMPORTANT:** If you do need to initiate a Cassandra-only repair (with no repair of the object data in the HyperStore File System), use this command rather than using the native Cassandra utility *nodetool* to initiate the repair. Using *hsstool repaircassandra* has multiple advantages over using *nodetool repair*, including that with *hsstool repaircassandra* you can track the repair's progress with [\*\*hsstool opstatus\*\*](#) and you can stop the repair if you need to for some reason.

### 11.1.11.1. Command Format

The *hsstool repaircassandra* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool repaircassandra Parameters**" (page 657).

```
[root]# hsstool -h <host> repaircassandra [-pr]
[-keyspace <keyspacename> [<columnfamily> <columnfamily>...]] [-stop [enforce]]
```

You can also run the *hsstool repaircassandra* command through the CMC UI:



**Note** As is shown in the CMC interface for *hsstool repaircassandra* and in the status response when you run the command, the command applies a "ranges=true" argument by default (the status response includes "ranges=true" in the list of arguments). With this method of Cassandra repair, each impacted token range is repaired one range at a time, sequentially. This approach improves the performance for Cassandra repair. Prior to HyperStore release 7.1.5 using this method was optional, but starting with release 7.1.5 it became the default repair behavior.

### 11.1.11.2. hsstool repaircassandra Parameters

**-h <host>**

(Mandatory) Hostname or IP address of the node to repair.

**Note** In the CMC UI for this command this parameter is called "Target Node".

**-pr**

(Optional) Only repair data that falls within the target node's primary ranges (data for which the hash of the data row key falls into one of the token ranges assigned to the node). Do not repair data that falls into other nodes' primary ranges and that is replicated on to the target node.

If each node in the cluster is being repaired in succession, using this option makes the successive repair operations less duplicative and more efficient.

`-keyspace <keyspacename> [<columnfamily> <columnfamily>...]`

(Optional) Use the `-keyspace` option if you want to repair only a specific keyspace in Cassandra (rather than repairing all keyspaces, which is the default behavior). You can also optionally specify one or more column families within that keyspace, if you want to repair just that column family or those column families. If you do not specify a column family then all column families in the specified keyspace will be repaired.

You can only specify one `-keyspace` option per `hsstool repaircassandra` run. Specifying multiple keyspaces is not supported. Note again that the default behavior of `hsstool repaircassandra` -- if you omit the `-keyspace` option -- is to repair all of the Cassandra keyspaces.

Example of using the `-keyspace` option together with a target column family name:

```
... -keyspace UserData_a36d2c44fbfcc12222e9587b3a42997f CLOUDIAN_OBJMETADATA
```

**Note** The `-keyspace` option is supported only on the command line, not in the CMC UI.

`-stop [enforce]`

(Optional) Use `hsstool -h <host> repaircassandra -stop` to terminate an in-progress Cassandra repair that was initiated via `hsstool` (whether by you or automatically by the system, such as with auto-repair).

If you need to terminate an in-progress Cassandra repair that was initiated via the native Cassandra utility `nodetool`, use `hsstool -h <host> repaircassandra -stop enforce`. Note that using `nodetool` to initiate a Cassandra repair is not recommended.

**Note** In the CMC UI the `-stop enforce` option is called "force stop".

The `hsstool repaircassandra` command does not support a 'resume' option. If you stop an in-progress Cassandra repair, to do the repair again use the `hsstool repaircassandra` command again and the repair will start over.

**Note** The `-stop` option stops a single in-progress Cassandra repair on a single node. It does **not** disable the HyperStore [scheduled auto-repair](#) feature.

### 11.1.12. hsstool repairec

Use this [`hsstool`](#) command to evaluate and repair erasure coded object data. When you run `hsstool repairec` on a target node:

- For **erasure coding storage policies within a single data center**, the `hsstool repairec` operation repairs all erasure coded data on all nodes in the data center in which the target node resides (not just the target node).
- For **replicated erasure coding storage policies spanning multiple data centers**, the `hsstool repairec` operation repairs all erasure coded data on all nodes in the data center in which the target node resides.

- For **distributed erasure coding storage policies spanning multiple data centers**, the *hsstool repair* operation repairs all erasure coded data on all nodes in all of the data centers included in the storage policies.

The command also supports options for repairing just a single disk or just a single token range.

The repair process entails replacing any erasure coded object fragments that are missing, outdated, or corrupted. Replacement of missing fragments is implemented by decoding erasure coded S3 objects, re-encoding them, and then re-writing the missing, outdated, or corrupted fragment(s).

**IMPORTANT:** Within a service region do not run *hsstool repair* on more than one node at a time. If *hsstool repair* is already running on one node (initiated either by an administrator or by the system's [auto-repair](#) feature), and you try to execute *hsstool repair* on a second node in the same service region, the repair operation on the second node will immediately fail. To check to see whether a repair is currently running you can use the CMC's [Operation Status](#) page.

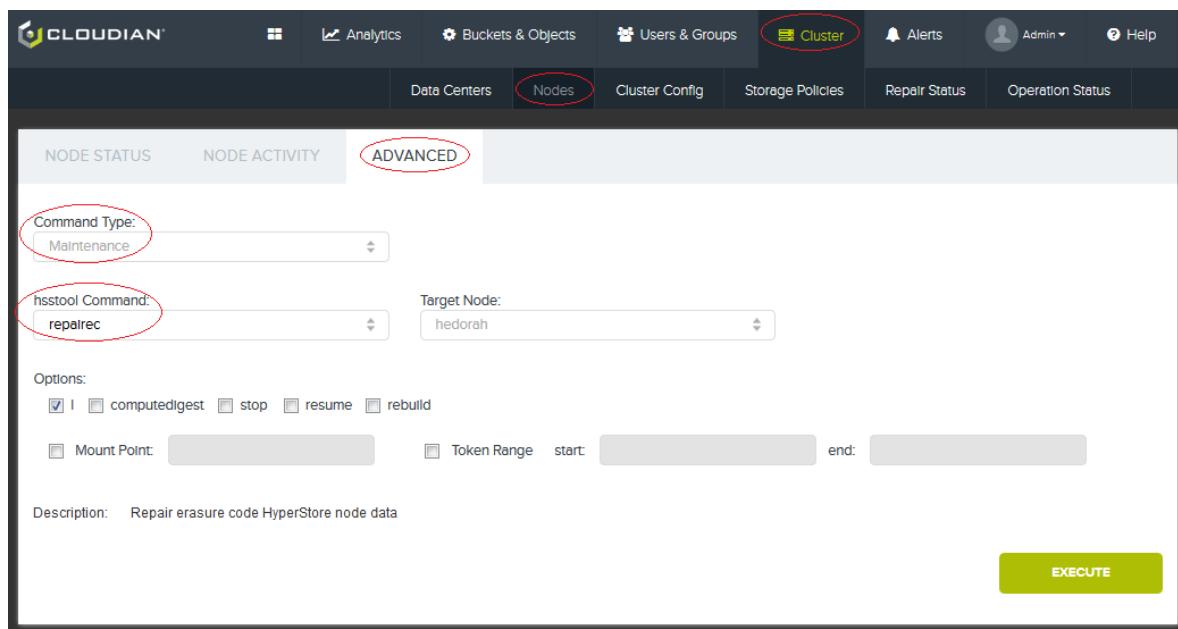
**Note** The *hsstool repair* operation is **potentially a very long running operation**. In a large cluster with high data volume it may take multiple weeks or even multiple months to complete.

#### 11.1.12.1. Command Format

The *hsstool repair* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "["hsstool repair" Parameters](#)" (page 662).

```
[root]# hsstool -h <host> repair [-l <true|false>] [-computedigest] [-stop]
[-resume] [-d <mount-point>] [-rebuild] [-range <start-token,end-token>]
[-f <input-file-path>]
```

You can also run the *hsstool repair* command through the CMC UI:



**Note** If you launch the operation through the CMC UI, you can track the operation progress through the CMC's [Operation Status](#) page. This way of tracking operation progress is not supported if you launch the operation on the command line. However, regardless of how you launch the operation you can periodically check on its progress by using the [hsstool opstatus](#) command.

### 11.1.12.2. Command/Response Example

The example below shows a default run of *repairec*, using no options. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "[hsstool repairec and hsstool opstatus repairec Response Items](#)" (page 665).

```
[root]# hsstool -h cloudian-node1 repairec
Executing repairec. logging=true check-metadata=true computedigest=false
REPAIREC cmdno#: 1 status: COMPLETED
arguments: logging=true check-metadata=true computedigest=false
operation ID: 362bb112-6f06-1d55-bd05-525400d0b090
start: Fri Dec 08 17:53:05 PDT 2017
end: Fri Dec 08 17:53:05 PDT 2017
duration: 0.416 sec
progress percentage: 100%
task count: 1064
completed count: 1064
repaired count: 0
failed count: 0
skipped count: 1064
```

**Note** In this example there is very little data in the system and so the operation completes almost instantly. In a real-world environment this is a long-running operation and the command response will not return until the operation completes or fails. In the meanwhile you can track operation progress as described in the Command Format section above.

### 11.1.12.3. Using *hsstool repairec*

The HyperStore system automatically uses a combination of [read repair, proactive repair, and scheduled auto-repair](#) to keep the erasure coded data on each node complete and current. Consequently, you should rarely need to manually initiate a *repairec* operation.

However, if you use erasure coding in your system, there are these uncommon circumstances when you should manually initiate a *repairec* operation:

- If you are removing a "dead" node from your cluster. In this circumstance, after removing the dead node you will run *repairec* on one node in each of your data centers. See "[Removing a Node](#)" (page 390) for details.
- If a node is unavailable for longer than the configurable "**hyper-store.proactiverepair.queue.max.time**" (page 522) (default = 4 hours). In this case then the metadata required for implementing [proactive repair](#) on the node will stop being written to Cassandra, and an alert will display in the CMC's [Alerts](#) page. When the node comes back online you will need to:
  1. Monitor the automatic proactive repair that initiates on the node when the node starts up, until it

completes. You can check the CMC's [Repair Status](#) page periodically to see whether proactive repair is still running on the node that you've brought back online. This proactive repair will repair the objects from during the period when proactive repair metadata was still being written to the Cassandra for the node.

2. After proactive repair on the node completes, manually initiate a full repair of the node (using *hsstool repairec* and [hsstool repair](#)). This will repair objects that were written after the proactive repair queueing time maximum was reached.

#### 11.1.12.4. Repairing Objects Specified in a File

The *hsstool repairec* command supports an option for repairing one or more specific objects that you list in an input file:

```
[root]# /opt/cloudian/bin/hsstool -h <host> repairec -f <input-file-path>
```

For *<input-file-path>*, specify the full absolute path to the input file (including the file name). Both the input file and the directory in which it is located **must be readable by the 'cloudian' user** or else the repair command will immediately fail and report an error.

The target node can be any node that is part of the erasure coding storage policies used by the objects that you specify in the input file.

**Note** The CMC does not support the *-f <input-file-path>* option. This option is only supported if you use *hsstool repairec* on the command line.

#### Input File Format

In the HyperStore File System on each of your nodes, objects are stored as "chunks". Objects smaller than or equal to the chunk size threshold (default 10MB) are stored as a single chunk. Objects larger than the chunk size threshold are broken into and stored as multiple chunks, with no chunk exceeding the threshold in size. (For more information on the size threshold see "[System Settings](#)" (page 297)). In the case of large objects that S3 client applications upload to HyperStore by the Multipart Upload method, HyperStore breaks the individual parts into chunks if the parts exceed the chunk size threshold.

In the context of erasure coding storage policies, after objects (or object parts) are broken into chunks, each object chunk is erasure coded.

The *hsstool repairec -f <input-file-path>* feature operates on individual chunks, and so in the input file each line must specify a chunk name and the full path to the chunk, in the following format:

```
chunkName | chunkPath
```

There is no limit on the number of lines that you can include in the file (no limit on the number of chunks that you can specify in the file).

Here is an example in which the object is smaller than the chunk size threshold and so the object is stored as just one chunk. In this case the chunk name is simply in the form of *<bucketname>/<objectname>*. (For background information about chunk paths within the HyperStore File System see "[HyperStore Service and the HSFS](#)" (page 12).)

```
bucket1/HyperFileInstallGuide_v-3.6.pdf|/cloudian1/ec/std8ZdRJDskcPvmOg4/
0bb5332b429ccb76466e05bee2915d34/074/156/90721763863541208072539249099911078458.
1554130786616795163-0A3232C9
```

**Note** Although the example above and those that follow below break to multiple lines in this documentation, in the actual input file each *chunkName|chunkPath* combination constitutes just one line in the file.

Here is a second example, for an object that was uploaded by a regular PUT operation (not a Multipart Upload) but which is larger than the chunk size threshold and so has been broken into multiple chunks. The example shows an input file entry for one of those chunks. Note that the chunk name here includes a chunk number suffix (shown in bold).

```
bucket1/HyperFileAdminGuide_v-3.6.pdf..0001|/cloudian1/ec/std8ZdRJDsKcPvmOg4/
0bb5332b429ccb76466e05bee2915d34/087/073/13080395222414127681573583484873262519.
1554124662019670529-0A3232C9
```

In this third example, the object has been uploaded via Multipart Upload. In this example which specifies one of the object's chunks, the chunk name includes a prefix based on the upload ID, as well as a number suffix (both shown in bold).

```
m-MDA1NTE5NjExNTU0MTIzODU3Mjg1/bucket1/cloudian-hyperfile_v-3.6.tar.gz..0001|/cloud-
ian1/
ec/std8ZdRJDsKcPvmOg4/0bb5332b429ccb76466e05bee2915d34/082/100/
39535768889303436494640495599026926454.1554123857285865726-0A3232C9
```

### Collecting Chunk Names and Paths

You have two options for obtaining the chunk name and chunk path for chunks that you want to target for repair. The first option is to use the [hsstool whereis](#) command for each object that you want to repair. The *whereis* response includes the chunk name(s) and chunk path(s) for the object that you specify. You can copy the chunk name and path from the *whereis* response into your input file.

**Note** The *whereis* response has information for each erasure coded fragment, on each node on which the fragments reside. For a given object chunk, each erasure coded fragment has the same chunk name and chunk path, so you can get this information from any of the fragments, regardless of which node a particular fragment is stored on.

The second option for obtaining the chunk name and chunk path for chunks that you want to target for repair is to collect failed task information from [cloudian-hyperstore-repair.log](#). In this log, all repairec failed task log entries have this format:

```
FailedTask|timestamp|chunkName|chunkPath|failedReason
```

You can extract the required information from the repair log. An example Linux command for doing so is:

```
zgrep "FailedTask" /var/log/cloudian/cloudian-hyperstore-repair.log | cut -d "|" -f3,4 >> /tmp/failedTasks.txt
```

This command extracts chunk name and chunk path from the failed task entries in *cloudian-hyperstore-repair.log* and exports that information to */tmp/failedTasks.txt*. You can then use */tmp/failedTasks.txt* as the input file for the *repairec -f <input-file-path>* command.

#### 11.1.12.5. *hsstool repairec* Parameters

This command supports these parameters:

`-h <host>`

(Mandatory) Hostname or IP address of the target node on which to execute the *repairec* operation. By default the *repairec* operation will repair the erasure coded object data on all nodes in the data center in which the target node resides. Consequently as the target node you can choose any node in the data center for which you want to repair erasure coded data.

The exception is if you want to use the "-i" (proactive repair), "-d <mountpoint>", or "-range <start-token,end-token>" option. For those options it does matter which host you specify as the target. See the description of those options for more detail.

**Note** In the CMC UI for this command this parameter is called "Target Node".

`-l <true|false>`

(Optional, defaults to true) If this option is true, write to a log file a list of all the objects that were repaired. Defaults to true, so you only need to specify an `-l` option if you do **not** want repair object logging (in which case you'd specify `-l false`).

The log is named *cloudian-hyperstore-repair.log* and is written into the Cloudian HyperStore log directory of the target node. Activity associated with a particular instance of a command run is marked with a unique command number.

`-computedigest`

(Optional) Use this option if you want to check for and repair not only missing erasure coded fragments but also any fragments that are present but corrupted. When doing the repair with the `-computedigest` option, the system computes a fresh digest for each fragment rather than using cached digests. The re-computed digest of each fragment is compared to the original digest of those fragment (stored alongside the fragment data) and if there is a mismatch the fragment is considered to be corrupted. The object is then decoded from healthy fragments and then re-encoded, and the corrupted fragment is replaced by a new and correct fragment.

This way of running *repairec* is resource-intensive.

**Note** If you wish, you can have some or all of the [scheduled auto-repairs](#) of erasure coded data use the "`-computedigest`" option to combat bit rot. This aspect of auto-repair is controlled by the "**auto\_repair\_computedigest\_run\_number**" (page 468) setting in *common.csv*. By default "`-computedigest`" is not used in auto-repair runs.

`-stop`

(Optional) Use *hsstool -h <host> repairec -stop* to abort an in-progress erasure coded data repair. This stops the repair immediately. You can subsequently use the "**hsstool opstatus**" (page 630) command to confirm that the repair has been stopped (status = TERMINATED) and to see how much repair progress had been made before the stop.

If you stop the default type of erasure-coded data repair -- a repair launched as *hsstool -h <host> repairec*, which repairs the whole data center -- you can later resume the repair by using the `-resume` option. This resumes the repair operation from the point of progress at which it was when you stopped the operation.

**Note** If the repair that you want to stop is a rebuild of the erasure coded data on a particular disk -- a repair launched as `hsstool -h <host> repairec -d <mountpoint> -rebuild` -- you can stop it with `hsstool -h <host> repairec -stop`. Do not include the `-d <mountpoint> -rebuild` options when executing the stop command.

If you stop the rebuild of the erasure coded data on a particular disk, you cannot resume that operation. You will need to run the disk rebuild operation again from scratch.

**Note** The `-stop` option stops a single in-progress repair. It does not disable the HyperStore [scheduled auto-repair](#) feature.

#### `-resume`

(Optional) Use this option if you want to resume a previous erasure coded repair operation that did not complete. This will resume the evaluation and repair of erasure coded fragments in the data center, starting from the point where the uncompleted repair left off. Specifically, the operation will resume its process of scanning object metadata to identify erasure coded objects, evaluating those objects, and if necessary repairing them -- with the resumption starting from the point where the object metadata scan was interrupted.

You may want to resume an erasure coded data repair in either of these circumstances:

- A previous erasure coded data repair was interrupted by the `repairec -stop` command.
- A previous erasure coded data repair was interrupted by a restart of the target node or a restart of the HyperStore Service on the target node

**Note** The `-resume` option **does not apply to rebuilds of erasure-coded data on a particular disk** (a repair launched as `hsstool -h <host> repairec -d <mountpoint> -rebuild`). If you run the resume option it will resume the **last full (data center wide) erasure-coded data repair operation**, if that operation was stopped prematurely. For example, if needed you could perform this sequence of actions:

1. Run `hsstool -h <host> repairec` to start a full repair of the data center.
2. Run `hsstool -h <host> repairec -stop` to stop the data center repair prematurely.
3. Run `hsstool -h <host> repairec -d /cloudian5 -rebuild` to rebuild the erasure coded data on a specific disk.
4. After the disk rebuild completes, run `hsstool -h <host> repairec -resume` to resume the data center repair.

Note that in action 4 the resume command ignores the just-completed disk rebuild, and resumes the last data center repair.

**Note** If you run `repairec -resume`, then in the command results the Arguments section will include "`resuming-cmd=<n>`", where `<n>` is the number of times that `repairec -resume` has been run on the node since the last restart of the HyperStore Service. This Argument string distinguishes `repairec -resume` result output from regular `repairec` result output.

#### `-d <mount-point>`

(Optional) If you use this option the repair is performed only for object fragments mapped to the vNodes that

are assigned to the specified HyperStore data mount point (for example `/cloudian1`) on the target node. This option may be useful in circumstances where data is known or suspected to be missing or incorrect on a particular disk.

**Note** For best repair efficiency in repairing a mount point, use the `-rebuild` option together with the `-d <mount-point>` option -- for example `hsstool -h localhost repairec repairec -d /cloudian1 -rebuild`.

**Note** If you use the `-d <mount-point>` option do not use the `-range <start-token,end-token>` option. The system does not support using the `-d` option together with the `-range` option.

#### `-rebuild`

(Optional) Use this option together with the `-d <mount-point>` option -- for example `hsstool -h localhost repairec -d /cloudian1 -rebuild`. Using the `-rebuild` option makes the mount point repairec process much more efficient and much less time-consuming than it would be without using the `-rebuild` option.

#### `-range <start-token,end-token>`

(Optional) If you use this option the repair is performed only for objects mapped to your specified token range. You specify the range by indicating the start token and end token (an example of a token is 18315119863185730105557340630830311535). The repair operation will repair all objects that fall within the token range bounded by the start and end tokens.

The **end-token must be a token that is assigned to the target host**. To see what tokens are assigned to each node you can use the "**hsstool ring**" (page 672) command. The **start-token must be the next-lower token in the ring** from the end-token. Put differently, the start-token is the token that forms the lower boundary of the vNode that's identified by the end-token.

This option may be useful if a previous full node repair reported failures.

**Note** If you use the `-range <start-token,end-token>` option do not use the `-d <mount-point>` option. The system does not support using the `-d` option together with the `-range` option.

#### `-f <input-file-path>`

(Optional) See "**Repairing Objects Specified in a File**" (page 661).

### 11.1.12.6. *hsstool repairec* and *hsstool opstatus repairec* Response Items

#### `cmdno#`

Command number of the run. Each run of a command is assigned a number.

#### `status`

Status of the command run: INPROGRESS, COMPLETED, FAILED, or TERMINATED.

A COMPLETED status means only that the operation did not error out and prematurely end. It does not mean that the operation succeeded in respect to every object checked by the operation. For example in the case of repair, a COMPLETED status means that all objects in the scope of the operation were checked to see if they needed repair. It does not mean that all objects determined to need repair were successfully repaired. For

high-level information about object repair successes and failures (if any), see the other fields in the *repairec* response.

A FAILED status means that the operation ended prematurely due to errors. For additional status detail see the other fields in the *repair* response. For details on any FAILED operation you can also scan *cloudian-hyper-store.log* for error messages from the period during which the operation was running.

A TERMINATED status means that the repair run was terminated by an operator, using *repairec -stop*.

#### *arguments*

Value of the command arguments used for the run, if any. The status results use internal system names for the arguments which may not exactly match the command-line arguments that are defined in a command's syntax, but the relationships should be clear. For example, *hsstool repair* command-line syntax supports a "-pr" option, and within "arguments" response item the use or non-use of this option is indicated as "primary-range=true" or "primary-range=false".

#### *operation ID*

Globally unique identifier of the *repairec* run. This may be useful if Cloudian Support is helping you troubleshoot a repair failure.

**Note** The "cmd#" (described further above) cannot serve as a globally unique identifier because that counter resets to zero -- and subsequently starts to increment again -- when the HyperStore Service is restarted.

#### *start*

Start time of the operation.

#### *end, duration*

End time and duration of a completed operation.

#### *progress percentage*

Of the total work that the operation has identified as needing to be done, the percentage of work that has been completed so far.

#### *task count*

The number of erasure coded objects that the operation identified, based on a scan of object metadata in Cassandra. For each such object there is one "task", which involves evaluating the object to determine whether it is in need of repair -- such as if one or more of the object's erasure coded fragments is missing from its proper location in the cluster -- and repairing the object if needed.

An interruption of a repair operation can result in the task count being less than the total number of erasure coded objects

#### *completed count*

From the task count, the number of tasks completed. For the definition of a "task" see the description of "task count". A "completed" task means that an erasure coded object (having been identified by a scan of object metadata in Cassandra) was evaluated and if appropriate an attempt was made to repair it. It does not necessarily mean that the attempt to repair the object succeeded (for more information on success or failure of object repair attempts, see the other status metrics below).

If the "completed count" is less than the "task count" this means that the repair was interrupted in such a way that some erasure objects were identified by the object metadata scan (and thus credited toward the "task count") but were not yet evaluated or repaired.

#### *repaired count*

The number of erasure coded objects for which a repair was found to be necessary and was successfully completed. The "repaired count", "failed count", and "skipped count" should add up to equal the "completed count".

#### *failed count*

The number of erasure coded objects for which a repair was found to be necessary, but the repair attempt failed. For information about such failures, on the target node for the repair you can scan `/var/log/cloudian/cloudian-hyperstore-repair.log` for the time period during which the repair operation was running. The "repaired count", "failed count", and "skipped count" should add up to equal the "completed count".

#### *skipped count*

The number of erasure coded objects for which the object evaluation determined that no repair was needed (i.e., all of the object's fragments were in the proper locations within the cluster). The "repaired count", "failed count", and "skipped count" should add up to equal the "completed count".

### 11.1.13. hsstool repairqueue

The HyperStore "auto-repair" feature implements a periodic automatic repair of replicated object data, erasure coded object data, and Cassandra metadata on each node in your system. With the *hsstool repairqueue* command you can:

- **Check on the upcoming auto-repair schedule** as well as the status from the most recent auto-repair runs.
- **Temporarily disable auto-repair** for a particular repair type or all types.
- **Re-enable auto-repair**, if it has previously been disabled by the *hsstool repairqueue* command.

For background information on the auto-repair feature, see "**Data Repair Feature Overview**" (page 75).

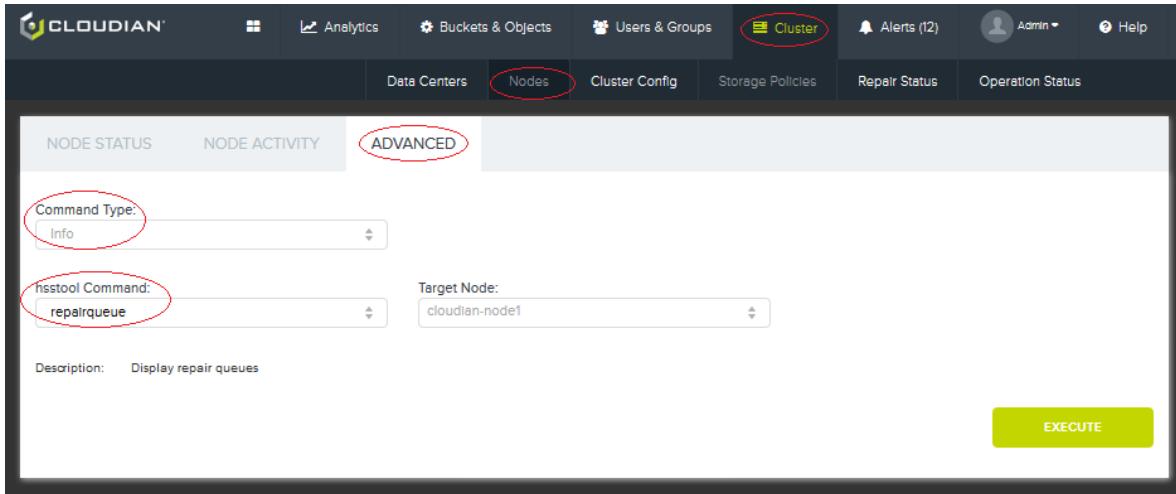
**Note** You cannot enable or disable auto-repair for just one particular node — the auto-repair feature is either enabled or disabled for the cluster as a whole.

#### 11.1.13.1. Command Format

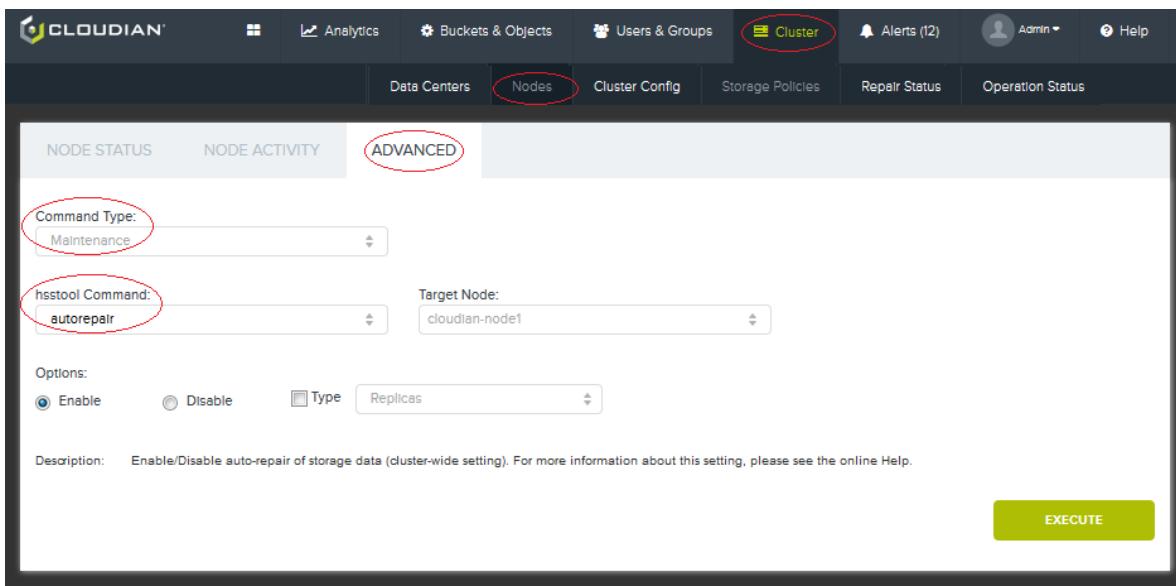
The *hsstool repairqueue* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "**hsstool repairqueue Parameters**" (page 669).

```
[root]# hsstool -h <host> repairqueue [-enable true|false] [-t replicas|ec|cassandra]
[-inc]
```

In the CMC UI you can run the *hsstool repairqueue* command's auto-repair queue status reporting function through this interface:



The function for disabling and re-enabling auto-repair has its own separate CMC interface and the command is there renamed as "autorepair" (although `hsstool repairqueue` is being invoked behind the scenes):



**Note** If you use this command to disable or re-enable auto-repair and you do not specify a repair type, then the disabling or re-enabling applies also to the "proactive repair" feature. (If you want to disable or re-enable only proactive repair without impacting the scheduled auto-repair feature see [hsstool pro-activerepair](#).)

### 11.1.13.2. Command/Response Example

The first `repairqueue` example below shows the "replicas" auto-repair queue status for a recently installed six-node cluster. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "[hsstool repairqueue Response Items](#)" (page 671).

```
[root]# hsstool -h cloudian-node1 repairqueue -t replicas

Queue: replicas Auto-repair: enabled #endpoints: 6
1: endpoint: 192.168.204.201 next repair at: Tue Mar 17 22:32:57 PDT 2015 last repair
```

```
status:
INIT interval: 43200 mins count: 0
2: endpoint: 192.168.204.202 next repair at: Tue Mar 17 22:32:58 PDT 2015 last repair
status:
INIT interval: 43200 mins count: 0
3: endpoint: 192.168.204.203 next repair at: Tue Mar 17 22:32:59 PDT 2015 last repair
status:
INIT interval: 43200 mins count: 0
4: endpoint: 192.168.204.204 next repair at: Tue Mar 17 22:33:01 PDT 2015 last repair
status:
INIT interval: 43200 mins count: 0
5: endpoint: 192.168.204.205 next repair at: Tue Mar 17 22:33:02 PDT 2015 last repair
status:
INIT interval: 43200 mins count: 0
6: endpoint: 192.168.204.206 next repair at: Tue Mar 17 22:33:03 PDT 2015 last repair
status:
INIT interval: 43200 mins count: 0
```

**Note** The response attributes for the "ec" and "cassandra" auto-repair queues would be the same as for the "replicas" queue ("next repair at", "last repair status", and so on) -- except that for the "cassandra" repair queue the response also includes a "repairScope" attribute which distinguishes between "INCREMENTAL" (for Cassandra incremental repairs) and "DEFAULT" (for Cassandra full repairs).

The next example command disables "replicas" auto-repair. Note that this disables replicated object data auto-repair for the **whole cluster**. It does not matter which node you submit the command to.

```
[root]# hsstool -h cloudian-node1 repairqueue -enable false -t replicas
Auto replicas repair disabled
```

### 11.1.13.3. Using *hsstool repairqueue*

With the *hsstool repairqueue* command you can disable (and subsequently re-enable) the HyperStore auto-repair feature. You should disable auto-repair before performing the following cluster operation:

- **"Removing a Node"** (page 390)

**IMPORTANT:** If you disable auto-repair in order to perform an operation, be sure to re-enable it afterward.

**Note** The system automatically disables the auto-repair feature when you upgrade your HyperStore software version or when you add nodes to your cluster; and the system automatically re-enables auto-repair after these operations are completed.

### 11.1.13.4. *hsstool repairqueue* Parameters

**-h <host>**

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command applies to all nodes in the specified host's service region.

**Note** In the CMC UI for this command this parameter is called "Target Node".

**-enable true|false**

(Optional) Enable or disable the auto-repair feature. By default the feature is enabled.

If you use `hsstool -h <host> repairqueue -enable false` to disable the auto-repair feature, this applies to **all nodes in the service region of the specified host**. So, it doesn't matter which host you specify in the command as long as it's in the right service region. Likewise `hsstool -h <host> repairqueue -enable true` re-enables the auto-repair feature for all nodes in the service region.

If you do not use the optional `-t` flag (described below) to specify an auto-repair type, then the disabling or re-enabling applies to **all auto-repair types and also to the proactive repair feature**. (If you want to disable or re-enable only proactive repair without impacting the scheduled auto-repair feature see "**hsstool proactiverepairq**" (page 637).)

Note that disabling the auto-repair feature **does not abort in-progress auto-repairs**. Rather, it prevents any additional scheduled auto-repairs from launching. (For information about stopping in-progress repairs, see "**hsstool repair**" (page 647) and "**hsstool repairec**" (page 658)).

**IMPORTANT:** The scheduled auto-repair feature is important for maintaining data integrity in your system. Do not leave it disabled permanently.

**Note** In the CMC UI the enable/disable option is presented as part of a **Maintenance -> autorepair** command rather than the **Info -> repairqueue** command.

**-t replicas|ec|cassandra**

(Optional) You can use this option if you want to restrict the `repairqueue` command action to a particular type of auto-repair -- `replicas` or `ec` or `cassandra`:

- In combination with the `-enable true|false` option, you can use the `-t` option to disable or re-enable just a particular type of auto-repair. For example, use `hsstool -h <host> repairqueue -enable false -t ec` to disable auto-repairs of erasure coded object data. In this example auto-repairs would continue to be enabled for replicated object data and for Cassandra metadata.

**Note:** If you do not use the optional `-t` flag to specify an auto-repair type, then the disabling or re-enabling applies to **all auto-repair types and also to the proactive repair feature**. (If you want to disable or re-enable only proactive repair without impacting the scheduled auto-repair feature see "**hsstool proactiverepairq**" (page 637).)

- Without the `-enable true|false` option, you can use the `-t` option to retrieve scheduling information for just a particular type of scheduled auto-repair. For example, use `hsstool -h <host> repairqueue -t replicas` to retrieve scheduling information for auto-repairs of replicated object data. Using `hsstool -h <host> repairqueue` by itself with no `-t` flag will retrieve scheduling information for all auto-repair types.

**-inc**

(Optional) You can use this option in combination with the `-enable true|false -t cassandra` option if you want the enabling or disabling action to apply only to Cassandra incremental auto-repair. For Cassandra, two types of

auto-repair are executed on schedule for each node: incremental repair (once a day) and full repair (once a week). For further information see "**"Auto-Repair Schedule Settings"** (page 305).

If you use the `-enable true|false -t cassandra` option without the `-inc` option then your enabling or disabling action applies to both types of Cassandra auto-repair (incremental and full).

**Note** The `-inc` option is only supported on the command line, not in the CMC.

### 11.1.13.5. *hsstool repairqueue* Response Items

When you use the `repairqueue` command to retrieve auto-repair queue information, the command results have three sections — one for each repair type. Each section consists of the following items:

#### *Queue*

Auto-repair type — either "replicas", "ec", or "cassandra"

#### *Auto-repair*

Enabled or disabled

#### *#endpoints*

Number of nodes in the cluster. Each node is separately scheduled for repair, for each repair type.

#### *<Queue position>*

This is an integer that indicates the position of this node within the cluster-wide queue for auto-repairs of this type. The node at the head of the queue has queue position "1" and is listed first in the command results.

#### *endpoint*

IP address of a node

#### *next repair at*

For each repair type, each node's *next repair at* value is determined by adding the configurable auto-repair *interval* for that repair type to the start-time of the last repair of that type done on that node. It's important to note that *next repair at* values are used to **order** the cluster-wide queues for each repair type, but the next repair of that type on that node won't necessarily start at that exact time. This is because the queue processing logic takes into account several other considerations along with the scheduled repair time.

**Note** For erasure coded (EC) object repair, the "next repair at" values are not relevant. Ignore these values. This is because auto-repair for erasure coded objects is run against just one randomly selected target host in each data center each 29-day auto-repair period (and this results in repair of all EC objects in the whole data center).

#### *last repair status*

This can be INIT (meaning no repair has been performed on that node yet), INPROGRESS, COMPLETED, TERMINATED (meaning that an in-progress repair was aborted by the operator using the "stop" option for "**"hsstool repair"** (page 647) or "**"hsstool repairec"** (page 658)), or FAILED.

If a node restart interrupts a repair, that repair job is considered FAILED and it goes to the head of the queue.

*interval*

The **configurable interval** at which this type of repair is automatically initiated on each node, in number of minutes. (Note though the qualifiers indicated in the "next repair at" description above).

*count*

The number of repairs of this type that have been executed on this node since HyperStore was installed on the node.

### 11.1.14. hsstool ring

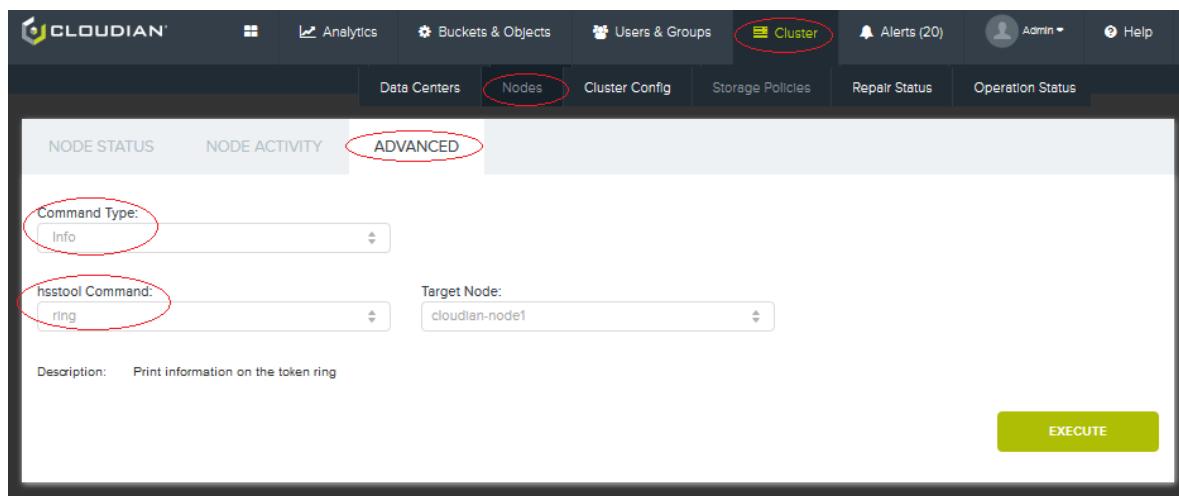
This **hsstool** command provides status information for each of the dozens or hundreds of **virtual nodes** (vNodes) in your storage cluster. It is very granular and verbose. In most circumstances you will find more value in using the **hsstool info** or **hsstool status** commands rather than **ring**.

#### 11.1.14.1. Command Format

The *hsstool ring* command line syntax is as follows.

```
[root]# hsstool -h <host> ring
```

You can also run the *hsstool ring* command through the CMC UI:



#### 11.1.14.2. Command/Response Example

The *ring* command results display a status line for each virtual node (vNode) in the storage cluster. In a typical cluster the *ring* command may return hundreds of lines of information. The returned information is sorted by ascending vNode token number.

The example below is an excerpt from a *ring* command response for a four node HyperStore system that spans two data centers. Each of the four physical nodes has 32 vNodes, so the full response has 128 data lines. The list is sorted by ascending vNode token number. Note that although the command is submitted to a particular node ("cloudian-node1"), it returns information for the whole cluster. It doesn't matter which node you submit the command to. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool ring Response Items**" (page 673).

```
[root]# hsstool -h cloudian-node1 ring
Address DC Rack Cassandra Cassandra-Load Hss State Token
```

```

105.236.130.70 DC1 RAC1 Up 2.05 MB Up Normal
2146203117201265879150333284323068618
105.236.130.70 DC1 RAC1 Up 2.05 MB Up Normal
5542328528654630725776532927863868383
105.236.218.176 DC2 RAC1 Up 1.35 MB Up Normal
12000523287982171803377514999547780254
105.236.218.176 DC2 RAC1 Up 1.35 MB Up Normal
13878365488241145156735727847865047180
198.199.106.194 DC1 RAC1 Up 1.9 MB Up Normal
18315119863185730105557340630830311535
...
...

```

### 11.1.14.3. *hsstool ring* Parameters

*-h <host>*

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**Note** In the CMC UI for this command this parameter is called "Target Node".

### 11.1.14.4. *hsstool ring* Response Items

#### *Address*

IP address of the physical node on which the vNode resides.

#### *DC*

Data center in which the vNode resides.

#### *Rack*

Rack in which the vNode resides.

#### *Cassandra*

Cassandra Service status of the vNode. Will be one of: "Up", "Down", "Joining" (in the process of joining the cluster), "Leaving" (in the process of decommissioning or being removed from the cluster), or "?" (physical host cannot be reached). All vNodes on a physical node will have the same Cassandra status.

#### *Cassandra-Load*

Cassandra load (quantity of data stored in Cassandra) for the physical host on which the vNode resides. There will be some Cassandra load even if all S3 objects are stored in the HyperStore File System or the erasure coding file system. For example, Cassandra is used for storage of object metadata and service usage data, among other things. Note that Cassandra load information is available only for the physical node as a whole; it is not available on a per-vNode basis.

#### *HSS*

HyperStore Service status for the vNode. Will be one of: "Up", "Down", or "?" (physical host cannot be reached). All vNodes on a physical node will have the same HSS status.

**State**

HyperStore Service state for the vNode. Will be one of: "Normal" or "Decommissioning". All vNodes on a given physical node will have the same HSS state.

**Token**

The vNode's token (from an integer token space ranging from 0 to  $2^{127} - 1$ ). This token is the top of the token range that constitutes the vNode. Each vNode's token range spans from the next-lower token (exclusive) in the cluster up to its own token (inclusive).

## 11.1.15. hsstool status

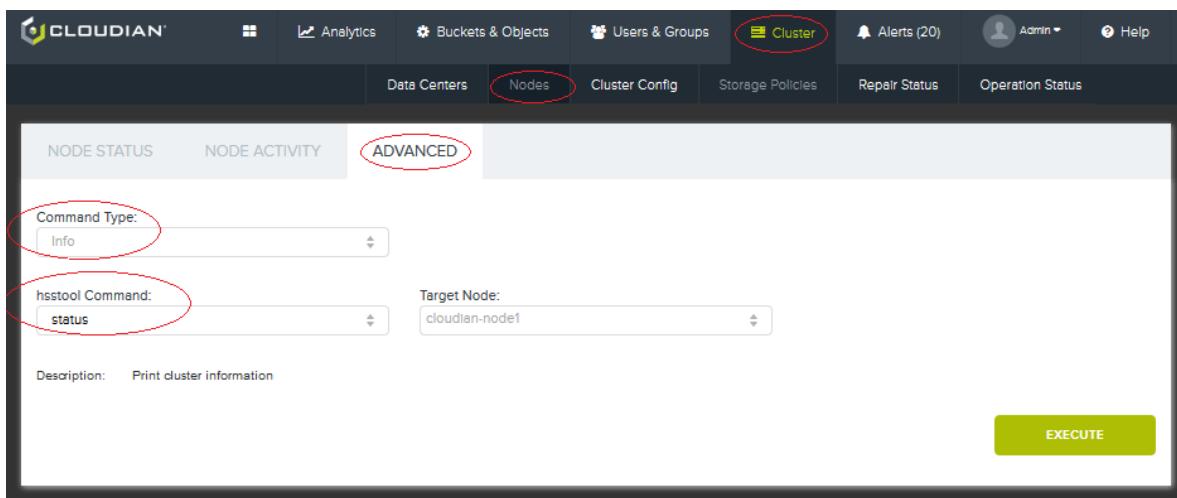
This [hsstool](#) command returns status information for a storage cluster as a whole.

### 11.1.15.1. Command Format

The *hsstool status* command line syntax is as follows.

```
[root]# hsstool -h <host> status
```

You can also run the *hsstool status* command through the CMC UI:



### 11.1.15.2. Command/Response Example

The *status* command example below retrieves the status of a four-node cluster. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "["hsstool status Response Items"](#) (page 675).

```
[root]# hsstool -h localhost status
Datacenter: DC1
=====
Address DC Rack Cassandra Tokens Cassandra-Load Hss State Hyperstore-Disk
Host-ID Hostname
10.20.2.54 DC1 RAC1 Up 32 0.24118415 Up Normal 427.66MB/7.75GB (5.39%)
cfa851df-4170-4006-a516-d5b56f99a820 cld05-04
10.20.2.57 DC1 RAC1 Down 32 0.2239672 Up Normal 79.45MB/11.17GB
(0.69%) 6bf98091-5318-4ae6-acc3-f1524f5e7da3 cld05-07
```

|                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------|
| 10.20.2.55 DC1 RAC1 Up 32 0.23458646 Up Normal 1.1GB/20.48GB (5.36%)                                                   |
| c77ff93a-5579-4f9f-b5a3-9e22d68fab84 cld05-05                                                                          |
| 10.20.2.52 DC1 RAC1 Up 32 0.3002622 Up Normal 692.7MB/12.61GB<br>(5.37%) 914117cb-3ec5-4c79-9dff-5a9f33002ce3 cld05-02 |

### 11.1.15.3. *hsstool status* Parameters

*-h <host>*

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**Note** In the CMC UI for this command this parameter is called "Target Node".

### 11.1.15.4. *hsstool status* Response Items

#### *Address*

IP address of the node.

#### *DC*

Data center in which the node resides.

#### *Rack*

Rack in which the node resides.

#### *Cassandra*

Cassandra Service status of the node. Will be one of: "Up", "Down", "Joining" (in the process of joining the cluster), "Leaving" (in the process of leaving the cluster), or "?" (host cannot be reached).

#### *Tokens*

Number of tokens (also known as vNodes) assigned to the physical node. The tokens assigned to a physical node determine which S3 objects will be stored on the node. For more information on how tokens are assigned to physical nodes, see "**How vNodes Work**" (page 32).

#### *Cassandra-Load*

From among the total token space in the storage cluster, the portion of token space that is owned by this node. This is expressed as a decimal value. For example, if 25% of the cluster's total token space is owned by this node, this field displays .25.

Note that this is a different meaning of "Cassandra-Load" than is used for *hsstool ring* and *hsstool info* results.

#### *Hss*

HyperStore Service status for the node. Will be one of: "Up", "Down", or "?" (physical host cannot be reached).

#### *State*

HyperStore Service state for the node. Will be one of: "Normal" or "Decommissioning".

#### *HyperStore-Disk*

The total volume of S3 object data stored in the HyperStore File System on the node. This includes S3 objects

for which the node serves as a secondary or tertiary replica as well as S3 objects for which the node is the primary replica.

This field also shows the total amount of disk space designated for S3 object storage on the node.

#### *Host-ID*

System-generated hexadecimal number uniquely identifying the physical node. Note that with the use of vNodes, tokens uniquely identify vNodes while Host IDs uniquely identify each physical node within the cluster.

#### *Hostname*

Hostname of the node.

### 11.1.16. hsstool trmap

#### *[Command]*

This [hsstool](#) command returns a list of token range map snapshot IDs along with information about each snapshot such as the snapshot creation time. You can also use the command to return the contents of a specified token range map snapshot.

The system creates a token range map snapshot each time you [add a new node to your cluster](#). The token range map identifies, for each storage policy in your system, the nodes (endpoints) that store data from each token range. The data from each token range will be stored on multiple nodes, with the number of nodes depending on the storage policy (for example, in a 3X replication storage policy each token range would be mapped to three storage endpoints). When you've added new nodes to your cluster, the system uses token range maps to manage the rebalancing of S3 object data from existing nodes to the new nodes.

**Typically you should not need to use this command** unless you are working with Cloudian Support to troubleshoot a failed attempt to add nodes to your HyperStore cluster or a failed attempt to rebalance the cluster after adding nodes.

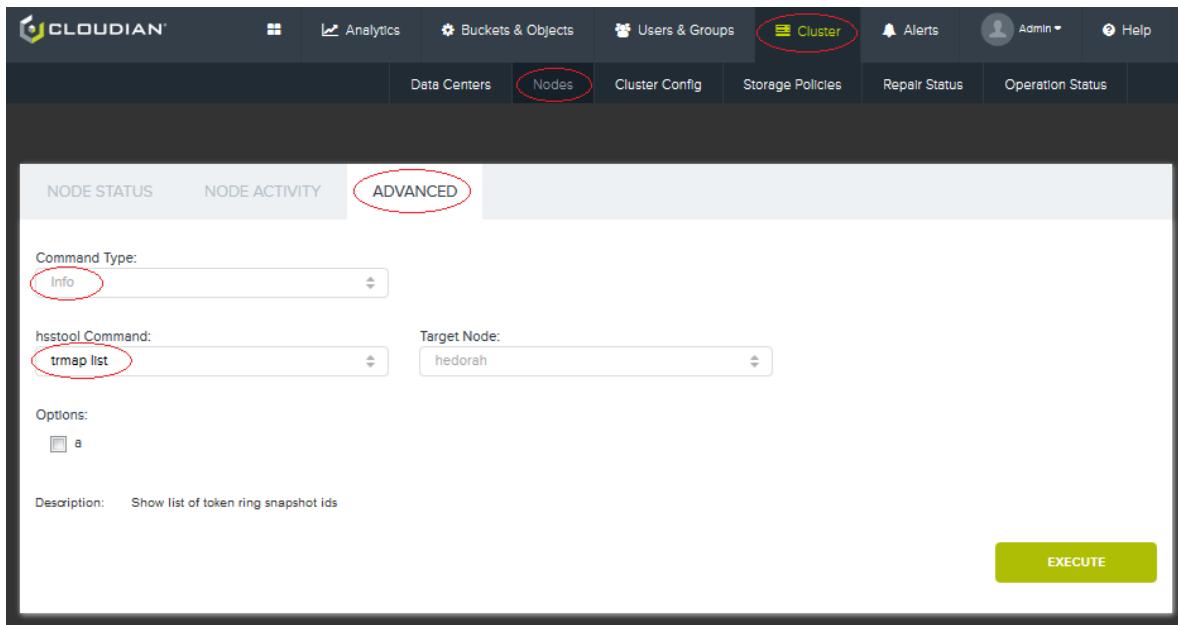
#### 11.1.16.1. Command Format

The *hsstool trmap* command line syntax is as follows. For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[hsstool trmap Parameters](#)" (page 678).

```
[root]# /opt/cloudian/bin/hsstool -h <host> trmap [list \[-a\]] [show <snapshotId>]
[set <snapshotId> <active/disabled>] [delete <snapshotId>]
```

**IMPORTANT:** Do not use the *set* or *delete* options unless instructed to do so by Cloudian Support.

You can also run the *hsstool trmap* command through the CMC UI:



### 11.1.16.2. Command/Response Examples

In this first example a list of active token range map snapshot IDs is retrieved. These are token range map snapshots for which the associated rebalancing operation has not yet completed.

```
[root]# /opt/cloudian/bin/hsstool -h localhost trmap list
061e9190-6d62-429d-85dc-e4baec07f49e Wed Jun 19 15:26:07 EDT 2019
22351e62-e921-4130-afaf-1ca3183046e1 Thu Jun 20 14:17:04 EDT 2019
1c84d57e-dccb-47b0-af6d-015e823eab07 Wed Jun 19 21:42:42 EDT 2019
9e096b8f-5914-42fc-8dc7-063de85deb4a Thu Jun 20 06:56:17 EDT 2019
```

In this next example, a token range map snapshot is retrieved (this is from a different system and is not one of the snapshots listed in the first example). The map is in JSON format. The response is very large, and is truncated below. In practice, if you use this command you should redirect the output to a text file. For description of a particular response item, click on the response item; or for the full list of response item descriptions see **"hsstool trmap Response Items"** (page 679).

```
[root]# /opt/cloudian/bin/hsstool -h cloudian-node1 trmap show fbcdf5be-7c15-40a4-be59-
955df70e7fb2
{
 "id" : "fbcdf5be-7c15-40a4-be59-955df70e7fb2",
 "version" : "1",
 "timestamp" : 1477266709616,
 "rebalance" : "REQUIRED",
 "policies" : {
 "5871e62dae4ccfd618112ca3403b3251" : {
 "policyId" : "5871e62dae4ccfd618112ca3403b3251",
 "keyspaceName" : "UserData_5871e62dae4ccfd618112ca3403b3251",
 "replicationScheme" : {
 "DC2" : "1",
 "DC1" : "1",
 "DC3" : "1"
 }
 }
 }
}
```

```

},
"ecScheme" : null,
"ecMap" : null,
"replicasMap" : [{
 "left" : 163766745962987615139826681359528839019,
 "right" : 164582023203604297970082254372849331112,
 "endPointDetails" : [{
 "endpoint" : "10.10.10.114",
 "datacenter" : "DC3",
 "rack" : "RAC1"
 }, {
 "endpoint" : "10.10.10.115",
 "datacenter" : "DC2",
 "rack" : "RAC1"
 }, {
 "endpoint" : "10.10.10.111",
 "datacenter" : "DC1",
 "rack" : "RAC1"
 }]
}, {
 "left" : 6341660663762096831290541188712444913,
 "right" : 6449737778660216877727472971404857143,
 "endPointDetails" : [{
 ...
 ...
 }]
}
]

```

**See Also:**

- **"How vNodes Work"** (page 32)

**11.1.16.3. *hsstool trmap* Parameters****-h <host>**

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**Note** In the CMC UI for this command this parameter is called "Target Node".

***list [-a]***

(Optional) Using *list* **without** the optional *-a* flag returns only a list of active token range map snapshot IDs. These are token range map snapshots for which the associated rebalancing operation has not yet completed.

If you use the *-a* flag -- that is, *trmap list -a* -- then the command returns the IDs of **all** snapshots -- the active snapshots and also the disabled snapshots (snapshots for which the associated rebalancing operation has completed). When you use the *-a* flag the return includes a status field for each snapshot, to distinguish active snapshots from disabled snapshots. For example:

```
[root]# /opt/cloudian/bin/hsstool -h localhost trmap list -a
59af5410-b303-41bd-94a0-31ce92058e11 Tue Jun 26 09:03:56 EDT 2018 DISABLED
061e9190-6d62-429d-85dc-e4baec07f49e Wed Jun 19 15:26:07 EDT 2019 ACTIVE
```

```
22351e62-e921-4130-afaf-1ca3183046e1 Thu Jun 20 14:17:04 EDT 2019 ACTIVE
1c84d57e-dccb-47b0-af6d-015e823eab07 Wed Jun 19 21:42:42 EDT 2019 ACTIVE
9e096b8f-5914-42fc-8dc7-063de85deb4a Thu Jun 20 06:56:17 EDT 2019 ACTIVE
3bf72d7e-a2a1-4551-972c-1223a4485335 Mon Jun 25 17:23:32 EDT 2018 DISABLED
dbf98852-2557-4cf8-8708-0398009fc6d7 Mon Jun 25 18:38:47 EDT 2018 DISABLED
```

`show <snapshotId>`

(Optional) This returns the content of the specified token range map snapshot. This option is only supported on the command line, not in the CMC interface.

`set <snapshotId> <active/disabled>`

(Optional) **Do not use this option unless instructed to do so by Cloudian Support.** This sets the status of the specified token range map snapshot to *active* (rebalancing in connection with this snapshot still needs to be completed) or *disabled* (rebalancing in connection with this snapshot has been completed). This option is only supported on the command line, not in the CMC interface.

`delete <snapshotId>`

(Optional) **Do not use this option unless instructed to do so by Cloudian Support.** This deletes the specified token range map snapshot. This option is only supported on the command line, not in the CMC interface.

#### 11.1.16.4. *hsstool trmap* Response Items

*id*

System-generated unique identifier of this token range map snapshot.

*version*

Version of the token range map snapshot. This integer is incremented each time a new snapshot is created.

*timestamp*

Timestamp indicating when the token range map snapshot was created.

*rebalance*

Status of the *hsstool rebalance* operation in regard to this token range map snapshot, such as REQUIRED or COMPLETED. Each time you add a node to your system (using the CMC's function for adding a node, in the **Data Centers** page), the system automatically generates a token range snapshot. After adding a node, you then must run *hsstool rebalance* on the new node (which you can do from the CMC's **Nodes Advanced** page). The rebalance operation utilizes the token range map snapshot. For complete instructions on adding nodes, see "**Adding Nodes**" (page 369).

*policies*

This marks the beginning of the per-policy token range map information. The token range map will have separate token range map information for each of your storage policies.

*policyId*

System-generated unique identifier of a storage policy. Note that this ID appears three times: at the outset of the policy block, then again as the *policyId* attribute, then again within the *keyspaceName*.

*keyspaceName*

Name of the Cassandra keyspace in which object metadata is stored for this storage policy. The name is in

*format* *UserData\_<policyId>*.

*replicationScheme*

Specification of the replication scheme, if this policy block is for a replication storage policy. In the example the policy's replication scheme calls for one replica in each of three data centers.

*ecScheme*

Specification of the erasure coding scheme, if this policy block is for an erasure coding storage policy. In the example, the policy block is for a replication policy so the *ecScheme* value is null.

*ecMap*

Content of the token range map for the policy scheme, if this policy block is for an erasure coding storage policy. In the example, the policy block is for a replication policy so the *ecMap* value is null.

*replicasMap*

Content of the token range map for the policy scheme, if this policy block is for a replication storage policy. The map consists of lists of endpoints per token range.

*left*

Token at the low end of the token range (exclusive). This is from the consistent hashing space of 0 to  $2^{127}$  from which HyperStore generates tokens for the purpose of allocating data across the cluster.

*right*

Token at the high end of the token range (inclusive). This is from the consistent hashing space of 0 to  $2^{127}$  from which HyperStore generates tokens for the purpose of allocating data across the cluster.

*endPointDetails*

Endpoint mapping information for this particular token range, for this storage policy. This is a list of endpoints (nodes), with each endpoint identified by IP address as well as data center name and rack name. The number of endpoints per token range will depend on the storage policy scheme. In the example the policy is a 3X replication policy, so there are three endpoints listed for each token range. Objects for which the object token (based on a hash of the bucket name / object name combination) falls into this token range will have replicas placed on each of these nodes.

## 11.1.17. hsstool whereis

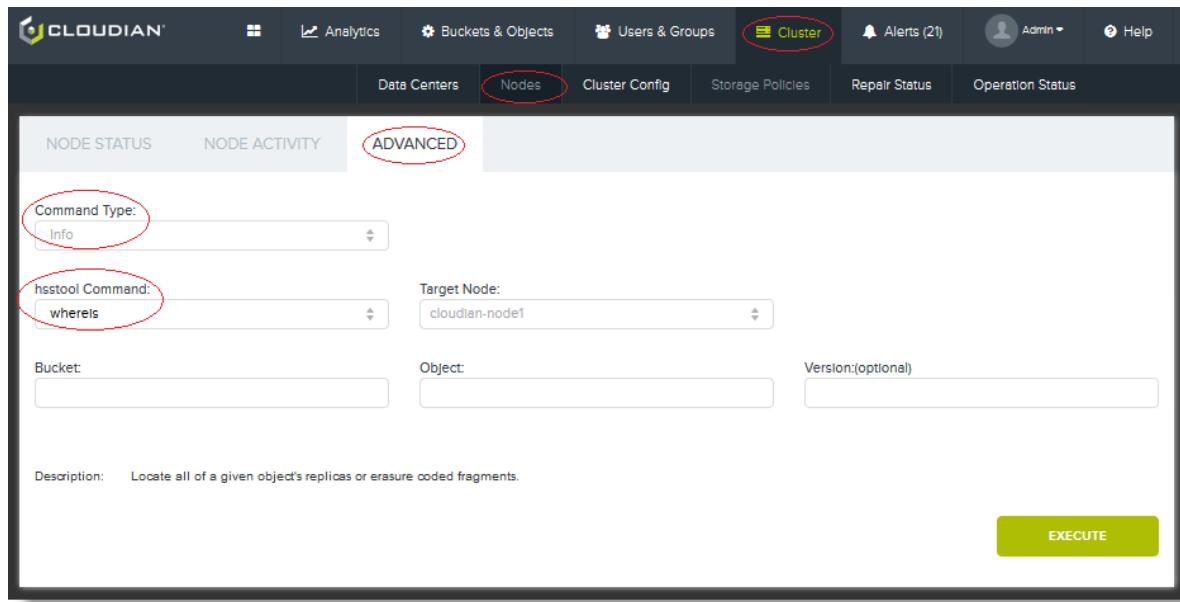
This [hsstool](#) command returns the current storage location of each replica of a specified S3 object (or in the case of erasure coded objects, the location of each of the object's fragments). The command response also shows the specified object's metadata such as last modified timestamp and object digest.

### 11.1.17.1. Command Format

The *hsstool whereis* command line syntax is as follows. For description of a particular parameter, click on the parameter; or for the full list of parameter descriptions see "[hsstool whereis Parameters](#)" (page 682).

```
[root]# hsstool -h <host> whereis <bucket>/<object> [-v <version>] [-a]
```

You also can run the *hsstool whereis* command through the CMC UI:



### 11.1.17.2. Command/Response Examples

The first `whereis` command example below retrieves location information for an object named "Guide.pdf". This is from a single-node HyperStore system, so there is just one replica of the object. The detail information for the object replica is truncated in this example. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool whereis Response Items**" (page 683).

```
[root]# hsstool -h sirius whereis bucket2/Guide.pdf
Key: bucket2/Guide.pdf
Policy ID: c4a276180b0c99346e2285946f60e59c
Version: null
Compression: NONE
Create Time: 2017-02-20T16:38:09.783Z
Last Modified: 2017-02-20T16:38:09.783Z
Last Access Time: 2017-02-20T16:38:10.273Z
Size: 7428524
Type: REPLICAS
Region: region1
[DC1 10.50.10.21 sirius] bucket2/Guide.pdf file://10.50.10.21:/var/lib/cloudian/hsfs/1L1t...
```

The second `whereis` command example retrieves location information for an object named "obj1.txt". This is from a two data center HyperStore system, using replicated 2+1 erasure coding (a configuration that is no longer supported). Note that the response indicates that one of the expected six fragments is missing. The detail information for each found fragment is truncated in this example. For description of a particular response item, click on the response item; or for the full list of response item descriptions see "**hsstool whereis Response Items**" (page 683).

```
[root]# hsstool -h cloudian-node3 whereis bucket1/obj1.txt
Key: bucket1/obj1
Policy ID: 1d140a90b17285cf2d1502cbd424d621
Version: null
Compression: NONE
Create Time: 2018-01-29T23:07:52.626Z
Last Modified: 2018-01-29T23:07:52.626Z
```

```

Last Access Time: 2018-01-29T23:07:52.626Z
Size: 11231
Type: EC
Region: region1
5 out of 6 fragments were found, 1 fragments missing
[DC1 2 10.100.186.47 cloudian-node3] bucket1/obj1.txt
ec://10.100.186.47:/var/lib/cloudian/ec/...
[DC1 3 10.100.76.90 cloudian-node4] bucket1/obj1.txt ec://10.100.76.90:/var/lib/cloudian/ec/...
[DC2 1 10.100.61.17 cloudian-node6] bucket1/obj1.txt
ec://10.100.61.17:/var/lib/cloudian/ec/...
[DC2 2 10.100.80.35 cloudian-node7] bucket1/obj1.txt ec://10.100.80.35:/var/lib/cloudian/ec/...
[DC2 3 10.100.218.23 cloudian-node8] bucket1/obj1.txt
ec://10.100.218.23:/var/lib/cloudian/ec/...
[DC1 1 10.100.186.42 cloudian-node1] 0 replication found

```

**Note** For objects for which the Type is "TRANSITIONED" (auto-tiered), the response will also include a [URL](#) field.

### 11.1.17.3. Log File for "whereis -a"

When you use the *whereis -a* command, information about all replicas and erasure coded fragments of all objects in the entire service region is written to a log file. The log file is written on the HyperStore host that you connect to when you run *whereis -a*, and by default the log file path is:

```
/var/log/cloudian/whereis.log
```

In the *whereis.log* file, the start and completion of the output from a single run of *whereis -a* is marked by "START" and "END" timestamps. Within those timestamps, the output is organized by user. The start of output for a particular user is marked by "#user:<canonical UID>". This line is then followed by lines for the user's buckets and objects, with the same object detail information as described in the *whereis* command results documentation above. Users who do not have any buckets will not be included in the log file.

The output of multiple runs of *whereis -a* may be written to the same log file, depending on the size of the output. Because the output of *whereis -a* may be very large, it's also possible that the output of a single run may be spread across multiple log files, if maximum file size is reached and log rotation occurs.

By default this log is rotated if it reaches 10MB in size or at the end of the day, whichever occurs first. The oldest rotated *whereis* log file is automatically deleted if it reaches 180 days in age or if the aggregate size of all rotated *whereis* log files (after compression) reaches 100MB. These rotation settings are configurable in the *RollingRandomAccessFile name="APP"* section of the */etc/cloudian-<version>-puppet/modules/cloudians3/templates/log4j-hsstool.xml.erb* file. For information about changing these settings see "[Log Configuration Settings](#)" (page 593).

### 11.1.17.4. *hsstool whereis* Parameters

**-h <host>**

(Mandatory) You can specify the hostname or IP address of any node in the cluster. The command retrieves cluster-wide information that is available from any node that belongs to the cluster.

**Note** In the CMC UI for this command this parameter is called "Target Node".

<bucket>/<object>

(Mandatory unless using the -a option) Bucket name, followed by a forward slash, followed by the full object name (including "folder path", if any). For example, *mybucket/file1.txt* or *mybucket/Videos/Vacation/Italy\_2016-06-27.mpg*.

If the object name has spaces in it, enclose the *bucket/object* name pair in quotes. For example, "*mybucket/big document.doc*".

The *bucket/object* name is case-sensitive.

**Note** In the CMC UI implementation of this command, you enter the bucket name and the full object name (including folder path) in separate fields. For example, bucket name *mybucket* and full object name *Videos/Vacation/Italy\_2016-06-27.mpg*.

-v <version>

(Optional) If versioning is enabled on the bucket that contains the object, you can optionally specify the version ID of a particular version of the object. Version IDs are system-generated hexadecimal values (for example, *fe1be647-5f3b-e87f-b433-180373cf31f5*). If versioning has been used for the object but you do not specify a version ID, location information will be retrieved for the most recent version of the object.

-a

(Optional) Use the -a option if you want a full list of **all replicas of all objects in the entire service region**. This information will be written to a log file. For more information on the log file, see the section that follows the command/response example.

If you use the -a option do not use the <bucket/object> parameter or the -v <version> parameter. Run the command simply as *hsstool -h <host> whereis -a*

**Note** The CMC does not support the -a option. To use this option you need to use *hsstool whereis* on the command line.

### 11.1.17.5. *hsstool whereis* Response Items

#### Key

Key that uniquely identifies the S3 object, in format <bucketname>/<objectname>. For example, *buck-1/Documents/Meetings\_2018-06-27.docx*.

#### PolicyID

System-generated identifier of the storage policy that applies to the bucket in which this object is stored.

#### Version

Object version, if versioning has been used for the object. Versions are identified by timeuuid values in hexadecimal format. If versioning has not been used for the object, the Version field displays "null".

#### Compression

Type of server-side compression applied to the object, if any. Possible values are NONE, SNAPPY, ZLIB, or LZ4. The type of compression applied depends on the storage policy used by the bucket. Each storage policy has its own configuration as to whether compression is used and the compression type.

*Create Time*

Timestamp for the original creation of the object. Format is ISO 8601 and the time is in Coordinated Universal Time (UTC).

*Last Modified*

Timestamp for last modification of the object. Format is ISO 8601 and time is in UTC.

*Last Access Time*

Timestamp for last access of the object. An object's Last Access Time is updated if the object is accessed either for retrieval (GET or HEAD) or modification (PUT/POST/Copy). Format is ISO 8601 and time is in UTC.

*Size*

The object's size in bytes.

*Type*

One of:

- REPLICAS — The object is replicated in the HyperStore File System (HSFS).
- EC — The object is erasure coded in the HSFS.
- TRANSITIONED — The object has been transitioned ([auto-tiered](#)) to a different storage system such as Amazon S3.

*Region*

The HyperStore service region in which the object is stored.

*URL*

This field appears only in the case of TRANSITIONED objects. For such objects, this field shows the URL that identifies the location of the object in the tiering destination system. For example:

```
http://s3.amazonaws.com/bucket2.mdazyjgxnxndu2ody4mji1nty3/notes.txt
```

In this example, the tiering destination is Amazon S3; the bucket name in the destination system is *buck-  
et2.mdazyjgxnxndu2ody4mji1nty3* (which is the HyperStore source bucket name — *bucket2* in this case — appended by a 28 character random string); and the object name is *notes.txt*. Note that the URL field will specify the transfer protocol as *http*, whereas to actually access the object in the destination system the protocol would typically be *https*.

*Location detail*

For objects stored locally (objects that are not of type TRANSITIONED), the lower part of the response shows the location of **each object replica** (for replicated objects) or of **each erasure coded object fragment** (for EC objects).

For HSFS replicated objects each location is specified as:

```
[<datacenter> <IP-address> <hostname>] <bucket>/<objectname>
file://<IP-address>:<mountpoint>/hsfs/<base62-encoded-vNode-token>/<policyid>/
<000-255>/<000-255>/<filename> <last-modified> <version> <digest>
```

For erasure coded object fragments each location is specified as:

```
[<datacenter> <key-suffix-digit> <IP-address> <hostname>] <bucket>/<objectname>
ec://<IP_address>:<mountpoint>/ec/<base62-encoded-vNode-token>/<policyid>/
<000-255>/<000-255>/<filename> <last-modified> <version> <digest>
```

- The *<base62-encoded-vNode-token>* is a base-62 encoding of the token belonging to the vNode to which the object instance or fragment is assigned (for background information see "**How vNodes Work**" (page 32)).
- The *<policyid>* segment is the unique identifier of the storage policy applied to the bucket in which the object is stored.
- The two *<000-255>* segments of the path are based on a hash of the *<filename>*, normalized to a 255\*255 number.
- The *<filename>* is a dot-separated concatenation of the object's hash token and the object's Last Modified Time timestamp. The timestamp is formatted as *<UnixTimeMillis><6digitAtomicCounter>-<nodeIPaddrHex>* (the last element is the IP address -- in hexadecimal format -- of the S3 Service node that processed the object upload request). Note: For objects last modified prior to HyperStore version 6.1, the timestamp is simply Unix time in milliseconds. This was the timestamp format used in HyperStore 6.0.x and older..
- For EC objects only, the *<key\_suffix\_digit>* at the beginning of each location is a digit that the system generates and uses to ensure that each fragment goes to a different node.

**Note** If an object replica or fragment is supposed to be on a node (according to system metadata) but is missing, the node's IP address is listed in the command results along with a message stating "0 replication found". For example, "[10.10.3.52] 0 replication found".

**Note** For multipart objects (large objects uploaded via the S3 multipart upload method), storage location detail is shown **for each part**.

#### Fragment count summary

For erasure coded objects, the *hsstool whereis* response includes a line stating the number of fragments found and (if applicable) the number of fragments missing. For objects for which all expected fragments were found, the line will state "x out of x fragments were found". For objects for which one or more of the expected fragments are missing, the line will state "x out of y fragments were found, z fragments missing".

## 11.2. Redis Monitor Commands

The HyperStore "**Redis Monitor Service**" (page 17) monitors Redis Credentials and Redis QOS cluster health and implements automatic failover of the Redis master node role in each cluster. The Redis Monitor runs on two nodes, with one instance being the primary and the other being the backup. When you submit Redis Monitor commands **you must submit them to the primary node**, not the backup. To check which of your nodes is running the primary Redis Monitor, go to the CMC's [Cluster Information](#) page.

You can submit commands to the Redis Monitor primary host through the Redis Monitor CLI or through a JMX client like JConsole. A couple of the more useful commands can also be executed through the CMC.

**To initiate a CLI session with the Redis Monitor**, use *netcat* to connect to port 9078 on the node on which the primary Redis Monitor is running:

```
$ nc <redismon_primary_host> 9078
```

Specify the hostname or IP address (do not use 'localhost'). Once connected, you can then use any of the Redis Monitor CLI commands listed below. When you want to terminate your Redis Monitor CLI session, enter **quit** on the command line. (To subsequently end your *netcat* session and return to the terminal prompt, enter **<ctrl>-D**).

**To access Redis Monitor commands in the CMC**, go to the [Node Advanced](#) page and from the "Command Type" drop-down list select "Redis Monitor Operations".

**To submit commands to Redis Monitor through JConsole**, first launch JConsole on any of your HyperStore nodes. JConsole comes bundled with the Java platform that HyperStore uses. By default the full path to the JConsole executable is */usr/java/latest/bin/jconsole*.

Using the JConsole GUI, connect to the Redis Monitor primary host at port 19083. Once connected, in the JConsole GUI change to the MBeans tab. The domain under which the relevant MBeans appear is *com.gemini.cloudian.util*. Under this domain select *RedisCluster*. Under *RedisCluster* you will see an MBean for each of your Redis clusters: one global Redis Credentials cluster (named *redis.credentials*) and one Redis QoS cluster for each of your service regions (each named *redis.qos.<regionName>*). After selecting a cluster-specific MBean you can execute the Redis Monitor actions listed below.

## 11.2.1. Redis Monitor Commands

### 11.2.1.1. get cluster

#### get cluster

Use this operation to retrieve basic status information that the Redis Monitor currently has for a specified Redis cluster. The cluster status information includes:

- The identity of the Redis master node within the cluster
- Whether monitoring of the cluster by the Redis Monitor is enabled
- Whether the sending of cluster status notifications to clients of the cluster is enabled
- A list of cluster member nodes, with an indication of the status (UP/DOWN) of each node
- A list of cluster clients (which write to and/or read from this Redis database), with an indication of the status (UP/DOWN) of each client. The clients will include S3 Service nodes (identified by JMX listening socket *<host>:19080*), Admin Service nodes (*<host>:19081*), and HyperStore Service nodes (*<host>:19082*). These are the clients to which the Redis Monitor sends notifications regarding the Redis cluster's status.
- "state" information that includes a timestamp indicating the last date and time that the master role status was updated (if there have been any fail-overs of the master role, the timestamp is the time of the last fail-over -- otherwise, it's the time of the last start-up of the Redis Monitor)

#### Command Line Syntax

```
get cluster redis.credentials|redis.qos.<region>
```

**Note** For this and all other Redis Monitor commands, the Redis QOS cluster identifier includes the name of the service region in which the cluster resides. This is necessary since in a multi-region Hyper-

Store system each region has its own Redis QoS cluster. By contrast the Redis Credentials cluster, since it is global (extending across all service regions), does not include a region name in its identifier.

**Example:**

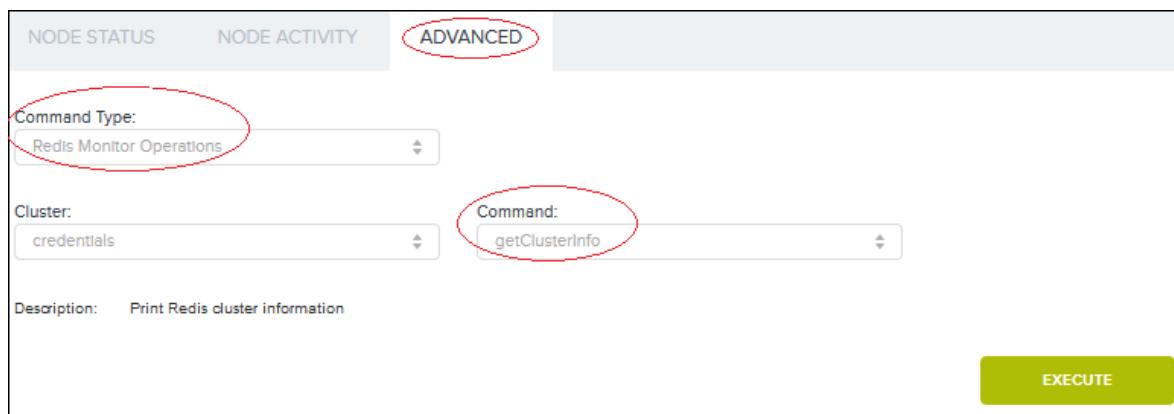
```
$ nc 10.50.20.12 9078
get cluster redis.credentials
OK master: store1(10.50.20.1):6379, monitoring: enabled, notifications: enabled
nodes: [[store1(10.50.20.1):6379,UP,master], [store4(10.50.20.4):6379,UP,slave],
[store5(10.50.20.5):6379,UP,slave]]
clients: [[store1(10.50.20.1):19080,UP,store1], [store2(10.50.20.2):19080,UP,store1],
[store3(10.50.20.3):19080,UP,store1], [store4(10.50.20.4):19080,UP,store1],
[store5(10.50.20.5):19080,UP,store1], [store6(10.50.20.6):19080,UP,store1],
[store1(10.50.20.1):19081,UP,store1], [store2(10.50.20.2):19081,UP,store1],
[store3(10.50.20.3):19081,UP,store1], [store4(10.50.20.4):19081,UP,store1],
[store5(10.50.20.5):19081,UP,store1], [store6(10.50.20.6):19081,UP,store1],
[store1(10.50.20.1):19082,UP,store1], [store2(10.50.20.2):19082,UP,store1],
[store3(10.50.20.3):19082,UP,store1], [store4(10.50.20.4):19082,UP,store1],
[store5(10.50.20.5):19082,UP,store1], [store6(10.50.20.6):19082,UP,store1]]
state: redis.credentials: master= 10.50.20.1 updatetime= Fri Sep 21 16:17:23 PDT 2018
```

## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Attributes → Cluster

## CMC UI

Path: Cluster → Nodes → Advanced



### 11.2.1.2. get master

#### get master

Use this operation to retrieve from the Redis Monitor the identity of the current master node within a specified Redis cluster.

#### Command Line Syntax

```
get master redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
get master redis.qos.region1
OK store2(10.50.20.2):6380
```

## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Attributes → ClusterMaster

### 11.2.1.3. get nodes

#### get nodes

Use this operation to retrieve from the Redis Monitor a list of all current members of a specified Redis cluster. The command response also indicates the status (UP/DOWN) of each member node.

#### Command Line Syntax

```
get nodes redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
get nodes redis.credentials
OK [[[store1(10.50.20.1):6379,UP,master], [store4(10.50.20.4):6379,UP,slave],
[store5(10.50.20.5):6379,UP,slave]]]
```

## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Attributes → ClusterNodes

### 11.2.1.4. get clients

#### get clients

Use this operation to retrieve from the Redis Monitor a list of clients of a specified Redis cluster (client nodes that write to and/or read from the Redis database). These are the clients to which the Redis Monitor sends notifications regarding the Redis cluster's status. For example, if the cluster's master role changes from one node to another, the Redis Monitor will notify these clients of the change.

The clients will include S3 Service nodes (identified by JMX listening socket <host>:19080), Admin Service nodes (<host>:19081), and HyperStore Service nodes (<host>:19082). The command response also indicates the status (UP/DOWN) of each client, and for each client it shows which node the client thinks is the Redis master.

#### Command Line Syntax

```
get clients redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
get clients redis.credentials
OK [[[store1(10.50.20.1):19080,UP,store1], [store2(10.50.20.2):19080,UP,store1],
```

```
[store3(10.50.20.3):19080,UP,store1], [store4(10.50.20.4):19080,UP,store1],
[store5(10.50.20.5):19080,UP,store1], [store6(10.50.20.6):19080,UP,store1],
[store1(10.50.20.1):19081,UP,store1], [store2(10.50.20.2):19081,UP,store1],
[store3(10.50.20.3):19081,UP,store1], [store4(10.50.20.4):19081,UP,store1],
[store5(10.50.20.5):19081,UP,store1], [store6(10.50.20.6):19081,UP,store1],
[store1(10.50.20.1):19082,UP,store1], [store2(10.50.20.2):19082,UP,store1],
[store3(10.50.20.3):19082,UP,store1], [store4(10.50.20.4):19082,UP,store1],
[store5(10.50.20.5):19082,UP,store1], [store6(10.50.20.6):19082,UP,store1]]]
```

In the above example, "store1" is the current Redis Credentials master node. All the clients correctly have this information.

### Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Attributes → ClusterClients

#### 11.2.1.5. enable monitoring

##### enable monitoring

Use this operation to enable monitoring of a specified Redis cluster by the Redis Monitor.

**Note** Monitoring is enabled by default. This operation is relevant only if you have previously disabled monitoring.

### Command Line Syntax

```
enable monitoring redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
enable monitoring redis.credentials
OK enabled
```

### Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → EnableClusterMonitoring

#### 11.2.1.6. disable monitoring

##### disable monitoring

Use this operation if you want to temporarily disable Redis Monitor's monitoring of a specified Redis cluster — for example if you are performing maintenance work on the Redis cluster. (You can subsequently use the *enable monitor* command re-enable monitoring of that cluster.)

### Command Line Syntax

```
disable monitoring redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
disable monitoring redis.qos.region1
OK disabled
```

### Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → disableClusterMonitoring

#### 11.2.1.7. enable notifications

##### enable notifications

Use this operation to enable Redis Monitor's sending of notifications to the clients of a specified Redis cluster. (The clients are the S3 Service nodes, Admin Service nodes, and HyperStore Service nodes that write to and/or read from that Redis cluster).

The Redis Monitor sends notifications to inform clients of the identity of the Redis cluster's master node, in either of these circumstances:

- The Redis master role has switched from one host to another. (This could happen if the original master goes down and Redis Monitor detects this and fails the master role over to one of the slave nodes; or if an operator uses the Redis Monitor CLI to move the master role from one node to another).
- The Redis Monitor in its regular polling of cluster clients' status detects that one of the clients has incorrect information about the identity of the Redis cluster master node. In this case the Redis Monitor notifies the client to give it the correct information.

**Note** Notifications are enabled by default. This operation is relevant only if you have previously disabled notifications using the *disable notifications* command.

### Command Line Syntax

```
enable notifications redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
enable notifications redis.credentials
OK enabled
```

### Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → enableClusterNotifications

#### 11.2.1.8. disable notifications

##### disable notifications

Use this operation to temporarily disable Redis Monitor's sending of Redis cluster status notifications to the clients of that cluster. For more information on the notification feature see "**enable notifications**" (page 690).

### Command Line Syntax

```
disable notifications redis.credentials|redis.qos.<region>
```

Example:

```
$ nc 10.50.20.12 9078
disable notifications redis.qos.region1
OK disabled
```

## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → disableClusterNotifications

### 11.2.1.9. set master

#### set master

Use this operation to assign the Redis master role to a different node within a specified Redis cluster. **The node to which you assign the master role must be one of the current slaves within the same Redis cluster.**

The Redis master node within a cluster is the node to which Redis clients submit writes. The writes are asynchronously replicated to the slave(s) within that cluster. Redis clients read from the slave(s).

An example of when you would move the Redis master role is if you want to remove the current Redis master host from your cluster.

Using this command is part of a broader procedure for moving a Redis master role to a slave. For the full procedure including the use of this command within the procedure, see "**Move the Redis Credentials Master or QoS Master Role**" (page 405).

#### Command Line Syntax

```
set master redis.credentials|redis.qos.<region> [<host:redisPort>]
```

Example:

```
$ nc 10.50.20.12 9078
set master redis.credentials store5:6379
OK set new master store5(10.50.20.5):6379
```

**Note** If you do not specify a <host:redisPort> value, the Redis Monitor chooses a slave node at random (from within the cluster) to elevate to the master role.

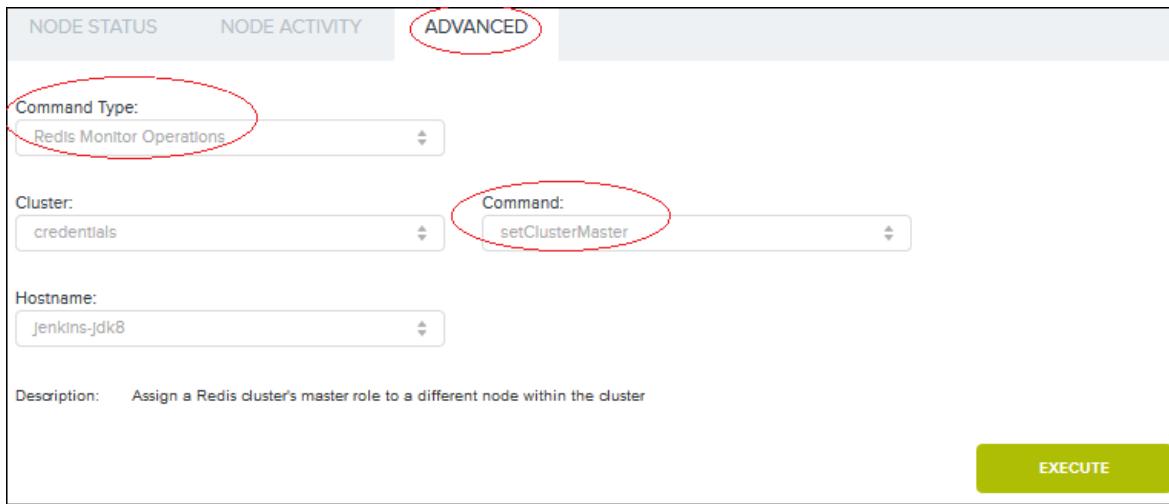
## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → setClusterMaster

(set "p1" to the node's hostname or IP address and "p2" to 6379 for Redis Credentials or 6380 for RedisQoS)

## CMC UI

Path: **Cluster** → **Nodes** → **Advanced**



In the CMC UI, use the "Hostname" field to specify the host to which you want to move the Redis master role.

#### 11.2.1.10. add node

##### **add node**

Use this operation to add a Redis node to the list of nodes that Redis Monitor is monitoring, for a specified Redis cluster. This would be if you have used the installer (`cloudianInstall.sh` in the installation directory on your Puppet master node) to activate Redis on a HyperStore node that wasn't previously running Redis. In this circumstances you have two options to make Redis Monitor aware of the new member of the Redis cluster:

- Restart Redis Monitor
- OR
- Use this command.

##### **Command Line Syntax**

```
add node redis.credentials|redis.qos.<region> <host:redisPort>
```

Example:

```
$ nc 10.50.20.12 9078
add node redis.qos.region1 store3:6380
OK added node store3(10.50.20.3):6380 to redis
```

##### **Path in JConsole**

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → addClusterNode

(set "p1" to the node's hostname or IP address and "p2" to 6379 for Redis Credentials or 6380 for RedisQoS)

#### 11.2.1.11. add client

##### **add client**

Use this operation to add a new S3 Service, Admin Service, and/or HyperStore Service node to the list of

clients to which Redis Monitor will send notifications regarding the status of a specified Redis cluster.

In normal circumstances you should not have to use this command. If you add a new node to your HyperStore cluster (as described in **"Adding Nodes"** (page 369)), the system automatically makes Redis Monitor aware of the new clients of the Redis Credentials and Redis QoS clusters.

If you do use this command, add only one client per command run. A client is identified by its JMX socket (for example "vm12:19080" for an S3 Service instance running on host vm12, or "vm12:19081" for an Admin Service instance running on host vm12).

## Command Line Syntax

```
add client redis.credentials|redis.qos.<region> <host:JMXport>
```

Example:

```
$ nc 10.50.20.12 9078
add client redis.credentials store7:19080
OK added client store7(10.50.20.7):19080 to redis
```

## Path in JConsole

com.gemini.cloudian.util → RedisCluster → <clusterMBean> → Operations → addClusterClient

(set "p1" to the node's hostname or IP address and "p2" to 19080 for an S3 Server's JMX port, 19081 for an Admin Server's JMX port, or 19082 for a HyperStore Server's JMX port)

This page left intentionally blank

# Chapter 12. Admin API

## 12.1. Introduction

### 12.1.1. HyperStore Admin API Introduction

**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.

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 Admin Service, which runs on the same nodes as your S3 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 701) and "**HTTP/S Basic Authentication for Admin API Access**" (page 703)).

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. When you access the Admin API directly, you can submit requests to any HyperStore node in your **default service region**.

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.

#### 12.1.1.1. RBAC Versions of Admin API Methods

For some read-only Admin API methods, there are alternative versions of the method implemented in the HyperStore IAM Service. This feature provides for granular role-based access control (RBAC) to a subset of HyperStore administrative functions, invoked by making calls to the HyperStore IAM Service (rather than the Admin Service). For more information on this feature, including information about the client tool that HyperStore provides to help you use this feature, see "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

#### 12.1.1.2. 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\*\*](#)

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 four 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.

#### See Also:

- [\*\*"Admin API Methods List"\*\* \(page 696\)](#)
- [\*\*"Common Request and Response Headers"\*\* \(page 699\)](#)
- [\*\*"Common Response Status Codes"\*\* \(page 700\)](#)
- [\*\*"cURL Examples"\*\* \(page 700\)](#)
- [\*\*"HTTP and HTTPS for Admin API Access"\*\* \(page 701\)](#)
- [\*\*"HTTP/S Basic Authentication for Admin API Access"\*\* \(page 703\)](#)
- [\*\*"Admin API Logging"\*\* \(page 704\)](#)

### 12.1.2. 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                                       |
|----------------------------------|--------------------------------|-----------------------------------------------|
| <a href="#"><u>billing</u></a>   | GET /billing                   | Get a bill for a user or group                |
|                                  | POST /billing                  | Create a bill for a user or group             |
| <a href="#"><u>bppolicy</u></a>  | GET /bppolicy/bucketsperpolicy | Get list of buckets using each storage policy |
|                                  | GET /bppolicy/listpolicy       | Get list of storage policy IDs                |
| <a href="#"><u>bucketops</u></a> | 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            |

| Resource                    | Method                               | Purpose                                                |
|-----------------------------|--------------------------------------|--------------------------------------------------------|
| <a href="#">group</a>       | 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                                     |
| <a href="#">monitor</a>     | 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 |
|                             | GET /monitor/history                 | Get historical monitoring statistics for a node        |
|                             | GET /monitor/notificationrules       | Get the list of notification rules                     |
|                             | POST /monitor/acknowledgeevents      | Acknowledge monitoring events                          |
|                             | POST /monitor/notificationruleenable | Enable or disable notification rules                   |
|                             | POST /monitor/notificationrule       | Change a notification rule                             |
| <a href="#">permissions</a> | GET /permissions/publicUrl           | Get public URL permissions for an object               |
|                             | POST /permissions/publicUrl          | Create or change public URL permissions for an object  |
| <a href="#">qos</a>         | 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                |
| <a href="#">ratingPlan</a>  | 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                               |

| Resource                | Method                               | Purpose                                                                            |
|-------------------------|--------------------------------------|------------------------------------------------------------------------------------|
| <a href="#">system</a>  | GET /system/audit                    | Get summary counts for system                                                      |
|                         | GET /system/bucketcount              | Get count of buckets 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/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/version                  | Get HyperStore system version                                                      |
|                         | POST /system/processProtectionPolicy | Process pending storage policy deletion or creation jobs                           |
| <a href="#">tiering</a> | POST /system/repairusercount         | Reconcile user counts in Redis and Cassandra                                       |
|                         | DELETE /tiering/credentials          | Delete a tiering credential for Amazon, Google, or other S3-compliant destination  |
|                         | DELETE /tiering/azure/credentials    | Delete a tiering credential for Azure                                              |
|                         | DELETE /tiering/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 /tiering/spectra/credentials    | Post a tiering credential for Spectra                                              |

| Resource                  | Method                            | Purpose                                                    |
|---------------------------|-----------------------------------|------------------------------------------------------------|
| <a href="#">usage</a>     | 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                  |
| <a href="#">user</a>      | 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              |
|                           | GET /user                         | Get a user's profile                                       |
|                           | GET /user/credentials             | Get a user's S3 security credential                        |
|                           | 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/list                    | Get a list of user profiles                                |
|                           | GET /user/password/verify         | Verify a user's CMC 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/password               | Create or change a user's CMC password                     |
|                           | POST /user/ratingPlanId           | Assign a rating plan to a user                             |
|                           | PUT /user                         | Create a new user                                          |
|                           | PUT /user/credentials             | Create a new S3 credential for a user                      |
| <a href="#">whitelist</a> | GET /whitelist                    | Get whitelist content                                      |
|                           | POST /whitelist                   | Change whitelist content (by request body object)          |
|                           | POST /whitelist/list              | Change whitelist content (by query parameters)             |

### 12.1.3. Common Request and Response Headers

#### 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.

## 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.

### 12.1.4. Common Response Status Codes

The following HTTP Status Codes are relevant for every Admin API method. Each method may return these codes, in addition to method-specific status codes indicated in the method documentation.

| Status Code | Description                                                                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 200         | OK                                                                                                                                                                                                                                                                                                                                                             |
| 500         | Internal Server Error                                                                                                                                                                                                                                                                                                                                          |
| 404         | Not found.<br><br><b>Note</b> URIs for the Admin API are <b>case-sensitive</b> . 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. |

The following HTTP Status Code is relevant for all Admin API methods that are forbidden in the non-default regions of a multi-region HyperStore system.

| Status Code | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 403         | Not allowed. Only the Admin API service in the default region can execute this method.<br><br>The only Admin API methods that are allowed in non-default regions of a multi-region deployment are: <ul style="list-style-type: none"><li>o <a href="#">POST /usage/storage</a></li><li>o <a href="#">POST /usage/storageall</a></li><li>o <a href="#">POST /usage/rollup</a></li><li>o <a href="#">POST /usage/repair/dirtyusers</a></li></ul> For any other Admin API method, submitting the method request to a non-default region's Admin Service will result in the 403 response. |

### 12.1.5. 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 local 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:public \
https://localhost:19443/group?groupId=QA | python -mjson.tool
```

With this example you would replace "sysadmin" and "public" with whatever [HTTP Basic Authentication](#) user name and password you've configured (if you've changed them from those default values); replace "localhost" with the IP address of one of your HyperStore nodes in the default service region (unless you run the command from one of those hosts); and 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 (the "--insecure" option) 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:public \
-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:public \
'https://localhost:19443/user?userId=John&groupId=QA'
```

## 12.1.6. HTTP and HTTPS for Admin API Access

The Admin Service's support for HTTP and HTTPS depends on what version your HyperStore system was **when you originally installed it**.

### If Your Original HyperStore Install Was Version 6.0.2 or Newer

If your original HyperStore install was version 6.0.2 or newer -- for example, if you've done a fresh install of the current version of HyperStore; or if you originally installed HyperStore 6.0.2 or newer and have upgraded to the current version -- then the Admin Service by default accepts **only HTTPS** requests from clients (through port 19443). Regular HTTP requests (through port 18081) will be rejected. Also, if your original HyperStore install was version 6.0.2 or newer, the CMC uses exclusively HTTPS when submitting requests to the Admin Service. For security, this is the recommended configuration.

*If you want the Admin Service to allow regular HTTP requests...*

If you want the Admin Service to accept HTTP requests (through port 18081) as well as HTTPS requests (through port 19443), you can follow the steps below to implement that change. Following these steps will also result in the CMC using exclusively HTTP when submitting requests to the Admin Service.

**Note** Some [system maintenance cron jobs](#) will continue to use HTTPS when sending requests to the Admin Service, even if you take the steps below. So, while it is possible to configure the Admin Service to support regular HTTP, you cannot configure it to support regular HTTP exclusively. Its HTTPS port (19443) will continue to be utilized by the cron jobs.

1. On your Puppet master, open this configuration file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

2. Find the "admin\_secure" setting and change it from "true" to "false", as below:

```
admin_secure, false
```

Save your change and close the file.

3. Still on your Puppet master node, launch the HyperStore installer:

```
root# /opt/cloudian-staging/7.2/cloudianInstall.sh
```

4. From the main menu select "Cluster Management" and then from the sub-menu that displays select "Push Configuration Settings to Cluster". Follow the prompts to trigger a Puppet push out to the cluster.
5. Return to the "Cluster Management" menu, then select "Manage Services". Select the S3 Service, then enter "restart". This automatically restarts the Admin Service as well as the S3 Service.
6. From the same menu, restart your CMC service. The CMC needs to be restarted so that it can update its configuration settings and start using HTTP to communicate with the Admin Service.

## If Your Original HyperStore Install Was Older Than Version 6.0.2

If your original HyperStore install was older than version 6.0.2 and you have upgraded to the current version, 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. Some [system maintenance cron jobs](#), on the other hand, use HTTPS when sending requests to the Admin Service.

*If you want the Admin Service to accept only HTTPS requests...*

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 your Puppet master, open this configuration file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

2. Anywhere in the "S3/Admin Services" section of *common.csv*, add this line:

```
admin_secure,true
```

**Note** By default the "admin\_secure" setting does not appear in the *common.csv* file for systems that were originally installed as version 6.0.2 or older (even after the upgrade process). You must manually add an "admin\_secure" line to the file, set to "true" as shown above.

Save your change and close the file.

3. Still on your Puppet master node, launch the HyperStore installer:

```
root# /opt/cloudian-staging/7.2/cloudianInstall.sh
```

4. From the main menu select "Cluster Management" and then from the sub-menu that displays select "Push Configuration Settings to Cluster". Follow the prompts to trigger a Puppet push out to the cluster.
5. Return to the "Cluster Management" menu, then select "Manage Services". Select the S3 Service, then enter "restart". This automatically restarts the Admin Service as well as the S3 Service.
6. From the same menu, restart your CMC service. The CMC needs to be restarted so that it can update its configuration settings and start using exclusively HTTPS to communicate with the Admin Service.

**Note** You do not need to take any action in regard to an SSL certificate for the Admin Service (for HTTPS support). A self-signed certificate -- unique to your system -- is generated automatically during HyperStore installation and is used by the Admin Service.

### 12.1.7. HTTP/S Basic Authentication for Admin API Access

If your **original HyperStore install was version 6.0.2 or newer** -- for example, if you've done a fresh install of the current version of HyperStore; or if you originally installed HyperStore 6.0.2 and have upgraded to the current version -- the Admin Service by default requires HTTP/S Basic Authentication from clients. The required user name and password are as configured by the "**admin\_auth\_user**" (page 471) and "**admin\_auth\_pass**" (page 471) settings in the *common.csv* file on your Puppet master. The default user name is "sysadmin" and the default password is "public".

If your **original HyperStore install was older than version 6.0.2** and you have upgraded to the current version, the Admin Service by default does not require Basic Authentication.

*If your original install was older than 6.0.2 and you want the Admin Service to require Basic Authentication...*

If your original install was older than version 6.0.2 and you want the Admin Service to require HTTP/S Basic Authentication from clients, do the following:

1. If you want to use an Admin Service HTTP/S Basic Authentication password other than HyperStore's default password for this purpose (which is a Jetty-obfuscation of "public"), use the Jetty password tool that's included in your HyperStore package to generate a different Jetty-obfuscated password. For example, generating a Jetty-obfuscation of the password "test1234":

```
$ cd /opt/cloudian/lib
$ java -cp jetty-util* org.eclipse.jetty.util.security.Password test1234
test1234
OBF:1mf31j8x1lts11tu11q411q61j651mbj
MD5:16d7a4fca7442dda3ad93c9a726597e4
```

If you generate a password, make a note of the OBF version of it; you will use it in Step 3 of this procedure.

2. On your Puppet master, open this configuration file in a text editor:

```
/etc/cloudian-<version>-puppet/manifests/extdata/common.csv
```

3. In *common.csv* edit these settings, then save and close the file.

- *admin\_auth\_user* → Set to the username you want to use for HTTP Basic Authentication for the Admin Service (or just leave this at the default which is "sysadmin")
- *admin\_auth\_pass* → Set to a quote-enclosed comma-separated pair: "<Jetty\_OBF\_obfuscated\_password>, <cleartext\_version\_of\_password>". For example, "1mf31j8x1ts1tu1lq41lq61j651mbj,test1234" (or just leave this at the default which is "1uvglx1n1tv91tvt1x0z1uuq,public")
- *admin\_auth\_enabled* → Set to true.

**Note** You can leave *admin\_auth\_realm* at its default of "CloudianAdmin"

4. If you edited the *admin\_auth\_user* and/or *admin\_auth\_pass* setting in *common.csv*, open the *<region-name>\_region.csv* file(s) for each of your service regions and edit the username and/or password component of the *admin\_service\_info* setting:

```
Setting format
admin_service_info,"<admin-service-endpoint>:19443,<admin_auth_user>,
<admin_auth_obfuscated_password>,<admin_auth_realm>"

Default value
admin_service_info,"<your-admin-service-endpoint>:19443,sysadmin,
1uvglx1n1tv91tvt1x0z1uuq,CloudianAdmin"
```

5. Still on your Puppet master node, launch the HyperStore installer:

```
root# /<your-installation-staging-directory>/cloudianInstall.sh
```

6. From the main menu select "Cluster Management" and then from the sub-menu that displays select "Push Configuration Settings to Cluster".
7. Return to the "Cluster Management" menu, then select "Manage Services". Select the S3 Service, then enter "restart". This automatically restarts the Admin Service as well as the S3 Service.
8. From the same menu, restart your CMC service. The CMC needs to be restarted so that it can update its configuration settings and start using Basic Authentication when communicating with the Admin Service.

### 12.1.8. Admin API Logging

Admin API transactions are logged in the Admin Service application log, for which the default location is */var/-log/cloudian/cloudian-admin.log*. API transactions are logged at log level INFO. As with all Admin Service application log entries, the format for API transaction entries is:

```
Date(ISO8601) PriorityLevel [ThreadId] ClassName:MethodName (Line#) MESSAGE
```

In the following example, first a request to retrieve the current list of user groups in the system is successfully processed. Then a request is received which is asking to retrieve a specific group that doesn't actually exist in the system (perhaps the requestor made a typo in the group ID). That request results in a 204 HTTP error response from the system. Note that each API transaction is identified by an integer within the MESSAGE element — "7" for the first transaction in the example and "8" for the second transaction.

```
2016-01-02 09:45:07,841 INFO [qtp21028611-57] LoggingFilter:log(153) 7 * Server has
received a request on thread qtp21028611-57
7 > GET http://192.168.2.16:18081/group/list
7 > Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
7 > Accept-Encoding: gzip, deflate
7 > Accept-Language: null
```

```

7 > Connection: keep-alive
7 > DNT: 1
7 > Host: 192.168.2.16:18081
7 > User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0

2016-01-02 09:45:08,607 INFO [qtp21028611-57] LoggingFilter:log(153) 7 * Server responded with a response on thread qtp21028611-57
7 < 200
7 < Content-Type: application/json
2016-01-02 09:48:17,596 INFO [qtp21028611-59] LoggingFilter:log(153) 8 * Server has received a request on thread qtp21028611-59
8 > GET http://192.168.2.16:18081/group?groupId=SalesGroup
8 > Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
8 > Accept-Encoding: gzip, deflate
8 > Accept-Language: null
8 > Connection: keep-alive
8 > DNT: 1
8 > Host: 192.168.2.16:18081
8 > User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0

2016-01-02 09:48:17,621 INFO [qtp21028611-59] GroupResource:getGroup(88) Group Id not found:SalesGroup

2016-01-02 09:48:17,623 INFO [qtp21028611-59] LoggingFilter:log(153) 8 * Server responded with a response on thread qtp21028611-59
8 < 204

```

## 12.2. 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\(s\)](#).

For an overview of the HyperStore billing feature, see "[Billing Feature Overview](#)" (page 63).

### 12.2.1. GET /billing

#### GET /billing Get a bill for a user or group

The request line syntax for this method is as follows.

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

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[billing Query Parameters](#)" (page 709).

There is no request payload.

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.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in **"RBAC Version of this Method"** (page 707).

## Example Using cURL

The [example](#) below generates a billable activity report for the user "glad" from the "eng" group, for the month of July 2017. This is an existing billable activity report that has previously been generated by the [POST /billing](#) method.

```
curl -X GET -k -u sysadmin:public \
'https://localhost:19443/billing?userId=glad&groupId=eng&billingPeriod=201707' \
| python -mjson.tool
```

The response payload is a JSON-formatted *Bill* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see **"Bill Object"** (page 710).

```
{
 "billId": "936265a2-fbd5-47c2-82ed-d62298299a1b",
 "canonicalUserId": "d47151635ba8d94efe981b24db00c07e",
 "currency": "USD",
 "endCal": 1501545599000,
 "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": 1498867200000,
 "total": 10.94,
 "userId": "glad",
 "whitelistTotal": 0
}
```

## Response Format

The response payload is a JSON-formatted *Bill* object (see example above). For response status code this method will return one of the [Common Status Codes](#) 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} |

### 12.2.1.1. 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 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 "[IAM Extensions for Role-Based Access to HyperStore Admin Functions](#)" (page 95).

- 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=d47151635ba8d94e-
fe981b24db00c07e
&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>

```

## 12.2.2. POST /billing

### POST /billing Create a bill for a user or group

The request line syntax for this method is as follows.

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

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**billing Query Parameters**" (page 709).

There is no request payload.

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 *mts.properties.erb*: "**reports.rolluphour.ttl**" (page 511). The default retention period is 65 days. Once this rollup data is deleted it can no longer be used to generate users' bills.

### Example Using cURL

The **example** below generates a billable activity report for the user "glad" from the "eng" group, for the month of July 2017.

```
curl -X POST -k -u sysadmin:public \
'https://localhost:19443/billing?userId=glad&groupId=eng&billingPeriod=201707' \
| python -mjson.tool
```

The response payload is a JSON-formatted *Bill* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "["Bill Object"](#)" (page 710).

```
{
 "billID": "936265a2-fbd5-47c2-82ed-d62298299a1b",
 "canonicalUserId": "d47151635ba8d94efe981b24db00c07e",
 "currency": "USD",
 "endCal": 1501545599000,
 "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": 1498867200000,
 "total": 10.94,
 "userId": "glad",
 "whitelistTotal": 0
}
```

## Response Format

The response payload is a JSON-formatted *Bill* object (see example above). For response status code this method will return one of the [Common Status Codes](#) or one of these method-specific status codes:

| Status Code | Description                                                |
|-------------|------------------------------------------------------------|
| 204         | No billing data                                            |
| 400         | Invalid billing period                                     |
| 400         | Missing required parameter : {billingPeriod}               |
| 400         | Conflicting parameters: {canonicalUserId, groupId, userId} |

### 12.2.3. billing Query Parameters

*userId*, *groupId*, *canonicalUserId*

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

- To generate or 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 *userId=martinez@groupId=operations*), or use the "canonicalUserId" parameter by itself (with no "groupId" parameter).
- To generate or retrieve 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 or retrieve 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 "201708" for August 2017. Note that the system uses GMT time when demarcating exactly when a month begins and ends.

### 12.2.4. billing Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Billing related Admin API methods.

**Note** For examples of this object see the API method request and response examples.

- **"Bill Object"** (page 710)

#### 12.2.4.1. Bill Object

The *Bill* object consists of the following attributes and nested objects:

##### *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": 1501545599000
```

#### 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 GB-Month (average number of GBs of data stored for the billing month). This is calculated by summing the month's hourly readings of stored bytes, converting to GB, then dividing by the number of hours in the month. In the example above the usage quantity during the billing period was 108 GB-months (that is, the user's storage bytes volume average 108GBs 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 GB-month is charged at \$0.14, the next 5 GB-months is charged at \$0.12 per GB-month, and all GB-months above that are charged at \$0.10 per GB-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 GB-months usage quantity ( $[1 \times .14] + [5 \times .12] + [102 \times .10] = 10.94$ ).

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": 1498867200000
```

*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
```

## 12.3. bppolicy

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

For an overview of the HyperStore storage policy feature, see "[Storage Policies Feature Overview](#)" (page 140). To create or change storage policies use the CMC's [Storage Policies](#) page.

### 12.3.1. GET /bppolicy/bucketsperpolicy

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

The request line syntax for this method is as follows.

```
GET /bppolicy/bucketsperpolicy
```

There is no request payload.

#### Example Using cURL

The [example](#) below retrieves the list of buckets using each storage policy.

```
curl -X GET -k -u sysadmin:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "[BucketsInPolicy Object](#)" (page 716).

```
[
```

```
{
```

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

## Response Format

The response payload is a JSON-formatted list of *BucketsInPolicy* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.3.2. GET /bppolicy/listpolicy

#### GET /bppolicy/listpolicy    Get list of storage policy IDs

The request line syntax for this method is as follows.

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

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**bppolicy Query Parameters**" (page 715).

**Note** 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.

There is no request payload.

Use this method if you want to retrieve the system-generated policy IDs associated with each of your storage policies.

#### Example Using cURL

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

```
curl -X GET -k -u sysadmin:public \
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 "policyName" and "policyId". 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",
 ...
 ...
 }
]
```

### Response Format

The response payload is a JSON-formatted list of *BucketProtectionPolicy* objects (see excerpt above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.3.3. bppolicy Query Parameters

#### *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, string) 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.

## 12.3.4. bppolicy Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the bucket protection policy related Admin API methods.

**Note** For examples of this object see the API method request and response examples.

- "**BucketsInPolicy Object**" (page 716)

### 12.3.4.1. BucketsInPolicy Object

The *BucketsInPolicy* object consists of the following attributes:

*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"
```

## 12.4. bucketops

### 12.4.1. GET /bucketops/id

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

The request line syntax for this method is as follows.

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

For parameter description click on the parameter name or see "**bucketops Query Parameters**" (page 719).

There is no request payload.

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.

## Example Using cURL

The [example](#) below retrieves the canonical ID of a bucket named "bucket1".

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

## Response Format

The response payload is a plain text string (see example above). For response status code this method will return either one of the ["Common Response Status Codes"](#) (page 700) or one of these method-specific status codes:

| Status Code | Description                               |
|-------------|-------------------------------------------|
| 400         | Missing required parameter : {bucketName} |

## 12.4.2. GET /bucketops/gettags

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

The request line syntax for this method is as follows.

```
GET /bucketops/gettags?groupID=xxx [&limit=xxx] [&userId=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see ["bucketops Query Parameters"](#) (page 719).

There is no request payload.

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 [PUT Bucket tagging](#)). 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.

#### 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.
- \* Only buckets that have bucket tags are listed in the response. Buckets that do not have bucket tags are excluded from the response.

## 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:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**BucketTags**" (page 720).

```
{
 "groupId": "Cloudian",
 "nextUserId": null,
 "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.

```
}
```

## Response Format

The response payload is a JSON-formatted *BucketTags* object (see example above). For response status code this method will return either one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

| Status Code | Description                            |
|-------------|----------------------------------------|
| 400         | Missing required parameter : {groupId} |

### 12.4.3. POST /bucketops/purge

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

The request line syntax for this method is as follows.

```
POST /bucketops/purge?bucketName=xxx
```

For parameter description click on the parameter name or see "**bucketops Query Parameters**" (page 719).

There is no request payload.

This operation marks all the objects in the bucket for deletion. The actual deletion of objects will then be executed in the background by a [system cron job](#) that runs hourly.

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

The bucket itself continues to exist after this operation. If you run an S3 *GET Bucket (List Objects)* call on the bucket -- or get the bucket in the CMC -- after successfully calling the *POST /bucketops/purge* operation, the *GET Bucket* 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. Any objects that you upload at this point -- **after** you've successfully called *POST /bucketops/purge* -- will not be deleted by the cron job.

Note that:

- Any S3 multipart upload operations in-progress for the bucket at the time that you execute the *POST /bucketups/purge* operation will be aborted.
- If you have [versioning](#) configured on the bucket, the *POST /bucketups/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).
- The *POST /bucketups/purge* operation is not allowed on buckets that have a [bucket lock policy](#) in place.
- Usage data (for stored bytes and stored objects) will be updated for the bucket owner by system cron jobs that run each day.
- In a multi-region system, the *POST /bucketups/purge* call should be submitted to the Admin API service in the default region regardless of which region the target bucket is in.

## Example Using cURL

The [example](#) below purges the contents of a bucket named "bucket1".

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

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

```
Bucket: bucket1 purged.
```

## Response Format

The response payload is a plain text string (see example above). For response status code this method will return either one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

| Status Code | Description                               |
|-------------|-------------------------------------------|
| 400         | Missing required parameter : {bucketName} |
| 403         | Can't delete locked bucket                |

### 12.4.4. bucketops Query Parameters

*bucketName*

(Mandatory, string) Name of the bucket.

*groupId*

(Mandatory, string) For a *GET /bucketops/gettags* operation, the group for which to retrieve a list of users

and their bucket tags.

*limit*

(Optional, integer) For a *GET /bucketops/gettags* operation, 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) For a *GET /bucketops/gettags* operation, 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.

## 12.4.5. bucketops Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the *bucketops* related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- **"BucketTags"** (page 720)

### 12.4.5.1. BucketTags

The *BucketTags* object consists of the following attributes:

*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 alpha-numerically 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 alphabetically 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"}},
 "gashington":
 {"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.

## 12.5. 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.

### 12.5.1. DELETE /group

**DELETE /group** Delete a group

**Note** Before you can delete a group you must first delete all users associated with the group, using the [DELETE /user](#) method.

The request line syntax for this method is as follows.

```
DELETE /group?groupId=xxx
```

For parameter description click on the parameter name or see "[group Query Parameters](#)" (page 730).

There is no request payload.

## Example Using cURL

The [example](#) below deletes the "QA" group.

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

## Response Format

There is no response payload. For response status code this method will return either one of the ["Common Response Status Codes"](#) (page 700) 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.      |

### 12.5.2. GET /group

#### GET /group Get a group's profile

The request line syntax for this method is as follows.

```
GET /group?groupId=xxx
```

For parameter description click on the parameter name or see ["group Query Parameters"](#) (page 730).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in ["RBAC Version of this Method"](#) (page 723).

## Example Using cURL

The [example](#) below retrieves the "QA" group.

```
curl -X GET -k -u sysadmin:public 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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see ["GroupInfo Object"](#) (page 731).

```
{
 "active": "true",
 "groupId": "QA",
 "groupName": "Quality Assurance Group",
 "ldapEnabled": false,
 "ldapGroup": "",
 "ldapMatchAttribute": "",
 "ldapSearch": ""}
```

```

 "ldapSearchUserBase": "",
 "ldapServerURL": "",
 "ldapUserDNTemplate": "",
 "s3endpointshttp": "ALL",
 "s3endpointshttps": "ALL",
 "s3websiteendpoints": "ALL"
}

```

## Response Format

The response payload is a JSON-formatted `GroupInfo` object (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

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

### 12.5.2.1. 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 **"IAM Extensions for Role-Based Access to HyperStore Admin Functions"** (page 95).

- 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 Hyper-Store 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>
```

### 12.5.3. GET /group/list

**GET /group/list** Get a list of group profiles

The request line syntax for this method is as follows.

GET /group/list[?prefix=xxx][&limit=xxx][&offset=xxx]

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see **"group Query Parameters"** (page 730).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in ["RBAC Version of this Method"](#) (page 725).

## Example Using cURL

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

```
curl -X GET -k -u sysadmin:public 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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see ["GroupInfo Object"](#) (page 731).

```
[
 {
 "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"
 }
]

```

## Response Format

The response payload is a JSON-formatted list of *GroupInfo* objects (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

Status Code	Description
400	Limit should be greater than zero

### 12.5.3.1. 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 **"IAM Extensions for Role-Based Access to HyperStore Admin Functions"** (page 95).

- 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>
```

## 12.5.4. GET /group/ratingPlanId

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

The request line syntax for this method is as follows.

```
GET /group/ratingPlanId?groupId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**group Query Parameters**" (page 730).

There is no request payload.

### Example Using cURL

The [example](#) below retrieves the ID of the rating plan assigned to the "QA" group.

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

### Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

## 12.5.5. POST /group

**POST /group** Change a group's profile

The request line syntax for this method is as follows.

```
POST /group
```

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

### 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:public \
-d @group_QA.txt https://localhost:19443/group
```

Note that in editing the *GroupInfo* object in the *group\_QA.txt* file before doing the POST operation you could 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](#).

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

## 12.5.6. POST /group/ratingPlanId

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

The request line syntax for this method is as follows.

```
POST /group/ratingPlanId?groupId=xxx&ratingPlanId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**group Query Parameters**" (page 730).

There is no request payload.

## Example Using cURL

The [example](#) below assigns the "Gold" rating plan to the "QA" group.

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

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

## 12.5.7. PUT /group

### PUT /group Create a new group

The request line syntax for this method is as follows.

```
PUT /group
```

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

#### 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:public \
-d @group_QA.txt https://localhost:19443/group
```

The *group\_QA.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "["GroupInfo Object"](#) (page 731).

```
{
 "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.

#### Response Format

There is no response payload. For response status code this method will return one of the "["Common Response Status Codes"](#) (page 700) 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

Status Code	Description
409	Unique constraint violation : {groupId}

## 12.5.8. group Query Parameters

*groupId*

(Mandatory, string) Unique identifier of the group.

*prefix*

(Optional, string) With a *GET /group/list* request: 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) With a *GET /group/list* request: 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) With a *GET /group/list* request: 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.

*ratingPlanId*

(Mandatory, string) With a *POST /group/ratingPlanId* request: 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=d=Engineering&ratingPlanId=Gold&region=East*, then the Gold rating plan will be applied to the Engineering group's service activity in the East region.

## 12.5.9. group Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Group related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- "GroupInfo Object" (page 731)

### 12.5.9.1. GroupInfo Object

The *GroupInfo* object consists of the following attributes:

*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. The groupId "0" is reserved for the system admin group.

Example:

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

**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 Feature Overview**" (page 101).

Example:

```
"ldapEnabled": false
```

**Note** If you enable LDAP Authentication for an existing group to which users have already been added, 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) If this is an LDAP-enabled group, you can use this attribute to specify the exact name of the group as it exists in the LDAP system. This would typically be the "ou" (Organization Unit) value in the LDAP system, but could also be for example the "l" (Location) value or "memberOf" value -- depending on which of these attributes you're using to identify the group.

Note that the "ldapGroup" attribute does not have the character restrictions that the "groupId" attribute has. For example, if the group's "ou" value in LDAP is "Marketing", when creating the group in HyperStore you can use "Marketing" as both the "groupId" and the "ldapGroup" name. By contrast, if the group's "ou" value in LDAP is "Quality Assurance (U.S.)" you would use that name exactly as the "ldapGroup" attribute:

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

But as the "groupId" attribute you would need to use something like "Quality\_Assurance\_US" to satisfy that attribute's character restrictions. The CMC will use the "ldapGroup" value for purposes of LDAP authentication, while using the "groupId" value internally for adherence to HyperStore data schemas.

**Note** Users within the group must use the "Group Name" value when selecting from the Group Name drop-down list when logging into the CMC. In the second example above, users would log into the CMC with the group name "Quality\_Assurance\_US".

#### *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 *ldapUser-DNTemplate* 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 *ldapUser-DNTemplate*, 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.

- **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 -- such as might be used in an Active Directory environment -- LDAP-enabled users from this group will log in with their LDAP *userPrincipalName* 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, 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, 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, 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"
```

## 12.6. 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.

### 12.6.1. DELETE /monitor/notificationrule

#### **DELETE /monitor/notificationrule** Delete a notification rule

The request line syntax for this method is as follows.

```
DELETE /monitor/notificationrule?ruleId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**monitor Query Parameters**" (page 754).

There is no request payload.

#### Example Using cURL

The [example](#) below deletes a notification rule.

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

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

### 12.6.2. GET /monitor/events

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

The request line syntax for this method is as follows.

```
GET /monitor/events?nodeId=xxx [&showAck=xxx] [&limit=xxx] [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**monitor Query Parameters**" (page 754).

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

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 737).

#### 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:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**MonitoringEvent Object**" (page 756).

```
[
 {
 "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"
}
]
```

## Response Format

The response payload is a JSON-formatted list of *MonitoringEvent* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

### 12.6.2.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>
```

### 12.6.3. GET /monitor/nodelist

#### GET /monitor/nodelist Get the list of monitored nodes

The request line syntax for this method is as follows.

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

For parameter description click on the parameter name or see "**monitor Query Parameters**" (page 754).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 739).

#### Example Using cURL

The [example](#) below retrieves the list of monitored nodes in the default service region.

```
curl -X GET -k -u sysadmin:public \
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"
]
```

## Response Format

The response payload is a JSON-formatted list of hostnames (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700).

### 12.6.3.1. 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 **"IAM Extensions for Role-Based Access to HyperStore Admin Functions"** (page 95).

- 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>
```

## 12.6.4. GET /monitor/host

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

The request line syntax for this method is as follows.

```
GET /monitor/host?nodeId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**monitor Query Parameters**" (page 754).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 744).

### Example Using cURL

The [example](#) below retrieves current monitoring statistics for the node that has hostname "store1".

```
curl -X GET -k -u sysadmin:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**MonitorNodeInfo Object**" (page 758).

```
{
 "adminHeapMax": {
 "timestamp": "1502799543355",
 "value": "409075712"
 },
 "adminHeapUsed": {
 "timestamp": "1502799543355",
 "value": "109849080"
 },
 "cassCMSGCCount": {
 "timestamp": "1502799543355",
 "value": "3"
 },
 "cassCMSGCTime": {
 "timestamp": "1502799543355",
 "value": "151"
 },
 "cassCopyGCCCount": null,
 "cassCopyGCTime": null,
 "cassHeapMax": {
 "timestamp": "1502799543355",
 "value": "2086666240"
 },
 "cassHeapUsed": {
 "timestamp": "1502799543355",
 "value": "1233215776"
 }
},
```

```

"cassParNewGCCountcassParNewGCTimecpudiskAvailKbdiskIORReaddiskIOWritediskTotalKbdiskUsedKbdisksInfo

```

```
 "diskIORRead": "3163136",
 "diskIOWrite": "50221056",
 "diskTotalKb": "20181628",
 "diskUsedKb": "45688",
 "mountPoint": "/cloudian1",
 "status": "OK",
 "storageUse": [
 "HS"
]
},
],
"timestamp": "1502799663530"
},
"hyperStoreHeapMaxhyperStoreHeapUsedioRxioTxs3GetLatencys3GetTPSs3GetThruputs3HeapMaxs3HeapUseds3PutLatencys3PutTPSs3PutThruput
```

```

 "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]"
}
}

```

## Response Format

The response payload is a JSON-formatted *MonitorNodeInfo* object (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {nodeId}

### 12.6.4.1. 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 **"IAM Extensions for Role-Based Access to HyperStore Admin Functions"** (page 95).

- 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>
```

## 12.6.5. GET /monitor

### GET /monitor Get current monitoring statistics for a service region

The request line syntax for this method is as follows.

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

For parameter description click on the parameter name or see "**monitor Query Parameters**" (page 754).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 747).

### Example Using cURL

The [example](#) below retrieves current monitoring statistics for the default service region.

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

The response payload is a JSON-formatted *MonitorSystemInfo* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**MonitorSystemInfo Object**" (page 765).

```
{
 "diskAvailKb": {
 "timestamp": "1502799843254",
```

```
 "value": "61855592"
 },
 "diskTotalKbdiskUsedKbnodeStatusess3GetLatencys3GetTPSs3GetThruputs3PutLatencys3PutTPS
```

```

},
"s3PutThruput": {
 "timestamp": "1502799843254",
 "value": "0"
},
"status": {
 "ipaddr": "",
 "status": [
 "OK"
],
 "timestamp": "1502799843254",
 "value": "[OK]"
}
}

```

## Response Format

The response payload is a JSON-formatted *MonitorSystemInfo* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Invalid region : {region}

### 12.6.5.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>
```

## 12.6.6. GET /monitor/history

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

The request line syntax for this method is as follows.

```
GET /monitor?nodeId=xxx [®ion=xxx] &statId=xxx&startTime=xxx&endTime=xxx
```

For parameter description click on the parameter name or see "**monitor Query Parameters**" (page 754).

There is no request payload.

#### Example Using cURL

The [example](#) below retrieves history for the "cpu" statistic, for a one hour period (2019 July 20th, midnight to 1AM). Note that the system interprets the start and end times as **GMT** times.

```
curl -X GET -k -u sysadmin:public \
'https://localhost:19443/monitor/history?nodeId=
hs1&statId=cpu&startTime=201907201200&endTime=201907201300' \
| 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 (see [statId](#) for detail). 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"
},
...
}
]
```

## Response Format

The response payload is a JSON-formatted list of timestamp/value pairs (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

## 12.6.7. GET /monitor/notificationrules

### GET /monitor/notificationrules Get the list of notification rules

The request line syntax for this method is as follows.

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

For parameter description click on the parameter name or see "**monitor Query Parameters**" (page 754).

There is no request payload.

### Example Using cURL

The [example](#) below retrieves the current list of notification rules for the default service region.

```
curl -X GET -k -u sysadmin:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**NotificationRule Object**" (page 768).

```
[
{
 "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"
},
...
...
]
```

## Response Format

The response payload is a JSON-formatted list of *NotificationRule* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.6.8. POST /monitor/acknowledgeevents

#### POST /monitor/acknowledgeevents Acknowledge monitoring events

The request line syntax for this method is as follows.

```
POST /monitor/acknowledgeevents?nodeId=xxx[®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**monitor Query Parameters**" (page 754).

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

#### 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:public \
-d @event_acknowledge.txt \
https://localhost:19443/monitor/acknowledgeevents?nodeId=store1
```

The *event\_acknowledge.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**EventsAck Object**" (page 756).

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

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

## 12.6.9. POST /monitor/notificationruleenable

### POST /monitor/notificationruleenable    Enable or disable notification rules

The request line syntax for this method is as follows.

```
POST /monitor/notificationruleenable
```

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

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.

## 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:public \
-d @rule_disable.txt https://localhost:19443/monitor/notificationruleenable
```

The *rule\_disable.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**NotificationRulesEnable Object**" (page 773).

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

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

## 12.6.10. POST /monitor/notificationrule

### POST /monitor/notificationrule Change a notification rule

The request line syntax for this method is as follows.

```
POST /monitor/notificationrule
```

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

#### 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:public \
-d @rule_s3GetLatency.txt https://localhost:19443/monitor/notificationrule
```

The *rule\_s3GetLatency.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**NotificationRule Object**" (page 768).

```
{
 "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.

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

### 12.6.11. PUT /monitor/notificationrule

#### PUT /monitor/notificationrule Create a new notification rule

The request line syntax for this method is as follows.

```
PUT /monitor/notificationrule
```

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

**Note** The HyperStore system comes with many pre-configured notification rules. To see the existing set of rules go to the CMC's [Alert Rules](#) page. The CMC interface uses the term "alert rules" rather than "notification rules".

#### 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:public \
-d @rule_s3GetLatency.txt https://localhost:19443/monitor/notificationrule
```

The *rule\_s3GetLatency.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**NotificationRule Object**" (page 768).

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

```
 "statId": "s3GetLatency"
}
```

## Response Format

There is no response payload. For response status code this method will return one of the ["Common Response Status Codes"](#) (page 700) or one of these method-specific status codes:

Status Code	Description
400	Invalid JSON object

## 12.6.12. monitor Query Parameters

### ruleId

(Mandatory, string) For a *DELETE /monitor/notificationrule* request: 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 "8e4cc533-360a-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.

### nodeId

(Mandatory, string) For a *GET /monitor/events* or *GET /monitor/host* or *GET /monitor/history* or *POST /monitor/acknowledgeevents* request: The hostname of the target node.

### showAck

(Optional, boolean) For a *GET /monitor/events* request: 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) For a *GET /monitor/events* request: The maximum number of events to return in the response.

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

### statId

(Mandatory, string) For a *GET /monitor/history* request: 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 ["MonitorNodeInfo Object"](#) (page 758).

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

**startTime**

(Mandatory, string) For a *GET /monitor/history* request: the start time of the interval for which to retrieve the statistic history, in format *yyyyMMddHHmm* (for example 201907200000). 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) For a *GET /monitor/history* request: the end time of the interval for which to retrieve the statistic history, in format *yyyyMMddHHmm* (for example 201907201200). 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.

### 12.6.13. monitor Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Monitor related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- "**EventsAck Object**" (page 756)
- "**MonitoringEvent Object**" (page 756)
- "**MonitorNodeInfo Object**" (page 758)
- "**MonitorSystemInfo Object**" (page 765)

- "**NotificationRule Object**" (page 768)
- "**NotificationRulesEnable Object**" (page 773)

### 12.6.13.1. EventsAck Object

The *EventsAck* object consists of the following attributes:

*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 a time period configured by the *events.acknowledged.ttl* property in the *mts.properties.erb* configuration file. By default this period is 86400 seconds (one day).

Example:

```
"delete": true
```

*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"
```

### 12.6.13.2. MonitoringEvent Object

The *MonitoringEvent* object consists of the following attributes:

*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 "**NotificationRule Object**" (page 768) 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"
```

### 12.6.13.3. MonitorNodeInfo Object

The *MonitorNodeInfo* object consists of the following attributes and nested objects:

**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>"}.

#### 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"}
```

#### cassCMSGCCCount

**(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:

```
"cassCMSGCCCount": {"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"}
```

#### cassCopyGCCCount

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

```
"cassCopyGCCCount": 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 kilobytes.

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 "**Automatic Stop of Writes to a Disk at 90% Usage**" (page 85).

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*.

*diskIORRead*

([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:

```
"diskIORRead": {"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 kilobytes. 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 kilobytes. 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 KBs. 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"
```

*diskIORRead*

(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:

```
"diskIORRead": "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 KBs. 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, see "**View a Node's Disk Detail**" (page 272).

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. 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. 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 MB 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 MB 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": "<UTC Milliseconds>", "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": "<statusValueFor-mattedAsString>"}.

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]"}

```

#### 12.6.13.4. MonitorSystemInfo Object

The *MonitorSystemInfo* object consists of the following attributes:

**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 kilobytes.

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 "[Automatic Stop of Writes to a Disk at 90% Usage](#)" (page 85).

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 kilobytes.

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 kilobytes.

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 MB 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 MB 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]"}

```

### 12.6.13.5. NotificationRule Object

The *NotificationRule* object consists of the following attributes:

#### *condition* (*mandatory*)

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

```
"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 293) page), 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 293) page.

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
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	<"	".15"	Yes, for <.15

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
	any individual disk on the node crosses the defined threshold.			
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).	>"	"1000000000"	No
ioTx	Total network bytes per second transmitted by the node (S3 traffic plus any other network traffic from the node).	>"	"1000000000"	No
diskIORRead	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 a second separate rule for status "LOG_ERR". Do not specify multiple service values in a single notification rule.	"="	"SVC_DOWN"	Yes, one rule for SVC_DOWN and one rule for LOG_ERR
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_

statId	Description	Appropriate "condition"	Sample "conditionVal"	Pre-Packaged Rule Exist?
				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 rule for LOG_ERR

**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.

### 12.6.13.6. NotificationRulesEnable Object

The *NotificationRulesEnable* object consists of the following attributes:

#### *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"]
```

### 12.6.13.7. MonitorStat

Data type with the following format:

```
"<statName>": {"timestamp": "<UTCMilliseconds>","value": "<statValue>"}
```

### 12.6.13.8. ServiceStatus

Data type with the following format:

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

The meanings of the possible "status" values are:

- "OK" — The service is up and running on the node, and there are no unacknowledged events associated with the service.
- "SVC\_DOWN" — The service is down on the node.
- "LOG\_WARN" — There are unacknowledged warnings in the service application log on the node.
- "LOG\_ERR" — There are unacknowledged errors in the service application log on the node.

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

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

## 12.7. 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.

### 12.7.1. GET /permissions/publicUrl

#### GET /permissions/publicUrl Get public URL permissions for an object

The request line syntax for this method is as follows.

```
POST /permissions/publicUrl?userId=xxx&groupId=xxx&bucketName=xxx&objectName=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**permissions Query Parameters**" (page 776).

There is no request payload.

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).

#### Example Using cURL

The [example](#) below retrieves an existing public URL for an object named *Cloudian.pdf*.

```
curl -X GET -k -u sysadmin:public \
'https://
localhost:19443/permissions/publicUrl?userId=jim&groupId=Pubs&bucketName=bkt1&objectName=Cloudian.pdf' \
| python -mjson.tool
```

The response payload is a JSON-formatted *PublicUrlAccess* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**PublicUrlAccess Object**" (page 777).

```
{
 "allowRead": true,
 "currentDownloads": 0,
 "expiryTime": "1517385600000",
 "maxDownloadNum": 1000,
 "secure": true,
 "url": "https://s3-region1.mycloudianhyperstore.com/bkt1/Cloudian.pdf?
AWSAccessKeyId=00b3ec480eb5c844fb88&Expires=1517385600&Signature=
rxxJnEWoUusrj1kQ02A9PMcFQ4U%3D&x-amz-pt=MDAzMTE1NjIxNTE2MTE4NzIxMDc1"
}
```

#### Response Format

The response payload is a JSON-formatted *PublicUrlAccess* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameter : {userId, groupId, bucketName, objectName}
400	User does not exist

## 12.7.2. POST /permissions/publicUrl

**POST /permissions/publicUrl** Create or change public URL permissions for an object

The request line syntax for this method is as follows.

```
POST /permissions/publicUrl?userId=xxx&groupId=xxx&bucketName=xxx&objectName=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**permissions Query Parameters**" (page 776).

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

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>?AWSAccessKeyId=<accessKeyOfObjectOwner>
&Expires=<expiryTime>&Signature=<SignatureString>&x-amz-pt=<>
```

This format follows the AWS specification for signed URLs.

### Example Using cURL

The [example](#) below creates a public URL for an object named *Cloudian.pdf*. 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:public \
-d @postPublicUrlAccess.txt \
'https://
localhost:19443/permissions/publicUrl?userId=jim&groupId=Pubs&bucketName=bkt1&objectName=Cloudian.pdf'
```

The *postPublicUrlAccess.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**PublicUrlAccess Object**" (page 777).

```
{
 "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.

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

### 12.7.3. permissions Query Parameters

*userId*

(Mandatory, string) User ID for user who owns the object for which a public URL is being generated.

*groupId*

(Mandatory, string) Group ID for user who owns the object.

*bucketName*

(Mandatory, bucketname) 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.

### 12.7.4. permissions Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Permissions related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- **"PublicUrlAccess Object"** (page 777)

### 12.7.4.1. PublicUrlAccess Object

#### *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>?AWSAccessKeyId=<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"
```

## 12.8. 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**" (page 110).

### 12.8.1. DELETE /qos/limits

#### DELETE /qos/limits Delete QoS settings for a user or group

The request line syntax for this method is as follows.

```
DELETE /qos/limits?userId=xxx&groupId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**qos Query Parameters**" (page 782).

There is no request payload.

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.

#### 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:public \
'https://localhost:19443/qos/limits?userId=&groupId=Dev'
```

#### Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {userId, groupId}
400	Region {region} is invalid

## 12.8.2. GET /qos/limits

### GET /qos/limits Get QoS settings for a user or group

The request line syntax for this method is as follows.

```
GET /qos/limits?userId=xxx&groupId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**qos Query Parameters**" (page 782).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 780).

### Example Using cURL

The [example](#) below retrieves current QoS settings for the user "cody" in the "Dev" group.

```
curl -X GET -k -u sysadmin:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**QosLimitSettings Object**" (page 783).

```
{
 "groupId": "Dev",
 "labelId": "qos.userQosOverrides.title",
 "qosLimitList": [
 {
 "type": "STORAGE_QUOTA_KBYTES",
 "value": 5000000
 },
 {
 "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
 }
]
}
```

```

 "value": -1
 },
 {
 "type": "DATAKBYTES_OUT_LH",
 "value": -1
 },
 {
 "type": "STORAGE_QUOTA_COUNT",
 "value": -1
 }
],
"userId

```

## Response Format

The response payload is a JSON-formatted *QoSLimitSettings* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

### 12.8.2.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>
```

### 12.8.3. POST /qos/limits

#### POST /qos/limits Create QoS settings for a user or group

The request line syntax for this method is as follows.

```
POST /qos/limits?userId=xxx&groupId=xxx&storageQuotaKBytes=xxx&storageQuotaCount=xxx
&wlRequestRate=xxx&hlRequestRate=xxx&wlDataKBytesIn=xxx&hlDataKBytesIn=xxx
&wlDataKBytesOut=xxx&hlDataKBytesOut=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**qos Query Parameters**" (page 782).

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". There is no request body payload.

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.

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.

**Note** For the system to **enforce** QoS limits that you have assigned to users or groups, the QoS feature must be enabled in your system configuration. By default it is disabled. You can enable it in the CMC's [Configuration Settings](#) page. Note that you can enable QoS enforcement just for storage utilization limits, or for storage utilization limits and also request traffic limits.

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {userId, groupId, storageQuotaBytes, storageQuotaCount, wIRequestRate, hIRequestRate, wIDataBytesIn, hIDataBytesIn, wIDataBytesOut, hIDataBytesOut}
400	Invalid parameter
400	Region {region} is invalid

### 12.8.4. qos Query Parameters

#### 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.

#### storageQuotaKBytes

(Mandatory, number) With a `POST /qos/limits` request: Maximum allowed storage, in number of

kilobytes.

Storage overhead associated with replication or erasure coding does not count toward this limit. For example a 100KB object that's replicated three times (in accordance with a 3X replication storage policy) would count as 100KB toward this limit, not 300KB.

#### *wlStorageQuotaCount*

(Mandatory, number) With a *POST /qos/limits* request: Maximum allowed storage, in number of objects.

#### *wlRequestRate*

(Mandatory, number) With a *POST /qos/limits* request: Warning level for number of HTTP requests per minute. Exceeding this level triggers the writing of a message to the S3 Service application log (it does not return a warning to the S3 client).

#### *hlRequestRate*

(Mandatory, number) With a *POST /qos/limits* request: Maximum allowed number of HTTP requests per minute.

#### *wlDataKBytesIn*

(Mandatory, number) With a *POST /qos/limits* request: Warning level for number of uploaded kilobytes per minute. Exceeding this level triggers the writing of a message to the S3 Service application log (it does not return a warning to the S3 client).

#### *hlDataKBytesIn*

(Mandatory, number) With a *POST /qos/limits* request: Maximum allowed number of uploaded kilobytes per minute.

#### *wlDataKBytesOut*

(Mandatory, number) With a *POST /qos/limits* request: Warning level for number of downloaded kilobytes per minute. Exceeding this level triggers the writing of a message to the S3 Service application log (it does not return a warning to the S3 client).

#### *hlDataKBytesOut*

(Mandatory, number) With a *POST /qos/limits* request: Maximum allowed number of downloaded kilobytes per minute.

## 12.8.5. qos Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the QoS related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- **"QosLimitSettings Object"** (page 783)

### 12.8.5.1. QosLimitSettings Object

The *QosLimitSettings* object consists of the following attributes and nested objects:

*groupId*

(String) Group ID. This will be either a specific group ID, or "ALL", or "\*". For details of how this is used, see "**qos Query Parameters**" (page 782).

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<QoS Limit>) List of *QoS Limit* objects. There will be one *QoS Limit* object for each of the eight QoS limit types. Each *QoS Limit* object consists of the following attributes:

*type*

(String) One of the following QoS limit types:

- STORAGE\_QUOTA\_KBYTES
- STORAGE\_QUOTA\_COUNT
- REQUEST\_RATE\_LW
- REQUEST\_RATE\_LH
- DATAKBYTES\_IN\_LW
- DATAKBYTES\_IN\_LH
- DATAKBYTES\_OUT\_LW
- DATAKBYTES\_OUT\_LH

For descriptions of these types see "**QoS Limit Type Descriptions**" (page 784).

Example:

```
"type": "STORAGE_QUOTA_KBYTES"
```

*value*

(Number) The value assigned to this QoS limit type. A value of -1 indicates that the limit is disabled. Example:

```
"value": 5000000
```

*userId*

(String) User ID. This will be either a specific user ID, or "ALL", or "\*". For details of how this is used, see "**qos Query Parameters**" (page 782). Example:

```
"userId": "cody"
```

## QoS Limit Type Descriptions

The table below describes each QoS limit type, as it applies to the individual user level and to the group level.

**Note** When the system rejects a user request because of a storage quota, it returns an HTTP 403 response to the client application. When the system rejects a user request due to rate controls, it returns an HTTP 503 response to the client application.

**Note** Requests rejected due to QoS limits **are** counted toward usage tracking, for purposes of request volume based billing. For example, if a user has reached her storage quota and tries to do a PUT of more data, the system rejects the PUT request but counts the request toward the user's bill (specifically, toward the part of her rating plan that charges per volume of HTTP PUT requests).

QoS Limit Type	Description	Implementation
STORAGE_QUOTA_KBYTES	Storage quota limit, in number of KBs	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
STORAGE_QUOTA_COUNT	Storage quota limit, in total number of objects. Note that folders count as objects, as well as files	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
REQUEST_RATE_LW	Request rate warning limit, in total number of HTTP requests per minute.	<ul style="list-style-type: none"> <li>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, an informational message is written to the S3 Server's application log. At the end of the 60 seconds, the request counter for the user is reset. Subsequently, the next request that</li> </ul>

QoS Limit Type	Description	Implementation
		<p>comes in from the user triggers the start of a new 60 second interval, and the process repeats.</p> <ul style="list-style-type: none"> <li>For group QoS — The implementation is the same as for user QoS, except that it applies to requests from all users in the group. 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.</li> </ul> <p>Note that HTTP DELETE requests are not counted toward Request Rate controls.</p>
REQUEST_RATE_LH	Request rate maximum, in total number of HTTP requests per minute.	<ul style="list-style-type: none"> <li>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. 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.</li> <li>For group QoS — The implementation is the same as for user QoS, except that it applies to requests from all users in the group. 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.</li> </ul>

QoS Limit Type	Description	Implementation
		group.
DATAKBYTES_IN_LW	Inbound data rate warning limit, in KBs 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 kilobytes of data.
DATAKBYTES_IN_LH	Inbound data rate high limit, in KBs 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 kilobytes of data. Note that if a block is triggered by the Data Bytes IN (KB) High Limit being reached, the block applies to all HTTP request types (not just PUTs.)
DATAKBYTES_OUT_LW	Outbound data rate warning limit, in KBs 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 kilobytes of data.
DATAKBYTES_OUT_LH	Outbound data rate high limit, in KBs 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 kilobytes of data. Note that if a block is triggered by the Data Bytes OUT (KB) High Limit being reached, the block applies to all HTTP request types (not just GETs.)

## 12.9. 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.

For an overview of the HyperStore billing feature, see "["Billing Feature Overview"](#) (page 63).

**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.

### 12.9.1. DELETE /ratingPlan

#### DELETE /ratingPlan Delete a rating plan

The request line syntax for this method is as follows.

```
DELETE /ratingPlan?ratingPlanId=xxx
```

For parameter description click on the parameter name or see "**ratingPlan Query Parameters**" (page 793).

There is no request payload.

### Example Using cURL

The [example](#) below deletes a rating plan with ID "Plan-6".

```
curl -X DELETE -k -u sysadmin:public \
https://localhost:19443/ratingPlan?ratingPlanId=Plan-6
```

### Response Format

There is no response payload. For response status code this method will return either one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Rating Plan does not exist
400	Missing required parameter :{ratingPlanId}

## 12.9.2. GET /ratingPlan

### GET /ratingPlan Get a rating plan

The request line syntax is as follows.

```
GET /ratingPlan?ratingPlanId=xxx
```

For parameter description click on the parameter name or see "**ratingPlan Query Parameters**" (page 793).

There is no request payload.

### Example Using cURL

The [example](#) below retrieves the system default rating plan, which has ID "Default-RP".

```
curl -X GET -k -u sysadmin:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**RatingPlan Object**" (page 794).

```
{
 "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"
}
```

## Response Format

The response payload is a JSON-formatted *RatingPlan* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
204	Rating Plan does not exist
400	Missing required parameter: {ratingPlanId}

### 12.9.3. GET /ratingPlan/list

#### GET /ratingPlan/list Get the list of rating plans in the system

The request line syntax for this method is as follows.

```
GET /ratingPlan/list
```

There is no request payload.

#### 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:public \
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"
}
]

```

## Response Format

The response payload is in format `List<Map<string,string,string,string>>` (see example above). Strings are: `{id,-name,encodedid,currency}`. "encodedId" value is a URL-encoding of the "id" value. For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.9.4. POST /ratingPlan

#### POST /ratingPlan Change a rating plan

The request line syntax for this method is as follows.

```
POST /ratingPlan
```

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

#### 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:public \
-d @ratingStorageOnly.txt https://localhost:19443/ratingPlan
```

Note that in editing the *RatingPlan* object in the *ratingStorageOnly.txt* file before doing the POST operation 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](#).

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

## 12.9.5. PUT /ratingPlan

### PUT /ratingPlan Create a new rating plan

The request line syntax for this method is as follows.

```
PUT /ratingPlan
```

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

#### 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:public \
-d @ratingStorageOnly.txt https://localhost:19443/ratingPlan
```

The *ratingStorageOnly.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "["RatingPlan Object"](#)" (page 794).

```
{
 "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

```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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}

## 12.9.6. ratingPlan Query Parameters

*ratingPlanId*

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

## 12.9.7. ratingPlan Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Rating Plan related Admin API methods.

**Note** For examples of this object see the API method request and response examples.

- **"RatingPlan Object"** (page 794)

### 12.9.7.1. RatingPlan Object

The *RatingPlan* object consists of the following attributes and nested objects:

*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 795) 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".

## 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 **GB-month** — the average number of GBs of data stored for the month (which is calculated by summing the month's hourly readings of stored bytes, converting to GB, 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 GB-month, the charge is \$2 per GB-month.
- For **data transfer bytes in or out (BI or BO)**, the unit is number of GBs. 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 GBs, the charge is \$2 per GB.

- 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"
```

## 12.10. system

The Admin API methods built around the **system** resource are for retrieving system information or performing certain system maintenance tasks.

### 12.10.1. GET /system/audit

#### GET /system/audit Get summary counts for system

The request line syntax for this method is as follows.

```
GET /system/audit?[region=xxx]
```

There is no request payload.

**Note** 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.

#### Example Using cURL

The **example** below retrieves the summary counts for the system.

```
curl -X GET -k -u sysadmin:public https://localhost:19443/system/audit \
| python -mjson.tool
```

The response payload is a JSON-formatted *AuditData* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**AuditData Object**" (page 809).

```
{
 "byteCount": 647687490,
 "bytesInCount": 0,
 "bytesOutCount": 0,
 "licenseExpiration": 1590094491952,
 "nodes": [
 {
```

```

 "name": "10.50.50.201"
 },
 {
 "name": "10.50.50.202"
 },
 {
 "name": "10.50.50.203"
 }
],
"objectCountostieredBytesCounttimestampuserCount

```

## Response Format

The response payload is a JSON-formatted *AuditData* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or this method-specific status code:

Status Code	Description
400	Region {region} is not valid

## 12.10.2. GET /system/bucketcount

**GET /system/bucketcount** Get count of buckets owned by a group's members

The request line syntax for this method is as follows.

```
GET /system/bucketcount?groupId=xxx
```

There is no request payload.

## Example Using cURL

The [example](#) below retrieves the count of buckets owned by users within the group "testgroup1".

```
curl -X GET -k -u sysadmin:public \
'https://localhost:19443/system/bucketcount?groupId=testgroup1'
```

The response payload is a text string, which in this example is as follows:

```
5
```

## Response Format

The response payload is as shown in the example above. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or this method-specific status code:

Status Code	Description
204	No content

### 12.10.3. GET /system/bucketlist

#### GET /system/bucketlist Get list of buckets owned by a group's members

The request line syntax for this method is as follows.

```
GET /system/bucketlist?groupId=xxx
```

There is no request payload.

#### Example Using cURL

The [example](#) below retrieves the list of buckets owned by users within the group "testgroup1".

```
curl -X GET -k -u sysadmin:public \
'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"
 },
 {
 "bucketName": "bucket2",
 "createTime": "1554755542554"
 },
 {
 "bucketName": "bucket3",
 "createTime": "1554755548227"
 }
]
 },
 {
 "userId": "testuser2",
 "buckets": [
 {
 "bucketName": "testbucket4",
 "createTime": "1554755580759"
 },
 {
 "bucketName": "testbucket5",
 "createTime": "1554755587516"
 }
]
 }
]
```

## Response Format

The response payload is as shown in the example above. For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or this method-specific status code:

Status Code	Description
204	No content

### 12.10.4. GET /system/bytecount

**GET /system/bytecount** Get stored byte count for the system, a group, or a user

The request line syntax for this method is as follows.

```
GET /system/bytecount?groupId=xxx&userId=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see **"system Query Parameters"** (page 808).

There is no request payload.

**Note** The byte count is for "net" bytes. Overhead due to replication or erasure coding does not count toward this figure. For example, if a 1MB object is replicated three times in the system (as part of a replication storage policy), this counts as 1MB toward the total byte count -- not as 3MB.

**Note** If you want to retrieve the current byte count for a particular **bucket**, use the [POST /usage/repair/bucket](#) method.

## 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:public \
'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:public \
'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:public \
'https://localhost:19443/system/bytecount?groupId=Pubs&userId=PubsUser1'
```

The response payload is:

66712

## Response Format

The response payload is plain text (see examples above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or this method-specific status code:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}

### 12.10.5. GET /system/bytestiered

**GET /system/bytestiered** Get tiered byte count for the system, a group, or a user

The request line syntax for this method is as follows.

```
GET /system/bytestiered?groupID=xxx&userID=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see **"system Query Parameters"** (page 808).

There is no request payload.

**Note** 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.

## 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:public \
'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 GBs) or "<n>T" (for number of TBs) or "<n>P" (for number of PBs). If the tiered volume is currently less than 1GB then "0" is returned. If the tiered volume is

## Response Format

The response payload is a plain text string. For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or this method-specific status code:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}

## 12.10.6. GET /system/groupbytecount

**GET /system/groupbytecount** Get stored byte counts for all of a group's users

The request line syntax for this method is as follows.

```
GET /system/groupbytecount?groupID=xxx [&limit=xxx] [&offset=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**system Query Parameters**" (page 808).

There is no request payload.

**Note** The byte counts are for "net" bytes. Overhead due to replication or erasure coding does not count toward these figures. For example, if a 1MB object is replicated three times in the system (as part of a replication storage policy), this counts as 1MB toward the byte count -- not as 3MB.

### 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:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**UserUsage Object**" (page 814).

```
[
 {
 "canonicalUserId": "da870acdd136d60789fb5c761fef4a4a",
 "groupId": "Pubs",
 "usageVal": 220508,
 "userId": "brady"
 },
 {
 "canonicalUserId": "9bdcd44ce1f9266adb9f22a8313feb4",
 "groupId": "Pubs",
 "usageVal": 225365,
 "userId": "gilmore"
 },
 {
 "canonicalUserId": "9a00529cdfb6496a09c5105913b486ac",
 "groupId": "Pubs",
 "usageVal": 76744,
 "userId": "gronk"
 }
]
```

Response Format

The response payload is a JSON-formatted *UserUsage* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or this method-specific status code:

Status Code	Description
400	Missing required parameter: {groupId}
400	Limit should be greater than zero.

## 12.10.7. GET /system/groupobjectcount

**GET /system/groupobjectcount** Get stored object counts for all of a group's users

The request line syntax for this method is as follows.

```
GET /system/groupobjectcount?groupId=xxx [&limit=xxx] [&offset=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**system Query Parameters**" (page 808).

There is no request payload.

### 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:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**UserUsage Object**" (page 814).

```
[
 {
 "canonicalUserId": "da870acdd136d60789fb5c761fef4a4a",
 "groupId": "Pubs",
 "usageVal": 5,
 "userId": "brady"
 },
 {
 "canonicalUserId": "9bdcd44ce1f9266adb9f22a8313feb4",
 "groupId": "Pubs",
 "usageVal": 5,
 "userId": "gilmore"
 },
 {
 "canonicalUserId": "9a00529cdfb6496a09c5105913b486ac",
 "groupId": "Pubs",
 "usageVal": 2,
```

```

 "userId": "gronk"
}
]

```

## Response Format

The response payload is a JSON-formatted *UserUsage* object (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or this method-specific status code:

Status Code	Description
400	Missing required parameter: {groupId}
400	Limit should be greater than zero.

## 12.10.8. GET /system/license

### GET /system/license Get HyperStore license terms

The request line syntax for this method is as follows.

```
GET /system/license
```

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in **"RBAC Version of this Method"** (page 804).

**Note** For background information about HyperStore licensing, see **"Licensing and Auditing"** (page 6).

### 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:public \
https://localhost:19443/system/license | python -mjson.tool
```

The response payload is a JSON-formatted *LicenseData* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see **"LicenseData Object"** (page 811).

```
{
 "appliancesbucketLockModeenforcingexpirationgracePeriodmaxNetStoragemaxRawStorage

```

```
"maxTieredStoragestorageExceededstorageModetieringExceededwarnPeriod
```

## Response Format

The response payload is a JSON-formatted *LicenseData* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.10.8.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>2020-05-21T13:54:51.952-07:00</expiration>
etc...
...
...
```

```
</LicenseData>
</GetCloudianSystemLicenseResponse>
```

## 12.10.9. GET system/objectcount

### GET system/objectcount Get stored object count for the system, a group, or a user

The request line syntax for this method is as follows.

```
GET /system/objectcount?groupId=xxx&userId=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**system Query Parameters**" (page 808).

There is no request payload.

**Note** If you want to retrieve the current object count for a particular **bucket**, use the [POST /usage/repair/bucket](#) method.

## 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:public \
'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:public \
'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:public \
'https://localhost:19443/system/objectcount?groupId=Pubs&userId=PubsUser1'
```

The response payload is:

```
54
```

## Response Format

The response payload is plain text (see examples above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or this method-specific status code:

Status Code	Description
400	Missing required parameters: {groupId}, {userId}

### 12.10.10. GET /system/version

#### GET /system/version Get HyperStore system version

The request line syntax for this method is as follows.

```
GET /system/version
```

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 806).

#### Example Using cURL

The [example](#) below retrieves the HyperStore system version.

```
curl -X GET -k -u sysadmin:public https://localhost:19443/system/version
```

The response payload is the system version information in plain text, which in this example is as follows.

```
6.2 Compiled: 2017-07-20 11:13
```

#### Response Format

The response payload is plain text (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.10.10.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>
7.1 Compiled: 2018-08-16 16:32
</String>
</GetSystemVersionResponse>
```

### 12.10.11. POST /system/processProtectionPolicy

**POST /system/processProtectionPolicy** Process pending storage policy deletion or creation jobs

The request line syntax for this method is as follows.

```
POST /system/processProtectionPolicy
```

There is no request payload.

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 "**Storage Policy Deletion or Creation Processing**" (page 424) cron job.

#### Example Using cURL

The [example](#) below triggers the processing of any pending storage policy deletion or creation jobs.

```
curl -X POST -k -u sysadmin:public \
https://localhost:19443/system/processProtectionPolicy
```

#### Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.10.12. POST /system/repairusercount

**POST /system/repairusercount** Reconcile user counts in Redis and Cassandra

The request line syntax for this method is as follows.

```
POST /system/repairusercount
```

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 User-Info table.

There is no request payload.

#### Example Using cURL

The [example](#) below syncs the user counts in Redis with the Cassandra metadata.

```
curl -X POST -k -u sysadmin:public https://localhost:19443/system/repairusercount
```

#### Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700).

### 12.10.13. system Query Parameters

*groupId, userId*

(Mandatory, string) For the *GET /system/bytecount*, *GET /system/bytestiered*, and *GET system/objectcount* methods: 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*)

*groupId*

(Mandatory, string) For the *GET /system/groupbytecount* and *GET /system/groupobjectcount* methods: Use the groupId parameter to specify the group for which you want to retrieve counts for all users in the group.

*limit*

(Optional, integer) For the *GET /system/groupbytecount* and *GET /system/groupobjectcount* methods: 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) For the *GET /system/groupbybytecount* and *GET /system/groupobjectcount* methods: The user ID with which to start the response list of users for the current request, sorted alpha-numerically. 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.

*region*

(Optional, string) For the *GET /system/audit* method: 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.

## 12.10.14. system Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the System related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- **"AuditData Object"** (page 809)
- **"LicenseData Object"** (page 811)
- **"UserUsage Object"** (page 814)

### 12.10.14.1. AuditData Object

The *AuditData* object consists of the following attributes.

*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
```

#### 12.10.14.2. LicenseData Object

The *LicenseData* object consists of the following attributes.

*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
```

*bucketLockMode*

(String) The license's support or non-support of the HyperStore WORM (bucket lock) feature:

- DISABLED -- WORM is not supported.
- SEC17 -- WORM is supported, but having a WORM "privileged delete user" is not supported
- ENTERPRISE -- WORM is supported, and having a WORM "privileged delete user" is supported

For more information see "**WORM (Bucket Lock) Feature Overview**" (page 159).

Example:

```
"bucketLockMode": "SEC17"
```

*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**" (page 6).

Example:

```
"enforcing": true
```

#### *expiration*

(Number) License expiration date-time in UTC milliseconds. Example:

```
"expiration": 1590094491952
```

#### *gracePeriod*

(Number) After the license expiration date passes, the number of days until the HyperStore system is automatically disabled. Example:

```
"gracePeriod": 0
```

#### *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 1GB object protected by 3X replication counts as 3GB 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"
```

#### *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
```

### 12.10.14.3. UserUsage Object

The *UserUsage* object consists of the following attributes.

*canonicalUserId*

(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"
```

## 12.11. 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 Hyper-Store 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** Having a system default tiering credential is only supported for S3-compliant tiering destinations -- not for Azure or Spectra.

### 12.11.1. DELETE /tiering/credentials

**DELETE /tiering/credentials** Delete a tiering credential for Amazon, Google, or other S3-compliant destination

The request line syntax for this method is as follows.

```
DELETE /tiering/credentials[?bucketName=xxx]
```

For parameter description click on the parameter name or see "**[tiering Query Parameters](#)**" (page 821).

There is no request payload.

### 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:public \
https://localhost:19443/tiering/credentials?bucketName=bucket1
```

### Response Format

There is no response payload. For response status code this method will return one of the "**[Common Response Status Codes](#)**" (page 700).

## 12.11.2. DELETE /tiering/azure/credentials

### DELETE /tiering/azure/credentials Delete a tiering credential for Azure

The request line syntax for this method is as follows.

```
DELETE /tiering/azure/credentials?bucketName=xxx
```

For parameter description click on the parameter name or see "**[tiering Query Parameters](#)**" (page 821).

There is no request payload.

### 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:public \
https://localhost:19443/tiering/azure/credentials?bucketName=bucket2
```

### Response Format

There is no response payload. For response status code this method will return one of the "**[Common Response Status Codes](#)**" (page 700).

## 12.11.3. DELETE /tiering/spectra/credentials

### DELETE /tiering/spectra/credentials Delete a tiering credential for Spectra

The request line syntax for this method is as follows.

```
DELETE /tiering/spectra/credentials?bucketName=xxx
```

For parameter description click on the parameter name or see "**[tiering Query Parameters](#)**" (page 821).

There is no request payload.

### 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:public \
https://localhost:19443/tiering/spectra/credentials?bucketName=bucket1
```

### Response Format

There is no response payload. For response status code this method will return one of the ["Common Response Status Codes"](#) (page 700).

## 12.11.4. GET /tiering/credentials

**GET /tiering/credentials** Get a tiering credential for Amazon, Google, or other S3-compliant destination

The request line syntax for this method is as follows.

```
GET /tiering/credentials[?bucketName=xxx]
```

For parameter description click on the parameter name or see ["tiering Query Parameters"](#) (page 821).

There is no request payload.

### 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:public \
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
```

### Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the ["Common Response Status Codes"](#) (page 700) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

## 12.11.5. GET /tiering/credentials/src

**GET /tiering/credentials/src** Check whether a bucket uses a bucket-specific or

## system default tiering credential

The request line syntax for this method is as follows.

```
GET /tiering/credentials/src[?bucketName=xxx]
```

For parameter description click on the parameter name or see "**[tiering Query Parameters](#)**" (page 821).

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 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).

There is no request payload.

**Note** This method is not supported for buckets that tier to Azure or Spectra.

### 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:public \
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
```

### Response Format

The response payload is plain text string (see example above). For response status code this method will return one of the "**[Common Response Status Codes](#)**" (page 700).

## 12.11.6. GET /tiering/azure/credentials

### GET /tiering/azure/credentials Get a tiering credential for Azure

The request line syntax for this method is as follows.

```
GET /tiering/azure/credentials?bucketName=xxx
```

For parameter description click on the parameter name or see "**[tiering Query Parameters](#)**" (page 821).

There is no request payload.

### 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:public \
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,Oy1wMUklsF81331LIGY5R1Vqa8Rg+iWT6zEFT6I1
```

## Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

### 12.11.7. GET /tiering/spectra/credentials

#### GET /tiering/spectra/credentials Get a tiering credential for Spectra

The request line syntax for this method is as follows.

```
GET /tiering/spectra/credentials?bucketName=xxxx
```

For parameter description click on the parameter name or see **"tiering Query Parameters"** (page 821).

There is no request payload.

## 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:public \
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
```

## Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) or one of these method-specific status codes:

Status Code	Description
204	No Tiering Credentials found.

## 12.11.8. POST /tiering/credentials

**POST /tiering/credentials** Post a tiering credential for Amazon, Google, or other S3-compliant destination

The request line syntax for this method is as follows.

```
POST /tiering/credentials?accessKey=urlencode(xxx)&secretKey=urlencode(xxx)
[&bucketName=xxxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**tiering Query Parameters**" (page 821).

There is no request payload.

### Example Using cURL

The [example](#) below posts S3 auto-tiering credentials for a HyperStore source bucket named "bucket1".

```
curl -X POST -k -u sysadmin:public \
'https://
localhost:19443/tiering/credentials?accessKey=00cc33c4ble&secretKey=YuaOJ71OFqc&buck-
etName=bucket1'
```

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.

**Note** If an access key or secret key includes a non-alphanumeric character and you do not URL-encode the key, the API server will return a 403 error.

### Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accessKey, secretKey}

## 12.11.9. POST /tiering/azure/credentials

**POST /tiering/azure/credentials** Post a tiering credential for Azure

The request line syntax for this method is as follows.

```
POST /tiering/azure/credentials?accountName=urlencode (xxx)
&accountKey=urlencode (xxx) &bucketName=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**tiering Query Parameters**" (page 821).

There is no request payload.

## Example Using cURL

The [example](#) below posts Azure auto-tiering credentials for a HyperStore source bucket named "bucket2".

```
curl -X POST -k -u sysadmin:public \
'https://localhost:19443/tiering/azure/credentials?accountName=
123&accountKey=Oy1wMU&bucketName=bucket2'
```

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.

**Note** 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.

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accountName, accountKey}

## 12.11.10. POST /tiering/spectra/credentials

### POST /tiering/spectra/credentials Post a tiering credential for Spectra

The request line syntax for this method is as follows.

```
POST /tiering/spectra/credentials?accessKey=urlencode (xxx)
&secretKey=urlencode (xxx) &bucketName=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**tiering Query Parameters**" (page 821).

There is no request payload.

## Example Using cURL

The [example](#) below posts Spectra auto-tiering credentials for a HyperStore source bucket named "bucket1".

```
curl -X POST -k -u sysadmin:public \
'https://
localhost:19443/tiering/spectra/credentials?accessKey=00d5d&secretKey=PxvAH6Ks&buck-
etName=bucket1'
```

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.

**Note** If an access key or secret key includes a non-alphanumeric character and you do not URL-encode the key, the API server will return a 403 error.

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
204	Bucket does not exist.
400	Missing required attributes : {accessKey, secretKey}

### 12.11.11. tiering Query Parameters

#### *bucketName*

(Optional for S3-compliant tiering, mandatory for Azure or Spectra tiering. String) Name of the **HyperStore source bucket** that uses the credential for auto-tiering to a destination system.

For tiering to Amazon, Google, or other S3-compliant destinations, if the *bucketName* parameter is omitted then the request applies to the system default auto-tiering credential. For example, with the POST method for S3 tiering credentials, if you omit the bucket name then you are POSTing a system default credential for tiering to an S3 destination. With GET or DELETE methods if you omit the bucket name then you are retrieving or deleting the system default S3 tiering credential.

**Note** Having a system default tiering credential is only supported for S3-compliant tiering destinations -- not for Azure or Spectra.

#### *accessKey*

(Mandatory, string) Access key for the tiering destination account.

*secretKey*

(Mandatory, string) Secret key for the tiering destination account.

*accountName*

(Mandatory, string) Name of the Azure tiering destination account.

*accountKey*

(Mandatory, string) Account key for the Azure tiering destination account.

## 12.12. 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 "**Usage Reporting Feature Overview**" (page 153).

### 12.12.1. DELETE /usage

#### DELETE /usage Delete usage data

The request line syntax for this method is as follows.

```
DELETE /usage?granularity=xxx&startTime=xxx[&unitCount=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**usage Query Parameters**" (page 834).

There is no request payload.

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**" (page 157).

**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.

## 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:public \
'https://localhost:19443/usage?granularity=day&startTime=20170501&unitCount=1'
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

## 12.12.2. GET /usage

### GET /usage Get usage data for group, user, or bucket

The request line syntax for this method is as follows.

```
GET /usage?[id=xxx|canonicalUserId=xxx|bucket=xxx] &operation=xxx&startTime=xxx
&endTime=xxx&granularity=xxx&reversed=xxx [&limit=xxx] [&pageSize=xxx]
[&offset=xxx] [®ion=xxx] [®ionOffset=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[usage Query Parameters](#)" (page 834).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "[RBAC Version of this Method](#)" (page 825).

**Note** The `GET/usage?bucket=xxx...` 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 "[Enabling Non-Default Usage Reporting Features](#)" (page 156).

## 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:public \
'https://
localhost
:19443/usage?id=QA*&operation=SB&startTime=201707010000&endTime=201708010000&gran-
ularity=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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**UsageData Object**" (page 840).

```
[
 {
 "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:public \
'
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**UsageData Object**" (page 840).

```
[
 {
 "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.

## Response Format

The response payload is a JSON-formatted list of *UsageData* objects (see examples above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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}

### 12.12.2.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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=QA1*&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>
```

### 12.12.3. POST /usage/bucket

#### POST /usage/bucket Get raw usage data for multiple buckets

The request line syntax for this method is as follows.

```
POST /usage/bucket
```

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

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.

**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.

**Note** 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 ["Enabling Non-Default Usage Reporting Features"](#) (page 156).

#### 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:public \
-d @buckets_usage.txt https://localhost:19443/usage/bucket | python -mjson.tool
```

The *buckets\_usage.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see ["UsageBucketReq Object"](#) (page 839).

```
{
 "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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see ["UsageBucketRes Object"](#) (page 839).

**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.

```
[
 {
 "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"
 },
 ...
 ...
]
 }
]
```

## Response Format

The response payload is a JSON-formatted list of *UsageBucketRes* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required attributes : {buckets, startTime, endTime}
400	Invalid JSON Object

### 12.12.4. POST /usage/repair

#### POST /usage/repair Repair storage usage data for group or system

The request line syntax for this method is as follows.

```
POST /usage/repair?groupId=xxx [&summarizeCountsOnly=xxx] [®ion=xxxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**usage Query Parameters**" (page 834).

There is no request payload.

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.

For background information on storage usage data repair, see "**Validating Usage Data for Storage Levels**" (page 157).

**Note** 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.

## Example Using cURL

The [example](#) below checks and repairs storage usage data for the "engineering" group.

```
curl -X POST -k -u sysadmin:public \
https://localhost:19443/usage/repair?groupId=engineering
```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700).

### 12.12.5. POST /usage/repair/bucket

**POST /usage/repair/bucket** Retrieve total bytes and total objects for a bucket

The request line syntax for this method is as follows.

```
POST /usage/repair/bucket?bucket=xxx
```

For parameter description click on the parameter name or see "**usage Query Parameters**" (page 834).

There is no request payload.

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 Cassandra metadata for objects in the bucket.

**Note** This is potentially a resource-intensive operation, depending on how many objects are in the bucket.

**Note** This API method is supported even if bucket usage statistics are disabled in the system. Bucket usage statistics are disabled in the system by default. For more information on bucket statistics see "**Bucket Usage Statistics**" (page 154).

## 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:public \
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
}
```

## Response Format

The response payload is the TB and TO values in JSON (see example above). For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700).

### 12.12.6. POST /usage/repair/dirtyusers

**POST /usage/repair/dirtyusers**   Repair storage usage data for users with recent activity

The request line syntax for this method is as follows.

```
POST /usage/repair/dirtyusers[?summarizeCounts=xxx]
```

For parameter description click on the parameter name or see "["usage Query Parameters"](#)" (page 834).

There is no request payload.

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 configurable maximum number of those users per method execution (*mts.properties.erb: usage.repair.maxdirtyusers*; default = 1000).

For background information on storage usage data repair, see "["Validating Usage Data for Storage Levels"](#)" (page 157).

**Note** This method is invoked once every 12 hours by a HyperStore "["Usage Data Processing"](#)" (page 421) 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.

## 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:public https://localhost:19443/usage/repair/dirtyusers
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

## 12.12.7. POST /usage/repair/user

### POST /usage/repair/user Repair storage usage data for a user

The request line syntax for this method is as follows.

```
POST /usage/repair/user?groupID=xxx&userId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[usage Query Parameters](#)" (page 834).

There is no request payload.

This method checks and repairs storage usage data for a single specified user. For background information on storage usage data repair, see "[Validating Usage Data for Storage Levels](#)" (page 157).

**Note** 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.

## 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:public \
'https://localhost:19443/usage/repair/user?groupID=engineering&userId=gladdes'
```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

### 12.12.8. POST /usage/rollup

#### POST /usage/rollup Roll up usage data

The request line syntax for this method is as follows.

```
POST /usage/rollup?granularity=xxx&startTime=xxx&unitCount=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**usage Query Parameters**" (page 834).

There is no request payload.

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** The *POST /usage/rollup* method is called regularly by HyperStore "**Usage Data Processing**" (page 421) 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.

## 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:public \
'https://
localhost:19443/usage/rollup?granularity=hour&startTime=2017081500&unitCount=1'
```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing or invalid parameters

## 12.12.9. POST /usage/storage

### POST /usage/storage Post raw storage usage data for users with recent activity

The request line syntax for this method is as follows.

```
POST /usage/storage
```

There is no request payload.

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" key-space. 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 5 minutes by a HyperStore "Usage Data Processing" (page 421) 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.

### 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:public https://localhost:19443/usage/storage
```

### Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700).

## 12.12.10. POST /usage/storageall

### POST /usage/storageall Post raw storage usage data for all users

The request line syntax for this method is as follows.

```
POST /usage/storageall
```

There is no request payload.

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" (page 421) 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.

## 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:public https://localhost:19443/usage/storageall
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700).

### 12.12.11. usage Query Parameters

#### granularity

(Mandatory, string) With a *GET /usage* or *POST /usage/rollup* or *DELETE /usage* request: The time period granularity of the usage data to retrieve or generate or delete. Supported values are:

- *hour* — Hourly rollup data
- *day* — Daily rollup data
- *month* — Monthly rollup data
- *raw* — Raw transactional data (not rolled up). This is supported only for a GET or DELETE.

**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.

#### 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.

With a *POST /usage/rollup* or *DELETE /usage* request 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. This level of granularity is not supported for `POST /usage/rollup`.

*unitCount*

(Optional, integer) With a `POST /usage/rollup` or `DELETE /usage` request: The number of units of the specified "granularity" to generate or 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.

*endTime*

(Mandatory, string) With a `GET /usage` request: 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.

*id*

(Optional, string) With a `GET /usage` request: 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) With a `GET /usage` request: 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*

With a `GET /usage` request (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.

With *POST /usage/repair/bucket* (Mandatory, string): The bucket name.

**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 "**Enabling Non-Default Usage Reporting Features**" (page 156).

#### *operation*

(Mandatory, string) With a *GET /usage* request: 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 "**Enabling Non-Default Usage Reporting Features**" (page 156).

- *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.

#### *reversed*

(Optional, boolean) With a *GET /usage* request: 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) With a *GET /usage* request: 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) With a *GET /usage* request: 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) With a *GET /usage* request: 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)

With a *GET /usage* request: 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).

With a *POST /usage/repair* or *POST /usage/repair/user* request: 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.

**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) With a *GET /usage* request: 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.

#### *groupId*

(Mandatory, string)

With a *POST /usage/repair* request: The group for which to repair user-level and group-level storage usage counts. If groupId is "ALL", repair is performed for all groups.

With a *POST /usage/repair/user* request: The ID of the group to which the target user belongs.

#### *summarizeCountsOnly*

(Optional, boolean) With a *POST /usage/repair* request: 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.

#### *summarizeCounts*

(Optional, boolean) With a *POST /usage/repair/dirtyusers* request: 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".

*userId*

(Mandatory, string) With a *POST /usage/repair/user* request: The ID of the user for whom usage data repair is to be performed.

## 12.12.12. usage Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Usage related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- "**UsageBucketReq Object**" (page 839)
- "**UsageBucketRes Object**" (page 839)
- "**UsageData Object**" (page 840)

### 12.12.12.1. UsageBucketReq Object

The *UsageBucketReq* object consists of the following attributes:

*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"
```

### 12.12.12.2. UsageBucketRes Object

The *UsageBucketRes* object consists of the following attributes:

*bucket*

(String) Bucket with which the usage data is associated. Example:

```
"bucket": "b123"
```

*data*

(List<UsageData>) List of [UsageData](#) objects. Note that in the context of a *UsageBucketRes* object, the *UsageData* objects will always be for "raw" granularity. Example:

```
"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"
 }
]
```

### 12.12.12.3. UsageData Object

The *UsageData* object consists of the following attributes:

#### *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 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"
```

## Usage Data Topics

### 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
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 located, 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-	Decrement by size of old tag-	No change	No change	No	No

Operation	Storage Bytes	Storage Objects	Num Requests	Bytes IN	Bytes OUT
ping	ping string			change	change
DELETE policy	No change	No change	No change	No change	No change
DELETE uploadId (MP Abort)	If successfully deleted, decremented by Total-Size of uploaded parts and 1V value added in MP initiate	If successfully deleted, decremented by 1	No change	No change	No change
GET bucket, service, policy, location, acl, bucketlogging, versioning, 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 complete	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 replacement object, decrement 1	Incremented by 1	Add Transfer-Size of request	Add Transfer-Size of response
POST 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 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.	No change	Incremented	Add	Add

Operation	Storage Bytes	Storage Objects	Num Requests	Bytes IN	Bytes OUT
	If replacing an existing part, subtract Content-Length of old part body.		by 1	Transfer-Size of request	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 <a href="#"><u>(cross-region replication)</u></a>	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 1MB object and the request fails to complete, the 1MB 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

The HyperStore system supports an S3 "**PUT Bucket lifecycle**" (page 916) API extension whereby objects can, on a specified scheduled, be auto-tiered to Amazon S3, Amazon Glacier, a different Cloudian HyperStore service region, or a third party Cloudian HyperStore system. When an object is transitioned to Amazon or a different HyperStore region or 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 — 8KB,

regardless of the transitioned object size — is added to the local Storage Bytes count. For example, if a 100KB object is auto-tiered to Amazon or a different HyperStore region or system, the net local effect is a 92KB 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 8KB 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 8KB 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 8KB 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.

## 12.13. user

The Admin API methods built around the **user** resource are for managing HyperStore service user accounts. This includes support for creating, changing, and deleting user accounts. These methods also support management of users' security credentials and the assignment of rating plans to users.

### 12.13.1. DELETE /user

#### DELETE /user Delete a user

The request line syntax for this method is as follows.

```
DELETE /user?userId=xxx&groupId=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "["user Query Parameters"](#) (page 873).

There is no request payload.

The user will be removed from his group; his security credentials will be deleted; and his S3 buckets and stored objects will be deleted. **Once deleted, a user's buckets and objects will not be recoverable.**

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 `reports.rollup.ttl` setting in `mts.properties.erb`. You can retrieve usage data for a recently deleted user via the [GET /usage](#) method.

**Note** You cannot delete the default system administrator account. This is not allowed.

## Example Using cURL

The [example](#) below deletes a user with ID "John" who is in the "QA" group.

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

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId}
400	User does not exist

## 12.13.2. DELETE /user/credentials

### DELETE /user/credentials Delete a user's S3 security credential

The request line syntax for this method is as follows.

```
DELETE /user/credentials?accessKey=xxx
```

For parameter description click on the parameter name or see "[user Query Parameters](#)" (page 873).

There is no request payload.

## 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:public \
https://localhost:19443/user/credentials?accessKey=21289bab1738ffdc792a
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {accessKey}

Status Code	Description
400	Invalid Access Key

### 12.13.3. DELETE /user/deleted

#### DELETE /user/deleted Purge profile data of a deleted user or users

The request line syntax for this method is as follows.

```
DELETE /user/deleted[?canonicalUserId=xxx|groupId=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "["user Query Parameters"](#) (page 873).

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 a *canonicalUserId* and a *groupId* together.

There is no request payload.

**Note** 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.

#### Example Using cURL

The [example](#) below purges a single deleted user's profile data.

```
curl -X DELETE -k -u sysadmin:public \
https://localhost:19443/user/deleted?canonicalUserId=bd0796cd9746ef9cc4ef656ddaacfac4
```

#### Response Format

There is no response payload. For response status code this method will return one of the "["Common Response Status Codes"](#) (page 700) 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.

#### 12.13.4. GET /user

##### GET /user Get a user's profile

The request line syntax for this method is as follows.

```
GET /user?userId=xxx&groupId=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 853).

#### Example Using cURL

The [example](#) below retrieves a user with ID "John" who is in the "QA" group.

```
curl -X GET -k -u sysadmin:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**Userinfo Object**" (page 876).

```
{
 "active": "true",
 "address1": "",
 "address2": "",
 "canonicalUserId": "bd0796cd9746ef9cc4ef656ddaacfac4",
 "city": "",
 "country": "",
 "emailAddr": "",
 "fullName": "John Thompson",
 "groupId": "QA",
 "ldapEnabled": false,
 "phone": "",
 "state": "",
 "userId": "John",
 "userType": "User",
 "website": "",
 "zip": ""
}
```

#### Response Format

The response payload is a JSON-formatted *Userinfo* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
204	User does not exist
400	Missing Required parameters : {userId, groupId}

#### 12.13.4.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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>
```

### 12.13.5. GET /user/credentials

#### GET /user/credentials Get a user's S3 security credential

The request line syntax for this method is as follows.

```
GET /user/credentials?accessKey=xxx
```

For parameter description click on the parameter name or see "**user Query Parameters**" (page 873).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 855).

#### 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:public \
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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**SecurityInfo Object**" (page 876).

```
{
 "accessKey": "009c156c79e64e0e4928",
 "active": true,
 "createDate": 1502279336024,
 "expireDate": null,
 "secretKey": "wVHk2nA0M03RWSM1rFHFAtuhow6S1DKN0gWjPhDG"
}
```

#### Response Format

The response payload is a JSON-formatted *SecurityInfo* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
204	No Data Found

Status Code	Description
400	Missing required parameters : {accessKey}

### 12.13.5.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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 Hyper-Store 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>

```

### 12.13.6. GET /user/credentials/list

#### GET /user/credentials/list Get a user's list of S3 security credentials

The request line syntax for this method is as follows.

```
GET /user/credentials/list?[userId=xxx&groupId=xxx | canonicalUserId=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

This retrieves all of the user's S3 credentials (active credentials as well as inactive [disabled] credentials).

Specify the user either by using *userId* and *groupId*, or by using *canonicalUserId*.

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 857).

#### 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:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**SecurityInfo Object**" (page 876).

```
[
 {
 "accessKeyactivecreateDateexpireDatesecretKey

```

```

 "createDate": 1502283467021,
 "expireDate": null,
 "secretKey": "o5jqJtqV36+sENGLozEUg1EXEmQp9V6yfCHLFCJk"
}
]

```

## Response Format

The response payload is a JSON-formatted list of *SecurityInfo* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

### 12.13.6.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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-
s="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>
```

### 12.13.7. GET /user/credentials/list/active

**GET /user/credentials/list/active    Get a user's list of active S3 security credentials**

The request line syntax for this method is as follows.

```
GET /user/credentials/list/active?[userId=xxx&groupId=xxx | canonicalUserId=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

This retrieves the user's active S3 credentials. Inactive (disabled) credentials are not returned.

Specify the user either by using *userId* and *groupId*, or by using *canonicalUserId*.

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "**RBAC Version of this Method**" (page 859).

#### Example Using cURL

The [example](#) below retrieves the active S3 credentials for user "John" in the "QA" group.

```
curl -X GET -k -u sysadmin:public \
'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). For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**SecurityInfo Object**" (page 876).

```
[
 {
 "accessKey": "009c156c79e64e0e4928",
 "active": true,
 "createDate": 1502279336024,
 "expireDate": null,
 "secretKey": "wVHk2nA0M03RWSM1rFHFAtuhow6S1DKN0gWjPhDG
 }
]
```

## Response Format

The response payload is a JSON-formatted list of *SecurityInfo* objects (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

### 12.13.7.1. 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 "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

- 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-
s="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>
```

## 12.13.8. GET /user/list

### GET /user/list Get a list of user profiles

The request line syntax for this method is as follows.

```
GET /user/list?group_id=xxx&user_type=xxx&user_status=xxx [&prefix=xxx] [&limit=xxx]
[&offset=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[user Query Parameters](#)" (page 873).

There is no request payload.

**Note** The HyperStore IAM Service supports a role-based access version of this API method as described in "[RBAC Version of this Method](#)" (page 862).

## Example Using cURL

The [example](#) below retrieves a list of all active users in the "QA" group.

```
curl -X GET -k -u sysadmin:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "[UserInfo Object](#)" (page 876).

```
[
 {
 "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",
 "zip": ""
 }
]
```

```

"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": ""
}
]

```

## Response Format

The response payload is a JSON-formatted list of *UserInfo* objects (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) 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

### 12.13.8.1. 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 **"IAM Extensions for Role-Based Access to HyperStore Admin Functions"** (page 95).

- 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 Hyper-Store 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&User-Status=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>
```

## 12.13.9. GET /user/password/verify

### GET /user/password/verify Verify a user's CMC password

The request line syntax for this method is as follows.

```
GET /user/password/verify?userId=xxx&groupId=xxx&password=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

This verifies that the supplied CMC password is the correct password for the user.

There is no request payload.

### Example Using cURL

The [example](#) below verifies the supplied password for a user "John" in the "QA" group.

```
curl -X GET -k -u sysadmin:public \
 'https://localhost:19443/user/password/verify?userId=John&groupId=QA&password=P1a2s3s4!'
```

The response payload is a plain text value "true" or "false", which in this example is as follows.

```
true
```

The "true" response indicates that the supplied password is the correct password for the user.

### Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or this method-specific status code:

Status Code	Description
400	Missing Required parameters : {userId, groupId, password}

## 12.13.10. GET /user/ratingPlan

### GET /user/ratingPlan Get a user's rating plan content

The request line syntax for this method is as follows.

```
GET /user/ratingPlan?userId=xxx&groupId=xxx[®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

### Example Using cURL

The [example](#) below retrieves the content of the rating plan that is assigned to user "John" in group "QA".

```
curl -X GET -k -u sysadmin:public \
 '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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**RatingPlan Object**" (page 794).

```
{
 "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}
```

## Response Format

The response payload is a JSON-formatted *RatingPlan* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

### 12.13.11. GET /user/ratingPlanId

#### GET /user/ratingPlanId Get a user's rating plan ID

The request line syntax for this method is as follows.

```
GET /user/ratingPlanId?userId=xxx&groupId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

#### Example Using cURL

The example below retrieves the rating plan ID for user "John" in group "QA".

```
curl -X GET -k -u sysadmin:public \
'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
```

## Response Format

The response payload is a plain text string (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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

## 12.13.12. POST /user

### POST /user Change a user's profile

The request line syntax for this method is as follows.

```
POST /user
```

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

#### 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:public \
-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 of the default system administrator account. This is not allowed.

#### Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	User does not exist
400	Missing Required parameters : {userId, groupId, userType}
400	Invalid JSON object
400	Invalid User Name

### 12.13.13. POST /user/credentials

**POST /user/credentials** Post a user's supplied S3 credential

The request line syntax for this method is as follows.

```
POST /user/credentials?userId=xxxx&groupId=xxxx&accessKey=xxxx&secretKey=xxxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

#### 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:public \
'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.

#### Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing Required parameters : {userId, groupId, accessKey, secretKey}
400	User does not exist
403	Reached maximum number of credentials allowed
409	Access Key already exists

### 12.13.14. POST /user/credentials/status

**POST /user/credentials/status** Deactivate or reactivate a user's S3 credential

The request line syntax for this method is as follows.

```
POST /user/credentials/status?accessKey=xxx&isActive=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

## 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:public \
'https://localhost:19443/user/credentials/status?accessKey=21289bab1738ffdc792a&isActive=false'
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required parameters : {accessKey}
400	Invalid Access Key

### 12.13.15. POST /user/password

#### POST /user/password Create or change a user's CMC password

The request line syntax for this method is as follows.

```
POST /user/password?userId=xxx&groupId=xxx&password=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "[user Query Parameters](#)" (page 873).

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.

There is no request payload.

Passwords must meet the following conditions:

- Minimum of nine characters, maximum of 64 characters
- Must contain at least three of these four types of characters:
  - Lower case letters
  - Upper case letters
  - Numbers
  - Special characters such as !, @, #, \$, %, ^, etc.

## Example Using cURL

The [example](#) below posts a CMC password for the user "John" in the "QA" group.

```
curl -X POST -k -u sysadmin:public \
'https://localhost:19443/user/password?userId=John&groupId=QA&password=Pla2s3s4!'
```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) 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.

### 12.13.16. POST /user/ratingPlanId

#### POST /user/ratingPlanId Assign a rating plan to a user

The request line syntax for this method is as follows.

```
POST /user/ratingPlanId?userId=xxx&groupId=xxx&ratingPlanId=xxx [®ion=xxx]
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

#### Example Using cURL

The [example](#) below assigns the "Gold" rating plan to user "John" in the "QA" group.

```
curl -X POST -k -u sysadmin:public \
'https://localhost:19443/user/ratingPlanId?userId=John&groupId=QA&ratingPlanId=Gold'
```

## Response Format

There is no response payload. For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

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

### 12.13.17. PUT /user

#### PUT /user Create a new user

The request line syntax for this method is as follows.

```
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 create a CMC password for the user.

## 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:public \
-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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "[UserInfo Object](#)" (page 876).

Request payload:

```
{
 "active": "true",
 "address1": "",
 "address2": "",
 "city": "",
 "country": "",
 "emailAddr": "",
 "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": "",
 "fullName": "John Thompson",
 "groupId": "QA",
```

```
 "ldapEnabled": false,
 "phone": "",
 "state": "",
 "userId": "John",
 "userType": "User",
 "website": "",
 "zip": ""
}
```

## Response Format

The response payload is a JSON-formatted *UserInfo* object (see example above). For response status code this method will return one of the "**Common Response Status Codes**" (page 700) or one of these method-specific status codes:

Status Code	Description
400	Missing required attributes : {userId, groupId, userType}
400	Invalid JSON object
400	Group Id does not exist
400	User Id is not allowed : {userId}
400	Invalid User Name
409	Unique constraint violation : {userId}

### 12.13.18. PUT /user/credentials

#### PUT /user/credentials Create a new S3 credential for a user

The request line syntax for this method is as follows.

```
PUT /user/credentials?userId=xxx&groupId=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "**user Query Parameters**" (page 873).

There is no request payload.

#### Example Using cURL

The [example](#) below creates a new S3 credential for user "John" in the "QA" group.

```
curl -X PUT -k -u sysadmin:public \
'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. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "**SecurityInfo Object**" (page 876).

```
{
 "accessKey": "28d945de2a2623fc9483",
```

```

 "active": true,
 "createDate": 1502285593100,
 "expireDate": null,
 "secretKey": "j2OrPGHF69hp3YsZHRHOCWdAQDabppsBtD7kttr9"
}

```

## Response Format

The response payload is a JSON-formatted `SecurityInfo` object (see example above). For response status code this method will return one of the **"Common Response Status Codes"** (page 700) 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

### 12.13.19. user Query Parameters

#### `userId`

(Optional for methods that also support a `canonicalUserId` parameter and mandatory for methods that do not; string) Unique identifier of the user. This is the user ID that was supplied by the user (or whoever created the user) at the time of user creation -- not the `canonicalUserId` that is automatically generated by the system when a new user is created.

#### `groupId`

(Optional for methods that also support a `canonicalUserId` parameter and mandatory for methods that do not; string) Unique identifier of the group to which the user belongs.

**Note** The group ID for system admins is "0".

#### `canonicalUserId`

(Optional, string) System-generated canonical user ID of the user.

If you don't know the user's canonical ID you can retrieve it via the Admin API method `GET /user/list`.

#### `accessKey`

(Mandatory, string) With a `POST`, `GET`, or `DELETE /user/credentials` request: The S3 access key.

**Note** An S3 security credential is a key pair consisting of an "access key" (public key) and a "secret key" (private key).

#### `secretKey`

(Mandatory, string) With a `POST /user/credentials` request: The S3 secret key.

#### `isActive`

(Optional, boolean) With a *POST /user/credentials/status* request: The status to apply to the credentials — *true* for active or *false* for inactive. Defaults to *false* if this query parameter is not supplied in the request.

#### *userType*

(Mandatory, string) With a *GET /user/list* request: Retrieve users of this type. Options are:

- *admin* — Administrators. If the group ID is "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, string) With a *GET /user/list* request: 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 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.
- *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.
- *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.

#### *prefix*

(Optional, string) With a *GET /user/list* request: 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) With a *GET /user/list* request: 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) With a *GET /user/list* request: 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.

#### *ratingPlanId*

(Mandatory, string) With a *POST /user/ratingPlanId* request: 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.

#### *password*

(Mandatory, string) With a *POST /user/password* or *GET /user/password/verify* request: The user's supplied CMC password.

Passwords must meet the following conditions:

- Minimum of nine characters, maximum of 64 characters
- Must contain at least three of these four types of characters:
  - Lower case letters
  - Upper case letters
  - Numbers
  - Special characters such as !, @, #, \$, %, ^, etc.

## 12.13.20. user Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the User related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- **"SecurityInfo Object"** (page 876)
- **"UserInfo Object"** (page 876)

### 12.13.20.1. SecurityInfo Object

The *SecurityInfo* object consists of the following attributes:

#### 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"
```

### 12.13.20.2. UserInfo Object

The *Userinfo* object consists of the following attributes:

#### 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."
```

*address2*

(Optional, string) User's street address line 2. Example:

```
"address2": ""
```

*canonicalUserId*

(Optional, string) Canonical user ID, globally unique within the HyperStore system. This is automatically generated by the system when a new user is created.

- For users created prior to HyperStore 3.0 (before the canonical ID feature existed), the canonicalUserId is the user's <groupId>|<userId> combination.
- For users created in HyperStore 3.0 and later, the canonicalUserId is a system-generated, globally unique printable string. 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.
- 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"
```

*fullName*

(Optional, string) User's full name. Alphanumeric, maximum length 64 characters. Example:

```
"fullName": "John Thompson"
```

*groupId*

(Mandatory, string) Group ID of the group to which the user belongs.

Note that the groupId of the system admin group is "0".

Example:

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

*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 Feature Overview**" (page 101). 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, and underscores are allowed.
- Length must be at least 1 character and no more than 64 characters.
- The following IDs are reserved for system use and are not available to individual users: "anonymous", "public", "null", "none", "admin", "0".

Example:

```
"userId": "John"
```

*userType*

(Mandatory, string) User type. One of {"User", "GroupAdmin", "SystemAdmin"}. Example:

```
"userType": "User"
```

*website*

(Optional, string) User's website URL. Example:

```
"website": "www.me.com"
```

*zip*

(Optional, string) User's postal zip code. Example:

```
"zip": "12345"
```

**Note** In the "address1", "address2", "zip", "email", "website", and "phone" fields, the Admin Service prohibits the use of any of these characters:

` | ; & > <

**Note** The *PUT/user* 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 create a CMC password for the user.

## 12.14. whitelist

The Admin API methods built around the **whitelist** resource are for managing a billing "whitelist" 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 whitelist feature, including how to enable the feature, see "**Creating a "Whitelist" for Free Traffic**" (page 66). The whitelist feature is disabled by default.

**Note** If you are using load balancers in front of the HyperStore S3 Service, the whitelist 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 "**s3\_proxy\_protocol\_enabled**" (page 475) in [common.csv](#). 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 whitelist feature. You must use PROXY Protocol if you have load balancers in front of the S3 Service and want to use the whitelist feature .

### 12.14.1. GET /whitelist

#### GET /whitelist Get whitelist content

The request line syntax for this method is as follows.

```
GET /whitelist?whitelistId=Default-WL
```

For parameter description click on the parameter name or see "**whitelist Query Parameters**" (page 882).

There is no request payload.

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

## Example Using cURL

The [example](#) below retrieves the current contents of the whitelist with ID "Default-WL".

```
curl -X GET -k -u sysadmin:public \
https://localhost:19443/whitelist?whitelistId=Default-WL | python -mjson.tool
```

The response payload is a JSON-formatted *Whitelist* object, which in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "["Whitelist Object"](#) (page 883).

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

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

## Response Format

The response payload is a JSON-formatted *Whitelist* object (see example above). There is no response payload. For response status code this method will return one of the "["Common Response Status Codes"](#) (page 700) or one of these method-specific status codes:

Status Code	Description
204	Whitelist does not exist
400	Missing required parameter : whitelistId

## 12.14.2. POST /whitelist

### POST /whitelist Change whitelist content (by request body object)

The request line syntax for this method is as follows.

```
POST /whitelist
```

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

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

## Example Using cURL

The [example](#) below uploads whitelist content as specified in the request body. In this example the JSON-formatted *Whitelist* object is specified in a text file named *default\_whitelist.txt* which is then referenced as the data input to the cURL command.

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

The *default\_whitelist.txt* file content in this example is as follows. For description of a particular attribute, click on the attribute name; or for the full list of attribute descriptions see "["Whitelist Object"](#) (page 883).

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

## Response Format

There is no response payload. For response status code this method will return one of the "["Common Response Status Codes"](#) (page 700) or one of these method-specific status codes:

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

### 12.14.3. POST /whitelist/list

#### POST /whitelist/list Change whitelist content (by query parameters)

The request line syntax for this method is as follows.

```
POST /whitelist/list?whitelistId=xxx&list=xxx
```

For description of a particular parameter, click on the parameter name; or for the full list of parameter descriptions see "["whitelist Query Parameters"](#) (page 882).

There is no request payload.

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

## Example Using cURL

The [example](#) below replaces the existing whitelist contents with a new list of IP addresses.

```
curl -X POST -k -u sysadmin:public \
'https://localhost:19443/whitelist/list?whitelistId=Default-
WL&list=10.20.2.10,10.20.2.11,10.20.2.12'
```

## Response Format

There is no response payload. For response status code this method will return one of the "[Common Response Status Codes](#)" (page 700) or one of these method-specific status codes:

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

## 12.14.4. whitelist Query Parameters

### *whitelistId*

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

### *list*

(Mandatory, string) With a *POST /whitelist/list* request: A comma-separated list of IP addresses or subnets. This list will **overwrite** the existing whitelist 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.

## 12.14.5. whitelist Objects

This section describes the JSON-formatted data objects that constitute the request or response payloads for the Whitelist related Admin API methods.

**Note** For examples of these objects see the API method request and response examples.

- ["Whitelist Object"](#) (page 883)

### 12.14.5.1. Whitelist Object

The *Whitelist* object consists of the following attributes:

*id*

(Mandatory, string) Unique ID of the whitelist.

In the current HyperStore release only one whitelist 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 whitelist. The default whitelist object has display name "Default Whitelist". This is editable.

Example:

```
"name": "Default Whitelist"
```

*ratingPlanId*

(Mandatory, string) Unique ID of the rating plan assigned to the whitelist. The default whitelist object is assigned rating plan "Whitelist-RP". This system-provided default whitelist 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 whitelist. You can also edit the "Whitelist-RP" rating plan, using the usual rating plan APIs.

Example:

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

This page left intentionally blank

# Chapter 13. S3 API

## 13.1. Introduction

### 13.1.1. HyperStore Support for the Amazon S3 API

The Cloudian HyperStore system supports the great majority of the Amazon S3 REST API, including advanced features.

This documentation provides the details of the HyperStore system's compliance with the S3 REST API. The organization of this documentation parallels that of the Amazon S3 API Reference. Links are provided to specific parts of the Amazon S3 API Reference so you can easily view additional information about individual API operations.

This document takes the approach of specifying in detail the things that the HyperStore system **does support** from the Amazon S3 REST API — from service, bucket, and object operations, down to the level of particular request parameters, request headers, request elements, response headers, response elements, and special errors. **If it's not listed in this HyperStore S3 API Support documentation, the HyperStore system does not currently support it.**

This document also describes ways in which the HyperStore system extends the Amazon S3 API, to support additional functionality. Most of these extensions are in the form of additional request headers that add enhanced functionality to standard Amazon S3 operations on buckets. These extensions are described within the sections that document HyperStore compliance with standard Amazon S3 operations. The extensions are always identified by a sub-heading that says **HyperStore Extension to the S3 API**. (For a summary of the extensions see "**HyperStore Extensions to the S3 API**" (page 890).)

**Note** For high-level information about your options for utilizing the HyperStore S3 API see:

- \* "**Using the CMC as Your S3 Client**" (page 115)
- \* "**Using Third Party S3 Applications**" (page 116)
- \* "**Developing Custom S3 Applications for HyperStore**" (page 116)

### 13.1.2. Error Responses

From the "Error Responses" section of the Amazon S3 API, the HyperStore system supports the error codes listed below. If an error code from the Amazon S3 API specification is not listed below, the HyperStore system does not support it. For error descriptions and for mapping of these errors to HTTP status codes, refer to the "["Error Responses" section of the Amazon S3 API](#)".

**Note** The HyperStore system supports the same REST error format specified in the "["REST Error Responses" section of the Amazon S3 API](#)".

- AccessDenied
- AccountProblem
- AmbiguousGrantByEmailAddress

- BadDigest
- BucketAlreadyExists
- BucketAlreadyOwnedByYou
- BucketNotEmpty
- CrossLocationLoggingProhibited
- EntityTooLarge
- EntityTooSmall
- IllegalVersioningConfigurationException
- IncorrectNumberOfFilesInPostRequest
- InternalError
- InvalidAccessKeyId
- InvalidArgument
- InvalidBucketName
- InvalidBucketState
- InvalidDigest
- InvalidEncryptionAlgorithmError
- InvalidLocationConstraint
- InvalidObjectState
- InvalidPart
- InvalidPartOrder
- InvalidPolicyDocument
- InvalidRange
- InvalidRequest
- InvalidSecurity
- InvalidTargetBucketForLogging
- InvalidURI
- KeyTooLong
- MalformedACLError
- MalformedPOSTRequest
- MalformedXML
- MaxMessageLengthExceeded
- MaxPostPreDataLengthExceededError
- MetadataTooLarge
- MethodNotAllowed
- MissingContentLength
- MissingSecurityHeader
- NoSuchBucket
- NoSuchBucketPolicy
- NoSuchKey

- NoSuchLifecycleConfiguration
- NoSuchReplicationConfiguration
- NoSuchUpload
- NoSuchVersion
- NotImplemented
- PermanentRedirect
- PreconditionFailed
- Redirect
- RestoreAlreadyInProgress
- RequestIsNotMultiPartContent
- RequestTimeout
- RequestTimeTooSkewed
- SignatureDoesNotMatch
- ServiceUnavailable
- SlowDown
- TemporaryRedirect
- TooManyBuckets
- UnexpectedContent
- UnresolvableGrantByEmailAddress
- UserKeyMustBeSpecified

### 13.1.3. Authenticating Requests (AWS Signature Version 4)

The HyperStore system supports AWS Signature Version 4 for authenticating inbound API requests. The HyperStore implementation of this feature is compliant with Amazon's specification of the feature. For example, you can express authentication information in the HTTP Authorization header or in query string parameters; and you can compute a checksum of the entire payload prior to transmission, or for large uploads, you can use chunked upload.

For more information on this Amazon S3 feature, refer to the ["Authenticating Requests \(AWS Signature Version 4\)" section of the Amazon S3 REST API](#).

**Note** If you use chunked upload, each chunk must be no larger than the configurable HyperStore setting **"System Settings"** (page 297) (editable in the CMC Configuration Settings page). The default is 10MB. Requests with chunks larger than this will be rejected with an HTTP 400 Bad Request error.

**Note** For HyperStore, the region name validation aspect of Signature Version 4 is disabled by default. You can enable it with the **"cloudian.s3.authorizationV4.singleregioncheck"** (page 503) and/or **"cloudian.s3.authorizationV4.multiregioncheck"** (page 503) settings in *mts.properties.erb*. Even if you do enable region name validation, the HyperStore S3 Service employs a fall-back device where if the region name specified in the request's authorization header does not match against the local region name, the system checks whether the specified region name matches against the S3 service

domain. If both checks fail then the request is rejected. This is to accommodate legacy HyperStore systems where the S3 service endpoint may not necessarily include the region name.

The HyperStore system continues to support AWS Signature Version 2 as well.

#### 13.1.4. Common Request Headers

From the "Common Request Headers" section of the Amazon S3 REST API, the HyperStore system supports the headers listed below. If a common request header from the Amazon S3 API specification is not listed below, the HyperStore system does not support it. For header descriptions, refer to the "["Common Request Headers" section of the Amazon S3 REST API.](#)

- Authorization
- Content-Length
- Content-Type
- Content-MD5
- Date
- Expect
- Host
- x-amz-content-sha256
- x-amz-date

#### 13.1.5. Common Response Headers

From the "Common Response Headers" section of the Amazon S3 REST API, the HyperStore system supports the headers listed below. If a common response header from the Amazon S3 API specification is not listed below, the HyperStore system does not support it. For header descriptions, refer to the "["Common Response Headers" section of the Amazon S3 REST API.](#)

- Content-Length
- Content-Type
- Connection
- Date
- ETag
- Server
- x-amz-delete-marker
- x-amz-request-id
- x-amz-version-id

#### 13.1.6. Access Control List (ACL)

For the Amazon S3 "Access Control List (ACL)" functionality, the HyperStore system supports the items listed below. If a grantee group, permission type, or canned ACL type from the Amazon S3 documentation is not listed below, the HyperStore system does not support it.

For ACL usage information and for descriptions of ACL items, see [Access Control List \(ACL\) Overview](#) in the Amazon S3 documentation.

### 13.1.6.1. Amazon S3 Predefined Groups

- Authenticated users group
- All users group
- Log delivery group

### 13.1.6.2. Permission Types

- READ
- WRITE
- READ\_ACP
- WRITE\_ACP
- FULL\_CONTROL

### 13.1.6.3. Canned ACL

- private
- public-read
- public-read-write
- authenticated-read
- bucket-owner-read
- bucket-owner-full-control
- log-delivery-write

*HyperStore Extension to the S3 API*

The HyperStore system supports these additional canned ACLs:

Canned ACL	Applies to	Permissions added to ACL
group-read	Bucket and object	Owner gets FULL_CONTROL. All other members of the owner's HyperStore service user group get READ access.
group-read-write	Bucket and object	Owner gets FULL_CONTROL. All other members of the owner's HyperStore service user group get READ and WRITE access.

**Note** To grant access to groups other than the requester's own group, you cannot use canned ACLs. Instead, when using standard Amazon S3 methods for assigning privileges to a grantee (via request headers or request body), specify "<groupID>|" as the grantee. The "<groupID>|" format (with vertical bar) indicates that the grantee is a group — for example, "Group5|".

**Note** When access privileges have through separate requests been granted to a group and to a specific member of the group, the user gets the broader of the privilege grants. For example, if Group5 is granted read-write privileges and a specific user within Group5 is separately granted read privileges, the user gets read-write privileges.

### 13.1.7. HyperStore Extensions to the S3 API

The HyperStore S3 Service supports the following extensions to the AWS S3 REST API:

Type	Extension	Purpose	Detail
Additional S3 API methods	<ul style="list-style-type: none"> <li>• <i>POST Bucket lock-policy</i></li> <li>• <i>POST Bucket lockId</i></li> <li>• <i>GET Bucket lock-policy</i></li> <li>• <i>DELETE Bucket lock-policy</i></li> </ul>	Execute and manage "bucket lock" (WORM) for a bucket	<ul style="list-style-type: none"> <li>• "<b>POST Bucket lock-policy</b>" (page 908)</li> <li>• "<b>POST Bucket lockId</b>" (page 911)</li> <li>• "<b>GET Bucket lock-policy</b>" (page 901)</li> <li>• "<b>DELETE Bucket lock-policy</b>" (page 894)</li> <li>• "<b>Setting Up a Bucket Lock</b>" (page 163)</li> </ul>
Additional headers or body elements for existing AWS S3 API methods	<i>x-gmt-policyid</i> as optional request header for "PUT Bucket" and response header for "GET Bucket Object (List Objects) version 1", "GET Bucket Object (List Objects) version 2", and "HEAD Bucket"	Specify the HyperStore storage policy to use for a new bucket	<ul style="list-style-type: none"> <li>• "<b>PUT Bucket</b>" (page 913)</li> <li>• "<b>GET Bucket (List Objects) Version 1</b>" (page 896)</li> <li>• "<b>GET Bucket (List Objects) Version 2</b>" (page 897)</li> <li>• "<b>HEAD Bucket</b>" (page 907)</li> </ul>

Type	Extension	Purpose	Detail
			<ul style="list-style-type: none"> <li>"Storage Policies Feature Overview" (page 140)</li> </ul>
	x-gmt-tieringinfo and x-gmt-compare and x-gmt-post-tier-copy as optional request headers for "PUT Bucket lifecycle" and response headers for "GET Bucket lifecycle"	Set up auto-tiering for a bucket	<ul style="list-style-type: none"> <li>"PUT Bucket lifecycle" (page 916)</li> <li>"GET Bucket lifecycle" (page 900)</li> <li>"Auto-Tiering Feature Overview" (page 53)</li> </ul>
	x-gmt-crr-endpoint and x-gmt-crr-credentials as optional request headers for "PUT Bucket replication" and response headers for "GET Bucket replication"	Set up cross-region replication to a destination outside of HyperStore	<ul style="list-style-type: none"> <li>"PUT Bucket replication" (page 927)</li> <li>"GET Bucket replication" (page 905)</li> <li>"Cross-Region Replication" (page 132)</li> </ul>
	"NotPrincipal":{"UserType":"SystemAdmin"} and "Action":["s3:PrivilegedDelete"] as supported policy statement elements for "PUT Bucket policy"	Set up a privileged delete user for a locked bucket	<ul style="list-style-type: none"> <li>"PUT Bucket policy" (page 923)</li> <li>"Setting Up a Privileged Delete User for a Bucket" (page 164)</li> </ul>
	x-gmt-error-code and x-gmt-message as supported response headers for "GET Object" and "Head Object"	Provide additional information about HTTP 4xx errors	<ul style="list-style-type: none"> <li>"GET Object" (page 932)</li> <li>"HEAD Object" (page 935)</li> </ul>

## 13.2. Operations on the Service

From the ["Operations on the Service" portion of the Amazon S3 REST API](#), the HyperStore system supports the operations listed in this section. If a service operation from the Amazon S3 API specification is not listed in this section, the HyperStore system does not support it.

### 13.2.1. GET Service

HyperStore supports the S3 API operation "GET Service". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 ["Common Request Headers"](#) (page 888) and ["Common Response Headers"](#) (page 888), which HyperStore supports. For operation syntax and examples, see [GET Service](#) in the Amazon S3 REST API specification.

#### *HyperStore Extension to the S3 API*

The HyperStore system supports the following Request Parameter as an extension to the "GET Service" operation:

**Note** Support for this extension is disabled by default. To enable support for this extension, in ["mts.-properties.erb"](#) (page 496) set `cloudian.s3.enablesharedbucket` to true, then do a Puppet push and then restart the S3 Service.

Name	Description	Required
shared	<p>If the <code>shared</code> parameter is included in the request, the GET Service method returns a list of buckets that other users have shared with the requesting user. This will be buckets that have been shared specifically with the requesting user, plus buckets that have been shared with the group to which the requesting user belongs, plus buckets that have been shared with everyone.</p> <p>Example:</p> <pre>GET /?shared HTTP/1.1. Host: s3-region1.enterprise4.mobi-cloud.com. Accept-Encoding: identity. Date: Fri, 05 Apr 2019 15:34:01 GMT. Content-Length: 0. Authorization: AWS akey2:jTcwd1Ta+5sZftVHGtEEyweojdk=. User-Agent: Boto/2.42.0 Python/2.7.5 Linux/3.10.0-693.el7.x86_64.  HTTP/1.1 200 OK. Date: Fri, 05 Apr 2019 15:34:01 GMT. x-amz-request-id: 1721b414-267b-1341-93e6-d4ae52ce5402. Content-Type: application/xml; charset=UTF-8. Content-Length: 432. Server: CloudianS3.  &lt;?xml version="1.0" encoding="UTF-8"?&gt;&lt;ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"&gt; &lt;Owner&gt;&lt;ID&gt;8ce1c49e532edc91b0a43e0c7e7d5975&lt;/ID&gt; &lt;DisplayName&gt;robot1&lt;/DisplayName&gt;&lt;/Owner&gt;</pre>	No

Name	Description	Required
	<pre>&lt;Buckets&gt;&lt;Bucket&gt;&lt;Name&gt;sharedbucket1&lt;/Name&gt; &lt;CreationDate&gt;2019-04-05T15:30:03.897Z&lt;/CreationDate&gt;&lt;/Bucket&gt; &lt;Bucket&gt;&lt;Name&gt;sharedbucket2&lt;/Name&gt; &lt;CreationDate&gt;2019-04-05T15:27:26.300Z&lt;/CreationDate&gt; &lt;/Bucket&gt;&lt;/Buckets&gt;&lt;/ListAllMyBucketsResult&gt;</pre> <p><b>Note</b> When the 'shared' parameter is used, the <i>GET Service</i> call returns only buckets that have been shared with the requesting user -- <b>not buckets owned by the requesting user</b>. So to retrieve all buckets that a user has access to, an S3 client application must submit two <i>GET Service</i> calls -- one without the 'shared' parameter (to retrieve the user's own buckets) and one with the 'shared' parameter (to retrieve buckets that have been shared with the user).</p> <p><b>Note</b> When the 'shared' parameter is used, in the <i>GET Service</i> response body the "Owner" is the requesting user, not the actual owner(s) of the shared bucket(s).</p>	

## 13.3. Operations On Buckets

From the "Operations on Buckets" portion of the Amazon S3 REST API, the HyperStore system supports the operations listed in this section. If a bucket operation from the Amazon S3 API specification is not listed in this section, the HyperStore system does not support it.

### 13.3.1. DELETE Bucket

HyperStore supports the S3 API operation "DELETE Bucket". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket](#) in the Amazon S3 REST API specification.

### 13.3.2. DELETE Bucket cors

HyperStore supports the S3 API operation "DELETE Bucket cors". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket cors](#) in the Amazon S3 REST API specification.

### 13.3.3. DELETE Bucket encryption

HyperStore supports the S3 API operation "DELETE Bucket encryption". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket encryption](#) in the Amazon S3 REST API specification.

### 13.3.4. DELETE Bucket lifecycle

HyperStore supports the S3 API operation "DELETE Bucket lifecycle". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket lifecycle](#) in the Amazon S3 REST API specification.

### 13.3.5. DELETE Bucket lock-policy

#### 13.3.5.1. Description

This method is a HyperStore extension to the S3 API.

The *DELETE Bucket lock-policy* method aborts a bucket lock process on a bucket, if and only if both of these conditions apply:

- The bucket lock initiated on the bucket by a "**POST Bucket lock-policy**" (page 908) call has not yet been completed by a subsequent "**POST Bucket lockId**" (page 911) call. Once *POST Bucket lockId* is successfully called, the bucket lock policy transitions from the *InProgress* state to the *Locked* state and the *DELETE Bucket lock-policy* method is no longer allowed.
- The bucket lock initiated on the bucket by a *POST Bucket lock-policy* call has not yet expired. The bucket lock automatically expires 24 hours after initiation if no corresponding *POST Bucket lockId* call is made. If the bucket lock has expired then the *DELETE Bucket lock-policy* call has no effect since there is no longer a bucket lock to delete.

The use case for the *DELETE Bucket lock-policy* method is if you change your mind about a bucket lock that you have initiated and you want to remove the lock immediately rather than waiting for the automatic expiration of the lock that will occur after 24 hours.

Only the bucket owner is allowed to execute this method.

#### 13.3.5.2. Requests

##### Syntax

```
DELETE /?lock-policy HTTP/1.1
Host: bucketName.s3.example.com
Date: Date
Authorization: SignatureValue
```

##### Request Parameters

This operation does not use request parameters.

##### Request Headers

This operation uses only request headers that are common to all operations.

##### Request Body

This operation does not have a request body.

### 13.3.5.3. Responses

#### Response Headers

This operation uses only response headers that are common to most responses.

#### Response Elements

This operation does not return a response body.

#### Special Errors

- If the lock is in the *Locked* state when this operation is requested (rather than the *InProgress* state), then the operation returns an 403 Forbidden, AccessDeniedException error.
- If the request is made except by bucket owner, then 403 Forbidden, AccessDenied is returned.

### 13.3.5.4. Example

This example removes an *InProgress* lock from a bucket named "mybucket".

```
DELETE /?lock-policy HTTP/1.1
Host: mybucket.s3.example.com
Date: Date
Authorization: SignatureValue
```

If the request succeeds HyperStore returns an HTTP 204 No Content response.

### 13.3.6. DELETE Bucket policy

HyperStore supports the S3 API operation "DELETE Bucket policy". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket Policy](#) in the Amazon S3 REST API specification.

### 13.3.7. DELETE Bucket replication

HyperStore supports the S3 API operation "DELETE Bucket replication". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket replication](#) in the Amazon S3 REST API specification.

### 13.3.8. DELETE Bucket tagging

HyperStore supports the S3 API operation "DELETE Bucket tagging". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket tagging](#) in the Amazon S3 REST API specification.

### 13.3.9. DELETE Bucket website

HyperStore supports the S3 API operation "DELETE Bucket website". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Bucket website](#) in the Amazon S3 REST API specification.

### 13.3.10. GET Bucket (List Objects) Version 1

HyperStore supports the S3 API operation "GET Bucket (List Objects) Version 1". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters, response headers, and response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket \(List Objects\) Version 1](#) in the Amazon S3 REST API specification.

**Note** Amazon recommends using the newer version of this API -- "[GET Bucket \(List Objects\) Version 2](#)" (page 897) -- for application development.

**Note** When using GET Bucket (List Objects) Version 1, use the *marker* request parameter to improve performance in listing the content of buckets that contain many objects. For detail see the Amazon documentation for GET Bucket (List Objects) Version 1.

#### 13.3.10.1. Request Parameters

- delimiter
- encoding-type
- marker
- max-keys
- prefix

**Note:** The HyperStore system does not support %c2%85(U+0085) as a delimiter value

**Note** The HyperStore S3 extension request parameter *meta=true* is no longer supported.

#### 13.3.10.2. Response Headers

##### *HyperStore Extension to the S3 API*

The HyperStore system supports the following Response Header as an extension to the "GET Bucket (List Objects) Version 1" operation:

Name	Description	Required
x-gmt-policyid	This header specifies the unique ID of the storage policy assigned to the bucket. For more information see " <a href="#">PUT Bucket</a> " (page 913).	No

### 13.3.10.3. Response Elements

- Contents
- CommonPrefixes
- Delimiter
- DisplayName
- Encoding-Type
- ETag
- ID
- IsTruncated
- Key
- LastModified
- ListBucketResult
- Marker
- MaxKeys
- Name
- NextMarker
- Owner
- Prefix
- Size
- StorageClass (values STANDARD and GLACIER only)

### 13.3.11. GET Bucket (List Objects) Version 2

HyperStore supports the S3 API operation "GET Bucket (List Objects) Version 2". Along with supporting the S3 **"Common Request Headers"** (page 888) and **"Common Response Headers"** (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters, response headers, and response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket \(List Objects\) Version 2](#) in the Amazon S3 REST API specification.

**Note** Amazon recommends using GET Bucket (List Objects) Version 2 for application development. However, for backward-compatibility Amazon and HyperStore continue to also support "["GET Bucket \(List Objects\) Version 1"](#) (page 896).

**Note** When using GET Bucket (List Objects) Version 2, use the *continuation-token* request parameter to improve performance in listing the content of buckets that contain many objects. For detail see the Amazon documentation for GET Bucket (List Objects) Version 2.

#### 13.3.11.1. Request Parameters

- delimiter

**Note:** The HyperStore system does not support %c2%85(U+0085) as a delimiter value

- encoding-type
- max-keys
- prefix
- list-type
- continuation-token
- fetch-owner
- start-after

**Note** The HyperStore S3 extension request parameter *meta=true* is no longer supported.

### 13.3.11.2. Response Headers

*HyperStore Extension to the S3 API*

The HyperStore system supports the following Response Header as an extension to the "GET Bucket (List Objects) Version 2" operation:

Name	Description	Required
x-gmt-policyid	This header specifies the unique ID of the storage policy assigned to the bucket. For more information see " <b>PUT Bucket</b> " (page 913).	No

### 13.3.11.3. Response Elements

- Contents
- CommonPrefixes
- Delimiter
- DisplayName
- Encoding-Type
- ETag
- ID
- IsTruncated
- Key
- LastModified
- ListBucketResult
- MaxKeys
- Name
- Owner
- Prefix
- Size
- StorageClass (values STANDARD and GLACIER only)

- ContinuationToken
- KeyCount
- NextContinuationToken
- StartAfter

### 13.3.12. GET Bucket acl

HyperStore supports the S3 API operation "GET Bucket acl". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket acl](#) in the Amazon S3 REST API specification.

#### 13.3.12.1. Response Elements

- AccessControlList
- AccessControlPolicy
- DisplayName
- Grant
- Grantee
- ID
- Owner
- Permission

### 13.3.13. GET Bucket cors

HyperStore supports the S3 API operation "GET Bucket cors". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket cors](#) in the Amazon S3 REST API specification.

#### 13.3.13.1. Response Elements

- CORSConfiguration
- CORSRule
- AllowedHeader
- AllowedMethod
- AllowedOrigin
- ExposeHeader
- ID
- MaxAgeSeconds

### 13.3.14. GET Bucket encryption

HyperStore supports the S3 API operation "GET Bucket encryption". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any

---

operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket encryption](#) in the Amazon S3 REST API specification.

### 13.3.14.1. Response Elements

- ApplyServerSideEncryptionByDefault
- Rule
- ServerSideEncryptionConfiguration
- SSEAlgorithm

### 13.3.15. GET Bucket lifecycle

HyperStore supports the S3 API operation "GET Bucket (List Objects) Version 2". Along with supporting the S3 **"Common Request Headers"** (page 888) and **"Common Response Headers"** (page 888) that may be used with any operation, HyperStore supports the operation-specific response headers, response elements, and special errors listed below. For description of these items and for operation syntax and examples, see [GET Bucket lifecycle](#) in the Amazon S3 REST API specification.

#### 13.3.15.1. Response Headers

*HyperStore Extension to the S3 API*

The HyperStore system supports the following Response Headers as extensions to the "GET Bucket lifecycle" operation:

Name	Description	Required
x-gmt-tieringinfo	See <a href="#">"PUT Bucket lifecycle"</a> (page 916).	No
x-gmt-compare	See <a href="#">"PUT Bucket lifecycle"</a> (page 916).	No
x-gmt-post-tier-copy	See <a href="#">"PUT Bucket lifecycle"</a> (page 916).	No

#### 13.3.15.2. Response Elements

- AbortIncompleteMultipartUpload
- Date
- Days
- DaysAfterInitiation
- Expiration
- ExpiredObjectDeleteMarker
- Filter
- ID
- LifecycleConfiguration
- NoncurrentDays
- NoncurrentVersionExpiration
- NoncurrentVersionTransition

- Prefix
- Rule
- Status
- StorageClass
- Transition

### 13.3.15.3. Special Errors

- NoSuchLifecycleConfiguration

## 13.3.16. GET Bucket location

HyperStore supports the S3 API operation "GET Bucket location". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket location](#) in the Amazon S3 REST API specification.

### 13.3.16.1. Response Elements

- LocationConstraint

## 13.3.17. GET Bucket lock-policy

### 13.3.17.1. Description

This method is a HyperStore extension to the S3 API.

The *GET Bucket lock-policy* method retrieves the current lock policy of a bucket, if one exists for the bucket. For background information see "[POST Bucket lock-policy](#)" (page 908) and "[WORM \(Bucket Lock\) Feature Overview](#)" (page 159).

Only the bucket owner is allowed to execute this method.

### 13.3.17.2. Requests

#### Syntax

```
GET /?lock-policy HTTP/1.1
Host: bucketName.s3.example.com
Date: Date
Authorization: SignatureValue
```

#### Request Parameters

This operation does not use request parameters.

#### Request Headers

This operation uses only request headers that are common to all operations.

## Request Body

This operation does not have a request body.

### 13.3.17.3. Responses

#### Response Headers

This operation uses only response headers that are common to most responses.

#### Response Body

The response body is a JSON object representing the policy:

```
{
 "Id": "string",
 "Version": "string",
 "Statement": JSONArray,
 "State": "string",
 "LockId": "string",
 "ExpirationDate": "string",
 "CreationDate": "string"
}
```

The response elements are as follows:

Name	Description
Id	Fixed to “lock-policy”  Type: String
Version	Fixed to “2012-10-17”  Type: String
Statement	The policy statement. For description see <b>“POST Bucket lock-policy”</b> (page 908).  Type: JSONArray
State	The state of the bucket lock: either <i>InProgress</i> or <i>Locked</i> .  Type: String
LockId	The bucket’s lockId generated by the POST lock-policy request.  Type: String  Example: 0a770a288f5507b153a96ddd139a2149
ExpirationDate	The UTC date and time at which the lock ID expires, if the bucket lock is still in an <i>InProgress</i> state. This expiration will be 24 hours after the lock process is initiated by a <b>“POST Bucket lock-policy”</b> (page 908) call.  This value is null if the bucket lock is in a <i>Locked</i> state. Once a lock is transitioned from an <i>InProgress</i> state to a <i>Locked</i> state by a successful <b>“POST Bucket lockId”</b> (page 911) call, the lock does not expire.  Type: A string representation of ISO 8601 date format.
CreationDate	The UTC date and time at which the bucket lock was put into the <i>InProgress</i> state

Name	Description
	by a successful " <b>POST Bucket lock-policy</b> " (page 908) call. Type: A string representation of ISO 8601 date format.

## Special Errors

- If there is no lock policy set on the bucket, then 404 Not Found is returned.
- If the request is made except by bucket-owner, then 403 Forbidden, AccessDenied is returned.

### 13.3.17.4. Example

This example retrieves the current bucket lock policy for the bucket "mybucket".

```
GET /?lock-policy HTTP/1.1
Host: mybucket.s3.example.com
Date: Date
Authorization: SignatureValue

HTTP/1.1 200 OK
Date: Date
Content-Type: application/json
Content-Length: length

{
 "Id": "lock-policy",
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "deny-based-on-age",
 "Effect": "Deny",
 "Action": ["s3:DeleteObject", "s3:DeleteObjectVersion"],
 "Resource": ["arn:aws:s3:::mybucket/*"],
 "Condition": {"NumericLessThan": {"s3:AgeInDays": "3650"}}
 }
],
 "State": "INPROGRESS",
 "LockId": "0a770a288f5507b153a96ddd139a2149",
 "ExpirationDate": "2017-10-04T01:08:25.180Z",
 "CreationDate": "2017-10-03T01:08:25.180Z"
}
```

### 13.3.18. GET Bucket logging

HyperStore supports the S3 API operation "GET Bucket logging". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket logging](#) in the Amazon S3 REST API specification.

### 13.3.18.1. Response Elements

- BucketLoggingStatus
- Grant
- Grantee
- LoggingEnabled
- Permission
- TargetBucket
- TargetGrants
- TargetPrefix

### 13.3.19. GET Bucket Object versions

HyperStore supports the S3 API operation "GET Bucket Object versions". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters and response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket Object versions](#) in the Amazon S3 REST API specification.

#### 13.3.19.1. Request Parameters

- delimiter
- encoding-type
- key-marker
- max-keys
- prefix
- version-id-marker

#### 13.3.19.2. Response Elements

- DeleteMarker
- DisplayName
- Encoding-Type
- ETag
- ID
- IsLatest
- IsTruncated
- Key
- KeyMarker
- LastModified
- ListVersionsResult
- MaxKeys
- Name

- NextKeyMarker
- NextVersionIdMarker
- Owner
- Prefix
- Size
- StorageClass
- Version
- VersionId
- VersionIdMarker

### 13.3.20. GET Bucket policy

HyperStore supports the S3 API operation "GET Bucket policy". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket policy](#) in the Amazon S3 REST API specification.

#### 13.3.20.1. Response Elements

The response contains the (JSON) policy of the specified bucket.

### 13.3.21. GET Bucket replication

HyperStore supports the S3 API operation "GET Bucket replication". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response headers, response elements, and special errors listed below. For description of these items and for operation syntax and examples, see [GET Bucket replication](#) in the Amazon S3 REST API specification.

#### 13.3.21.1. Response Headers

##### *HyperStore Extension to the S3 API*

HyperStore will return these headers if and only if they were included in the "**PUT Bucket replication**" (page 927) request that created the bucket's replication configuration:

- x-gmt-crr-endpoint
- x-gmt-crr-credentials

These headers would have been used in the PUT request if the replication destination bucket is not in the HyperStore system in which the source bucket is located (such as in the case of replicating from a HyperStore bucket to Amazon S3 as the destination). For more information about these HyperStore extension headers see "**PUT Bucket replication**" (page 927).

### 13.3.21.2. Response Elements

- ReplicationConfiguration
- Role
- Rule
- ID
- Status
- Prefix
- Destination
- Bucket
- StorageClass

### 13.3.21.3. Special Errors

- NoSuchReplicationConfiguration

## 13.3.22. GET Bucket tagging

HyperStore supports the S3 API operation "GET Bucket tagging". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements and special errors listed below. For description of these items and for operation syntax and examples, see [GET Bucket tagging](#) in the Amazon S3 REST API specification.

**Note** The HyperStore Admin API supports a method for retrieving all the bucket tags for all users in a specified group. Because it is implemented through the Admin API, that method does not require the users' S3 access credentials. For more information see [GET /bucketops/gettags](#).

### 13.3.22.1. Response Elements

- Tagging
- TagSet
- Tag
- Key
- Value

### 13.3.22.2. Special Errors

- NoSuchTagsetError

## 13.3.23. GET Bucket versioning

HyperStore supports the S3 API operation "GET Bucket versioning". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any

operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket versioning](#) in the Amazon S3 REST API specification.

### 13.3.23.1. Response Elements

- Status
- VersioningConfiguration

### 13.3.24. GET Bucket website

HyperStore supports the S3 API operation "GET Bucket website". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Bucket website](#) in the Amazon S3 REST API specification.

### 13.3.24.1. Response Elements

The response XML includes the same elements that were uploaded when the bucket was configured as a website.

### 13.3.25. HEAD Bucket

HyperStore supports the S3 API operation "HEAD Bucket". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response headers listed below. For description of these items and for operation syntax and examples, see [HEAD Bucket](#) in the Amazon S3 REST API specification.

### 13.3.25.1. Response Headers

- x-amz-bucket-region

*HyperStore Extension to the S3 API*

The HyperStore system supports the following Response Header as an extension to the "HEAD Bucket" operation:

Parameter	Description
x-gmt-policyid	This header specifies the unique ID of the storage policy assigned to the bucket. For more information see " <a href="#">PUT Bucket</a> " (page 913).

### 13.3.26. List Multipart Uploads

HyperStore supports the S3 API operation "List Multipart Uploads". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters and response elements listed

below. For description of these items and for operation syntax and examples, see [List Multipart Uploads](#) in the Amazon S3 REST API specification.

### 13.3.26.1. Request Parameters

- delimiter
- encoding-type
- max-uploads
- key-marker
- prefix
- upload-id-marker

### 13.3.26.2. Response Elements

- ListMultipartUploadsResult
- Bucket
- KeyMarker
- UploadIdMarker
- NextKeyMarker
- NextUploadIdMarker
- Encoding-Type
- MaxUploads
- IsTruncated
- Upload
- Key
- UploadId
- Initiator
- ID
- DisplayName
- Owner
- StorageClass
- Initiated
- ListMultipartUploadsResult.Prefix
- Delimiter
- CommonPrefixes
- CommonPrefixes.Prefix

## 13.3.27. POST Bucket lock-policy

### 13.3.27.1. Description

This method is a HyperStore extension to the S3 API.

---

The *POST Bucket lock-policy* method initiates the process of "locking" a bucket such that objects in the bucket cannot be altered or deleted until the object age exceeds the retention period specified in the lock policy. Only the bucket owner is allowed to execute this method, and the method is only supported on buckets for which versioning is used and no bucket lock is currently in place. For general information about the HyperStore bucket lock feature including licensing requirements see "**"WORM (Bucket Lock) Feature Overview"** (page 159).

The *POST Bucket lock-policy* method initiates the bucket lock process by:

- Attaching the lock policy to the bucket
- Transitioning the bucket lock to the *InProgress* state
- Returning a unique lock ID

After the bucket lock enters the *InProgress* state, **you have 24 hours to complete the lock** by calling the "**"POST Bucket lockId"** (page 911) method, which changes the lock to the *Locked* state. Once a bucket is in the *Locked* state, the lock policy can never be removed from the bucket or modified.

If you do not call the *POST Bucket lockId* method within 24 hours after the bucket lock enters the *InProgress* state, the bucket automatically exits the *InProgress* state and the lock is removed.

Anytime during the 24 hours that a bucket lock is in the *InProgress* state, you can immediately abort the locking process by calling the "**"DELETE Bucket lock-policy"** (page 894) method. You can get the current state of the lock by calling the "**"GET Bucket lock-policy"** (page 901) method.

The *InProgress* state provides you the opportunity to test your bucket lock policy before you permanently lock the bucket. During the *InProgress* state your bucket lock policy will be in full effect in the sense that it will prevent objects from being deleted from the bucket (unless they are already older than your defined retention period). What distinguishes the *InProgress* state from the *Locked* state is that while the bucket lock state is *InProgress* you still have the option to remove the lock policy from the bucket, either by calling *DELETE Bucket lock-policy* or by letting 24 hours pass without calling the *POST Bucket lockId* method. Once an *InProgress* bucket lock has been removed by either of these approaches you can apply a different lock policy to the bucket if you wish, by again calling the *POST Bucket lock-policy* method.

### 13.3.27.2. Requests

#### Syntax

```
POST ?lock-policy HTTP/1.1
Host: bucketName.s3.example.com
Date: Date
Authorization: SignatureValue
Content-Length: Length

{
 "Id": "some-id",
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "some-statement-id",
 "Effect": "Deny",
 "Action": ["s3:DeleteObject", "s3:DeleteObjectVersion"],
 "Resource": ["arn:aws:s3:::bucketname/*"],
 "Condition": {"NumericLessThan": {"s3:AgeInDays": "Days"} }
 }
]
}
```

```
]
}
```

## Request Parameters

This operation does not use request parameters.

## Request Headers

This operation uses only request headers that are common to all operations.

## Request Body

The request body is a JSON string specifying the policy statement. The JSON string uses "\\" as an escape character. The JSON fields are as follows:

Name	Description	Required
Id	Any string value. This value is ignored.	No
Version	Recommend using "2012-10-17".	No
Statement	Policy statement	Yes
Sid	Any value accepted.	No
Effect	Must be "Deny".	Yes
Action	JSONArray or JSONPrimitive. Must be ["s3:DeleteObject","s3:DeleteObjectVersion"].	Yes
Resource	JSONArray. The bucket name is extracted as the substring at the end after the last '.' character and before the last '/' character. Any string before the bucket name is acceptable, for example [arn:aws:s3:::<BUCKETNAME>/].	Yes
Condition	Must be {"NumericLessThan":{"s3:AgeInDays": "<DAYS>"}}, where <DAYS> is an integer. For example {"NumericLessThan": {"s3:AgeInDays": "365"}},	Yes

All other fields are ignored.

### 13.3.27.3. Responses

## Response Headers

A successful response includes the following response header, in addition to the response headers that are common to all operations.

Name	Description
x-hs-lock-id	The unique lock ID, which is used to complete the vault locking process by supplying it with the <b>"POST Bucket lockId"</b> (page 911) method.  Type: String  Example: 0a770a288f5507b153a96ddd139a2149

## Response Body

This operation does not return a response body.

## Special Errors

- If *POST Bucket lock-policy* is called when the bucket already has a bucket lock policy (regardless of lock state), a 403 Forbidden AccessDeniedException error is returned.
- If bucket versioning is not enabled, then 409 Conflict, BucketLockPolicyError is returned.
- If the request is made except by bucket owner, then 403 Forbidden, AccessDenied is returned.

### 13.3.27.4. Example

The following example initiates a lock policy on the bucket "mybucket" such that objects in the bucket will not be allowed to be deleted until 10 years after object creation.

```
POST ?lock-policy HTTP/1.1
Host: mybucket.s3.example.com
Date: Date
Authorization: SignatureValue
Content-Length: Length

{
 "Id": "some-id",
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "TenYearWORM",
 "Effect": "Deny",
 "Action": ["s3:DeleteObject", "s3:DeleteObjectVersion"],
 "Resource": ["arn:aws:s3:::mybucket/*"],
 "Condition": {"NumericLessThan": {"s3:AgeInDays": "3650"} }
 }
]
}
```

If the request succeeds HyperStore returns an HTTP 201 Created response, which includes the *x-hs-lock-id* response header.

## 13.3.28. POST Bucket lockId

### 13.3.28.1. Description

**This method is a HyperStore extension to the S3 API.**

The *POST Bucket lockId* method completes the process of "locking" a bucket, if this method is called **within 24 hours** of having successfully called the "**POST Bucket lock-policy**" (page 908) method on the same bucket. The *POST Bucket lockId* method transitions the bucket lock from the *InProgress* state to the *Locked* state. Once a bucket is in the *Locked* state, **the lock policy can never be removed from the bucket or modified**.

Only the bucket owner is allowed to execute this method.

### 13.3.28.2. Requests

#### Syntax

```
POST /?lock-policy&lockId=XXX HTTP/1.1
Host: bucketName.s3.example.com
Date: Date
Authorization: SignatureValue
Content-Length: Length
```

#### Request Parameters

Name	Description	Required
lockId	The bucket lock ID that was returned by the <i>POST Bucket lock-policy</i> method.  Example: 0a770a288f5507b153a96ddd139a2149	No

#### Request Headers

This operation uses only request headers that are common to all operations.

#### Request Body

This operation does not have a request body.

### 13.3.28.3. Responses

#### Response Headers

This operation uses only response headers that are common to most responses.

#### Response Elements

This operation does not return a response body.

#### Special Errors

- If an request's lock ID is expired (more than 24 hours have passed since the *POST Bucket lock-policy* call) or does not match the lock policy's lock ID, the operation returns a 400 Bad Request, InvalidArgument error.
- If bucket versioning is not enabled, then 409 Conflict, BucketLockPolicy is returned.
- If the request is made except by bucket owner, then 403 Forbidden, AccessDenied is returned.

### 13.3.28.4. Example

This example completes a bucket lock on the bucket "mybucket", using the lock ID that had been returned by a prior *POST Bucket lock-policy* call on the bucket.

```
POST /?lock-policy&lockId=EA863rKkWZU53SLW5be4DUCW HTTP/1.1
Host: mybucket.s3.example.com
Date: Date
```

```
Authorization: SignatureValue
Content-Length: 0
```

If the request succeeds HyperStore returns an HTTP 204 No Content response.

### 13.3.29. PUT Bucket

HyperStore supports the S3 API operation "PUT Bucket". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket](#) in the Amazon S3 REST API specification.

**Note** By default each user is allowed a maximum of 100 buckets. You can change this setting in the CMC's [Configuration Settings](#) page.

#### 13.3.29.1. Request Headers

- x-amz-acl
- x-amz-grant-read
- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control

*HyperStore Extension to the S3 API*

The HyperStore system supports the following Request Header as an extension to the "PUT Bucket" operation:

Name	Description	Required
x-gmt-policyid	<p>This header specifies the unique ID of the storage policy to assign to the newly created bucket. The storage policy determines how data in the bucket will be distributed and protected through either replication or erasure coding. System administrators can create multiple storage policies through the CMC and the system automatically assigns each a unique policy ID that becomes part of the policy definition. (To obtain a list of storage policies for your system and their policy IDs, you can use the Admin API's <a href="#">GET /bppolicy/listpolicy</a> method).</p> <p>With the "x-gmt-policyid" request header for "PUT Bucket", you specify the ID of the desired storage policy when you create a new bucket. Note however that some policies may not be available to all user groups — a policy's availability is specified by system administrators at the time of policy creation, and this information becomes part of the policy definition. When you specify an "x-gmt-policyid" value with a "PUT Bucket" request, the policy ID must be for a policy that is available to the group to which the bucket owner belongs.</p> <p>Also the policy ID must be for a storage policy from the service region that is specified in the "PUT Bucket" request's LocationConstraint element.</p>	No

Name	Description	Required
	<p>If the "PUT Bucket" request does not include the "x-gmt-policyid" request header, then the system will automatically assign the system default storage policy to the bucket during bucket creation.</p> <p><b>Note</b> After a bucket is created, it cannot be assigned a different storage policy. The storage policy assigned to the bucket at bucket creation time will continue to be bucket's storage policy for the life of the bucket.</p> <p><b>Note</b> A 403 error response is returned if you specify a policy ID that does not exist, has been disabled, is not available to the region in which the bucket is being created, or is not available to the group to which the bucket owner belongs. A 403 is also returned if you do not specify an "x-gmt-policyid" header and the system does not yet have an established default storage policy.</p> <p>Example header:</p> <pre>x-gmt-policyid: 1bc90238f9f11cb32f5e4e901675d50b</pre> <p>For more information on storage policies, see "<b>Storage Policies Feature Overview</b>" (page 140).</p>	

### 13.3.29.2. Request Elements

- CreateBucketConfiguration
- LocationConstraint

**Note** The HyperStore system enforces the same [bucket naming restrictions](#) as does Amazon S3. Also, if you use an underscore in a bucket name you will not be able to enable auto-tiering for the bucket (for transitioning objects to Amazon or other remote destinations on a configurable schedule). It's best not to use underscores when naming new buckets, in case you may want to enable auto-tiering on the bucket immediately or in the future.

### 13.3.30. PUT Bucket acl

HyperStore supports the S3 API operation "PUT Bucket acl". Along with supporting the S3 ["Common Request Headers"](#) (page 888) and ["Common Response Headers"](#) (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket acl](#) in the Amazon S3 REST API specification.

### 13.3.30.1. Request Headers

- x-amz-acl
- x-amz-grant-read
- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control

### 13.3.30.2. Request Elements

- AccessControlList
- AccessControlPolicy
- DisplayName
- Grant
- Grantee
- ID
- Owner
- Permission

### 13.3.31. PUT Bucket cors

HyperStore supports the S3 API operation "PUT Bucket cors". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket cors](#) in the Amazon S3 REST API specification.

### 13.3.31.1. Request Headers

- Content-MD5

### 13.3.31.2. Request Elements

- CORSConfiguration
- CORSRule
- ID
- AllowedMethod
- AllowedOrigin
- AllowedHeader
- MaxAgeSeconds
- ExposeHeader

### 13.3.32. PUT Bucket encryption

HyperStore supports the S3 API operation "PUT Bucket encryption". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket encryption](#) in the Amazon S3 REST API specification.

**Note** For information about HyperStore's support for server-side encryption -- including the interaction of object level, bucket level, and storage policy level encryption settings -- see "[Server-Side Encryption Feature Overview](#)" (page 117).

**Note** In the current HyperStore release, **only the bucket owner is allowed to perform operations relating to bucket encryption**. HyperStore does not currently support the use of bucket policies to extend bucket encryption permissions to users other than the bucket owner. Specifically, with regard to "[PUT Bucket policy](#)" (page 923), HyperStore does not currently support the "s3:PutEncryptionConfiguration" or "s3:GetEncryptionConfiguration" actions.

#### 13.3.32.1. Request Elements

- ApplyServerSideEncryptionByDefault
- KMSMasterKeyID
- Rule
- ServerSideEncryptionConfiguration
- SSEAlgorithm

### 13.3.33. PUT Bucket lifecycle

HyperStore supports the S3 API operation "PUT Bucket lifecycle". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket lifecycle](#) in the Amazon S3 REST API specification.

**Note** With the HyperStore system, only the bucket owner can create Lifecycle rules.

#### 13.3.33.1. Request Headers

- Content-MD5

##### *HyperStore Extension to the S3 API*

The HyperStore system supports the following Request Headers as extensions to the "PUT Bucket lifecycle" operation:

Name	Description	Required
x-gmt-tieringinfo	<p>This request header enables you to configure a bucket for schedule-based automatic transitioning of objects from local HyperStore storage to a remote storage system. For background information on the HyperStore auto-tiering feature, see "<b>Auto-Tiering Feature Overview</b>" (page 53).</p> <p>The <i>x-gmt-tieringinfo</i> header is formatted as follows:</p> <pre>x-gmt-tieringinfo: PROTOCOL EndPoint:Endpoint,Action:Action[,Mode:-proxy] [,TieringBucket:TieringBucket]</pre> <ul style="list-style-type: none"> <li>• <i>PROTOCOL</i> (mandatory) — Specify one of these values: <ul style="list-style-type: none"> <li>◦ S3 -- Transition the objects to Amazon S3 storage.</li> <li>◦ S3GLACIER -- Transition the objects to Amazon Glacier.</li> <li>◦ GCS -- Transition the objects to Google Cloud Storage.</li> <li>◦ AZURE -- Transition the objects to Microsoft Azure.</li> <li>◦ SPECTRA -- Transition the objects to a Spectra Logic BlackPearl destination.</li> </ul> </li> </ul> <div style="background-color: #e0f2e0; padding: 10px;"> <p><b>Note:</b> If you are tiering to an S3-compliant system other than Amazon S3, Glacier, or Google Cloud Storage, use "S3" as the protocol. This would include, for instance, tiering to a remote HyperStore region or system.</p> </div> <div style="background-color: #e0f2e0; padding: 10px;"> <p><b>Note:</b> Auto-tiering restrictions based on destination type:  * By default the largest object size that can be auto-tiered to Amazon, Google, or other S3-compliant destinations is 50GB. If you want to tier objects larger than this, consult with Cloudian Support. This 50GB limit does not apply to tiering to Azure or Spectra BlackPearl.  * Tiering to Azure or Spectra BlackPearl is not supported for source buckets that have versioning enabled or that have had versioning enabled in the past.  * When auto-tiering to Spectra BlackPearl is used for a bucket, objects in the bucket will not be auto-tiered unless they are larger than 5MB. Objects 5MB or smaller will remain in HyperStore.</p> </div> <ul style="list-style-type: none"> <li>• <i>EndPoint:Endpoint</i> (mandatory) — The service endpoint URL to use as your auto-tiering destination. For example with Amazon S3, choose the region endpoint that's most suitable for your location (such as <i>s3-us-west-1.amazonaws.com</i> if your organization is in northern California). Or in the case of Spectra BlackPearl, specify the URL for your Spectra BlackPearl destination.</li> </ul> <p>If your ultimate tiering destination is Glacier, you must specify an Amazon S3 endpoint here, <b>not</b> a Glacier endpoint. The HyperStore system will first transition the objects to your specified Amazon S3 endpoint and then from there they will be immediately transitioned to the corresponding Glacier location.</p>	No

Name	Description	Required
	<p>If you want to auto-tier to an external HyperStore system -- not a different region within the same HyperStore system but rather a different HyperStore system altogether -- see "<b>Note About Tiering to a Different HyperStore Region or System</b>" (page 58), in regard to the format requirements for the tiering endpoint.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>Note:</b> You must use nested URL encoding. First URL encode the Endpoint value (the endpoint itself), and then URL encode the whole <i>x-gmt-tieringinfo</i> value.</p> </div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>Note:</b> Once you've configured an auto-tiering lifecycle on a bucket you cannot subsequently change the tiering endpoint for that bucket.</p> </div> <ul style="list-style-type: none"> <li>• Action:<i>Action</i> (mandatory) — This parameter specifies how the HyperStore system will handle S3 <i>GET Object</i> requests for objects that have been transitioned to the tiering destination. The choices are: <ul style="list-style-type: none"> <li>◦ stream — If the client submits a <i>GET Object</i> request to HyperStore, the HyperStore system retrieves the object from the destination and streams it through to the client. This method is supported only if the <i>Protocol</i> is S3, GCS, or AZURE.</li> <li>◦ nostream — If the client submits a <i>GET Object</i> request to HyperStore, the HyperStore system rejects the GET request. Instead, clients must submit a <i>POST Object restore</i> request in order to temporarily restore a copy of the object to local HyperStore storage.</li> </ul> </li> <li>• If the <i>Protocol</i> is S3, GCS, or AZURE you can use either "stream" or "nostream". If the <i>Protocol</i> is S3GLACIER or SPECTRA you must use "nostream" (the "stream" option is not supported for those destinations).</li> <li>• Mode:<i>proxy</i> (optional) — If you specify this option, then: <ul style="list-style-type: none"> <li>◦ All objects uploaded to the bucket from this point forward (all objects uploaded after you successfully submit the <i>PUT Bucket lifecycle</i> request) will be <b>immediately</b> transitioned to the destination system.</li> <li>◦ Any objects already in the bucket at the time that you submit the <i>PUT Bucket lifecycle</i> request will be subject to the transition schedule defined in the request body.</li> </ul> </li> </ul> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>Note</b> If you want to use proxy mode and do not want schedule-based tiering of any objects that are already in the bucket, you <b>still must include a request body</b>. In this case in the lifecycle configuration in the request body you can set the "Status" attribute to "Disabled" (for an example see "<b>Request Body if Using "Proxy Mode"</b>" (page 922)). The result will be that any objects already in the bucket will not be auto-tiered, and all objects subsequently uploaded to the bucket (after successfully submitting the <i>PUT Bucket lifecycle</i> request) will be immediately moved to the des-</p> </div>	

Name	Description	Required
	<p>tination system.</p> <p>Proxy mode is supported only if the <i>Protocol</i> is S3, GCS, or AZURE (proxy mode is not supported for S3GLACIER or SPECTRA tiering). For more information on proxy mode -- also known as "bridge mode" -- see "<b>Auto-Tiering Feature Overview</b>" (page 53).</p> <ul style="list-style-type: none"> <li>• TieringBucket:<i>TieringBucket</i> (optional) — The name of the bucket to transition objects into, in the tiering destination system. This can be either: <ul style="list-style-type: none"> <li>◦ The <b>name of a bucket that already exists in the destination system</b>, and for which you are the bucket owner. In this case HyperStore will use this existing bucket as the tiering destination.</li> <li>◦ The <b>name of a bucket that you want HyperStore to create in the destination system</b>, to use as the tiering destination. Be sure to choose a bucket name that is very likely to be unique in the destination system. If your supplied bucket name is not unique in the destination system, HyperStore will be unable to create the bucket and the <i>PUT Bucket lifecycle</i> request will fail.</li> </ul> </li> </ul> <p>If you <b>omit</b> the tiering bucket parameter, then in the destination system HyperStore will create a tiering bucket named as follows:</p> <p style="padding-left: 40px;"><i>&lt;origin-bucket-name-truncated-to-34-characters&gt;-&lt;28-character-random-string&gt;</i></p> <p><b>Example x-gmt-tieringinfo request headers:</b></p> <pre># Example 1 (before URL encoding) Tiering to Amazon S3, into target bucket # named 'bucket12'. Streaming for local GETs will be supported.  x-gmt-tieringinfo: S3 EndPoint:ht- tp://s3.amazonaws.com,Action:stream, TieringBucket:bucket12  # Example 1 after nested URL encoding (endpoint value first, then whole # header value)  x-gmt-tieringinfo: S3%7CEndPoint%3Aht- tp%253A%252F%252Fs3.amazonaws.com %2CAction%3Astream%2CTieringBucket%3Abucket12  # Example 2 (before URL encoding) Tiering to Azure. HyperStore will derive target # bucket name from source bucket name. Streamed local GETs will not be supported, # clients must use Restore.</pre>	

Name	Description	Required
	<pre>x-gmt-tieringinfo: AZURE EndPoint:ht- tps://blob.core.windows.net,Action:nostream  # Example 2 after nested URL encoding (endpoint value first, then whole # header value)  x-gmt-tieringinfo: AZURE%7CEndPoint%3Aht- tps%253A%252F%252Fblob.core.windows.net %2CAction%3Anostream</pre> <p><b>IMPORTANT:</b> If you use the <i>x-gmt-tieringinfo</i> request header, then <b>for the request body element "StorageClass" you must specify "GLACIER"</b>. Set the Storage Class to GLACIER even if your final tiering destination is Amazon S3 or some other S3-compliant destination.</p>	
x-gmt-compare	<p>If you include this header in your "PUT Bucket lifecycle" request and set the header value to "LAT", then in lifecycle rules that you configure with the "Days" comparator the rule will be implemented as number of days since the object's <b>Last Access Time</b>.</p> <p>If you do not use this extension header, or if you include the header but assign it no value or any value other than "LAT", then "Days" based lifecycle rules will be implemented as number of days since the object's Creation Time (the default Amazon S3 behavior).</p> <p>You can use this header to create:</p> <ul style="list-style-type: none"> <li>• Last Access Time based auto-tiering rules (use this header and also the <i>x-gmt-tierinfo</i> header).</li> <li>• Last Access Time based expiration rules (use this header but not the <i>x-gmt-tierinfo</i> header).</li> </ul> <p><b>Note</b> An object's Last Access Time is updated if the object is accessed either for retrieval (GET or HEAD) or modification (PUT/POST/Copy). If an object is created and then never accessed, its Last Access Time will be its Creation Time.</p> <p><b>Note</b> If you use the <i>x-gmt-compare</i> header and set it to "LAT", it does not apply to any in <i>NoncurrentVersionTransition</i> or <i>NoncurrentVersionExpiration</i> rules within the lifecycle policy (for non-current versions of versioned objects). These types of rules are always based on the time elapsed since an object version became non-current (was replaced by a new version of the object).</p>	No
x-gmt-post-tier-copy	If you use the <i>x-gmt-tieringinfo</i> request header to configure auto-tiering for a bucket, you can optionally also use the <i>x-gmt-post-tier-copy</i> request header to <b>specify a number of days for which a local copy of auto-tiered objects should be retained</b> . For example if you set <i>x-gmt-post-tier-copy: 7</i> then after each object is auto-tiered to the tiering destination, a copy of the object will be kept in the HyperStore source bucket for 7 days.	No

Name	Description	Required
	<p>After that the local copy will be deleted and only object metadata will be retained locally.</p> <p>There is no upper limit on this value. So if you want the local copy retention period to be practically limitless, you could for example set this header to 36500 to indicate a local copy retention period of 100 years.</p> <p>If you <b>omit</b> the <i>x-gmt-post-tier-copy</i> request header, then by default local objects are deleted after they are successfully auto-tiered to the tiering destination system, and only object metadata is retained locally.</p>	

### 13.3.33.2. Request Elements

- AbortIncompleteMultipartUpload
- Date
- Days
- DaysAfterInitiation
- Expiration
- ExpiredObjectDeleteMarker
- Filter
- ID
- LifecycleConfiguration
- NoncurrentDays
- NoncurrentVersionExpiration
- NoncurrentVersionTransition
- Prefix

**Note:** If you are using "Bridge Mode" (whereby objects are auto-tiered immediately after being uploaded to HyperStore), leave the "Object Prefix" field empty. Bridge Mode does not support filtering by prefix.

- Rule
- Status
- StorageClass

**Note:** If you are using the HyperStore extension request header *x-gmt-tieringinfo*, then for the request element "StorageClass" you must specify "GLACIER". Set the Storage Class to GLACIER even if your final tiering destination is Amazon S3, Google, Azure, or some other system.

Note that in the case of tiering to Azure, if you do a *GET Bucket (List Objects)* call on a bucket, for any objects that have been tiered to Azure the response will indicate that the storage class of those tiered objects is STANDARD (despite your having specified GLACIER as the storage class in the *PUT Bucket Lifecycle* call). This behavior is expected. With objects tiered to any

other destination, a *GET Bucket (List Objects)* call will indicate that tiered objects have the storage class that you specified in the *PUT Bucket Lifecycle* call.

- Transition

### Request Body if Using "Proxy Mode"

If you are applying a "proxy mode" bucket lifecycle (see "HyperStore Extension to the S3 API" in the "Request Headers" section above), you still must include a request body. The lifecycle configuration specified in the request body will apply to objects already in the bucket (if any) while the proxy mode (for immediate transitioning) will apply to all objects uploaded after the *PUT Bucket lifecycle* request is successfully submitted. If there are already objects in the bucket and you don't want any transitioning applied to them, make the request body a lifecycle configuration with the "Status" attribute set to "Disabled" such as in the following example:

```
<LifecycleConfiguration>
 <Rule>
 <ID>DummyLifecycleConfig</ID>
 <Filter>
 <Prefix></Prefix>
 </Filter>
 <Status>Disabled</Status>
 <Transition>
 <StorageClass>GLACIER</StorageClass>
 <Days>0</Days>
 </Transition>
 </Rule>
</LifecycleConfiguration>
```

#### 13.3.34. PUT Bucket logging

HyperStore supports the S3 API operation "PUT Bucket logging". Along with supporting the S3 **"Common Request Headers"** (page 888) and **"Common Response Headers"** (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket logging](#) in the Amazon S3 REST API specification.

**Note** For a bucket that has bucket logging enabled, bucket logs (server access logs) are generated every 10 minutes by a HyperStore system cron job, if there was activity for that bucket during that interval.

**Note** If you are using bucket logging in your service, and if you use a load balancer in front of your S3 Service nodes, you should configure your S3 Service to support the HTTP X-Forwarded-For header. This will enable bucket logs to record the true originating IP address of S3 requests, rather than the load balancer IP address. By default the S3 Service does not support the X-Forwarded-For header. You can enable support for this header using the system configuration file `s3.xml.erb`.

### 13.3.34.1. Request Elements

- BucketLoggingStatus
- EmailAddress
- Grant
- Grantee
- LoggingEnabled
- Permission
- TargetBucket
- TargetGrants
- TargetPrefix

### 13.3.35. PUT Bucket policy

HyperStore supports the S3 API operation "PUT Bucket policy". Along with supporting the S3 **"Common Request Headers"** (page 888) and **"Common Response Headers"** (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket policy](#) in the Amazon S3 REST API specification.

**Note** Regardless of any bucket policies, a bucket owner always has full access to her bucket and the objects in it; and an object owner always has full access to her object.

**Note** As an extension to the S3 API, HyperStore supports a special type of bucket policy *Statement* that establishes a privileged delete user for a bucket that is locked. This is supported only if you have a HyperStore Enterprise WORM license. For more information see ["Setting Up a Privileged Delete User for a Bucket"](#) (page 164) and ["WORM \(Bucket Lock\) Feature Overview"](#) (page 159).

### 13.3.35.1. Request Elements

The request body is a JSON-formatted bucket policy containing one or more policy statements. Within a policy's *Statement* block(s), HyperStore support for policy statement elements and their values is as follows:

- *Sid* -- Same as Amazon: Custom string identifying the statement, for example "Statement1" or "Only allow access from partner source IPs"
- *Effect* -- Same as Amazon: "Allow" or "Deny"
- *Principal* -- Must be one of the following:
  - "\*" -- Statement applies to all users.
  - {"CanonicalUser": "<canonicalUserId>"} -- Statement applies to the specified user.
  - {"CanonicalUser": ["<canonicalUserId>", "<canonicalUserId>"]} -- Statement applies to the specified users.

**Note:** You can obtain a HyperStore user's canonical ID by retrieving the user through the

CMC's "**Manage Users**" (page 229) page or by using the Admin API method [GET /user](#).

**Note:** HyperStore does not support having an IAM user as the Principal in a bucket policy. For more information on HyperStore support for IAM see "**Identity and Access Management (IAM) Feature Overview**" (page 91).

- *Action* -- See details below.
- *Resource* -- Same as Amazon; must be one of:
  - "arn:aws:s3:::<bucketName>" -- For bucket actions (such as "s3>ListBucket") and bucket sub-resource actions (such as "s3:GetBucketAcl").
  - "arn:aws:s3:::<bucketName>/\*" or "arn:aws:s3:::<bucketName>/<objectName>" -- For object actions (such as "s3>PutObject").
- *Condition* -- See details below.

## Supported "Action" Values

Within bucket policy statements, HyperStore supports **only** the following *Action* values (also known as permission keywords).

**Note** For information about how to use *Action* values in a bucket policy, see the AWS documentation on [Specifying Permissions in a Policy](#).

## Object Actions

- s3>AbortMultipartUpload
- s3>DeleteObject
- s3>DeleteObjectTagging
- s3>DeleteObjectVersion
- s3>DeleteObjectVersionTagging
- s3>GetObject
- s3>GetObject
- s3>GetObjectAcl
- s3>GetObjectTagging
- x3>GetObjectTorrent
- s3>GetObjectVersion
- s3>GetObjectVersionAcl
- s3>GetObjectVersionTagging
- s3>ListMultipartUploadParts
- s3>PutObject
- s3>PutObjectAcl
- s3>PutObjectTagging
- s3>PutObjectVersionAcl

- s3:PutObjectVersionTagging
- s3:RestoreObject

### Bucket Actions

- s3:CreateBucket
- s3:DeleteBucket
- s3>ListBucket
- s3>ListBucketMultipartUploads
- s3>ListBucketVersions

### Bucket Subresource Actions

- s3>DeleteBucketPolicy
- s3>DeleteBucketWebsite
- s3>GetBucketAcl
- s3>GetBucketCORS
- s3>GetBucketLocation
- s3>GetBucketLogging
- s3>GetBucketNotification
- s3>GetBucketPolicy
- s3>GetBucketRequestPayment
- s3>GetBucketTagging
- s3>GetBucketVersioning
- s3>GetBucketWebsite
- s3>GetLifecycleConfiguration
- s3>GetReplicationConfiguration
- s3>PutBucketAcl
- s3>PutBucketCORS
- s3>PutBucketLogging
- s3>PutBucketNotification
- s3>PutBucketPolicy
- s3>PutBucketRequestPayment
- s3>PutBucketTagging
- s3>PutBucketVersioning
- s3>PutBucketWebsite
- s3>PutLifecycleConfiguration
- s3>PutReplicationConfiguration

**Note** Like Amazon, the HyperStore system supports the use of a wildcard in your Action configuration ("Action": "s3:\*"). When an Action wildcard is used together with an object-level Resource element ("arn:aws:s3:::<bucketName>//\*" or "arn:aws:s3:::<bucketName>/<objectName>"), the wildcard denotes

all the Object actions **that HyperStore supports**. When an Action wildcard is used together with bucket-level Resource element ("arn:aws:s3:::<bucketName>"), the wildcard denotes all the Bucket actions and Bucket Subresource actions **that HyperStore supports**.

## Supported "Condition" Values

Within bucket policy statements, HyperStore supports **only** the following *Condition* operators and keys.

**Note** For information about how to use condition operators and keys in a bucket policy, see the AWS documentation on [Specifying Conditions in a Policy](#).

### Condition Operators

- IpAddress

**Note:** If you are using load balancers in front of the HyperStore S3 Service, then IP address based bucket policies 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 "["s3\\_proxy\\_protocol\\_enabled"](#)" (page 475) in "["common.csv"](#)" (page 458). 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 IP address based bucket policies. You must use PROXY Protocol if you have load balancers in front of the S3 Service and want to use IP address based bucket policies .

- NotIpAddress
- NumericEquals
- NumericNotEquals
- NumericLessThan
- NumericLessThanEquals
- NumericGreaterThan
- NumericGreaterThanOrEqual
- StringEquals
- StringNotEquals
- StringEqualsIgnoreCase
- StringNotEqualsIgnoreCase
- StringLike
- ForAllValues:StringLike
- ForAnyValue:StringLike
- StringNotLike

## Condition Keys

- aws:Referer
- aws:SourceIp

**Note:** If you create a bucket policy that restricts access based on source IP address, these restrictions will not apply to IP addresses within your HyperStore cluster. IP addresses from within your cluster are automatically "whitelisted".

- s3:delimiter
- s3:ExistingObjectTag/<tag-key>
- s3:LocationConstraint
- s3:max-keys
- s3:prefix
- s3:RequestObjectTag/<tag-keys>
- s3:RequestObjectTagKeys
- s3:VersionId
- s3:x-amz-acl
- s3:x-amz-copy-source
- s3:x-amz-grant-full-control
- s3:x-amz-grant-read
- s3:x-amz-grant-read-acp
- s3:x-amz-grant-write
- s3:x-amz-grant-write-acp
- s3:x-amz-metadata-directive
- s3:x-amz-server-side-encryption

For examples of the kinds of things you can do with bucket policies, see the AWS documentation on [Bucket Policy Examples](#).

### 13.3.36. PUT Bucket replication

HyperStore supports the S3 API operation "PUT Bucket replication. Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket replication](#) in the Amazon S3 REST API specification. For an overview of this Amazon S3 feature see [Cross-Region Replication](#) in the Amazon S3 Developer's Guide. For information about HyperStore implementation of this feature, see "**Cross-Region Replication**" (page 132).

**IMPORTANT:** Unlike Amazon S3, HyperStore does not require that you set up an IAM Role (or anything analogous) in order to use bucket replication. Also, HyperStore does not require that the destination bucket be in a different region than the source bucket. With HyperStore you can replicate to a destination bucket that's in the same region as the source bucket, if you want to.

Apart from the above exceptions, HyperStore bucket replication has the same requirements as Amazon S3 bucket replication, including that **versioning must be enabled** (using the "**PUT Bucket versioning**" (page 930) operation) on both the source bucket and the destination bucket in order to use bucket replication.

### 13.3.36.1. Request Headers

- Content-MD5

#### *HyperStore Extension to the S3 API*

The HyperStore system supports the following Request Headers as extensions to the "PUT Bucket replication" operation. These headers are required **only if you are replicating to a destination bucket that is not in the same HyperStore system as the source bucket** (for example, if you are replicating from a HyperStore source bucket to a destination Amazon S3 bucket; or from a HyperStore source bucket to a destination bucket in a completely independent HyperStore system).

Name	Description	Required
x-gmt-crr-end-point	<p>Service endpoint of the destination S3 service, in format &lt;protocol&gt;://&lt;endpoint&gt;:&lt;port&gt;. For example <a href="https://s3.amazonaws.com">https://s3.amazonaws.com</a>:443. Since security credentials will be transmitted in this request (see "x-gmt-crr-credentials" below), HTTPS is the recommended protocol rather than regular HTTP. HyperStore does not force HTTP use here, but for security HTTP is advisable.</p> <p>This header is required only if the destination bucket is not in the same HyperStore system as the source bucket. Do not use this header if the destination bucket is in the same HyperStore region as the source bucket, or if it is in a different region within the same HyperStore system as the source bucket.</p>	See description
x-gmt-crr-credentials	<p>Access key and secret key for the user account that HyperStore should use to write to the destination bucket in the destination S3 system, in format &lt;access-key&gt;:&lt;secret-key&gt;. For example, 00caf3940d-c923c59406:Ku0bMR0H5nSA7t8N+ngP6uPPTINSxJ/Q2oICMexx. This user account must have write permissions on the destination bucket. For example, if the destination bucket is in the Amazon S3 system, this header is used to specify the Amazon S3 access key and secret key for an account that has write permissions on the destination bucket.</p> <p>This header is required only if the destination bucket is not in the same HyperStore system as the source bucket. Do not use this header if the destination bucket is in the same HyperStore region as the source bucket, or if it is in a different region within the same HyperStore system as the source bucket.</p>	See description

### 13.3.36.2. Request Elements

- ReplicationConfiguration
- Role

**Note:** As with the Amazon S3 API specification, for HyperStore the "Role" element must be included in the PUT Bucket replication request. However, HyperStore ignores the "Role" element's value (so, you can use any random string as its value). HyperStore does not use an IAM role or anything analogous when implementing cross-region replication.

- Rule
- ID
- Status
- Prefix
- Destination
- Bucket

**Note:** Use the same "Bucket" value formatting as in the Amazon S3 API spec, i.e.

- arn:aws:s3:::<bucketname>.
- StorageClass

**Note:** If you include this optional element in the request, HyperStore ignores its value.

### 13.3.37. PUT Bucket tagging

HyperStore supports the S3 API operation "PUT Bucket tagging". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket tagging](#) in the Amazon S3 REST API specification.

**Note** The HyperStore Admin API supports a method for retrieving all the bucket tags for all users in a specified group. Because it is implemented through the Admin API, that method does not require the users' S3 access credentials. For more information see [GET /bucketops/gettags](#).

#### 13.3.37.1. Request Headers

- Content-MD5

#### 13.3.37.2. Request Elements

- Tagging
- TagSet
- Tag
- Key
- Value

### 13.3.37.3. Special Errors

- InvalidTagError
- MalformedXMLError
- OperationAbortedError
- InternalError

### 13.3.38. PUT Bucket versioning

HyperStore supports the S3 API operation "PUT Bucket versioning". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket versioning](#) in the Amazon S3 REST API specification.

#### 13.3.38.1. Request Elements

- Status
- VersioningConfiguration

### 13.3.39. PUT Bucket website

HyperStore supports the S3 API operation "PUT Bucket website". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements listed below. For description of these items and for operation syntax and examples, see [PUT Bucket website](#) in the Amazon S3 REST API specification.

#### 13.3.39.1. Request Elements

- WebsiteConfiguration
- RedirectAllRequestsTo
- HostName
- Protocol
- WebsiteConfiguration
- IndexDocument
- ErrorDocument

## 13.4. Operations On Objects

From the "Operations on Objects" portion of the Amazon S3 REST API, the HyperStore system supports the operations listed in this section. If an operation is not listed in this section, the HyperStore system does not support it.

---

### 13.4.1. Delete Multiple Objects

HyperStore supports the S3 API operation "Delete Multiple Objects". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, request elements, and response elements listed below. For description of these items and for operation syntax and examples, see [Delete Multiple Objects](#) in the Amazon S3 REST API specification.

**Note** The HyperStore S3 Service allows a maximum of 1000 object deletes per Delete Multiple Objects request.

**IMPORTANT:** Do not attempt to delete more than 100,000 objects from a single bucket in less than an hour. Doing so will result in TombstoneOverwhelmingException errors in the Cassandra logs and an inability to successfully execute an S3 "**GET Bucket (List Objects) Version 1**" (page 896) or "**GET Bucket (List Objects) Version 2**" (page 897) operation on the bucket. If the system is in this error condition, you can trigger a tombstone purge as described in "**Dealing with Excessive Tombstone Build-Up**" (page 423).

#### 13.4.1.1. Request Headers

- Content-MD5
- Content-Length

#### 13.4.1.2. Request Elements

- Delete
- Quiet
- Object
- Key
- VersionId

#### 13.4.1.3. Response Elements

- DeleteResult
- Deleted
- Key
- VersionId
- DeleteMarker
- DeleteMarkerVersionId
- Error
- Key
- VersionId

- Code
- Message

### 13.4.2. DELETE Object

HyperStore supports the S3 API operation "DELETE Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response headers listed below. For description of these items and for operation syntax and examples, see [DELETE Object](#) in the Amazon S3 REST API specification.

**Note** Do not attempt to delete more than 100,000 objects from a single bucket in less than an hour. Doing so will result in TombstoneOverwhelmingException errors in the Cassandra logs and an inability to successfully execute a "**GET Bucket (List Objects) Version 1**" (page 896) operation on the bucket. If the system is in this error condition, you can trigger a tombstone purge as described in "**Dealing with Excessive Tombstone Build-Up**" (page 423).

#### 13.4.2.1. Response Headers

- x-amz-delete-marker
- x-amz-version-id

### 13.4.3. DELETE Object tagging

HyperStore supports the S3 API operation "DELETE Object tagging". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [DELETE Object tagging](#) in the Amazon S3 REST API specification.

### 13.4.4. GET Object

HyperStore supports the S3 API operation "GET Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters, request headers, and response headers listed below. For description of these items and for operation syntax and examples, see [GET Object](#) in the Amazon S3 REST API specification.

#### 13.4.4.1. Request Parameters

- response-content-type
- response-content-language
- response-expires
- response-cache-control
- response-content-disposition
- response-content-encoding

#### 13.4.4.2. Request Headers

- Range
- If-Modified-Since
- If-Unmodified-Since
- If-Match
- If-None-Match
- x-amz-server-side-encryption-customer-algorithm

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see "**Server-Side Encryption Feature Overview**" (page 117).

- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

#### 13.4.4.3. Response Headers

- x-amz-delete-marker
- x-amz-expiration
- x-amz-meta-\*
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5
- x-amz-restore
- x-amz-tagging-count
- x-amz-version-id
- x-amz-website-redirect-location
- x-amz-server-side-encryption-customer-algorithm

##### *HyperStore Extension to the S3 API*

The HyperStore system supports the following Response Headers as extensions to the "GET Object" operation. These headers are returned **only in the event of an HTTP 4xx response**. They are not returned with HTTP 2xx, 3xx, or 5xx responses.

Name	Description
x-gmt-error-code	In the event of an HTTP 4xx response, these two response headers provide additional information about the nature of the error. The <i>x-gmt-error-code</i> header values will be from among the list in " <b>Error Responses</b> " (page 885).
x-gmt-message	

### 13.4.5. GET Object acl

HyperStore supports the S3 API operation "GET Object acl". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Object acl](#) in the Amazon S3 REST API specification.

#### 13.4.5.1. Response Elements

- AccessControlList
- AccessControlPolicy
- DisplayName
- Grant
- Grantee
- ID
- Owner
- Permission

### 13.4.6. GET Object tagging

HyperStore supports the S3 API operation "GET Object tagging". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific response elements listed below. For description of these items and for operation syntax and examples, see [GET Object tagging](#) in the Amazon S3 REST API specification.

#### 13.4.6.1. Response Elements

- Tagging
- TagSet
- Tag
- Key
- Value

### 13.4.7. GET Object torrent

HyperStore supports the S3 API operation "GET Object torrent". The operation does not use any operation-specific parameters, headers, or elements. It uses only the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888), which HyperStore supports. For operation syntax and examples, see [GET Object torrent](#) in the Amazon S3 REST API specification.

#### 13.4.7.1. Implementation Notes

- HyperStore does not provide a BitTorrent "tracker". You must either provide your own tracker or use one of the many publicly available trackers. **You must edit the "cloudian.s3.torrent.tracker" (page 520) property in [mts.properties](#).erb** to specify the URL of the tracker that you are using.

- HyperStore implements BitTorrent HTTP seeding for in accordance with the BEP19 specification ([http://www.bittorrent.org/beps/bep\\_0019.html](http://www.bittorrent.org/beps/bep_0019.html)). Therefore torrent files returned by HyperStore in response to *GET Object torrent* requests will include a "url-list" key and the value of that key will be the URL of the object in HyperStore.
- HyperStore objects that have been auto-tiered to a destination S3 system cannot be retrieved via BitTorrent, unless the objects are first restored to local HyperStore storage (via the S3 "**POST Object restore**" (page 938) method). Restored objects can be retrieved via BitTorrent.
- Like with Amazon S3, with HyperStore only publicly readable objects are eligible for BitTorrent retrieval. And like with Amazon S3, the following types of objects are **not** retrievable via BitTorrent:
  - Objects larger than 5GB
  - Non-current versions of versioned objects
  - Objects encrypted via SSE-C (SSE with Customer-managed key; by contrast, BitTorrent retrieval is supported for objects encrypted with regular SSE)

## 13.4.8. HEAD Object

HyperStore supports the S3 API operation "HEAD Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and response headers listed below. For description of these items and for operation syntax and examples, see [HEAD Object](#) in the Amazon S3 REST API specification.

### 13.4.8.1. Request Headers

- Range
- If-Modified-Since
- If-Unmodified-Since
- If-Match
- If-None-Match
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

### 13.4.8.2. Response Headers

- x-amz-expiration
- x-amz-meta-\*
- x-amz-restore
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5
- x-amz-tagging-count
- x-amz-version-id

The HyperStore system supports the following Response Headers as extensions to the "HEAD Object" operation. These headers are returned **only in the event of an HTTP 4xx response**. They are not returned with HTTP 2xx, 3xx, or 5xx responses.

Name	Description
x-gmt-error-code	In the event of an HTTP 4xx response, these two response headers provide additional information about the nature of the error. The <i>x-gmt-error-code</i> header values will be from among the list in " <b>Error Responses</b> " (page 885).
x-gmt-message	

## 13.4.9. OPTIONS Object

HyperStore supports the S3 API operation "OPTIONS Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and response headers listed below. For description of these items and for operation syntax and examples, see [OPTIONS Object](#) in the Amazon S3 REST API specification.

### 13.4.9.1. Request Headers

- Origin
- Access-Control-Request-Method
- Access-Control-Request-Headers

### 13.4.9.2. Response Headers

- Access-Control-Allow-Origin
- Access-Control-Max-Age
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers
- Access-Control-Expose-Headers

## 13.4.10. POST Object

HyperStore supports the S3 API operation "POST Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific form fields, response headers, and response elements listed below. For description of these items and for operation syntax and examples, see [POST Object](#) in the Amazon S3 REST API specification.

### 13.4.10.1. Form Fields

- AWSAccessKeyId
- acl
- Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires
- file
- key

- policy
- success\_action\_redirect, redirect
- success\_action\_status
- tagging
- x-amz-storage-class

**Note:** HyperStore ignores the value of the *x-amz-storage-class* field and treats all requests as being for storage class STANDARD.

- x-amz-meta-\*

**Note:** The metadata values must be UTF-8 and must not contain control characters less than 0x20 except for \r, \n, and \t. Also, normal XML escaping is required where appropriate.

- x-amz-website-redirect-location
- x-amz-server-side-encryption

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption* and *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see "**Server-Side Encryption Feature Overview**" (page 117).

- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

#### 13.4.10.2. Response Headers

- x-amz-expiration
- success\_action\_redirect, redirect
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5
- x-amz-version-id

#### 13.4.10.3. Response Elements

- Bucket
- ETag
- Key
- Location

### 13.4.11. POST Object restore

HyperStore supports the S3 API operation "POST Object restore". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and request elements listed below. For description of these items and for operation syntax and examples, see [POST Object restore](#) in the Amazon S3 REST API specification.

**Note** In the context of the HyperStore system, this standard S3 operation is for temporarily restoring a copy of an object that has been auto-tiered to a tiering destination, such as Amazon S3 or Amazon Glacier. For information about the HyperStore auto-tiering feature, see "[Auto-Tiering Feature Overview](#)" (page 53).

#### 13.4.11.1. Request Headers

- Content-MD5

#### 13.4.11.2. Request Elements

- RestoreRequest
- Days
- GlacierJobParameters

**Note:** For the sake of S3 API compatibility, HyperStore's S3 Service allows the request elements *GlacierJobParameters* and *Tier* to be included in a "POST Object restore" request -- but in the current HyperStore release these elements will have no effect on how the restore request is implemented.

- Tier

### 13.4.12. PUT Object

HyperStore supports the S3 API operation "PUT Object". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers and response headers listed below. For description of these items and for operation syntax and examples, see [PUT Object](#) in the Amazon S3 REST API specification.

**Note** For HyperStore S3 bucket capacity information, see "[Maximum Objects Per S3 Bucket](#)" (page 114).

#### 13.4.12.1. Request Headers

- Cache-Control
- Content-Disposition

- Content-Encoding
- Content-Length
- Content-MD5
- Content-Type
- Expect
- Expires
- x-amz-meta-\*

**Note:** The metadata values must be UTF-8 and must not contain control characters less than 0x20 except for \r, \n, and \t. Also, normal XML escaping is required where appropriate.

- x-amz-storage-class

**Note:** HyperStore ignores the value of the *x-amz-storage-class* header and treats all requests as being for storage class STANDARD.

- x-amz-tagging
- x-amz-website-redirect-location
- x-amz-acl
- x-amz-grant-read
- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control
- x-amz-server-side-encryption

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption* and *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see "**Server-Side Encryption Feature Overview**" (page 117).

- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

#### HyperStore Restrictions on Object Names

The following control characters cannot be used anywhere in an object name and will result in a 400 Bad Request response: 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A ("\\n"), 0x0B, 0x0C, 0x0D ("\\r"), 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Also unsupported are:

- 0x09 ("\\t") at the beginning of an object name
- 0xBF (inverted question mark) at the end of an object name

Also, an object name may result in 400 Bad Request or be stored as a different name if the supplied object name:

- Consists only of ".."
- Contains a combination of "." and "/", or ".." and "/"

Examples of object names that will result in 400 Bad Request:

- ..
- ..
- ./
- ../
- ./.
- ./..
- ../..
- ../a
- ../a/
- a/.../b

Examples of object names that will be stored as a different name:

Supplied Object Name	Stored As
.a	a
.a/	a/
a//b	a/b
a./b	a/b
a../b	b
a.../b	b

#### 13.4.12.2. Response Headers

- x-amz-expiration
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5
- x-amz-version-id

#### 13.4.13. PUT Object - Copy

HyperStore supports the S3 API operation "PUT Object - Copy". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, response headers, and response ele-

---

ments listed below. For description of these items and for operation syntax and examples, see [PUT Object - Copy](#) in the Amazon S3 REST API specification.

### 13.4.13.1. Request Headers

- x-amz-copy-source
- x-amz-metadata-directive
- x-amz-copy-source-if-match
- x-amz-copy-source-if-none-match
- x-amz-copy-source-if-unmodified-since
- x-amz-copy-source-if-modified-since
- x-amz-storage-class

**Note:** HyperStore ignores the value of the *x-amz-storage-class* header and treats all requests as being for storage class STANDARD.

- x-amz-tagging-directive
- x-amz-tagging
- x-amz-website-redirect-location
- x-amz-server-side-encryption

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption* and *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see ["Server-Side Encryption Feature Overview"](#) (page 117).

- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5
- x-amz-copy-source-server-side-encryption-customer-algorithm
- x-amz-copy-source-server-side-encryption-customer-key
- x-amz-copy-source-server-side-encryption-customer-key-MD5
- x-amz-acl
- x-amz-grant-read
- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control

### 13.4.13.2. Response Headers

- x-amz-expiration
- x-amz-copy-source-version-id

- x-amz-server-side-encryption
- x-amz-version-id

### 13.4.13.3. Response Elements

- CopyObjectResult
- ETag
- LastModified

## 13.4.14. PUT Object acl

HyperStore supports the S3 API operation "PUT Object acl". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, request elements, and response headers listed below. For description of these items and for operation syntax and examples, see [PUT Object acl](#) in the Amazon S3 REST API specification.

### 13.4.14.1. Request Headers

- x-amz-acl
- x-amz-grant-read
- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control

### 13.4.14.2. Request Elements

- AccessControlList
- AccessControlPolicy
- DisplayName
- Grant
- Grantee
- ID
- Owner
- Permission

### 13.4.14.3. Response Headers

- x-amz-version-id

## 13.4.15. PUT Object tagging

HyperStore supports the S3 API operation "PUT Object tagging". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any

---

operation, HyperStore supports the operation-specific request elements and special errors listed below. For description of these items and for operation syntax and examples, see [PUT Object tagging](#) in the Amazon S3 REST API specification.

#### 13.4.15.1. Request Elements

- Tagging
- TagSet
- Tag
- Key
- Value

#### 13.4.15.2. Special Errors

- InvalidTagError
- MalformedXMLError
- OperationAbortedError
- InternalError

### 13.4.16. Abort Multipart Upload

HyperStore supports the S3 API operation "Abort Multipart Upload". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific special errors listed below. For operation syntax and examples, see [Abort Multipart Upload](#) in the Amazon S3 REST API specification.

#### 13.4.16.1. Special Errors

- NoSuchUpload

### 13.4.17. Complete Multipart Upload

HyperStore supports the S3 API operation "Complete Multipart Upload". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request elements, response headers, response elements, and special errors listed below. For description of these items and for operation syntax and examples, see [Complete Multipart Upload](#) in the Amazon S3 REST API specification.

#### 13.4.17.1. Request Elements

- CompleteMultipartUpload
- Part
- PartNumber
- ETag

#### 13.4.17.2. Response Headers

- x-amz-expiration
- x-amz-server-side-encryption
- x-amz-version-id

#### 13.4.17.3. Response Elements

- CompleteMultipartUploadResult
- Location
- Bucket
- Key
- ETag

#### 13.4.17.4. Special Errors

- InvalidPart
- InvalidPartOrder
- NoSuchUpload

### 13.4.18. Initiate Multipart Upload

HyperStore supports the S3 API operation "Initiate Multipart Upload". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, response headers, and response elements listed below. For description of these items and for operation syntax and examples, see [Initiate Multipart Upload](#) in the Amazon S3 REST API specification.

#### 13.4.18.1. Request Headers

- Cache-Control
- Content-Disposition
- Content-Encoding
- Content-Type
- Expires
- x-amz-meta-
- x-amz-storage-class

**Note:** HyperStore ignores the value of the *x-amz-storage-class* header and treats all requests as being for storage class STANDARD.

- x-amz-website-redirect-location
  - x-amz-acl
  - x-amz-grant-read
-

- x-amz-grant-write
- x-amz-grant-read-acp
- x-amz-grant-write-acp
- x-amz-grant-full-control
- x-amz-server-side-encryption

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption* and *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see "**Server-Side Encryption Feature Overview**" (page 117).

- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

### 13.4.18.2. Response Headers

- x-amz-abort-date
- x-amz-abort-rule-id
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5

### 13.4.18.3. Response Elements

- InitiateMultipartUploadResult
- Bucket
- Key
- UploadId

## 13.4.19. List Parts

HyperStore supports the S3 API operation "List Parts". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request parameters and response elements listed below. For description of these items and for operation syntax and examples, see [List Parts](#) in the Amazon S3 REST API specification.

### 13.4.19.1. Request Parameters

- encoding-type
- uploadId
- max-parts
- part-number-marker

### 13.4.19.2. Response Elements

- x-amz-abort-date
- x-amz-abort-rule-id
- ListPartsResult
- Bucket
- Encoding-Type
- Key
- UploadId
- Initiator
- ID
- DisplayName
- Owner
- StorageClass
- PartNumberMarker
- NextPartNumberMarker
- MaxParts
- IsTruncated
- Part
- PartNumber
- LastModified
- ETag
- Size

### 13.4.20. Upload Part

HyperStore supports the S3 API operation "Upload Part". Along with supporting the S3 "**Common Request Headers**" (page 888) and "**Common Response Headers**" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, response headers, and special errors listed below. For description of these items and for operation syntax and examples, see [Upload Part](#) in the Amazon S3 REST API specification.

#### 13.4.20.1. Request Headers

- Content-Length
- Content-MD5
- Expect
- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm

**Note:** For information about HyperStore's support of the *x-amz-server-side-encryption-customer-\** request headers, and set-up steps that you must perform in order to use HyperStore's server-side encryption features, see "[Server-Side Encryption Feature Overview](#)" (page 117).

- x-amz-server-side-encryption-customer-key
- x-amz-server-side-encryption-customer-key-MD5

#### 13.4.20.2. Response Headers

- x-amz-server-side-encryption
- x-amz-server-side-encryption-customer-algorithm
- x-amz-server-side-encryption-customer-key-MD5

#### 13.4.20.3. Special Errors

- NoSuchUpload

### 13.4.21. Upload Part - Copy

HyperStore supports the S3 API operation "Upload Part - Copy". Along with supporting the S3 "[Common Request Headers](#)" (page 888) and "[Common Response Headers](#)" (page 888) that may be used with any operation, HyperStore supports the operation-specific request headers, response headers, response elements, and special errors listed below. For description of these items and for operation syntax and examples, see [Upload Part - Copy](#) in the Amazon S3 REST API specification.

#### 13.4.21.1. Request Headers

- x-amz-copy-source
- x-amz-copy-source-range
- x-amz-copy-source-if-match
- x-amz-copy-source-if-none-match
- x-amz-copy-source-if-unmodified-since
- x-amz-copy-source-if-modified-since

#### 13.4.21.2. Response Headers

- x-amz-copy-source-version-id
- x-amz-server-side-encryption

#### 13.4.21.3. Response Elements

- CopyPartResult
- ETag
- LastModified

#### 13.4.21.4. Special Errors

- NoSuchUpload
- InvalidRequest

# Chapter 14. IAM API

## 14.1. Identity and Access Management (IAM)

HyperStore provides limited support for the Amazon Identity and Access Management (IAM) API. For an overview of this HyperStore feature -- including limitations on the scope of HyperStore's IAM support, and how to access the HyperStore IAM Service -- see "**Identity and Access Management (IAM) Feature Overview**" (page 91).

Details about HyperStore's support for the IAM API are in these sections:

- "**IAM Supported Actions**" (page 950)
- "**IAM Supported Policy Document Elements**" (page 966)
- "**IAM Common Parameters**" (page 969)
- "**IAM Common Errors**" (page 970)

## 14.2. IAM Supported Actions

The HyperStore implementation of Amazon Web Service's IAM API supports the following Actions and associated parameters, elements, and errors. If an IAM Action, parameter, element, or error is not listed here, HyperStore does not support it. For descriptions of each Action and its associated parameters, elements, and errors, see the AWS documentation links.

**Note** As an extension to the IAM API, the HyperStore IAM Service also supports Actions for performing HyperStore administrative tasks. For a list of these actions see "**HyperStore Extension to the IAM API: Actions for HyperStore Admin Tasks**" (page 965), further below.

### *AddUserToGroup*

Request Parameters:

- GroupName
- UserName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [AddUserToGroup](#)

### *AttachGroupPolicy*

Request Parameters:

- GroupName
- PolicyArn

Errors:

- InvalidInput
- LimitExceeded
- NoSuchEntity
- PolicyNotAttachable
- ServiceFailure

Amazon Documentation: [AttachGroupPolicy](#)

### *AttachUserPolicy*

Request Parameters:

- PolicyArn
- UserName

Errors:

- InvalidInput
- LimitExceeded
- NoSuchEntity
- PolicyNotAttachable
- ServiceFailure

Amazon Documentation: [AttachUserPolicy](#)

#### *CreateAccessKey*

Request Parameters:

- UserName

Response Elements:

- AccessKey

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [CreateAccessKey](#)

**Note** By default the HyperStore system allows only two key pairs per IAM user. This restriction is configurable by the "**credentials.iamuser.max**" (page 514) setting in *mts.properties.erb*. Note that an IAM user's inactive credentials (if any) count toward this limit, as well as active credentials.

#### *CreateGroup*

Request Parameters:

- GroupName
- Path

Response Elements

- Group

**Note:** For HyperStore, within the "Group" object the system-generated "GroupId" attribute value will be in this format: <Canonical-UID-of-HyperStore-User>|<IAM-groupname>

For example: e97eb4557aea18781f53eb2b8f7e282e|iamgroup2

The canonical user ID is that of the HyperStore user account under which the IAM group is created. The IAM group name will be preceded by the path if any is specified when the group is created.

Similarly, the "Arn" attribute value will be in this format:

*arn:aws:iam:<Canonical-UID-of-HyperStore-User>:group/<IAM-groupname>*

Errors:

- EntityAlreadyExists
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [CreateGroup](#)

*CreatePolicy*

Request Parameters:

- Description
- Path
- PolicyDocument

**Note:** For information about HyperStore's IAM policy document support see "**IAM Supported Policy Document Elements**" (page 966).

- PolicyName

Response Elements:

- Policy

Errors:

- EntityAlreadyExists
- InvalidInput
- LimitExceeded
- MalformedPolicyDocument
- ServiceFailure

Amazon Documentation: [CreatePolicy](#)

*CreateUser*

Request Parameters:

- Path
- UserName

Response Elements:

- User

**Note:** For HyperStore, within the "User" object the system-generated "UserId" attribute value will be in this format: <Canonical-UID-of-HyperStore-User>|<IAM-username>

For example: e97eb4557aea18781f53eb2b8f7e282e|iamuser2

The canonical user ID is that of the HyperStore user account under which the IAM user is

created. The IAM user name will be preceded by the path if any is specified when the user is created.

Similarly, the "Arn" attribute value will be in this format:

`arn:aws:iam::<Canonical-UID-of-HyperStore-User>:user/<IAM-username>`

Errors:

- EntityAlreadyExists
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [CreateUser](#)

**Note** IAM users that you create under your HyperStore user account **will not be allowed to log into the CMC** or to use the CMC as their S3 client application. IAM users will need to use an S3 client application other than the CMC to access the HyperStore S3 Service.

**Note** In the current version of HyperStore there is no limit on the number of IAM users that you can create under your HyperStore user account.

#### *DeleteAccessKey*

Request Parameters:

- AccessKeyId
- UserName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeleteAccessKey](#)

#### *DeleteGroup*

Request Parameters:

- GroupName

Errors:

- DeleteConflict
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeleteGroup](#)

*DeleteGroupPolicy*

Request Parameters:

- GroupName
- PolicyName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeleteGroupPolicy](#)

*DeletePolicy*

Request Parameters:

- PolicyArn

Errors:

- DeleteConflict
- InvalidInput
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeletePolicy](#)

*DeleteUser*

Request Parameters:

- UserName

Errors:

- DeleteConflict
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeleteUser](#)

*DeleteUserPolicy*

Request Parameters:

- PolicyName
- UserName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DeleteUserPolicy](#)

*DetachGroupPolicy*

Request Parameters:

- GroupName
- PolicyArn

Errors:

- InvalidInput
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DetachGroupPolicy](#)

*DetachUserPolicy*

Request Parameters:

- PolicyArn
- UserName

Errors:

- InvalidInput
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [DetachUserPolicy](#)

*GetGroup*

Request Parameters:

- GroupName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- Group

**Note:** For HyperStore, within the "Group" object the system-generated "GroupId" attribute value will be in this format: <Canonical-UID-of-HyperStore-User>|<IAM-groupname>

For example: e97eb4557aea18781f53eb2b8f7e282e|iamgroup2

The canonical user ID is that of the HyperStore user account under which the IAM group was created. The IAM group name will be preceded by the path if any was specified when the group was created.

Similarly, the "Arn" attribute value will be in this format:

`arn:aws:iam::<Canonical-UID-of-HyperStore-User>:group/<IAM-groupname>`

- IsTruncated

**Note:** "IsTruncated" will always be "false".

- Users.member.N

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [GetGroup](#)

#### *GetGroupPolicy*

Request Parameters:

- GroupName
- PolicyName

Response Elements:

- GroupName
- PolicyDocument
- PolicyName

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [GetGroupPolicy](#)

#### *GetPolicy*

Request Parameters:

- PolicyArn

Response Elements:

- Policy

Errors:

- InvalidInput
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [GetPolicy](#)

#### *GetPolicyVersion*

Request Parameters:

- PolicyArn

**Note** The "VersionId" request parameter, if submitted, is ignored and always defaults to "v1". HyperStore does not currently support IAM managed policy versioning. However, the "GetPolicyVersion" action is supported because this is the only action that returns the actual policy document (within the "PolicyVersion" object).

Response Elements:

- PolicyVersion

Errors:

- InvalidInput
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [GetPolicyVersion](#)

*GetUser*

Request Parameters:

- UserName

Response Elements:

- User

**Note:** For HyperStore, within the "User" object the system-generated "UserId" attribute value will be in this format: <Canonical-UID-of-HyperStore-User>|<IAM-username>

For example: e97eb4557aea18781f53eb2b8f7e282e|iamuser2

The canonical user ID is that of the HyperStore user account under which the IAM user was created. The IAM user name will be preceded by the path if any was specified when the user was created.

Similarly, the "Arn" attribute value will be in this format:

arn:aws:iam:<Canonical-UID-of-HyperStore-User>:user/<IAM-username>

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation:  [GetUser](#)

*GetUserPolicy*

Request Parameters:

- PolicyName
- UserName

Response Elements:

- PolicyDocument
- PolicyName
- UserName

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [GetUserPolicy](#)

#### *ListAccessKeys*

Request Parameters:

- UserName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- AccessKeyMetadata.member.N
- IsTruncated

**Note:** "IsTruncated" will always be "false".

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListAccessKeys](#)

#### *ListAttachedGroupPolicies*

Request Parameters:

- GroupName
- PathPrefix

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- AttachedPolicies.member.N
- IsTruncated

**Note:** "IsTruncated" will always be "false".

Errors:

- InvalidInput
- NoSuchEntity

- ServiceFailure

Amazon Documentation: [ListAttachedGroupPolicies](#)

*ListAttachedUserPolicies*

Request Parameters:

- PathPrefix
- UserName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- AttachedPolicies.member.N
- IsTruncated

**Note:** "IsTruncated" will always be "false".

Errors:

- InvalidInput
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListAttachedUserPolicies](#)

*ListEntitiesForPolicy*

Request Parameters:

- EntityFilter
- PathPrefix
- PolicyArn

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- IsTruncated

**Note:** "IsTruncated" will always be "false".

- PolicyGroups.member.N
- PolicyUsers.member.N

Errors:

- InvalidInput
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListEntitiesForPolicy](#)

*ListGroupPolicies*

Request Parameters:

- GroupName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- IsTruncated

**Note:** "IsTruncated" will always be "false".

- PolicyNames.member.N

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListGroupPolicies](#)

*ListGroups*

Request Parameters:

- PathPrefix

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- Groups.member.N
- IsTruncated

**Note:** "IsTruncated" will always be "false".

Errors:

- ServiceFailure

Amazon Documentation: [ListGroups](#)

*ListGroupsForUser*

Request Parameters:

- UserName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- Groups.member.N
- IsTruncated

**Note:** "IsTruncated" will always be "false".

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListGroupsForUser](#)

*ListPolicies*

Request Parameters:

- OnlyAttached
- PathPrefix

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

**Note** The "Scope" request parameter, if submitted, is ignored and defaults to All. Note however that only Local policies are currently supported in HyperStore, so the policies returned by this command will all be Local policies.

Response Elements:

- IsTruncated
- Policies.member.N

**Note:** "IsTruncated" will always be "false".

Errors:

- ServiceFailure

Amazon Documentation: [ListPolicies](#)

*ListPolicyVersions*

Request Parameters:

- PolicyArn

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- IsTruncated

**Note:** "IsTruncated" will always be "false".

- Versions.member.N

**Note:** The only version returned will be "v1". HyperStore does not currently support IAM managed policy versioning.

Errors:

- InvalidInput
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListPolicyVersions](#)

#### *ListUserPolicies*

Request Parameters:

- UserName

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- IsTruncated
- PolicyNames.member.N

**Note:** "IsTruncated" will always be "false".

Errors:

- NoSuchEntity
- ServiceFailure

Amazon Documentation: [ListUserPolicies](#)

#### *ListUsers*

Request Parameters:

- PathPrefix

**Note** The "Marker" and "MaxItems" request parameters, if submitted, are ignored.

Response Elements:

- IsTruncated
- Users.member.N

**Note:** "IsTruncated" will always be "false".

Errors:

- ServiceFailure

Amazon Documentation: [ListUsers](#)

*PutGroupPolicy*

Request Parameters:

- GroupName
- PolicyDocument

**Note:** For information about HyperStore's IAM policy document support see "**IAM Supported Policy Document Elements**" (page 966).

- PolicyName

Errors:

- LimitExceeded
- MalformedPolicyDocument
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [PutGroupPolicy](#)

*PutUserPolicy*

Request Parameters:

- PolicyDocument

**Note:** For information about HyperStore's IAM policy document support see "**IAM Supported Policy Document Elements**" (page 966).

- PolicyName
- UserName

Errors:

- LimitExceeded
- MalformedPolicyDocument
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [PutUserPolicy](#)

*RemoveUserFromGroup*

Request Parameters:

- GroupName
- UserName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [RemoveUserFromGroup](#)

*UpdateAccessKey*

Request Parameters:

- AccessKeyId
- Status
- UserName

Errors:

- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [UpdateAccessKey](#)

**Note** By default the HyperStore system allows only two key pairs per IAM user. This restriction is configurable by the "**credentials.iamuser.max**" (page 514) setting in *mts.properties.erb*.

Note that an IAM user's inactive credentials (if any) count toward this limit, as well as active credentials.

*UpdateGroup*

Request Parameters:

- GroupName
- NewGroupName
- NewPath

Errors:

- EntityAlreadyExists
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [UpdateGroup](#)

*UpdateUser*

Request Parameters:

- NewPath
- NewUserName
- UserName

Errors:

- EntityAlreadyExists
- EntityTemporarilyUnmodifiable
- LimitExceeded
- NoSuchEntity
- ServiceFailure

Amazon Documentation: [UpdateUser](#)

### 14.2.1. HyperStore Extension to the IAM API: Actions for HyperStore Admin Tasks

The HyperStore IAM Service supports the use of the following Actions to perform HyperStore administrative tasks. For details about a particular Action -- including parameters, response format, and restrictions based on the role of the requesting user (system admin, group admin, or regular user) -- see the references in the Details column.

**Note** For an overview of this feature, including information about the client tool that HyperStore provides to help you use this feature, see "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

Action	Details
GetCloudianBill (see Important note below)	" <b>GET /billing Get a bill for a user or group</b> " (page 705)
GetCloudianGroup	" <b>GET /group Get a group's profile</b> " (page 722)
GetCloudianGroupList	" <b>GET /group/list Get a list of group profiles</b> " (page 724)
GetCloudianMonitorEvents	" <b>GET /monitor/events Get the event list for a node</b> " (page 736)
GetCloudianMonitorNodelist	" <b>GET /monitor/nodelist Get the list of monitored nodes</b> " (page 738)
GetCloudianMonitorHost	" <b>GET /monitor/host Get current monitoring statistics for a node</b> " (page 740)
GetCloudianMonitorRegion	" <b>GET /monitor Get current monitoring statistics for a service region</b> " (page 745)
GetCloudianQosLimits	" <b>GET /qos/limits Get QoS settings for a user or group</b> " (page 779)
GetCloudianSystemLicense	" <b>GET /system/license Get HyperStore license terms</b> " (page 803)
GetCloudianSystemVersion	" <b>GET /system/version Get HyperStore system version</b> " (page 806)
GetCloudianUsage	" <b>GET /usage Get usage data for group, user, or bucket</b> " (page 823)
GetCloudianUser	" <b>GET /user Get a user's profile</b> " (page 852)
GetCloudianUserCredentials	" <b>GET /user/credentials Get a user's S3 security credential</b> " (page 854)

Action	Details
GetCloudianUserCredentialsList	" <b>GET /user/credentials/list</b> <b>Get a user's list of S3 security credentials</b> " (page 856)
GetCloudianUserCredentialsListActive	" <b>GET /user/credentials/list/active</b> <b>Get a user's list of active S3 security credentials</b> " (page 858)
GetCloudianUserList	" <b>GET /user/list</b> <b>Get a list of user profiles</b> " (page 860)

**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 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.

## 14.3. IAM Supported Policy Document Elements

This topic describes HyperStore's support for IAM policy document content. IAM policies grant permissions to IAM groups and users. You can create IAM policy documents through the CMC (see "[Create and Manage Inline Policies for an IAM Group](#)" (page 261)) or by using a third party IAM client. Note that:

- The CMC only supports creating "inline" policies for groups. Members of the groups will then gain those permissions. In the CMC you cannot create inline policies for users, and you cannot create "managed" policies.
- The HyperStore IAM Service, if accessed by a third party client application (not the CMC), supports creating inline policies for groups or for users; and also supports creating managed policies and attaching those managed policies to groups or users.

In regard to IAM policy document content, HyperStore supports most of the standard AWS IAM policy elements that define S3 permissions or IAM permissions. HyperStore also supports most policy elements that define permissions for HyperStore administrative actions (using HyperStore extensions to the IAM specification).

### 14.3.1. Policy Document Content for Granting S3 or IAM Permissions

The CMC supports Amazon Web Services standard IAM policy formatting and most policy elements for granting S3 or IAM service permissions.

**Note** In the current release:

- \* HyperStore support for IAM policy document components is not fully comprehensive. HyperStore supports **most but not all** of the policy elements, actions, resources, and/or condition keys cited in the AWS documentation for IAM policy formation.
- \* IAM Actions are case sensitive.
- \* The use of the wildcard character "\*" to match missing text in an Action or Resource is not well supported. Explicit Action names or Resource ARNs can be grouped together in lists as a work-around.

For guidance on how to construct IAM policies for S3 or IAM service permissions, see the AWS documentation on this topic. For example:

- *Policies and Permissions*

[http://docs.aws.amazon.com/IAM/latest/UserGuide/access\\_policies.html](http://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html)

- *IAM JSON Policy Elements Reference*

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_elements.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html)

- *Actions, Resources, and Condition Keys for Amazon S3*

[https://docs.aws.amazon.com/IAM/latest/UserGuide/list\\_amazons3.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/list_amazons3.html)

- *Actions, Resources, and Condition Keys for Identity And Access Management*

[https://docs.aws.amazon.com/IAM/latest/UserGuide/list\\_identityandaccessmanagement.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/list_identityandaccessmanagement.html)

**Note:** HyperStore supports only the IAM actions listed in "**IAM Supported Actions**" (page 950).

Below is an example of a simple IAM policy document granting permission to list the contents of a bucket named "example\_bucket":

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "s3>ListBucket",
 "Resource": "arn:aws:s3:::example_bucket"
 }
]
}
```

### 14.3.2. Policy Document Content for Granting HyperStore Administrative Permissions

Along with granting S3 service permissions, you can also use IAM policies to grant IAM groups and users permissions to perform read-only HyperStore administrative actions. Note that:

- 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 below).
- The IAM Service **will not allow an IAM user to execute an administrative action that her parent HyperStore user is not allowed to execute**, even if an IAM policy grants the IAM user permission to the action.

**Note** For an overview of this feature see "**IAM Extensions for Role-Based Access to HyperStore Admin Functions**" (page 95).

The table below lists the administrative Action permissions supported by the HyperStore IAM Service, and indicates how, if an IAM user is granted permission to an Action, the IAM Service restricts the IAM user's use of the Action according to the role of the parent HyperStore user. Note that -- when specified as an "Action" in a policy document -- all HyperStore administrative actions are prefixed by "admin:" (analogous to how S3 actions are prefixed by "s3:"). For examples of policy documents granting HyperStore administrative permissions see below the table.

<b>IAM Action Permission</b>	<b>IAM User's Permission Scope Based On Parent User's Role</b>		
	<b>System Admin</b>	<b>Group Admin</b>	<b>Regular User</b>
admin:GetCloudianBill	Get any user's bill	Get bill of any user in own group	Get parent user's bill
admin:GetCloudianGroup	Get any group's profile	Get own group's profile	Not allowed
admin:GetCloudianGroupList	Get list of groups	Not allowed	Not allowed
admin:GetCloudianMonitorEvents	Get event list for a node	Not allowed	Not allowed
admin:GetCloudianMonitorNodelist	Get list of monitored nodes	Not allowed	Not allowed
admin:GetCloudianMonitorHost	Get monitoring stats for a node	Not allowed	Not allowed
admin:GetCloudianMonitorRegion	Get monitoring stats for a region	Not allowed	Not allowed
admin:GetCloudianQosLimits	Get QoS limits for any group or user	Get QoS limits for own group or users in own group	Get parent user's QoS limits
admin:GetCloudianSystemLicense	Get system license info	Not allowed	Not allowed
admin:GetCloudianSystemVersion	Get current system version	Get current system version	Get current system version
admin:GetCloudianUsage	Get usage info for any group or user	Get usage info for own group or users in own group	Get parent user's usage info
admin:GetCloudianUser	Get any user's profile	Get profile of any user in own group	Get parent user's profile
admin:GetCloudianUserCredentials	Get any user's S3 credential	Get S3 credential of any user in own group	Get parent user's S3 credential
admin:GetCloudianUserCredentialsList	Get any user's S3 credentials list	Get S3 credentials list of any user in own group	Get parent user's S3 credentials list
admin:GetCloudianUserCredentialsListActive	Get any user's active S3 credentials list	Get active S3 credentials list of any user in own group	Get parent user's active S3 credentials list
admin:GetCloudianUserList	Get list of users in any group	Get list of users in own group	Not allowed

**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**. HyperStore does not track usage, billing, or QoS information specifically for IAM users. This information is only tracked for the parent HyperStore user accounts.

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

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "admin:GetCloudianBill",
 "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.

## 14.4. IAM Common Parameters

From the "Common Parameters" section of the Amazon IAM API, the HyperStore system supports the parameters listed below. If a common parameter from the Amazon IAM API specification is not listed below, the HyperStore system does not support it. For common parameter descriptions, refer to the ["Common Parameters" section of the Amazon IAM API](#).

- Action
- Version

**Note:** Unlike Amazon's IAM implementation, in HyperStore's IAM implementation the "Version" request parameter is not required.

- X-Amz-Algorithm
- X-Amz-Credential
- X-Amz-Date
- X-Amz-Signature
- X-Amz-SignedHeaders

**Note** Like Amazon's IAM implementation, in HyperStore's IAM implementation you can either use query parameters or the HTTP header *Authorization* to submit the authentication data required by Amazon's Signature Version 2 or Signature Version 4 protocol. For more information on this topic see the Amazon documentation topic [Task 4: Add the Signature to the HTTP Request](#).

## 14.5. IAM Common Errors

From the "Common Errors" section of the Amazon IAM API, the HyperStore system supports the errors listed below. If a common error from the Amazon IAM API specification is not listed below, the HyperStore system does not support it. For error descriptions, refer to the ["Common Errors" section of the Amazon IAM API](#).

- AccessDeniedException
- IncompleteSignature
- InternalFailure
- InvalidAction
- InvalidClientTokenId
- InvalidParameterCombination
- InvalidParameterValue
- InvalidQueryParameter
- MalformedQueryString
- MissingAction
- MissingAuthenticationToken
- MissingParameter
- OptInRequired
- RequestExpired
- ServiceUnavailable
- ThrottlingException
- ValidationError

# Chapter 15. Open Source License Agreements

Cloudian, Inc. acknowledges the redistribution of open source components under the licenses shown below.

Component or Library	License	License URL	Copyright
Airlift	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright 2011 Dain Sundstrom dain@iq80.com Copyright 2010 Cedric Beust cedric@beust.com
Amazon S3 SDK	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright 2010-2014 Amazon.com, Inc. or its affiliates.
Antlr	BSD	<a href="http://www.antlr.org/license.html">http://www.antlr.org/license.html</a>	Copyright (c) 2012 Terence Parr and Sam Harwell
Apache Commons	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2018 The Apache Software Foundation.
Apache HTTPComponents	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2005-2018 The Apache Software Foundation
Apache Tomcat	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 1999-2018, The Apache Software Foundation
Apache Velocity	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2005-2018 The Apache Software Foundation
Avro	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2012 The Apache Software Foundation."
Blueimp	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright © 2010 Sebastian Tschan, <a href="https://blueimp.net">https://blueimp.net</a>
Bootstrap	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright (c) 2011-2018 Twitter, Inc. Copyright (c) 2011-2018 The Bootstrap Authors
Cassandra	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright 2009-2014 The Apache Software Foundation
CentOS	GPL and various	<a href="http://mirror.centos.org/centos/6/os/i386/EULA">http://mirror.centos.org/centos/6/os/i386/EULA</a>	Copyright © 2017 The CentOS Project
D3	BSD	<a href="https://opensource.org/licenses/BSD-3-Clause">https://opensource.org/licenses/BSD-3-Clause</a>	Copyright 2010-2017 Mike Bostock
DataStax Java Driver	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright 2012-2018, DataStax
DataTables	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright (C) 2008-2018, SpryMedia Ltd.
Disruptor	Apache	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None

Component or Library	License	License URL	Copyright
	2.0	2.0	
DropWizard	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Gson	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright 2008 Google Inc.
Guava	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Hector	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright (c) 2010 Ran Tavory
High-scale-lib	Public Domain	<a href="https://github.com/stephenc/high-scale-lib/blob/master/LICENSE">https://github.com/stephenc/high-scale-lib/blob/master/LICENSE</a>	None
Jackson	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Java	Oracle binary code license	<a href="http://www.oracle.com/technetwork/java/javase/terms/license/index.html">http://www.oracle.com/technetwork/java/javase/terms/license/index.html</a>	Copyright © 1995, 2018, Oracle and/or its affiliates.
JCraft	BSD	<a href="http://www.jcraft.com/jsch">http://www.jcraft.com/jsch</a>	Copyright (c) 2002-2015 Atsuhiko Yamanaka, JCraft,Inc.
Jedis	Custom: No limitation if copyright included	<a href="https://github.com/xetorthio/jedis/blob/master/LICENSE.txt">https://github.com/xetorthio/jedis/blob/master/LICENSE.txt</a>	Copyright (c) 2010 Jonathan Leibiusky
Jersey	CDDL v1.1	<a href="https://jersey.java.net/license.html">https://jersey.java.net/license.html</a>	Copyright ©2010-2017 Oracle Corporation.
Jetty	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2016 The Eclipse Foundation.
JNA	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Joda-Time	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright ©2002-2017 Joda.org.
Jquery	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright JS Foundation and other contributors, <a href="https://js.foundation/">https://js.foundation/</a>
jsviews	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright (c) 2015 Boris Moore, <a href="https://github.com/BorisMoore/jsviews">https://github.com/BorisMoore/jsviews</a>
JYaml	BSD	<a href="http://jyaml.sourceforge.net/license.html">http://jyaml.sourceforge.net/license.html</a>	None
log4j	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 1999-2018 The Apache Software Foundation.

Component or Library	License	License URL	Copyright
LZ4	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Netty	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
OpenCSV	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Paranamer	BSD	<a href="https://github.com/paul-hammant/paranamer/blob/master/LICENSE.txt">https://github.com/paul-hammant/paranamer/blob/master/LICENSE.txt</a>	Copyright (c) 2006 Paul Hammant & ThoughtWorks Inc
Puppet	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright (C) 2005-2016 Puppet, Inc.
Redis	3-clause BSD	<a href="http://redis.io/topics/license">http://redis.io/topics/license</a>	Copyright (c) 2006-2015, Salvatore Sanfilippo
RocksDB	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright (c) 2011 The LevelDB Authors.
SLF4J	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright (c) 2004-2017 QOS.ch
SnakeYaml	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
Snappy	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	None
SNMP4J	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2003-2018, SNMP4J.org
Spring	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright (c) 2013 GoPivotal, Inc. Copyright (c) 2000-2011 INRIA, France Telecom Copyright (c) 1999-2009, OW2 Consortium < <a href="http://www.ow2.org/">http://www.ow2.org/</a> >
Thrift	Apache 2.0	<a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>	Copyright © 2005-2018 The Apache Software Foundation
UUID	MIT	<a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>	Copyright © 2003-2013 Johann Burkard