

ECS

Version 3.2.x.x

Data Access Guide

302-004-491

03

Copyright © 2018 Dell Inc. or its subsidiaries. All rights reserved.

Published June 2018

Dell believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS-IS.” DELL MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. USE, COPYING, AND DISTRIBUTION OF ANY DELL SOFTWARE DESCRIBED IN THIS PUBLICATION REQUIRES AN APPLICABLE SOFTWARE LICENSE.

Dell, EMC, and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be the property of their respective owners.
Published in the USA.

Dell EMC
Hopkinton, Massachusetts 01748-9103
1-508-435-1000 In North America 1-866-464-7381
www.DellEMC.com

CONTENTS

Figures		7
Tables		9
Part 1	S3	11
Chapter 1	S3	13
	Amazon S3 API support in ECS.....	14
	S3 API supported and unsupported features.....	14
	Behavior where bucket already exists.....	17
	Bucket policy support.....	17
	Creating, assigning and managing bucket policies.....	19
	Bucket policy scenarios.....	20
	Supported bucket policy operations.....	22
	Supported bucket policy conditions.....	23
	S3 Extensions.....	24
	Byte range extensions.....	24
	Retention.....	29
	File system enabled.....	30
	Metadata Search.....	30
	Assign metadata index values to a bucket.....	31
	Using encryption with metadata search.....	33
	Assign metadata to objects using the S3 protocol.....	34
	Use metadata search queries.....	34
	Using Metadata Search from the ECS Java SDK	39
	ECS system metadata and optional attributes.....	40
	S3 and Swift Interoperability.....	41
	Create and manage secret keys.....	43
	Create a key for an object user.....	43
	Create an S3 secret key: self-service.....	44
	Authenticating with the S3 service.....	46
	Using s3curl with ECS.....	48
	Use SDKs to access the S3 service.....	48
	Using the Java Amazon SDK.....	49
	Java SDK client for ECS.....	50
	ECS S3 error codes.....	51
Part 2	OpenStack Swift	59
Chapter 2	OpenStack Swift	61
	OpenStack Swift support in ECS.....	62
	OpenStack Swift supported operations.....	62
	Swift extensions.....	64
	Swift byte range extensions.....	64
	Updating a byte range within an object.....	64
	Overwriting part of an object.....	66
	Appending data to an object.....	67

	Reading multiple byte ranges within an object.....	68
	Retention.....	68
	File system enabled.....	69
	S3 and Swift interoperability.....	69
	OpenStack Swift authentication.....	70
	Create Swift users in the ECS Portal.....	70
	OpenStack Version 1 authentication	72
	OpenStack Version 2 authentication.....	73
	Authentication using ECS Keystone V3 integration.....	75
	Authorization on Container.....	79
	ECS Swift error codes.....	80
Part 3	EMC Atmos	83
Chapter 3	EMC Atmos	85
	EMC Atmos API support in ECS.....	86
	Supported EMC Atmos REST API Calls.....	86
	Unsupported EMC Atmos REST API Calls.....	88
	Subtenant Support in EMC Atmos REST API Calls.....	88
	API Extensions.....	89
	Appending data to an object.....	89
	ECS support for retention and retention expiration periods for Atmos objects.....	90
	ECS Atmos error codes.....	94
Part 4	CAS	97
Chapter 4	CAS	99
	Setting up CAS support in ECS.....	100
	Cold Storage.....	100
	Compliance.....	101
	Platform hardening and Compliance.....	101
	Compliance and retention policy.....	102
	Compliance agent.....	103
	CAS retention in ECS.....	104
	Advanced retention for CAS applications: event-based retention, litigation hold, and the min/max governor.....	105
	Set up namespace retention policies.....	110
	Create and set up a bucket for a CAS user.....	111
	Set up a CAS object user.....	112
	Set up bucket ACLs for CAS.....	113
	ECS Management APIs that support CAS users.....	115
	Content Addressable Storage (CAS) SDK API support.....	116
	ECS CAS error codes.....	117
Part 5	ECS Management REST API	123
Chapter 5	ECS Management REST API	125
	ECS Management REST API introduction.....	126
	Authenticate with the ECS Management REST API.....	126
	Authenticate without cookies	126

	Logout.....	128
	ECS Management REST API whoami command.....	128
	ECS Management REST API summary.....	129
Part 6	HDFS	135
Chapter 6	ECS HDFS Introduction	137
	ECS HDFS Introduction.....	138
	Configuring Hadoop to use ECS HDFS	139
	Hadoop authentication modes.....	139
	Accessing the bucket as a file system.....	140
	Bucket Custom Group ACLs and Default Group.....	141
	Hadoop superuser and supergroup.....	141
	Multi-protocol (cross-head) access.....	142
	Proxy user.....	142
	Equivalence user.....	142
	Migration from a simple to a Kerberos Hadoop cluster.....	143
	Hadoop Kerberos authentication mode.....	143
	File system interaction.....	144
	Supported Hadoop applications.....	144
Chapter 7	Configure a simple Hadoop cluster with ECS HDFS	145
	Integrate a simple Hadoop cluster with ECS HDFS.....	146
	Install Hortonworks HDP using Ambari.....	146
	Create a bucket for HDFS using the ECS Portal.....	147
	Set custom group bucket ACLs.....	150
	Set user bucket ACLs.....	151
	Example Hadoop and ECS bucket permissions.....	152
	Plan the ECS HDFS and Hadoop integration.....	154
	Obtain the ECS HDFS installation and support package.....	155
	Deploy the ECS HDFS Client Library.....	155
	Configure ECS client properties.....	156
	Set up Hive.....	157
	Verify Hadoop access to ECS.....	159
	Secure the bucket.....	159
	Relocate the default file system from HDFS to an ECS bucket.....	160
Chapter 8	Configure a Kerberized Hadoop cluster with ECS HDFS	163
	Integrate a secure Hadoop cluster with ECS HDFS	164
	Plan migration from a simple to a Kerberos cluster.....	164
	Map group names.....	165
	Configure ECS nodes with the ECS service principal.....	165
	Enable Kerberos using Ambari.....	169
	Secure the ECS bucket using metadata.....	170
	Load metadata values to ECS using the Management REST API..	172
	Reconfigure ECS client properties.....	173
	Start Hadoop services and verify Hadoop access to ECS.....	174
Chapter 9	Troubleshooting	175
	Introduction.....	176
	Verify that AD/LDAP is correctly configured with a secure Hadoop cluster...	176

	Pig test fails: unable to obtain Kerberos principal.....	177
	Permission denied for AD user.....	177
	Permissions errors.....	177
	Failed to process request.....	180
	Enable Kerberos client-side logging and debugging.....	181
	Debug Kerberos on the KDC.....	181
	Eliminate clock skew.....	181
	Configure one or more new ECS nodes with the ECS service principal.....	182
	Workaround for Yarn directory does not exist error.....	184
Chapter 10	Appendix: Guidance on Kerberos Configuration	187
	Guidance on Kerberos configuration.....	188
	Set up the Kerberos KDC.....	188
	Configure AD user authentication for Kerberos.....	189
Chapter 11	Appendix: Hadoop core-site.xml properties for ECS HDFS	193
	Hadoop core-site.xml properties for ECS HDFS.....	194
	Sample core-site.xml for simple authentication mode.....	197
Chapter 12	Appendix: Secure bucket metadata example	199
	Secure bucket metadata.....	200

FIGURES

1	Set Compliance on a new namespace in the ECS Portal.....	103
2	Retention options for CAS buckets.....	106
3	EBR scenarios.....	108
4	Litigation hold scenarios.....	110
5	New Retention Policy.....	111
6	Retention policies for a namespace.....	111
7	CAS settings for object users.....	113
8	Edit bucket ACL.....	114
9	Bucket ACLs Management.....	114
10	ECS HDFS integration in a Hadoop cluster.....	138

FIGURES

TABLES

1	Supported S3 APIs.....	14
2	Additional features.....	16
3	Unsupported S3 APIs.....	16
4	Permissions for Object Operations.....	22
5	Permissions for Bucket Operations.....	22
6	Permissions for Bucket Sub-resource Operations.....	22
7	Supported generic AWS condition keys.....	23
8	Supported S3-specific condition keys for object operations.....	23
9	Supported S3-specific condition keys for bucket operations.....	24
10	OpenStack Swift supported calls.....	62
11	Additional features.....	63
12	OpenStack Swift unsupported calls.....	64
13	Keystone authentication provider settings.....	78
14	Supported Atmos REST API calls.....	86
15	Unsupported Atmos REST API calls.....	88
16	Atmos retention periods.....	90
17	Requirements for regular and cold archives compared.....	100
18	ECS Management API resources for retention.....	105
19	CAS API functions for event-based retention.....	108
20	CAS API functions for litigation hold.....	110
21	Bucket ACLs.....	114
22	Bucket ACL groups.....	115
23	ECS Management REST API- methods summary.....	129
24	Example bucket permissions for file system access in a simple Hadoop cluster.....	153
25	Example bucket permissions for file system access in a Kerberized Hadoop cluster.....	153
26	ECS HDFS configuration prerequisites.....	154
27	ECS HDFS Client Library.....	155
28	Hadoop configuration to enable ECS access.....	156
29	Hive templeton configuration.....	158
30	Hadoop configuration to enable Hive concurrency and ACID transactions.....	160
31	Hadoop configuration to enable ECS access.....	174
32	Hadoop core-site.xml properties.....	194

PART 1

S3

Chapter 1, "S3"

CHAPTER 1

S3

• Amazon S3 API support in ECS.....	14
• S3 API supported and unsupported features.....	14
• Bucket policy support.....	17
• S3 Extensions.....	24
• Metadata Search.....	30
• S3 and Swift Interoperability.....	41
• Create and manage secret keys.....	43
• Authenticating with the S3 service.....	46
• Using s3curl with ECS.....	48
• Use SDKs to access the S3 service.....	48
• ECS S3 error codes.....	51

Amazon S3 API support in ECS

ECS provides support for the Amazon Simple Storage Service (Amazon S3) Application Programming Interface (API).

The Amazon S3 Object Service is available on the following ports.

Protocol	Ports
HTTP	9020
HTTPS	9021

The following topics describe the support for the S3 API and the extension provided by ECS, and describe how to authenticate with the service and how to use the Software Development Kit (SDK) to develop clients to access the service:

- [S3 API supported and unsupported features](#) on page 14
- [S3 Extensions](#) on page 24
- [Metadata Search](#) on page 30
- [Create and manage secret keys](#) on page 43
- [Authenticating with the S3 service](#) on page 46
- [Use SDKs to access the S3 service](#) on page 48

Some aspects of bucket addressing and authentication are specific to ECS. If you want to configure an existing application to talk to ECS, or develop a new application that uses the S3 API to talk to ECS, you should refer to the Base URL section in the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

S3 API supported and unsupported features

ECS supports a subset of the Amazon S3 REST API.

The following sections detail the supported and unsupported APIs:

- [Supported S3 APIs](#) on page 14
- [Unsupported S3 APIs](#) on page 16

Supported S3 APIs

The following table lists the supported S3 API methods.

Table 1 Supported S3 APIs

Feature	Notes
GET Service	<p>ECS supports marker and max-keys parameters to enable paging of bucket list.</p> <pre>GET /?marker=<bucket>&max-keys=<num></pre> <p>For example:</p> <pre>GET /?marker=mybucket&max-keys=40</pre>

Table 1 Supported S3 APIs (continued)

Feature	Notes
DELETE Bucket	
DELETE Bucket cors	
DELETE Bucket lifecycle	Only the expiration part is supported in lifecycle. Policies related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
DELETE Bucket policy	
GET Bucket (List Objects)	For file system-enabled buckets, / is the only supported delimiter when listing objects in the bucket.
GET Bucket cors	
GET Bucket acl	
GET Bucket lifecycle	Only the expiration part is supported in lifecycle. Policies related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
GET Bucket policy	
GET Bucket Object versions	
GET Bucket versioning	
HEAD Bucket	
List Multipart Uploads	
PUT Bucket	Where <code>PUT</code> is performed on an existing bucket, refer to Behavior where bucket already exists on page 17.
PUT Bucket cors	
PUT Bucket acl	
PUT Bucket lifecycle	Only the expiration part is supported in lifecycle. Policies related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
PUT Bucket policy	Bucket policies cannot be configured for file system-enabled or CAS-enabled buckets. Bucket policies cannot be configured for operations that are not supported by ECS. More information about bucket policy support is provided in Bucket policy support .
PUT Bucket versioning	
DELETE Object	
Delete Multiple Objects	
GET Object	
GET Object ACL	
HEAD Object	
PUT Object	Supports chunked <code>PUT</code>
PUT Object acl	
PUT Object - Copy	
OPTIONS object	

Table 1 Supported S3 APIs (continued)

Feature	Notes
Initiate Multipart Upload	
Upload Part	
Upload Part - Copy	
Complete Multipart Upload	ECS returns an ETag of 00 for this request. This differs from the Amazon S3 response.
Abort Multipart Upload	
List Parts	

Note

- Creation of buckets using names with less than three characters will fail with 400 Bad Request, InvalidBucketName.
- When creating a bucket or object with empty content, ECS returns 400 invalid content-length value, this differs from AWS which returns 400 Bad Request.
- Copying an object to another bucket that indexes the same user metadata index key but with a different datatype is not supported and will fail with 500 Server Error.
- When listing the objects in a bucket, if you use a prefix and delimiter but supply an invalid marker, ECS throws 500 Server Error, or 400 Bad Request for a file system-enabled bucket. This differs from AWS which returns 200 OK and no objects are listed.

Table 2 Additional features

Feature	Notes
Pre-signed URLs	ECS supports use of pre-signed URLs to grant access to objects without needing credentials. More information can be found here .
Chunked PUT	PUT operation can be used to upload objects in chunks. This enables content to be sent before the total size of the payload is known. Chunked transfer uses the Transfer-Encoding header (Transfer-Encoding: chunked) to specify that content will be transmitted in chunks.

Unsupported S3 APIs

The following table lists the unsupported S3 API methods.

Table 3 Unsupported S3 APIs

Feature	Notes
DELETE Bucket tagging	
DELETE Bucket website	
GET Bucket location	ECS is only aware of a single Virtual Data Center (VDC).

Table 3 Unsupported S3 APIs (continued)

Feature	Notes
GET Bucket logging	
GET Bucket notification	Notification is only defined for reduced redundancy feature in S3. ECS does not currently support notifications.
GET Bucket tagging	
GET Bucket requestPayment	ECS uses its own model for payments.
GET Bucket website	
PUT Bucket logging	
PUT Bucket notification	Notification is only defined for the reduced redundancy feature in S3. ECS does not currently support notifications.
PUT Bucket tagging	
PUT Bucket requestPayment	ECS uses its own model for payments.
PUT Bucket website	
Object APIs	
GET Object torrent	
POST Object	
POST Object restore	This operation is related to AWS Glacier, which is not supported in ECS.

Behavior where bucket already exists

When an attempt is made to create a bucket with a name that already exists, the behavior of ECS can differ from AWS.

AWS always returns `409 Conflict` when a user who has `FULL CONTROL` permissions on the bucket, or any other permissions, attempts to recreate the bucket. When an ECS user who has `FULL_CONTROL` or `WRITE_ACP` on the bucket attempts to recreate the bucket, ECS returns `200 OK` and the ACL is overwritten, however, the owner is not changed. An ECS user with `WRITE/READ` permissions will get `409 Conflict` if they attempt to recreate a bucket.

Where an attempt to recreate a bucket is made by the bucket owner, ECS returns `200 OK` and overwrites the ACL. AWS behaves in the same way.

Where a user has no access privileges on the bucket, an attempt to recreate the bucket throws a `409 Conflict` error. AWS behaves in the same way.

Bucket policy support

ECS supports the setting of S3 bucket access policies. Unlike ACLs, which either permit all actions or none, access policies provide the ability to give specific users, or all users, conditional and granular permissions for specific actions. Policy conditions can be used to assign permissions for a range of objects that match the condition and can be used to automatically assign permissions to newly uploaded objects.

The way in which access to resources is managed when using the S3 protocol is described in <http://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html> and you can use this as the basis for understanding and using S3 bucket policies in ECS. The information in this section is provided to give basic information on the use of bucket policies, and to identify the differences when using bucket policies with ECS.

The following provides an example of an ECS bucket policy.

```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyIdNew2",
  "Statement": [
    {
      "Sid": "Granting PutObject permission to user2 ",
      "Effect": "Allow",
      "Principal": "user_n2",
      "Action": ["s3:PutObject"],
      "Resource": ["PolicyBuck1/*"],
      "Condition": {
        "StringEquals": {"s3:x-amz-server-side-encryption": [ "AES256"]}
      }
    }
  ]
}
```

Each policy is a JavaScript Object Notation (JSON) document that comprises a version, an identifier, and one or more statements.

Version

The Version field specifies the policy language version and can be either 2012-10-17 or 2008-10-17. 2008-10-17 is automatically inserted if no version is specified.

It is good practice to set the policy language for a new policy to the latest version, 2012-10-17.

Id

The Id field is optional.

Each statement comprises the following elements:

SID

A statement ID. This is a string that describes what the statement does.

Resources

The bucket or object that is the subject of the statement. The resource can be associated with a Resource or NotResource statement.

The resource name is the bucket and key name and is specified differently depending on whether you are using virtual host style addressing or path style addressing, as shown below:

```
Host Style: http://bucketname.ns1.emc.com/objectname
Path Style: http://ns1.emc.com/bucketname/objectname
```

In either case, the resource name is: bucketname/objectname .

You can use the * and ? wildcard characters, where asterisk (*) represents any combination of zero or more characters and a question mark (?) represents any

single character. For example, you can represent all objects in bucket called *bucketname*, using:

```
bucketname/*
```

Actions

The set of operations that you want to assign permissions to (allow or deny). The supported operations are listed in [Supported bucket policy operations](#) on page 22.

The operation can be associated with an `Action` or `NotAction` statement.

Effect

Can be set to `Allow` or `Deny` to determine whether you want to allow or deny the specified actions.

Principal

The ECS object user who is allowed or denied the specified actions.

To grant permissions to everyone, referred to as anonymous access, you can set the principal value to a wildcard, `"*"`, as below:

```
"Principal": "*"
```

Note

ECS bucket policies do not support federated users, nor do they support Amazon IAM users and roles.

Conditions

The condition under which the policy will be in effect. The condition expression is used to match a condition provided in the policy with a condition provided in the request.

The following condition operators are not supported: Binary, ARN, IfExists, Check Key Exists. The supported condition keys are listed in [Supported bucket policy operations](#).

More information on the elements that you can use in a policy are described in the Amazon S3 documentation, http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html.

Creating, assigning and managing bucket policies

You can create a bucket policy for a bucket from the ECS Portal (see the *ECS Administration Guide* available from the [ECS Product Documentation page](#)). It is also possible to create a policy using another editor, and associate the policy with a bucket using the ECS Management REST API or using the ECS S3 API.

The ECS Management REST API provides the following APIs to enable bucket policy sub-resources to be added, retrieved, and deleted:

- `PUT /object/bucket/{bucketName}/policy`
- `GET /object/bucket/{bucketName}/policy`

- `DELETE /object/bucket/{bucketName}/policy`

To set a policy using the ECS Management REST API you need to have either the ECS System Administrator or Namespace Administrator role.

The ECS S3 API provides the following APIs:

- `PUT Bucket Policy`
- `GET Bucket Policy`
- `DELETE Bucket Policy`

Note

To set a policy using the S3 API you must be the bucket owner.

Details of these APIs can be found in the [ECS API Reference](#).

Bucket policy scenarios

In general, the bucket owner has full control on a bucket and can grant permissions to other users and can set S3 bucket policies using an S3 client. In ECS, it is also possible for an ECS System or Namespace Administrator to set bucket policies using the Bucket Policy Editor from the ECS Portal.

You can use bucket policies in the following typical scenarios:

- Grant bucket permissions to a user
- Grant bucket permissions to all users
- Automatically assign permissions to created objects

Grant bucket permissions to a user

Where you want a user other than the bucket owner to be granted permissions on a bucket, you can specify the resource that you want to change the permissions for, set the principal attribute to the name of the user, and specify one or more actions that you want to allow.

The following example shows a policy that grants a user named `user1` the permission to update and read objects in the bucket named `mybucket`.

```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "Grant permission to user1",
      "Effect": "Allow",
      "Principal": ["user1"],
      "Action": [ "s3:PutObject", "s3:GetObject" ],
      "Resource": [ "mybucket/*" ]
    }
  ]
}
```

You can also add conditions. For example, if you only want the user to be able to read and write object when accessing the bucket from a specific IP address, you can add an `IpAddress` condition as shown in the following policy.

```
{
  "Version": "2012-10-17",
```

```

    "Id": "S3PolicyId1",
    "Statement": [
      {
        "Sid": "Grant permission ",
        "Effect": "Allow",
        "Principal": ["user1"],
        "Action": [ "s3:PutObject", "s3:GetObject" ],
        "Resource": [ "mybucket/*" ]
        "Condition": { "IpAddress": { "aws:SourceIp": "<Ip address>" }
      }
    ]
  }

```

Grant bucket permissions to all users

Where you want a user other than the bucket owner to be granted permissions on a bucket, you can specify the resource that you want to change the permissions for, set the principal attribute as anybody (*), and specify one or more actions that you want to allow.

The following example shows a policy that grants anyone permission to read objects in the bucket named `mybucket`.

```

{
  "Version": "2012-10-17",
  "Id": "S3PolicyId2",
  "Statement": [
    {
      "Sid": "statement2",
      "Effect": "Allow",
      "Principal": ["*"],
      "Action": [ "s3:GetObject" ],
      "Resource": [ "mybucket/*" ]
    }
  ]
}

```

Automatically assign permissions to created objects

You can use bucket policies to automatically enable access to ingested object data. In the following example bucket policy, `user1` and `user2` can create sub-resources (that is, objects) in the bucket named `mybucket` and can set object ACLs. With the ability to set ACLs, the users can then set permissions for other users. If you set the ACL in the same operation, a condition can be set such that a canned ACL `public-read` must be specified when the object is created. This ensures that all of the created objects can be read by anybody.

```

{
  "Version": "2012-10-17",
  "Id": "S3PolicyId3",
  "Statement": [
    {
      "Sid": "statement3",
      "Effect": "Allow",
      "Principal": ["user1", "user2"],
      "Action": [ "s3:PutObject", "s3:PutObjectAcl" ],
      "Resource": [ "mybucket/*" ]
      "Condition": { "StringEquals": { "s3:x-amz-acl": ["public-read"] } }
    }
  ]
}

```

Supported bucket policy operations

The following tables show the supported permission keywords and the operations on bucket, object, and sub-resource that they control.

Table 4 Permissions for Object Operations

Permission keyword	Supported S3 operations
s3:GetObject applies to latest version for a version-enabled bucket	GET Object, HEAD Object
s3:GetObjectVersion	GET Object, HEAD Object This permission supports requests that specify a version number
s3:PutObject	PUT Object, POST Object, Initiate Multipart Upload, Upload Part, Complete Multipart Upload PUT Object - Copy
s3:GetObjectAcl	GET Object ACL
s3:GetObjectVersionAcl	GET ACL (for a Specific Version of the Object)
s3:PutObjectAcl	PUT Object ACL
s3:PutObjectVersionAcl	PUT Object (for a Specific Version of the Object)
s3:DeleteObject	DELETE Object
s3:DeleteObjectVersion	DELETE Object (a Specific Version of the Object)
s3:ListMultipartUploadParts	List Parts
s3:AbortMultipartUpload	Abort Multipart Upload

Table 5 Permissions for Bucket Operations

Permission keyword	Supported S3 operations
s3:DeleteBucket	DELETE Bucket
s3:ListBucket	GET Bucket (List Objects), HEAD Bucket
s3:ListBucketVersions	GET Bucket Object versions
s3:GetLifecycleConfiguration	GET Bucket lifecycle
s3:PutLifecycleConfiguration	PUT Bucket lifecycle

Table 6 Permissions for Bucket Sub-resource Operations

Permission keyword	Supported S3 operations
s3:GetBucketAcl	GET Bucket acl
s3:PutBucketAcl	PUT Bucket acl
s3:GetBucketCORS	GET Bucket cors
s3:PutBucketCORS	PUT Bucket cors
s3:GetBucketVersioning	GET Bucket versioning

Table 6 Permissions for Bucket Sub-resource Operations (continued)

Permission keyword	Supported S3 operations
s3:PutBucketVersioning	PUT Bucket versioning
s3:GetBucketPolicy	GET Bucket policy
s3>DeleteBucketPolicy	DELETE Bucket policy
s3:PutBucketPolicy	PUT Bucket policy

Supported bucket policy conditions

The condition element is used to specify conditions that determine when a policy is in effect.

The following tables show the condition keys that are supported by ECS and that can be used in condition expressions.

Table 7 Supported generic AWS condition keys

Key name	Description	Applicable operators
aws:CurrentTime	Used to check for date/time conditions	Date operator
aws:EpochTime	Used to check for date/time conditions using a date in epoch or UNIX time (see Date Condition Operators).	Date operator
aws:principalType	Used to check the type of principal (user, account, federated user, etc.) for the current request.	String operator
aws:SourceIp	Used to check the requester's IP address.	String operator
aws:UserAgent	Used to check the requester's client application.	String operator
aws:username	Used to check the requester's user name.	String operator

Table 8 Supported S3-specific condition keys for object operations

Key name	Description	Applicable permissions
s3:x-amz-acl	Sets a condition to require specific access permissions when the user uploads an object.	s3:PutObject, s3:PutObjectAcl, s3:PutObjectVersionAcl
s3:x-amz-grant-permission (for explicit permissions), where permission can be:read, write, read-acp, write-acp, full-control	Bucket owner can add conditions using these keys to require certain permissions.	s3:PutObject, s3:PutObjectAcl, s3:PutObjectVersionAcl
s3:x-amz-server-side-encryption	Requires the user to specify this header in the request.	s3:PutObject, s3:PutObjectAcl
s3:VersionId	Restrict the user to accessing data only for a specific version of the object	s3:PutObject, s3:PutObjectAcl, s3:DeleteObjectVersion

Table 9 Supported S3-specific condition keys for bucket operations

Key name	Description	Applicable permissions
s3:x-amz-acl	Set a condition to require specific access permissions when the user uploads an object	s3:CreateBucket, s3:PutBucketAcl
s3:x-amz-grant-permission (for explicit permissions), where permission can be:read, write, read-acp, write-acp, full-control	Bucket owner can add conditions using these keys to require certain permissions	s3:CreateBucket, s3:PutBucketAcl
s3:prefix	Retrieve only the object keys with a specific prefix.	s3:ListBucket, s3:ListBucketVersions
s3:delimiter	Require the user to specify the delimiter parameter in the Get Bucket (List Objects) request.	s3:ListBucket, s3:ListBucketVersions
s3:max-keys	Limit the number of keys ECS returns in response to the Get Bucket (List Objects) request by requiring the user to specify the max-keys parameter.	s3:ListBucket, s3:ListBucketVersions

S3 Extensions

ECS supports a number of extensions to the S3 API.

The extensions and the APIs that support them are listed below.

- [Byte range extensions](#) on page 24
- [Retention](#) on page 29
- [File system enabled](#) on page 30
- [Metadata Search](#) on page 30

Byte range extensions

The following byte range extensions are provided:

- [Updating a byte range within an object](#) on page 25
- [Overwriting part of an object](#) on page 26
- [Appending data to an object](#) on page 27
- [Reading multiple byte ranges within an object](#) on page 28

Note

A byte range operation (update/append/overwrite) on a versioned object does not create a new version and latest version itself is updated.

A byte range operation (update/append/overwrite) on an old version of an object updates the latest version.

Updating a byte range within an object

You can use ECS extensions to the S3 protocol to update a byte range within an object.

Partially updating an object can be very useful in many cases. For example, to modify a binary header stored at the beginning of a large file. On Amazon or other S3 compatible platforms, it is necessary to send the full file again.

The following example demonstrates use of the byte range update. In the example, object1 has the value The quick brown fox jumps over the lazy dog.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:04:40 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:9qxKiHt2H7upUDPF86dvGp8VdvI=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:04:40 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:04:28 GMT
ETag: 6
Content-Type: application/json
Content-Length: 43

The quick brown fox jumps over the lazy dog.
```

To update a specific byte range within this object, the Range header in the object data request must include the start and end offsets of the object that you want to update. The format is: Range: bytes=<startOffset>-<endOffset>.

In the example below, the PUT request includes the Range header with the value bytes=10-14 indicating that bytes 10,11,12,13,14 are to be replaced by the value sent in the request. Here, the new value green is being sent.

```
PUT /bucket1/object1 HTTP/1.1
Content-Length: 5
Range: bytes=10-14
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:15:16 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:xHJcAYAEQansKLaF+/4PdLBHyaM=
Accept-Encoding: gzip, deflate, compress

green

HTTP/1.1 204 No Content
ETag: 10
x-amz-id-2: object1
x-amz-request-id: 027f037c-29ea-4670-8670-de82d0e9f52a
Content-Length: 0
Date: Mon, 12 Mar 2018 20:15:16 GMT
```

When reading the object again, the new value is now The quick green fox jumps over the lazy dog. A specific byte range within the object is updated, replacing the word brown with the word green.

```
GET /bucket1/object1 HTTP/1.1
Cookie: JSESSIONID=wdit99359t8rnvipinz4tbtu
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:16:00 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:OGVN4z8NV5vnSAilQTdpv/fcQzU=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:16:00 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:15:16 GMT
ETag: 10
Content-Type: application/json
Content-Length: 43

The quick green fox jumps over the lazy dog.
```

Overwriting part of an object

You can use ECS extensions to the S3 protocol to overwrite part of an object.

To overwrite part of an object, provide the data to be written and the starting offset. The data in the request is written starting at the provided offset. The format is:

Range: <startingOffset>- .

For example, to write the data brown cat starting at offset 10, you issue this PUT request:

```
PUT /bucket1/object1 HTTP/1.1
Content-Length: 9
Range: bytes=10-
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 12 Mar 2018 20:51:41 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:uWPjDAgmazCP5lu77Zvbo+CiT4Q=
Accept-Encoding: gzip, deflate, compress

brown cat

HTTP/1.1 204 No Content
ETag: 25
x-amz-id-2: object1
x-amz-request-id: 65be45c2-0ee8-448a-a5a0-fff82573aa3b
Content-Length: 0
Date: Mon, 12 Mar 2018 20:51:41 GMT
```

When the object is retrieved, part of the data is replaced at the provided starting offset (green fox is replaced with brown cat) and the final value is: The quick brown cat jumps over the lazy dog and cat.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
```

```
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbKl69GzhZmt4=
Accept-Encoding: gzip, deflate, compress
```

```
HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:51:55 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:51:41 GMT
ETag: 25
Content-Type: application/json
Content-Length: 51
```

```
The quick brown cat jumps over the lazy dog and cat.
```

Note that when you overwrite existing parts of an object, the size and numbers of the new parts is added to the size and numbers of the existing parts you overwrote. For example, in a bucket that has one part that is 20 KB in size, you overwrite 5 KB. When you query the bucket using `GET /object/billing/buckets/{namespace}/{bucketName}/info`, the output will show `total_mpu_size = 25 KB (not 20 KB)` and `total_mpu_parts = 2 (not 1)`.

Appending data to an object

You can use ECS extensions to the S3 protocol to append data to an object.

There may be cases where you need to append to an object, but determining the exact byte offset is not efficient or useful. For this scenario, ECS provides the ability to append data to the object without specifying an offset (the correct offset is returned to you in the response). For example, in order to append lines a log file, on Amazon or other S3 compatible platforms, you must send the full log file again.

A Range header with the special value `bytes=-1-` can be used to append data to an object. In this way, the object can be extended without knowing the existing object size. The format is: `Range: bytes=-1-`

A sample request showing appending to an existing object using a Range value of `bytes=-1-` is shown in the following example. Here the value `and cat` is sent in the request.

```
PUT /bucket1/object1 HTTP/1.1
Content-Length: 8
Range: bytes=-1-
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:46:01 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/sqOFL65riEBSWLg6t8hL0DFW4c=
Accept-Encoding: gzip, deflate, compress
```

```
and cat
```

```
HTTP/1.1 204 No Content
ETag: 24
x-amz-id-2: object1
x-amz-request-id: 087ac237-6ff5-43e3-b587-0c8fe5c08732
Content-Length: 0
Date: Mon, 12 Mar 2018 20:46:01 GMT
```

When the object is retrieved, and cat has been appended, and you can see the full value: The quick green fox jumps over the lazy dog and cat.

```
GET /bucket1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:46:56 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nGlgMDTg=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:46:56 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:46:01 GMT
ETag: 24
Content-Type: application/json
Content-Length: 51

The quick green fox jumps over the lazy dog and cat.
```

Reading multiple byte ranges within an object

You can use ECS extensions to the S3 protocol to read multiple byte ranges within an object.

Reading multiple parts of an object can be very useful in many cases. For example, to get several video parts. On Amazon or other S3 compatible platforms, it is necessary to send a different request for each part.

To read two specific byte ranges within the object named `object1`, you issue the following GET request for `Range: bytes==4-8,41-44`. The read response is words quick and lazy.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbKl69GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 206 Partial Content
Date: Mon, 12 Mar 2018 20:51:55 GMT
Content-Type: multipart/byteranges;boundary=bound04acf7f0ae3ccc
Last-Modified: Mon, 12 Mar 2018 20:51:41 GMT
Content-Length: 230

--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 4-8/50
quick
--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 41-44/50
lazy
--bound04acf7f0ae3ccc--
```

Retention

The ECS S3 head supports retention of objects to prevent them being deleted or modified for a specified period of time. This is an ECS extension and is not available in the standard S3 API.

Retention can be set in the following ways:

Retention period on object

Stores a retention period with the object. The retention period is set using an `x-emc-retention-period` header on the object.

Retention policy on object

A retention policy can be set on the object and the period associated with the policy can be set for the namespace. This enables the retention period for a group of objects to be set to the same value using a policy and can be changed for all objects by changing the policy. The use of a policy provides much more flexibility than applying the retention period to an object. In addition, multiple retention policies can be set for a namespace to allow different groups of objects to have different retention periods.

The retention policy applied to an object using an `x-emc-retention-policy` header on the object and the policy retention period must be set by your ECS administrator from the ECS Portal or using the ECS Management REST API.

Retention period on bucket

A retention period stored against a bucket sets a retention period for all objects, with the object level retention period or policy used to provide an object-specific setting where a longer retention is required. The retention period is set using an `x-emc-retention-period` header on the bucket.

When an attempt is made to modify or delete the object, the larger of the bucket retention period or the object period is used to determine whether the operation can be performed. The object period is set directly on the object or using the object retention policy.

S3 buckets can also be created from the ECS Management REST API or from the ECS Portal and the retention period for a bucket can be set from there.

Lifecycle (expiration) and retention

ECS supports S3 LifecycleConfiguration on both version-enabled buckets and non version-enabled buckets.

Where you need to modify objects and delete objects, but need to ensure that the objects are still retained for a period of time, you can enable versioning on a bucket and use the lifecycle capability to determine when deleted versions of objects will be removed from ECS.

Versioning and lifecycle are standard S3 features. However, lifecycle expiration is closely related to retention, which is an ECS extension. If the lifecycle expires before the retention period expires, the object will not be deleted until the retention period is over.

File system enabled

S3 buckets can also be filesystem (FS) enabled so that files written using the S3 protocol can be read using file protocols, such as NFS and HDFS, and vice-versa.

Enabling FS access

You can enable FS access using the `x-emc-file-system-access-enabled` header when creating a bucket using the S3 protocol. File system access can also be enabled when creating a bucket from the ECS Portal (or using the ECS Management REST API).

Limitation on FS support

The following limitations apply:

- When a bucket is FS enabled S3 lifecycle management cannot be enabled.
- When a bucket is FS enabled it is not possible to use retention.

Cross-head support for FS

Cross-head support refers to accessing objects written using one protocol using a different, ECS-supported protocol. Objects written using the S3 head can be read and written using NFS and HDFS file system protocols.

An important aspect of cross-head support is how object and file permissions translate between protocols and, in the case of file system access, how user and group concepts translate between object and file protocols.

You can find more information on the cross-head support with file systems in the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Metadata Search

The ECS S3-compatible API provides a metadata search extension that allows objects within a bucket to be indexed based on their metadata, and for the metadata index to be queried to find objects and their associated data.

Traditionally, metadata can be associated with objects using the ECS S3 API and, if you know the identity of the object you are interested in, you can read its metadata. However, without the ECS metadata search feature, it is not possible to find an object based on its metadata without iterating through the set of object in a bucket.

Metadata can be either *user metadata* or *system metadata*. System metadata is defined and automatically written to objects by ECS, user metadata is written by clients based on end-user requirements. Both system and user metadata can be indexed and used as the basis for metadata searches. The number of metadata values that can be indexed is limited to 30 and must be defined when the bucket is created.

Note

In the case of small objects (100KB and below), the ingest rate for data slightly reduces on increasing the number of index keys. Performance testing data showing the impact of using metadata indexes for smaller objects is available in the ECS Performance white paper.

When querying objects based on their indexed metadata, the objects matched by the query and the values of their indexed metadata are returned. You can also choose to return all of the system and/or user metadata associated with the returned objects. In addition to system metadata, objects also have attributes which can be returned as

part of metadata search results. The system metadata values that are available and can be indexed, and the metadata values that can optionally be returned with search query results, are listed [here](#).

The following topics cover the steps involved in setting up and using the metadata search feature:

- [Assign metadata index values to a bucket](#) on page 31
- [Assign metadata to objects using the S3 protocol](#) on page 34
- [Use metadata search queries](#) on page 34

Assign metadata index values to a bucket

You can set metadata index values on a bucket using the ECS Portal or ECS Management REST API, or using the S3 protocol. The index values must reflect the name of the metadata that they are indexing and can be based on system metadata or user metadata.

A list of the available system metadata is provided in [ECS system metadata and optional attributes](#) on page 40.

Index values are set when a bucket is created. You can disable the use of indexing on a bucket, but you cannot change or delete individual index values.

Setting index values using the Portal

The **Manage > Bucket** page enables buckets to be created and for index values to be assigned during the creation process. For more information, refer to the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Setting index values using the ECS Management REST API

The methods provided by the ECS Management REST API for working with indexes are listed in the table below and links are provided to the API reference.

API Path	Description
GET /object/bucket/searchmetadata	Lists the names of all system metadata keys available for assigning to a new bucket.
POST /object/bucket	Assigns the metadata index names that will be indexed for the specified bucket. The index names are supplied in the method payload.
GET /object/bucket	Gets a list of buckets. The bucket information for each bucket shows the metadata search details.
GET /object/bucket/{bucketname}/info	Gets the bucket details for the selected bucket. The information for the bucket includes the metadata search details.
DELETE /object/bucket/{bucketname}/searchmetadata	Stops indexing using the metadata keys.

Example: Get the list of available metadata names

The following example gets the entire list of metadata names available for indexing and that can be returned in queries.

```
s3curl.pl --id myuser -- http://{host}:9020/?searchmetadata
```

The results of the query are as follows.

```
<MetadataSearchList xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <IndexableKeys>
    <Key>
      <Name>LastModified</Name>
      <Datatype>datetime</Datatype>
    </Key>
    <Key>
      <Name>Owner</Name>
      <Datatype>string</Datatype>
    </Key>
    <Key>
      <Name>Size</Name>
      <Datatype>integer</Datatype>
    </Key>
    <Key>
      <Name>CreateTime</Name>
      <Datatype>datetime</Datatype>
    </Key>
    <Key>
      <Name>ObjectName</Name>
      <Datatype>string</Datatype>
    </Key>
  </IndexableKeys>
  <OptionalAttributes>
    <Attribute>
      <Name>ContentType</Name>
      <Datatype>string</Datatype>
    </Attribute>
    <Attribute>
      <Name>Expiration</Name>
      <Datatype>datetime</Datatype>
    </Attribute>
    <Attribute>
      <Name>ContentEncoding</Name>
      <Datatype>string</Datatype>
    </Attribute>
    <Attribute>
      <Name>Expires</Name>
      <Datatype>datetime</Datatype>
    </Attribute>
    <Attribute>
      <Name>Retention</Name>
      <Datatype>integer</Datatype>
    </Attribute>
  </OptionalAttributes>
</MetadataSearchList>
```

Example: Get the list of keys being indexed for a bucket

The following example gets the list of metadata keys currently being indexed for a bucket .

```
s3curl.pl --id myuser -- http://{host}:9020/mybucket/?searchmetadata
```

The results of this example are as follows.

```
<MetadataSearchList xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <MetadataSearchEnabled>true</MetadataSearchEnabled>
  <IndexableKeys>
    <Key>
      <Name>Size</Name>
      <Datatype>integer</Datatype>
```



```

    </Key>
    <Key>
      <Name>x-amz-meta-DAT</Name>
      <Datatype>datetime</Datatype>
    </Key>
  </IndexableKeys>
</MetadataSearchList>

```

Setting values using the S3 API

The methods provided by the S3 API for working with indexes are listed in the table below and links are provided to the API reference.

API Path	Description
GET /?searchmetadata	Lists the names of all system metadata available for indexing on new buckets.
PUT /{bucket} -H x-emc-metadata-search: {name[:datatype],...}	<p>Creates a bucket with the search metadata key indicated in the header.</p> <hr/> <p>Note</p> <p>A datatype must be associated with a user metadata key, but is not necessary for a system metadata key.</p> <hr/>
GET /{bucket}/?searchmetadata	Gets the list of metadata keys that are currently being indexed for the bucket.

Example

The example below shows how to create a bucket with metadata indexes for three system metadata keys and two user metadata keys.

```

s3curl.pl --id myuser --createbucket -- http://{host}:9020/mybucket
-H "x-emc-metadata-search:Size,CreateTime,LastModified,x-amz-meta-STR:String,x-amz-meta-INT;Integer"

```

Note

When adding a new object with x-amz-meta-, values containing special characters do not have to be url-encoded.

Using encryption with metadata search

When encryption is used on a bucket, object metadata keys that are indexed are stored in non-encrypted form, so it is always possible to perform metadata searches on encrypted buckets.

Where the encryption was performed using system-supplied keys, the object metadata returned by a query will be decrypted and shown in text form. However, if the data was encrypted using a user-supplied encryption key, metadata that is not indexed will still be encrypted when returned by a metadata search query as the user encrypted keys cannot be provided via the query.

Assign metadata to objects using the S3 protocol

End users can assign user metadata to objects using the "x-amz-meta-" header. The value assigned can be any text string and is case sensitive.

When the metadata is indexed so that it can be used as the basis of object searches (the metadata search feature), a data type is assigned to the data. When writing metadata to objects, clients should write data in the appropriate format so that it can be used correctly in searches.

The data types are:

String

If the search index term is marked as text, the metadata string will be treated as a string in all search comparisons.

Integer

If the search index term is marked as integer, the metadata string will be converted to an integer in search comparisons.

Decimal

If a search index term is marked as decimal, the metadata string will be converted to a decimal value so that the "." character is treated as a decimal point.

Datetime

If the search index term is marked as datetime, the metadata string will be treated as a date time with the expected format: `yyyy-MM-ddTHH:mm:ssZ`. If you want the string to be treated as datetime, you need to use the format `yyyy-MM-ddTHH:mm:ssZ` when specifying the metadata.

Example

The example below uses the S3 API to upload an object and two user metadata values on the object.

```
s3curl.pl --id myuser --put myfile -- http://{host}:9020/mybucket/file4 -i -H x-amz-meta-STR:String4 -H x-amz-meta-INT:407
```

Use metadata search queries

The metadata search feature provides a rich query language that enables objects that have indexed metadata to be searched.

The syntax is shown in the table below.

API Syntax	Response Body
<pre>GET /{bucket}/? query={expression} &attributes={fieldname,...} &sorted={selector} &include_older_version={true false} &max-keys=(num_keys) &marker=(marker_value)</pre>	<pre><BucketQueryResult xmlns:ns2="http:// s3.amazonaws.com/doc/2006-03-01/"> <Name>mybucket</Name> <Marker/> <NextMarker>NO MORE PAGES</NextMarker> <MaxKeys>0</MaxKeys> <ObjectMatches> <object> <objectName>file4</objectName> <objectId>09998027b1b7fbb21f50e13fabb481a237ba</pre>

API Syntax	Response Body
	<pre> 2f60f352d437c8da3c7c1c8d7589</objectId> <versionId>0</versionId> <queryMds> <type>SYSMD</type> <mdMap> <entry> <key>createtime</key> <value>1449081778025</value> </entry> <entry> <key>size</key> <value>1024</value> </entry> <entry> <key>mtime</key> <value>1449081778025</value> </entry> </mdMap> </queryMds> <queryMds> <type>USERMD</type> <mdMap> <entry> <key>x-amz-meta-INT</key> <value>407</value> </entry> <entry> <key>x-amz-meta-STR</key> <value>String4</value> </entry> </mdMap> </queryMds> <indexKey/> </object> <object ... </object> </ObjectMatches> </BucketQueryResult> </pre>

The expression keywords and their meanings are listed below:

expression

An expression in the form:

```
[ ( {condition1} [%20[and/or]%20{condition2}] ) ] [%20[and/or]%20...]
```

Where "condition" is a metadata key name filter in the form:

```
{selector} {operator}
{argument},
```

For example:

```
LastModified > 2018-03-01T11:22:00Z
```

selector

A searchable key name associated with the bucket.

operator

An operator. One of: ==, >, <, <=, >=

argument

A value that the selector is tested against.

attributes=[fieldname,...]

Specifies any optional object attributes that should be included in the report. Attribute values will be included in the report where that attribute is present on the object. The optional attribute values comprise:

- ContentEncoding
- ContentType
- Retention
- Expiration
- Expires

In addition, it is possible to return the non-indexed metadata associated with objects that are returned by the search query. The following:

ALL

Lists both system and user metadata associated with the returned objects.

ALL_SMD

Lists the system metadata associated with the returned objects.

ALL_UMD

Lists the user metadata associated with the returned objects.

sorted=[selector]

Specifies one searchable key name associated with the bucket. The key name must be a key that appears in the expression. In the absence of &sorted=keyname, the output will be sorted according to the first key name that appears in the query expression.

Note

If "or" operators are used in the expression, the sort order is indeterminate.

include-older-versions=[true|false]

When S3 versioning is enabled on a bucket, setting this to true will return current and older versions of objects that match the expression. Default is false.

max-keys

The maximum number of objects that match the query that should be returned. If there are more objects than the max-keys, a marker will be returned that can be used to retrieve more matches.

marker

The marker that was returned by a previous query and that indicates the point from which query matches should be returned.

Datetime queries

Datetime values in user metadata are specified in ISO-8601 format `yyyy-MM-dd'T'HH:mm:ssZ` and are persisted by ECS in that format. Metadata queries also use this format. However, ECS persists datetime values for system metadata as *epoch time*, the number of milliseconds since the beginning of 1970.

When a query returns results, it returns the datetime format persisted by ECS. An example of the two formats is shown below.

User metadata upload header example:

```
-H x-amz-meta-Foo:2018-03-06T12:00:00Z
```

User and System query expression format:

```
?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z
```

Query results fragment - system metadata

```
<key>createtime</key> <value>1449081777620</value>
```

Query results fragment - user metadata

```
<key>x-amz-meta-Foo</key> <value>2018-03-06T12:00:00Z</value>
```

Using markers and max-keys to paginate results

You can specify the maximum number of objects that will be returned by a query using the `max-keys` query parameter.

The example below specified a maximum number of objects as 3.

```
?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z&max-keys=3
```

Where a query matches more objects than the `max-keys` that has been specified, a marker will also be returned that can be used to return the next page objects that match the query but were not returned.

The query below specifies a marker that has been retrieved from a previous query:

```
?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z&max-keys=3&marker=r00ABXNyAD...
```

When the objects that are returned are the final page of objects, **NO MORE PAGES** is returned in the `NextMarker` of the response body.

```
<NextMarker>NO MORE PAGES</NextMarker>
```

Using special characters in queries

The use of url-encoding is required to ensure that special characters are received correctly by the ECS REST service and quoting can be required to ensure that when ECS parses the query it does not mis-interpret symbols. For example:

- When querying on x-amz-meta values, special characters must be url-encoded. For example: when using "%" (ASCII 25 hex), or "/" (ASCII 2F), they must be encoded as %25 and 2F, respectively.
- When querying on x-amz-meta values that have SQL-reserved characters the reserved characters must be escaped. This is to ensure that the SQL parser used by ECS does not consider them operators. For example: 'ab < cd' (that is, make sure a pair of quotes is passed into the service so that the SQL parser used by ECS does not consider them operators). The SQL-reserved characters include comparison operators (=, <, >, +, -, !, ~) and syntax separators (comma, semi-colon).

Different ways of quoting are possible and depend on the client being used. An example for Unix command-line tools like `S3curl.pl`, would be:

```
?query="'ab+cd<ed;ef'"
```

In this case, the search value is single-quoted and that is wrapped in double quotes.

Metadata search example

The example below uses the S3 API to search a bucket for a particular object size and user metadata value match.

Note

Some REST clients may require that you encode "spaces" with url code %20.

```
s3curl.pl --id myuser
-- "http://{host}:9020.mybucket?query=Size>1000%20and%20x-amz-meta-STR>=String4"
```

The result shows three objects that match the search.

```
<BucketQueryResult xmlns:ns2="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>mybucket</Name>
  <Marker/>
  <NextMarker>NO MORE PAGES</NextMarker>
  <MaxKeys>0</MaxKeys>
  <ObjectMatches>
    <object>
      <objectName>file4</objectName>
      <objectId>09998027b1b7fbb21f50e13fabb481a237ba2f60f352d437c8da3c7c1c8d7589</objectId>
      <versionId>0</versionId>
      <queryMds>
        <type>SYSMD</type>
        <mdMap>
          <entry>
            <key>createtime</key>
            <value>1449081778025</value>
          </entry>
          <entry>
            <key>size</key>
```

```

        <value>1024</value>
      </entry>
    <entry>
      <key>mtime</key>
      <value>1449081778025</value>
    </entry>
  </mdMap>
</queryMds>
<queryMds>
  <type>USERMD</type>
  <mdMap>
    <entry>
      <key>x-amz-meta-INT</key>
      <value>407</value>
    </entry>
    <entry>
      <key>x-amz-meta-STR</key>
      <value>String4</value>
    </entry>
  </mdMap>
</queryMds>
<indexKey/>
</object>
<object>
  <objectName>file5</objectName>
  <objectId>1ad87d86ef558ca0620a26855662da1030f7d9ff1d4bbc7c2ffdfef29943b9150</objectId>
  <queryMds>
    <type>SYSMD</type>
    <mdMap>
      <entry>
        <key>createtime</key>
        <value>1449081778396</value>
      </entry>
      <entry>
        <key>size</key>
        <value>1024</value>
      </entry>
      <entry>
        <key>mtime</key>
        <value>1449081778396</value>
      </entry>
    </mdMap>
  </queryMds>
</queryMds>
  <type>USERMD</type>
  <mdMap>
    <entry>
      <key>x-amz-meta-INT</key>
      <value>507</value>
    </entry>
    <entry>
      <key>x-amz-meta-STR</key>
      <value>String5</value>
    </entry>
  </mdMap>
</queryMds>
<indexKey/>
</object>
</ObjectMatches>
</BucketQueryResult>

```

Using Metadata Search from the ECS Java SDK

In the 3.0 SDK, there is an option to exclude the "search" and "searchmetadata" parameters from the signature if you are connecting to a pre-3.0 ECS. These

parameters were not part of the signature computation in ECS 2.x, but are now part of the computation to enhance security.

The following compatibility table is provided to show SDK support for the Metadata Search feature:

	ECS Version	
	2.x	3.x
SDK 2.x	Yes	No
SDK 3.x	Yes	Yes

ECS system metadata and optional attributes

System metadata is automatically associated with each object stored in the object store. Some system metadata is always populated and can be used as index keys, other metadata is not always populated but, where present, can be returned with metadata search query results.

System metadata

The system metadata listed in the table below can be used as keys for metadata search indexes.

Name (Alias)	Type	Description
ObjectName	string	Name of the object.
Owner	string	Identity of the owner of the object.
Size	integer	Size of the object.
CreateTime	datetime	Time at which the object was created.
LastModified	datetime	Time and date at which the object was last modified. Note Modification supported by ECS S3 byte-range update extensions, not by pure S3 API.

Optional metadata attributes

Optional system metadata attributes may or may not be populated for an object, but can be optionally returned along with search query results. The optional system metadata attributes are listed in the table below.

Name (Alias)	Type
ContentType	string
Expiration	datetime
ContentEncoding	string
Expires	datetime
Retention	integer

S3 and Swift Interoperability

S3 and Swift protocols can interoperate so that S3 applications can access objects in Swift buckets and Swift applications can access objects in S3 buckets.

When considering whether objects created using the S3 head will be accessible using the Swift head, and vice versa, you should first consider whether users can access the bucket (called a container in Swift). A bucket is assigned a bucket type (S3 or Swift, for example) based on the ECS head that created it and, if you want to ensure that an application can access both Swift and S3 buckets, you need to ensure that object users have appropriate permissions for the type of bucket. This requires some thought because the way in which permissions are determined for Swift and S3 buckets is different.

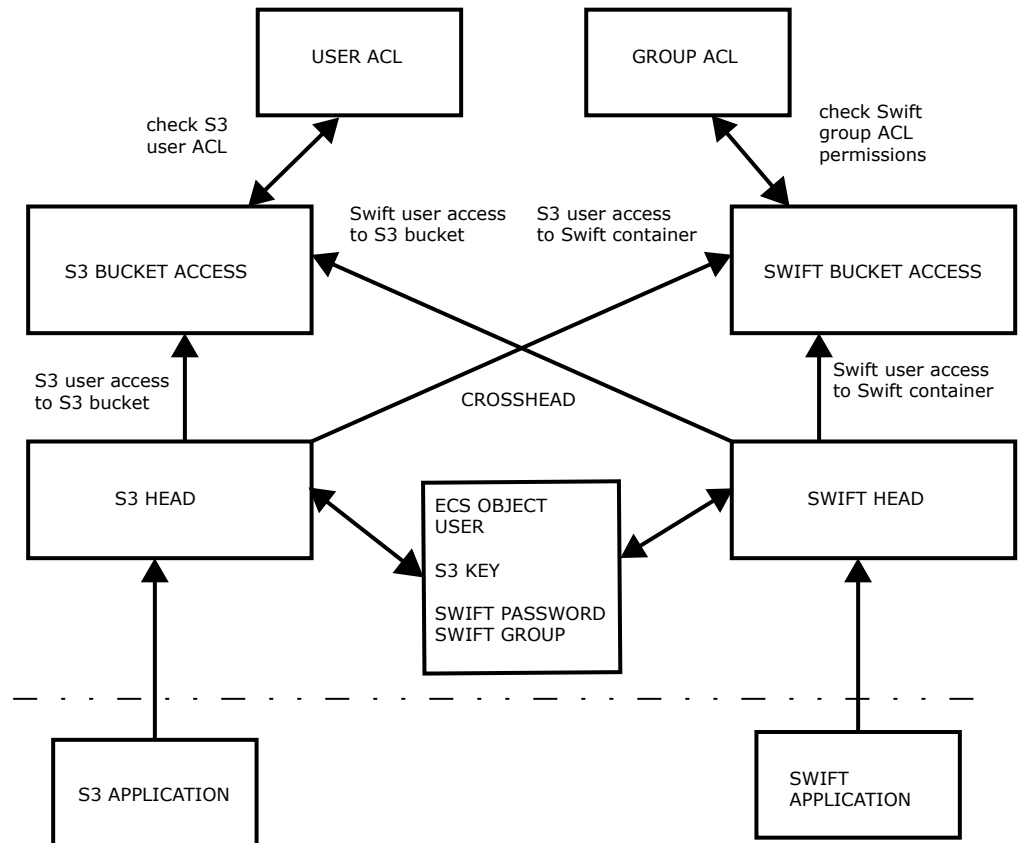
Note

S3 and Swift interoperability is not compatible with the use of bucket policies. Bucket policies apply only to bucket access using the S3 head and are not enforced when accessing a bucket using the Swift API.

In ECS, the same object user name can be given both S3 and Swift credentials. So, as far as ECS is concerned, a user called `john` who authenticates as a Swift user, can then access any S3 resources that `john` is allowed to access.

Access to a resource is determined either by being the bucket owner, or by being assigned permission on the bucket using ACLs. Where a bucket is created by an S3 user, for example, that bucket will be owned by the S3 user name and that user will have full permissions on the bucket, and a Swift user with the same name will similarly have full permissions on the bucket.

Where you want users other than the owner to be able to access a bucket, permissions can be assigned using ACLs. Access to Swift containers can be granted using group ACLs (in ECS, these are Custom Group ACLs), and the Swift head performs a check on group membership before checking group ACL permissions. Swift containers can be considered to have the `admin` group implicitly added, and, as such, any user that is a member of the `admin` group (an `admin` user) can access any other `admin` user's containers. Only `admin` users have permissions to create, delete, and list-all containers. The `admin` user's permissions only apply to the namespace to which the user belongs. Access to S3 buckets depends on user permissions (User ACLs), not group permissions. To determine access to a bucket, the S3 head checks if the user has ACL permissions on the bucket. This is illustrated in the figure below.



Because Swift uses groups to enable access to resources, for an S3 user to be able to access a Swift container, the S3 user must also be assigned to a Swift group, either the `admin` group, or a group that has been given Custom Group ACLs on the container.

In summary, one of the following conditions should be met for access to S3 buckets:

- The Swift or S3 user must be the bucket owner.
- The Swift or S3 user must have been added to the user ACL for the bucket.

One of the following conditions should be met for access to Swift containers:

- The S3 or Swift user must be the container owner.
- The S3 user must also be a Swift user and must have been added to a Swift group. The Swift group must be added as a custom group, unless the user is a member of the Swift `admin` group, which is added automatically to the custom groups.
- The Swift user must have been added to a group ACL for the container, or the user must be in the Swift `admin` group, which is added automatically to the custom groups.

Note

Reading a Swift DLO object through the S3 API will not work since the request will follow a generic code path for the read without acknowledging the presence of the `X-Object-Manifest` metadata key to be able to stitch the object back from its individual paths.

Note

For an MPU upload, the Swift `list parts` operation fails since it does not understand the `'?uploadId=<uploadId>'` sub-resource.

Create and manage secret keys

Users of the ECS object services require a secret key in order to authenticate with a service.

Secret keys can be created and made available to the object user in the following ways:

- An administrator creates a keys and distributes to the object user ([Create a key for an object user](#) on page 43).
- An domain user creates an object user account by creating a new secret key using the self-service API provided by the ECS Management REST API ([Create an S3 secret key: self-service](#) on page 44).

It is possible to have 2 secret keys for a user. When changing (sometimes referred to as "rolling over") a secret key, an expiration time in minutes can be set for the old key. During the expiration interval, both keys will be accepted for requests. This provides a grace period where an application can be updated to use the new key.

Create a key for an object user

ECS Management users can create a secret key for an object user.

- [Generate a secret key from the ECS Portal](#) on page 43
- [Create an S3 secret key using the ECS Management REST API](#) on page 44

For more information on ECS users, refer to *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Generate a secret key from the ECS Portal

You can generate a secret key at the ECS Portal.

Before you begin

- You must be an ECS System Administrator or Namespace Administrator.

If you are a System Administrator, you can create a secret key for an object user belonging to any namespace. If you are a Namespace Administrator, you can create a secret key for an object users who belongs to your namespace.

Procedure

1. In the ECS Portal, select the **Manage > Users** page.
2. In the Object Users table, select **New Object User** or select **Edit** for an existing user to which you want to assign a secret key.
3. For S3, select **Generate & Add Password**.

To change a secret key for a user, you can generate a second secret key and specify when the first key will expire.

4. Copy the generated key and email to the object user.

Create an S3 secret key using the ECS Management REST API

The ECS Management REST API enables a management user to create a secret key for an S3 object user.

The APIs is as follows:

API Path	Description
/object/user-secret-keys/{uid}	<p>API to allow secret keys to be assigned to object users and enable secret keys to be managed.</p> <p>A Namespace Administrator can create keys for users in their namespace. A System Administrator can assign keys to users in any namespace.</p> <p>To change a key, a second key can be assigned and the time at which the first key expires can be specified.</p>

You can find out more information about the API call in the [ECS API Reference](#).

Create an S3 secret key: self-service

The ECS Management REST API provides the ability to allow authenticated domain users to request a secret key to enable them to access the object store.

The [ECS API Reference](#) can be used where you want to create a custom client to perform certain ECS management operations. For simple operations domain users can use curl or a browser-based HTTP client to execute the API to create a secret key.

When a user runs the `object/secret-keys` API, ECS automatically creates an object user and assigns a secret key.

API Path	Description
/object/secret-keys	<p>API to allow S3 client users to create a new secret key that enables them to access objects and buckets within their namespace.</p> <p>This is also referred to as a self-service API.</p>

The payload for the `/object/secret-keys` can include an optional existing key expiry time.

```
<secret_key_create_param>
  <existing_key_expiry_time_mins></existing_key_expiry_time_mins>
</secret_key_create_param>
```

If you are creating a secret key for the first time, you can omit the `existing_key_expiry_time_mins` parameter and a call would be:

```
POST object/secret-keys

Request body
<?xml version="1.0" encoding="UTF-8"?>
<secret_key_create_param/>

Response
<user_secret_key>
  <secret_key>...</secret_key>
  <key_timestamp>...</key_timestamp>
```

```
<link rel="..." href="..." />
</user_secret_key>
```

Working with self-service keys

Examples provided here will help you use the ECS Management REST API to create, read, and manage secret keys.

To perform operations with secret keys you must first authenticate with the Management API. The examples provided use the `curl` tool.

- [Log in as a domain user](#) on page 45
- [Generate first key](#) on page 45
- [Generate second key](#) on page 45
- [Check keys](#) on page 46
- [Delete all secret keys](#) on page 46 .

Log in as a domain user

You can log in as a domain user and obtain an authentication token that can be used to authenticate subsequent requests.

```
curl -ik -u user@mydomain.com:<Password> https://10.241.48.31:4443/login
HTTP/1.1 200 OK
Date: Mon, 05 Mar 2018 17:29:38 GMT
Content-Type: application/xml
Content-Length: 107
Connection: keep-alive
X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwNzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODAlNTetYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8=

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loggedIn>
<user>tcas@corp.sean.com</user>
</loggedIn>
```

Generate first key

You can generate a secret key.

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwNzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODAlNTetYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{}"
https://10.241.48.31:4443/object/secret-keys | xmllint --format -

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user_secret_key>
  <link rel="self" href="/object/user-secret-keys/tcas@corp.sean.com"/>
  <secret_key>7hXZ9/EHTVvmFuYly/z3gHpihXtEUX/VZxdxDDbD</secret_key>
  <key_expiry_timestamp/>
  <key_timestamp>2018-03-05 17:39:13.813</key_timestamp>
</user_secret_key>
```

Generate second key

You can generate a second secret key and set the expiration for the first key.

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwNzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODAlNTetYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{\"existing_key_expiry_time_mins\": \"10\"}"
```

```
https://10.241.48.31:4443/object/secret-keys | xmllint --format -

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user_secret_key>
  <link rel="self" href="/object/user-secret-keys/tcas@corp.sean.com"/>
  <secret_key>l3fPCuFCG/bxoOXCPZoYuPwhXrSTwU0f1kFDaRUr</secret_key>
  <key_expiry_timestamp/>
  <key_timestamp>2018-03-05 17:40:12.506</key_timestamp>
</user_secret_key>
```

Check keys

You can check the keys that you have been assigned. In this case there are two keys with the first having an expiration date/time.

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcAVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAw
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTETmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
https://10.241.48.31:4443/object/secret-keys | xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user_secret_keys>
  <secret_key_1>7hXZ9/EHTVvmFuYly/z3gHpihXtEUX/VZxdxDDbD</secret_key_1>
  <secret_key_2>l3fPCuFCG/bxoOXCPZoYuPwhXrSTwU0f1kFDaRUr</secret_key_2>
  <key_expiry_timestamp_1>2018-03-05 17:50:12.369</key_expiry_timestamp_1>
  <key_expiry_timestamp_2/>
  <key_timestamp_1>2018-03-05 17:39:13.813</key_timestamp_1>
  <key_timestamp_2>2018-03-05 17:40:12.506</key_timestamp_2>
  <link rel="self" href="/object/secret-keys"/>
</user_secret_keys>
```

Delete all secret keys

If you need to delete your secret keys before regenerating them. You can use the following.

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcAVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAw
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTETmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{}" https://10.241.48.31:4443/object/
secret-keys/deactivate
```

Authenticating with the S3 service

The ECS S3 service allows authentication using Signature Version 2 and Signature Version 4. This topic identifies any ECS-specific aspects of the authentication process.

Amazon S3 uses an authorization header that must be present in all requests to identify the user and provide a signature for the request. The format of the authorization header differs between Signature Version 2 and Signature Version 4 authentication.

In order to create an authorization header, you need an AWS Access Key Id and a Secret Access Key. In ECS, the AWS Access Key Id maps to the ECS user id (UID). An AWS Access Key ID has 20 characters (some S3 clients, such as the S3 Browser, check this), but ECS data service does not have this limitation.

Authentication using Signature V2 and Signature V4 are introduced in:

- [Authenticating using Signature V2](#)
- [Authenticating using Signature V4](#)

The following notes apply:

- In the ECS object data service, the UID can be configured (through the ECS API or the ECS UI) with 2 secret keys. The ECS data service will try to use the first secret key, and if the calculated signature does not match, it will try to use the second secret key. If the second key fails, it will reject the request. When users add or change the secret key, they should wait 2 minutes so that all data service nodes can be refreshed with the new secret key before using the new secret key.
- In the ECS data service, namespace is also taken into HMAC signature calculation.

Authenticating using Signature V2

The Authorization header when using Signature V2 looks like this:

```
Authorization: AWS <AWSAccessKeyId>:<Signature>
```

For example:

```
GET /photos/puppy.jpg
?AWSAccessKeyId=user11&Expires=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D HTTP/
1.1
Host: myco.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

Authentication using Signature V2 is described in:

- <http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html>

Authenticating using Signature V4

The Authorization header when using Signature V4 looks like this:

```
Authorization: AWS4-HMAC-SHA256
Credential=user11/20130524/us/s3/aws4_request,
SignedHeaders=host;range;x-amz-date,
Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

The Credential component comprises your Access Key Id followed by the Credential Scope. The Credential Scope comprises Date/Region/Service Name/Termination String. For ECS, the Service Name is always s3 and the Region can be any string. When computing the signature, ECS will use the Region string passed by the client.

Authentication using Signature V4 is described in:

- <http://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html> , and
- <http://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-header-based-auth.html>

An example of a PUT bucket request using Signature V4 is provided below:

```
PUT /bucket_demo HTTP/1.1
x-amz-date: 20160726T033659Z
Authorization: AWS4-HMAC-SHA256 Credential=user11/20160726/us/s3/
aws4_request,SignedHeaders=host;x-amz-date;x-emc-
namespace,Signature=e75a150daa28a2b2f7ca24f6fd0e161cb58648a25121d3108f0af5c9451b09ce
x-emc-namespace: ns1
x-emc-rest-client: TRUE
x-amz-content-sha256: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Content-Length: 0
```

```
Host: 10.247.195.130:9021
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.2.1 (java 1.5)
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 26 Jul 2016 03:37:00 GMT
Server: ViPR/1.0
x-amz-request-id: 0af7c382:156123ab861:4192:896
x-amz-id-2: 3e2b2280876d444d6c7215091692fb43b87d6ad95b970f48911d635729a8f7ff
Location: /bucket_demo_2016072603365969263
Content-Length: 0
```

Using s3curl with ECS

A modified version of s3curl is required for use with ECS.

When using ECS custom headers (x-emc), the signature element of the Authorization header must be constructed to include the custom headers. In addition, when connecting to ECS 3.0 and later, the "search" and "searchmetadata" parameters are part of the signature computation.

You can obtain an ECS-specific version of s3curl that is modified to handle these conditions from the [EMCECS Git Repository](#).

Use SDKs to access the S3 service

When developing applications that talk to the ECS S3 service, there are a number of SDKs that will support your development activity.

The ECS Community provides information on the various clients that are available and provides guidance on their use: [ECS Community: Developer Resources](#).

The following topics describe the use of the Amazon S3 SDK and the use of the ECS Java SDK.

- [Using the Java Amazon SDK](#) on page 49
- [Java SDK client for ECS](#) on page 50

Note

If you want to make use of the ECS API Extensions (see [S3 Extensions](#) on page 24), support for these extensions is provided in the ECS Java SDK. If you do not need support for the ECS extensions, or you have existing applications that use it, you can use the Amazon Java SDK.

Note

Compatibility of the ECS Java SDK with the metadata search extension is described in [Using Metadata Search from the ECS Java SDK](#) on page 39.

Using the Java Amazon SDK

You can access ECS object storage using the Java S3 SDK.

By default the `AmazonS3Client` client object is coded to work directly against `amazon.com`. This section shows how to set up the `AmazonS3Client` to work against ECS.

In order to create an instance of the `AmazonS3Client` object, you need to pass it credentials. This is achieved through creating an `AWSCredentials` object and passing it the AWS Access Key (your ECS username) and your generated secret key for ECS.

The following code snippet shows how to set this up.

```
AmazonS3Client client = new AmazonS3Client(new BasicAWSCredentials(uid, secret));
```

By default the Amazon client will attempt to contact Amazon WebServices. In order to override this behavior and contact ECS you need to set a specific endpoint.

You can set the endpoint using the `setEndpoint` method. The protocol specified on the endpoint dictates whether the client should be directed at the HTTP port (9020) or the HTTPS port (9021).

Note

If you intend to use the HTTPS port, the JDK of your application must be set up to validate the ECS certificate successfully; otherwise the client will throw SSL verification errors and fail to connect.

In the snippet below, the client is being used to access ECS over HTTP:

```
AmazonS3Client client = new AmazonS3Client(new BasicAWSCredentials(uid, secret));
client.setEndpoint("http://ecs1.emc.com:9020");
```

When using path-style addressing (`ecs1.emc.com/mybucket`), you will need to set the `setPathStyleAccess` option, as shown below:

```
S3ClientOptions options = new S3ClientOptions();
options.setPathStyleAccess(true);

AmazonS3Client client = new AmazonS3Client(new BasicAWSCredentials(uid, secret));
client.setEndpoint("http://ecs1.emc.com:9020");
client.setS3ClientOptions(options);
```

The following code shows how to list objects in a bucket.

```
ObjectListing objects = client.listObjects("mybucket");
for (S3ObjectSummary summary : objects.getObjectSummaries()) {
    System.out.println(summary.getKey() + "    " + summary.getOwner());
}
```

The `CreateBucket` operation differs from other operations in that it expects a region to be specified. Against S3 this would indicate the datacenter in which the bucket should be created. However, ECS does not support regions. For this reason, when calling the `CreateBucket` operation, we specify the standard region, which stops the

AWS client from downloading the Amazon Region configuration file from Amazon CloudFront.

```
client.createBucket("mybucket", "Standard");
```

The complete example for communicating with the ECS S3 data service, creating a bucket, and then manipulating an object is provided below:

```
public class Test {
    public static String uid = "root";
    public static String secret = "KHBkaH0Xd7YKF43ZPFbWMBT9OP0vIcFAMkD/9dwj";
    public static String viprDataNode = "http://ecs.yourco.com:9020";

    public static String bucketName = "myBucket";
    public static File objectFile = new File("/photos/cat1.jpg");

    public static void main(String[] args) throws Exception {

        AmazonS3Client client = new AmazonS3Client(new BasicAWSCredentials(uid, secret));

        S3ClientOptions options = new S3ClientOptions();
        options.setPathStyleAccess(true);

        AmazonS3Client client = new AmazonS3Client(credentials);
        client.setEndpoint(viprDataNode);
        client.setS3ClientOptions(options);

        client.createBucket(bucketName, "Standard");
        listObjects(client);

        client.putObject(bucketName, objectFile.getName(), objectFile);
        listObjects(client);

        client.copyObject(bucketName, objectFile.getName(), bucketName, "copy-" +
            objectFile.getName());
        listObjects(client);
    }

    public static void listObjects(AmazonS3Client client) {
        ObjectListing objects = client.listObjects(bucketName);
        for (S3ObjectSummary summary : objects.getObjectSummaries()) {
            System.out.println(summary.getKey() + "    " + summary.getOwner());
        }
    }
}
```

Java SDK client for ECS

The ECS Java SDK builds on the Amazon S3 Java SDK and supports the ECS API extensions.

An example of using the ViPRS3client is shown below.

```
package com.emc.ecs.sample;

import com.amazonaws.util.StringInputStream;
import com.emc.viپر.services.s3.ViPRS3Client;

public class BucketCreate {

    private ViPRS3Client s3;
```

```

public BucketCreate() {

    URI endpoint = new URI("http://ecs.yourco.com:9020");
    String accessKey = "fred@yourco.com";
    String secretKey = "pcQQ20rDI2DHZOIWNkAug3wK4XJP9sQnZqbQJev3";
    BasicAWSCredentials creds = new BasicAWSCredentials(accessKey, secretKey);
    ViPRS3Client client = new ViPRS3Client(endpoint, creds);

}

public static void main(String[] args) throws Exception {
    BucketCreate instance = new BucketCreate();
    instance.runSample();
}

public void runSample() {

    String bucketName="mybucket";
    String key1 = "test1.txt";
    String content = "Hello World!";

    try {
        s3.createBucket(bucketName);
        s3.putObject(bucketName, key1, new StringInputStream(content), null);
    }

    catch (Exception e) {

    }

}
}

```

ECS S3 error codes

The error codes that can be generated by the ECS S3 head are listed in the following table.

Error Code	HTTP Status Code	Generic Error Code	Description Error
AccessDenied	403	AccessDenied	Access Denied
BadDigest	400	BadDigest	The Content-MD5 you specified did not match that received.
BucketAlreadyExists	409	BucketAlreadyExists	The requested bucket name is not available. The bucket namespace is shared by all users of the system. Please select a different name and try again.
BucketNotEmpty	409	BucketNotEmpty	The bucket you tried to delete is not empty.
ContentMD5Empty	400	InvalidDigest	The Content-MD5 you specified was invalid.
ContentMD5Missing	400	InvalidRequest	The required Content-MD5 header for this request is missing.

Error Code	HTTP Status Code	Generic Error Code	Description Error
EntityTooSmall	400	EntityTooSmall	The proposed upload is smaller than the minimum allowed object size.
EntityTooLarge	400	EntityTooLarge	The proposed upload exceeds the maximum allowed object size.
IncompleteBody	400	IncompleteBody	The number of bytes specified by the <code>Content-Length</code> HTTP header were not provided.
InternalServerError	500	InternalServerError	An internal error was encountered. Please try again.
ServerTimeout	500	ServerTimeout	An internal timeout error was encountered. Please try again.
InvalidAccessKeyId	403	InvalidAccessKeyId	The Access Key Id you provided does not exist.
InvalidArgument	400	InvalidArgument	Invalid Argument.
NoNamespaceForAnonymous Request	403	AccessDenied	ECS could not determine the namespace from the anonymous request. Please use a namespace BaseURL or include an <code>x-emc-namespace</code> header.
InvalidBucketName	400	InvalidBucketName	The specified bucket is not valid.
InvalidDigestBadMD5	400	InvalidDigest	The Content-MD5 you specified was invalid.
InvalidDigest	403	SignatureDoesNotMatch	The Content-MD5 you specified was an invalid.
InvalidRequest	400	InvalidRequest	Invalid Request.
InvalidPart	400	InvalidPart	One or more of the specified parts could not be found. The part might not have been uploaded.
InvalidPartOrder	400	InvalidPartOrder	The list of parts was not in ascending order. Parts list must specified in order by part number.
InvalidPartSizeZero	400	InvalidPartSizeZero	The upload part size cannot be zero.
MissingEncryption	400	InvalidRequest	The multipart upload initiate requested encryption. Subsequent part requests must include the appropriate encryption parameters.
NoEncryptionNeed	400	InvalidRequest	The multipart initiate request did not request encryption. Please resend the request without sending encryption parameters.

Error Code	HTTP Status Code	Generic Error Code	Description Error
BadMD5	400	InvalidRequest	The calculated MD5 hash of the key did not match the hash that was provided.
BadEncryptKey	400	InvalidRequest	The provided encryption parameters did not match the ones used originally.
InvalidRange	416	InvalidRange	The requested range cannot be satisfied.
KeyTooLong	400	KeyTooLong	The specified key is too long.
MalformedACLError	400	MalformedACLError	The XML provided was not well-formed or did not validate against the ECS published schema.
MalformedXML	400	MalformedXML	Malformed xml (that does not conform to the published xsd) for the configuration was sent.
MaxMessageLengthExceeded	400	MaxMessageLengthExceeded	The request was too big.
MetadataTooLarge	400	MetadataTooLarge	The metadata headers exceed the maximum allowed metadata size.
InvalidProject	400	InvalidProject	The specified project is Invalid.
InvalidVPool	400	InvalidVPool	The specified vPool (Replication Group) is Invalid.
InvalidNamespace	400	InvalidNamespace	The specified namespace is Invalid.
MethodNotAllowed	405	MethodNotAllowed	The specified method is not allowed against this resource.
MissingContentLength	411	MissingContentLength	The Content-Length HTTP header must be provided.
MissingRequestBodyError	400	MissingRequestBodyError	An empty XML document was sent. The error message is: Request body is empty.
MissingSecurityHeader	400	MissingSecurityHeader	The request was missing a required header.
IncompleteLifecycleConfig	400	IncompleteLifecycleConfig	At least one action needs to be specified in a rule.
MalformedLifecycleConfig	400	MalformedLifecycleConfig	The XML provided was not well-formed or did not validate against the published schema.
MalformedDateLifecycleConfig	400	MalformedDateLifecycleConfig	The XML provided was not well-formed or did not validate against the published schema. Invalid Date or Days.
NoSuchBucket	404	NoSuchBucket	The specified bucket does not exist.

Error Code	HTTP Status Code	Generic Error Code	Description Error
NoSuchBucketPolicy	404	NoSuchBucketPolicy	The bucket policy does not exist.
NoSuchKey	404	NoSuchKey	The specified key does not exist.
NoSuchRetention	404	NoSuchRetention	The specified retention does not exist.
ObjectUnderRetention	409	ObjectUnderRetention	The object is under retention and cannot be deleted or modified.
NoSuchUpload	404	NoSuchUpload	The specified multipart upload does not exist. The upload ID might be invalid.
NotImplemented	501	NotImplemented	The requested functionality is not implemented.
OperationAborted	409	OperationAborted	A conflicting conditional operation is currently in progress against this resource. Please try again.
PermanentRedirect	301	PermanentRedirect	The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.
PreconditionFailed	412	PreconditionFailed	At least one of the preconditions you specified did not hold.
RequestIsNotMultiPartContent	400	RequestIsNotMultiPartContent	Bucket POST must be of the enclosure type <code>multipart/form-data</code> .
RequestTimeout	400	RequestTimeout	The socket connection to the server was not read from or written to within the timeout period.
RequestTimeTooSkewed	403	RequestTimeTooSkewed	The difference between the request time and the server's time is too large.
DateIsRequired	403	AccessDenied	A valid Date or <code>x-amz-date</code> header is required.
SignatureDoesNotMatch	403	SignatureDoesNotMatch	The request signature calculated does not match the signature provided. Check the Secret Access Key and signing method.
ZeroAmzExpires	403	Forbidden	Zero value specified for <code>x-amz-expires</code> .
InvalidAmzExpires	400	Bad Request	Invalid value specified for <code>x-amz-expires</code> .
ServiceUnavailable	503	ServiceUnavailable	Please reduce your request rate.
TemporaryRedirect	307	TemporaryRedirect	Requests are being redirected to the bucket while DNS updates.

Error Code	HTTP Status Code	Generic Error Code	Description Error
TooManyBuckets	400	TooManyBuckets	The request attempted to create more buckets than allowed.
UnexpectedContent	400	UnexpectedContent	The request does not support this content.
UnresolvableGrantByEmailAddress	400	UnresolvableGrantByEmailAddress	The email address you provided does not match any account on record.
InvalidBucketState	409	InvalidBucketState	The request is not valid with the current state of the bucket.
SlowDown	503	SlowDown	Please reduce your request rate.
AccountProblem	403	AccountProblem	There is a problem with the specified account that prevents the operation from completing successfully.
CrossLocationLoggingProhibited	403	CrossLocationLoggingProhibited	Cross location logging is not allowed. Buckets in one geographic location cannot log information to a bucket in another location.
ExpiredToken	400	ExpiredToken	The provided token has expired.
IllegalVersioningConfigurationException	400	IllegalVersioningConfigurationException	The Versioning configuration specified in the request is invalid.
IncorrectNumberOfFilesInPostRequest	400	IncorrectNumberOfFilesInPostRequest	POST requires exactly one file upload per request.
InvalidAddressingHeader	500	InvalidAddressingHeader	The specified role must be Anonymous role.
InvalidLocationConstraint	400	InvalidLocationConstraint	The specified location constraint is not valid.
InvalidPolicyDocument	400	InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.
InvalidStorageClass	400	InvalidStorageClass	The storage class you specified is not valid.
InvalidTargetBucketForLogging	400	InvalidTargetBucketForLogging	The target bucket for logging does not exist, is not owned by you, or does not have the appropriate grants for the log delivery group.
InvalidToken	400	InvalidToken	The provided token is malformed or otherwise invalid.
InvalidURI	400	InvalidURI	Unable to parse the specified URI.
MalformedPOSTRequest	400	MalformedPOSTRequest	The body of the POST request is not well-formed <code>multipart/form-data</code> .

Error Code	HTTP Status Code	Generic Error Code	Description Error
MaxPostPreDataLengthExceededError	400	MaxPostPreDataLengthExceededError	The POST request fields preceding the upload file were too large.
NoLoggingStatusForKey	400	NoLoggingStatusForKey	There is no such thing as a logging status sub-resource for a key.
NoSuchLifecycleConfiguration	404	NoSuchLifecycleConfiguration	The lifecycle configuration does not exist.
NoSuchVersion	404	NoSuchVersion	Indicates that the version ID specified in the request does not match an existing version.
RequestTorrentOfBucketError	400	RequestTorrentOfBucketError	Requesting the torrent file of a bucket is not permitted.
UserKeyMustBeSpecified	400	UserKeyMustBeSpecified	The bucket POST must contain the specified field name. If it is specified please check the order of the fields.
AmbiguousGrantByEmailAddress	400	AmbiguousGrantByEmailAddress	The email address you provided is associated with more than one account.
BucketAlreadyOwnedByYou	409	BucketAlreadyOwnedByYou	The previous request to create the named bucket succeeded and you already own it.
CredentialsNotSupported	400	CredentialsNotSupported	The request does not support credentials.
InlineDataTooLarge	400	InlineDataTooLarge	The inline data exceeds the maximum allowed size.
InvalidPayer	403	InvalidPayer	All access to this object has been disabled.
TokenRefreshRequired	400	TokenRefreshRequired	The provided token must be refreshed.
AccessModeNotSupported	409	AccessModeNotSupported	The bucket does not support file access or the requested access mode is not allowed.
AccessModeInvalidToken	409	AccessModeInvalidToken	The token for the file access switch request is invalid.
NoSuchBaseUrl	400	NoSuchBaseUrl	The specified BaseUrl does not exist.
NoDataStoreForVirtualPool	404	NoDataStoreForVirtualPool	No Data Store found for Replication Group of the bucket.
VpoolAccessNotAllowed	400	Cannot Access Vpool	Bucket is hosted on a Replication Group that is not accessible from S3.
InvalidCorsRequest	403	InvalidCorsRequest	Invalid CORS request.
InvalidCorsRule	400	InvalidCorsRule	Invalid CORS rule.

Error Code	HTTP Status Code	Generic Error Code	Description Error
NoSuchCORSConfiguration	404	NoSuchCORSConfiguration	The CORS configuration does not exist.
InvalidAclRequest	404	NoACLFound	The ACL does not exist.
InsufficientStorage	507	Insufficient Storage	The server cannot process the request because there is not enough space on disk.
BadMaxParts	400	InvalidArgument	Argument max-parts must be an integer between 0 and 2147483647.
BucketNotFound	404	NoSuchBucket	The specified bucket does not exist.
NotSupported	400	Not Supported	The bucket may be locked.
InvalidContentLength	400	Invalid content length	The content length has invalid value.
InvalidVersioningRequest	403	Invalid request for version control	The bucket is in compliance mode.
InvalidLifeCycleRequest	403	Invalid request for life cycle	The bucket is in compliance mode.
RetentionPeriodRequired	400	Invalid request for bucket with compliance	The bucket requires a retention period.
Conflict	409	Conflict	The bucket may be locked.
MethodForbidden	403	Forbidden	Check if quota has been exceeded.
NotAcceptable	406	Content encoding not acceptable	The object <code>Content-Encoding</code> does not match requested <code>Accept-Content</code> .
InvalidEncoding	400	Invalid URL encoding	The URL encoding used is invalid.
InvalidMetadataQuery	400	Invalid metadata query entered	The metadata query entered does not conform to valid syntax
InvalidMetadataSearchList	400	Invalid metadata search list entered	A keyname on the request is not a valid indexable key, or the format of the request list is incorrect.
MetadataSearchNotEnabled	405	Metadata search not enabled	Metadata search is not enabled for this bucket.
MetadataSearchBadParameter	400	Metadata search invalid parameter used in query	Invalid search index key name, sort key name or attribute name value.
MetadataSearchInvalidArgument	400	Metadata search invalid parameter used in query	Invalid search index value format or operator used.
MetadataSearchInvalidValueforDatatype	400	Metadata search key indexing found invalid input value	Object operation failed because a user metadata value cannot be converted to its defined datatype.
MetadataOperationNotSupported	405	Metadata search operation not supported	Metadata query with both AND and OR logical operators not supported.
MetadataSearchBadSortParameter	400	Metadata search invalid sort parameter	The sort parameter has to be present in the query as a search parameter.

Error Code	HTTP Status Code	Generic Error Code	Description Error
MetadataSearchRestriction	400	Buckets that are encrypted or within an encrypted namespace can not have metadata search enabled	Metadata search is mutually exclusive with bucket/namespace encryption.
MetadataSearchTooManyIndexKeys	400	The number of Index keys exceeds the maximum allowed	The number of keys to be indexed exceeds the maximum number allowed, try with fewer keys.
InvalidOrNoCustomerProvidedEncryptionKey	400	Invalid or no customer provided encryption key	No encryption key, or an encryption key that did not match the one in the system, was provided.
DareUnavailable	403	Server side encryption (D@RE) is not supported	D@RE JAR/license is unavailable hence server side encryption requests are not supported.
SelfCopyInvalidRequest	400	InvalidRequest	The copy request is illegal because it is trying to copy an object to itself without changing the object's metadata or encryption attributes.
OverLappingPrefixes	400	Invalid Request	Found overlapping prefixes.
SamePrefix	400	Invalid Request	Found two rules with same prefix.
XAmzContentSHA256Mismatch	400	XAmzContentSHA256Mismatch	The Content-SHA256 you specified did not match what we received
InvalidJSON	400	InvalidJSON	Policies must be valid JSON and the first byte must be { .
InvalidBucketPolicy	400	InvalidBucketPolicy	Invalid Bucket Policy.
MalformedPolicy	400	MalformedPolicy	Malformed Policy.
MaxIDLengthExceeded	400	InvalidArgument	ID length should not exceed allowed limit of 255.
CrossHeadAccessBeforeUpgrade	400	InvalidRequest	Cross head access is not supported.
InvalidDate	400	InvalidArgument	Date must be no earlier than 1970-01-01T00:00:00.000Z.
BadContentLengthRequest	400	RequestTimeout	Content-Length specified is not matching with Length of the Content in the body.

PART 2

OpenStack Swift

[Chapter 2, "OpenStack Swift"](#)

CHAPTER 2

OpenStack Swift

• OpenStack Swift support in ECS	62
• OpenStack Swift supported operations	62
• Swift extensions	64
• Swift byte range extensions	64
• Retention	68
• File system enabled	69
• S3 and Swift interoperability	69
• OpenStack Swift authentication	70
• Authorization on Container	79
• ECS Swift error codes	80

OpenStack Swift support in ECS

ECS includes support for the OpenStack Swift API and can replace Swift in an OpenStack environment. This part describes the supported operations and the mechanisms for authorization and authentication.

The OpenStack Swift Service is made available on the following ports.

Protocol	Ports
HTTP	9024
HTTPS	9025

ECS supports the OpenStack Swift API and can be used with applications that support that API. The following topics describe supported methods, the ECS extensions, and the mechanism for authentication:

- [OpenStack Swift supported operations](#) on page 62
- [Swift extensions](#) on page 64
- [OpenStack Swift authentication](#) on page 70
- [Authorization on Container](#) on page 79

Examples showing the use of the OpenStack Swift API can be found here:

- [OpenStack API Examples](#)

In an OpenStack environment, ECS can be used as a replacement for the OpenStack Swift component or alongside an existing OpenStack Swift installation. While ECS can be used with any OpenStack distribution, it has been tested with Mirantis OpenStack 9.1. Please note that ECS has been tested as a Swift replacement for end user object storage and not as a Glance backend.

Using OpenStack with ECS requires you to configure ECS so that it can authenticate OpenStack users. You can refer to [Authentication using ECS Keystone V3 integration](#) for information on configuring authentication.

OpenStack Swift supported operations

The following sections list the OpenStack REST API requests that are supported by ECS.

- [Supported OpenStack Swift calls](#) on page 62
- [Unsupported OpenStack Swift calls](#) on page 63

This information is taken from the Object Storage API V1 section of the [OpenStack API Reference](#) documentation.

Supported OpenStack Swift calls

The following OpenStack Swift REST API calls are supported in ECS.

Table 10 OpenStack Swift supported calls

Method	Path	Description
GET	v1/{account}	Retrieve a list of existing storage containers ordered by names.

Table 10 OpenStack Swift supported calls (continued)

Method	Path	Description
POST	v1/{account}	Create or update an account metadata by associating custom metadata headers with the account level URI. These headers must take the format X-Account-Meta-.*.
GET	v1/{account}/{container}	Retrieve a list of objects stored in the container.
PUT	v1/{account}/{container}	Create a container.
DELETE	v1/{account}/{container}	Delete an empty container.
POST	v1/{account}/{container}	Create or update the arbitrary container metadata by associating custom metadata headers with the container level URI. These headers must take the format X-Container-Meta-.*.
HEAD	v1/{account}/{container}	Retrieve the container metadata. Currently does not include object count and bytes used. User requires administrator privileges.
GET	v1/{account}/{container}/{object}	Retrieve the object's data. Note GET range on a Static Large Object (SLO) will not work if the segments were created prior to ECS 3.0.
PUT	v1/{account}/{container}/{object}	Write, or overwrite, an object's content and metadata. Used to copy existing object to another object using X-Copy-From header to designate source. For a Dynamic Large Object (DLO) or a SLO the object can be a manifest, as described here .
DELETE	v1/{account}/{container}/{object}	Remove an object from the storage system permanently. In combination with the COPY command you can use COPY then DELETE to effectively move an object.
HEAD	v1/{account}/{container}/{object}	Retrieve object metadata and other standard HTTP headers.
POST	v1/{account}/{container}/{object}	Set and overwrite arbitrary object metadata. These metadata must take the format X-Object-Meta-.*. X-Delete-At or X-Delete-After for expiring objects can also be assigned by this operation. But other headers such as Content-Type cannot be changed by this operation.

Table 11 Additional features

Feature	Notes
Temporary URLs	ECS supports the use of temporary URLs to enable users to be given access to objects without needing credentials. More information can be found here .

Unsupported OpenStack Swift calls

The following OpenStack Swift REST API calls are not supported in ECS.

Table 12 OpenStack Swift unsupported calls

Method	Path	Description
COPY	v1/{account}/{container}/{object}	Copy operation can be achieved using PUT v1/{account}/{container}/{object} with X-Copy-From header.
HEAD	v1/{account}	Retrieve the account metadata. Not fully supported as returns zero for the bytes stored (X-Account-Bytes-Used).

Swift extensions

ECS supports a number of extensions to the Swift API.

The extensions and the APIs that support them are listed below.

- [Swift byte range extensions](#) on page 64
- [Retention](#) on page 68
- [File system enabled](#) on page 69

Swift byte range extensions

The following byte range extensions are provided:

- [Updating a byte range within an object](#) on page 64
- [Overwriting part of an object](#) on page 66
- [Appending data to an object](#) on page 67
- [Reading multiple byte ranges within an object](#) on page 68

Updating a byte range within an object

You can use ECS extensions to the Swift protocol to update a byte range within an object.

Partially updating an object is useful in many cases. For example, to modify a binary header stored at the beginning of a large file. On Swift or other Swift compatible platforms, it is necessary to send the full file again.

The following example demonstrates use of the byte range update. In the example, **object1** has the value **The quick brown fox jumps over the lazy dog**.

```
GET /container1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:04:40 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:9qxKiHt2H7upUDPF86dvGp8VdvI=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:04:40 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:04:28 GMT
ETag: 6
Content-Type: application/json
Content-Length: 43
```



```
The quick brown fox jumps over the lazy dog.
```

To update a specific byte range within this object, the Range header in the object data request must include the start and end offsets of the object that you are updating. The format is: Range: bytes=<startOffset>-<endOffset>.

In the example below, the PUT request includes the Range header with the value bytes=10-14 indicating that bytes 10,11,12,13,14 are replaced by the value sent in the request. Here, the new value green is being sent.

```
PUT /container1/object1 HTTP/1.1
Content-Length: 5
Range: bytes=10-14
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:15:16 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:xHJcAYAEQansKLaF+/4PdLBHyaM=
Accept-Encoding: gzip, deflate, compress

green

HTTP/1.1 204 No Content
ETag: 10
x-amz-id-2: object1
x-amz-request-id: 027f037c-29ea-4670-8670-de82d0e9f52a
Content-Length: 0
Date: Mon, 12 Mar 2018 20:15:16 GMT
```

When reading the object again, the new value is now The quick green fox jumps over the lazy dog. A specific byte range within the object is updated, replacing the word brown with the word green.

```
GET /container1/object1 HTTP/1.1
Cookie: JSESSIONID=wdit99359t8rnvipinz4tbtu
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:16:00 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:OGVN4z8NV5vnSAilQTdpv/fcQzU=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:16:00 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:15:16 GMT
ETag: 10
Content-Type: application/json
Content-Length: 43

The quick green fox jumps over the lazy dog.
```

Overwriting part of an object

You can use ECS extensions to the Swift protocol to overwrite part of an object.

To overwrite part of an object, you provide the data to be written and the starting offset. The data in the request is written starting at the provided offset. The format is:

Range: <startingOffset>-

For example, to write the data `brown cat` starting at offset 10, you issue the following PUT request:

```
PUT /container1/object1 HTTP/1.1
Content-Length: 9
Range: bytes=10-
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:51:41 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:uwPjDgmaZCP5lu77Zvbo+CiT4Q=
Accept-Encoding: gzip, deflate, compress

brown cat

HTTP/1.1 204 No Content
ETag: 25
x-amz-id-2: object1
x-amz-request-id: 65be45c2-0ee8-448a-a5a0-fff82573aa3b
Content-Length: 0
Date: Mon, 12 Mar 2018 20:51:41 GMT
```

When the object is retrieved, part of the data is replaced at the provided starting offset (green fox is replaced with brown cat) and the final value is: The quick brown cat jumps over the lazy dog and cat.

```
GET /container1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbKl69GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:51:55 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:51:41 GMT
ETag: 25
Content-Type: application/json
Content-Length: 51

The quick brown cat jumps over the lazy dog and cat.
```

Note that when you overwrite existing parts of an object, the size and numbers of the new parts is added to the size and numbers of the existing parts you overwrote. For example, in a bucket that has one part that is 20 KB in size, you overwrite 5 KB. When you query the bucket using `GET /object/billing/buckets/{namespace}/{bucketName}/info`, the output will show `total_mpu_size = 25 KB (not 20 KB)` and `total_mpu_parts = 2 (not 1)`.

Appending data to an object

You can use ECS extensions to the Swift protocol to append data to an object.

There may be cases where you need to append to an object, but determining the exact byte offset is not efficient or useful. For this scenario, ECS provides the ability to append data to the object without specifying an offset (the correct offset is returned to you in the response). For example, to append lines to a log file, on Swift or other Swift compatible platforms, you must send the full log file again.

A Range header with the special value `bytes=-1-` is used to append data to an object. In this way, the object is extended without knowing the existing object size. The format is: `Range: bytes=-1-`

A sample request showing appending to an existing object using a Range value of `bytes=-1-` is shown in the following example. Here the value `and cat` is sent in the request.

```
PUT /container1/object1 HTTP/1.1
Content-Length: 8
Range: bytes=-1-
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:46:01 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/sqOFL65riEBSWLg6t8hL0DFW4c=
Accept-Encoding: gzip, deflate, compress

and cat

HTTP/1.1 204 No Content
ETag: 24
x-amz-id-2: object1
x-amz-request-id: 087ac237-6ff5-43e3-b587-0c8fe5c08732
Content-Length: 0
Date: Mon, 12 Mar 2018 20:46:01 GMT
```

When the object is retrieved, and `cat` is appended, and you see the full value: The quick green fox jumps over the lazy dog and cat.

```
GET /container1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 12 Mar 2018 20:46:56 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nGlgMDTg=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 12 Mar 2018 20:46:56 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 12 Mar 2018 20:46:01 GMT
ETag: 24
Content-Type: application/json
Content-Length: 51

The quick green fox jumps over the lazy dog and cat.
```

Reading multiple byte ranges within an object

You can use ECS extensions to the Swift protocol to read multiple byte ranges within an object.

Reading multiple parts of an object is very useful in many cases. For example, to get several video parts. On Swift or other Swift compatible platforms, it is necessary to send a different request for each part

To read two specific byte ranges within the object named `object1`, you issue the following GET request for `Range: bytes==4-8,41-44`. The read response is the words `quick` and `lazy`.

```
GET /container1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbKl69GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 206 Partial Content
Date: Mon, 12 Mar 2018 20:51:55 GMT
Content-Type: multipart/byteranges;boundary=bound04acf7f0ae3ccc
Last-Modified: Mon, 12 Mar 2018 20:51:41 GMT
Content-Length: 230

--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 4-8/50
quick
--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 41-44/50
lazy
--bound04acf7f0ae3ccc--
```

Retention

The ECS Swift head supports retention of objects to prevent them being deleted or modified for a specified period of time. This is an ECS extension and is not available in the standard Swift API.

Retention can be set in the following ways:

Retention period on object

Stores a retention period with the object. The retention period is set using an `x-emc-retention-period` header on the object.

Retention policy on object

A retention policy can be set on the object and the period associated with the policy can be set for the namespace. This enables the retention period for a group of objects to be set to the same value using a policy and can be changed for all objects by changing the policy. The use of a policy provides much more flexibility than applying the retention period to an object. In addition, multiple retention policies can be set for a namespace to allow different groups of objects to have different retention periods.

The retention policy applied to an object using an `x-emc-retention-policy` header on the object and the policy retention period must be set using the ECS Management REST API (or from the ECS Portal).

Retention period on bucket

A retention period stored against a bucket sets a retention period for all objects, with the object level retention period or policy used to provide an object-specific setting where a longer retention is required. The retention period is set using an `x-emc-retention-period` header on the bucket.

When an attempt is made to modify or delete the object, the larger of the bucket retention period or the object period, set directly on the object or using the object retention policy, is used to determine whether the operation can be performed.

File system enabled

Swift buckets can also be file system (FS) enabled so that files written using the Swift protocol can be read using file protocols, such as NFS and HDFS, and vice-versa.

Enabling FS access

You can enable file system access using the `x-emc-file-system-access-enabled` header when creating a bucket using the Swift protocol. File system access can also be enabled when creating a bucket from the ECS Portal (using the ECS Management REST API).

Limitation on FS support

When a bucket is FS enabled it is not possible to use retention.

Cross-head support for FS

Cross-head support refers to accessing objects written using one protocol using a different, ECS-supported protocol. Objects written using the Swift head can be read and written using NFS and HDFS file system protocols.

An important aspects of cross-head support is how object/file permissions translate between protocols and, in the case of file system access, how user and group concepts translate between object and file protocols.

You can find more information on the cross-head support with file systems in the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

S3 and Swift interoperability

S3 and Swift protocols can interoperate so that S3 applications can access objects in Swift buckets and Swift applications can access objects in S3 buckets.

Refer to [S3 and Swift Interoperability](#) for more details about Swift and S3 interoperability.

Note

S3 and Swift interoperability is not compatible with the use of bucket policies. Bucket policies apply only to access using the S3 head and are not enforced when accessing a bucket using the Swift API.

OpenStack Swift authentication

ECS provides support for different versions of the OpenStack Swift Authentication protocol.

v1

ECS enables object users to authenticate with the ECS Swift service and obtain an authentication token that can be used when making subsequent API calls to the ECS Swift service. See [OpenStack Version 1 authentication](#) on page 72.

v2

ECS enables object users to authenticate with the ECS Swift service to obtain a scoped token, that is, a token associated with a tenant (equivalent to a project), that can be used when making subsequent API calls to the ECS Swift service. See [OpenStack Version 2 authentication](#) on page 73

v3

ECS validates Keystone V3 users that present tokens scoped to a Keystone project. See [Authentication using ECS Keystone V3 integration](#) on page 75.

For v1 and v2 protocols, access to the ECS object store using the OpenStack Swift protocol requires an ECS object user account and a Swift password.

For v3, users are created, and assigned to projects and roles, outside of ECS using a Keystone V3 service. ECS does not perform authentication, but validates the authentication token with the Keystone V3 service.

Assigning Swift credentials to ECS object users is described in [Create Swift users in the ECS Portal](#) on page 70.

Create Swift users in the ECS Portal

ECS object users can be given credentials to access the ECS object store using the OpenStack Swift protocol.

Before you begin

- This operation requires the System Administrator or Namespace Administrator role in ECS.
- A System Administrator can assign new object users into any namespace.
- A Namespace Administrator can assign new object users into the namespace in which they are the administrator.
- The Swift user must belong to an OpenStack group. A group is a collection of Swift users that have been assigned a role by an OpenStack administrator. Swift users that belong to the `admin` group can perform all operations on Swift buckets (containers) in the namespace to which they belong. You should not add ordinary Swift users to the `admin` group. For Swift users that belong to any group other than the `admin` group, authorization depends on the permissions that are set on the Swift bucket. You can assign permissions on the bucket from the OpenStack Dashboard UI or in the ECS Portal using the Custom Group ACL for the bucket. For more information on custom group ACLs and adding object users to ECS, see the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Procedure

1. In the ECS Portal, select **Manage > Users**.

2. On the **User Management** page, you can create a new object user who will access the ECS object store through the Swift object protocol in one of two ways:
 - a. Click **New Object User** to create a new object user.
 - On the **New Object User** page, in the **Name** field, type a name for the object user.
 - In the **Namespace** field, select the namespace to which the user belongs.
 - Click **Next to Add Passwords**.
 - b. Click **Edit** in the **Actions** column beside an existing user and add a Swift password to the existing user.
3. On the **Update Passwords for User <username>** page, in the **Swift Groups** field, enter the Swift group to which the user belongs.

If you specify the `admin` group, users will automatically be able to perform all container operations. If you specify a different group, that group must be given permissions on the container. Refer to [Authorization on Container](#) on page 79 for more information on container authorization.

The screenshot shows the 'Update Passwords for User fred' form. It contains the following fields and sections:

- Name ***: A text input field containing 'fred'.
- Namespace ***: A text input field containing '_a_ns2'.
- Object Access**: A section with a dropdown menu set to 'S3 / Atmos', a 'Show Secret Key' checkbox, and a 'Generate & Add Secret Key' button.
- Swift Groups**: A text input field with placeholder text 'Ex.: admin, users'. This field is highlighted with a red box.
- Swift password**: A text input field for the password. This field is also highlighted with a red box.
- Set Groups & Password**: A blue button located to the right of the password field.

4. In the **Swift password** field, type a password for the Swift user.
5. Click **Set Groups & Password**.

OpenStack Version 1 authentication

You can authenticate with the ECS OpenStack Swift service using V1 of the authentication protocol.

Procedure

1. Acquire a UID and password for an ECS object user.

You can do this from the ECS Portal (see [Create Swift users in the ECS Portal](#) on page 70) or you can call the following ECS REST API to generate a password.

Request:

```
PUT /object/user-password/myUser@emc.com
<user_password_create>
<password>myPassword</password>
<namespace>EMC_NAMESPACE</namespace>
</user_password_create>
```

Response:

```
HTTP 200
```

2. Call the OpenStack authentication REST API shown below. Use port 9024 for HTTP, or port 9025 for HTTPS.

Request:

```
GET /auth/v1.0
X-Auth-User: myUser@emc.com
X-Auth-Key: myPassword
```

Response:

```
HTTP/1.1
204 No
Content
Date: Mon, 12 Nov 2010 15:32:21 GMT
Server: Apache

X-Storage-Url: https://{hostname}/v1/account
X-Auth-Token: ECS_e6384f8ffcd849fd95d986a0492ea9a6
Content-Length: 0
```

Results

If the UID and password are validated by ECS, the storage URL and token are returned in the response header. Further requests are authenticated by including this token. The storage URL provides the host name and resource address. You can access containers and objects by providing the following X-Storage-Url header:

```
X-Storage-Url: https://{hostname}/v1/{account}/{container}/{object}
```


The generated token expires 24 hours after creation. If you repeat the authentication request within the 24 hour period using the same UID and password, OpenStack will return the same token. Once the 24 hour expiration period expires, OpenStack will return a new token.

In the following simple authentication example, the first REST call returns an X-Auth-Token. The second REST call uses that X-Auth-Token to perform a GET request on an account.

```
$ curl -i -H "X-Storage-User: tim_250@sanity.local" -H "X-Storage-
Pass: 1f09X3xyrVhfcokqy3U1UyTY029gha5T+k+vjLqS"

http://ecs.yourco.com:9024/auth/v1.0
```

```
HTTP/1.1 204 No Content
X-Storage-Url: http://ecs.yourco.com:9024/v1/s3
X-Auth-Token: ECS_8cf4a4e943f94711aad1c91a08e98435
Server: Jetty(7.6.4.v20120524)
```

```
$ curl -v -X GET -s -H "X-Auth-Token:
8cf4a4e943f94711aad1c91a08e98435"

http://
ecs.yourco.com:9024/v1/s3
```

```
* About to connect() to ecs.yourco.com port 9024 (#0)
* Trying 203.0.113.10...
* Adding handle: conn: 0x7f9218808c00
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7f9218808c00) send_pipe: 1, recv_pipe: 0
* Connected to ecs.yourco.com (203.0.113.10) port 9024 (#0)

> GET /v1/s3 HTTP/1.1
> User-Agent: curl/7.31.0
> Host: ecs.yourco.com:9024
> Accept: */*
> X-Auth-Token: 8cf4a4e943f94711aad1c91a08e98435
>
< HTTP/1.1 204 No Content
< Date: Mon, 16 Sep 2013 19:31:45 GMT
< Content-Type: text/plain
* Server Jetty(7.6.4.v20120524) is not blacklisted
< Server: Jetty(7.6.4.v20120524)
<

* Connection #0 to host ecs.yourco.com left intact
```

OpenStack Version 2 authentication

ECS includes limited support for OpenStack Version 2 (Keystone) authentication.

Before you begin

ECS provides an implementation of the OpenStack Swift V2 identity service which enables a Swift application that uses V2 authentication to authenticate users. Users

must be ECS object users who have been assigned OpenStack Swift credentials which enable them to access the ECS object store using the Swift protocol.

Only tokens that are scoped to an ECS namespace (equivalent to a Swift project) can be used to make Swift API calls. An unscoped token can be obtained and used to access the identity service in order to retrieve the tenant identity before obtaining a token scoped to a tenant and a service endpoint.

The scoped token and service endpoint can be used to authenticate with ECS as described in the previous section describing V1 authentication.

The two articles listed below provide important background information.

- [OpenStack Keystone Workflow and Token Scoping](#)
- [Authenticate for Admin API](#)

Procedure

1. To obtain an unscoped token from ECS you can use the `/v2.0/tokens` API and supply a username and password for the ECS Swift service.

```
curl -v -X POST -H 'ACCEPT: application/json' -H "Content-Type: application/json" -d '{"auth": {"passwordCredentials": {"username": "swift_user", "password": "123"}}}' http://203.0.113.10:9024/v2.0/tokens
```

The response looks like the following. The unscoped token is preceded by id and tokens generated by ECS are preceded by the "ecs_" prefix.

```
{"access": {"token": {"id": "ecs_d668b72a011c4edf960324ab2e87438b", "expires": "1376633127950"}, "user": {"name": "sysadmin", "roles": [ ], "role_links": [ ] }, "serviceCatalog": [ ] } }
```

2. Retrieve tenant information associated with the unscoped token.

```
curl -v http://203.0.113.10:9024/v2.0/tenants -H 'X-Auth-Token: d668b72a011c4edf960324ab2e87438b'
```

The response looks like the following.

```
{"tenants_links": [ ], "tenants": [{"description": "s3", "enabled": true, "name": "s3"}]}
```

3. Retrieve the scoped token along with the storageUrl.

```
curl -v -X POST -H 'ACCEPT: application/json' -H "Content-Type: application/json" -d '{"auth": {"tenantName": "s3",
```

```

        "token":{"id" :
ecs_d668b72a011c4edf960324ab2e87438b"}}}' http://
203.0.113.10:9024/v2.0/tokens

```

An example response follows. The scoped token is preceded by id.

```

{"access":{"token":{"id":"ecs_baf0709e30ed4b138c5db6767ba76a4e",
"expires":"1376633255485","tenant":
{"description":"s3","enabled":true,"name":"s3"}},
"user":{"name":"swift_admin","roles":[{"name":"member"},
{"name":"admin"}],"role_links":[]},
"serviceCatalog":[{"type":"object-store",
"name":"Swift","endpoints_links":[],"endpoint":
[{"internalURL":
"http://203.0.113.10:9024/v1/s3","publicURL":"http://
203.0.113.10:9024/v1/s3"}]}]}]}

```

4. Use the scoped token and the service endpoint URL for Swift authentication. This step is the same as in V1 of OpenStack.

```

curl -v -H "X-Auth-Token: baf0709e30ed4b138c5db6767ba76a4e"
http://203.0.113.10:9024/v1/s3/{container}/{object}

```

Authentication using ECS Keystone V3 integration

ECS provides support for Keystone V3 by validating authentication tokens provided by OpenStack Swift users. For Keystone V3, users are created outside of ECS using a Keystone V3 service. ECS does not perform authentication, but validates the authentication token with the Keystone V3 service.

Note

In the Keystone domain, a project can be thought of as a equivalent to an ECS tenant/namespace. An ECS namespace can be thought of as a tenant.

Keystone V3 enables users to be assigned to roles and for the actions that they are authorized to perform to be based on their role membership. However, ECS support for Keystone V3 does not currently support Keystone policies, so users must be in the `admin` group (role) to perform container operations.

Authentication tokens must be scoped to a project; unscoped tokens are not allowed with ECS. Operations related to unscoped tokens, such as obtaining a list of projects (equivalent to a tenant in ECS) and services, must be performed by users against the Keystone service directly, and users must then obtain a scoped token from the Keystone service that can then be validated by ECS and, if valid, used to authenticate with ECS.

To enable ECS validation, an authentication provider must have been configured in ECS so that when a project-scoped token is received from a user, ECS can validate it against the Keystone V3 authentication provider. In addition, an ECS namespace corresponding to the Keystone project must be created. More information is provided in [Configure OpenStack Swift and ECS integration](#) on page 76.

Authorization Checks

ECS uses the information provided by the Keystone tokens to perform authorization decisions. The authorization checks are as follows:

1. ECS checks whether the project that the token is scoped to matches the project in the URI.
2. If the operation is an object operation, ECS evaluates the ACLs associated with the object to determine if the operation is allowed.
3. If the operation is a container operation, ECS evaluates the requested operation. If the user has the `admin` role they can perform the following container operations: list, create, update, read, and delete.

Domains

in Keystone V3 all users belong to a domain and a domain can have multiple projects. Users have access to projects based on their role. If a user is not assigned to a domain, their domain will be *default*.

Objects and containers created using Swift Keystone V3 users will be owned by `<user>@<domain.com>`. If the user was not assigned to a domain, their username assigned to containers and objects will be `<user>@default`.

Configure OpenStack Swift and ECS integration

To ensure that an OpenStack Swift service that uses Keystone V3 can authenticate with ECS, you must ensure that the Swift and ECS configurations are consistent.

Before you begin

The following pre-requisites apply:

- Ensure you have credentials for the Swift service administrator account. These credentials will be required so that ECS can authenticate with the Keystone service.
- Ensure you have the identity of the Keystone project to which Swift users that will access ECS belong.
- Ensure that you have the credentials for an ECS System Administrator account.

Procedure

1. Ensure that the ECS endpoint has been added to the Swift service catalog and is correctly formatted.

You must ensure that the endpoints are located in the "default" Keystone domain.

2. Log into the ECS Portal as a System Administrator.
3. Create an authentication provider that specifies the Keystone V3 service endpoint and the credentials of an administrator account that can be used to validate tokens.

See [Add a Keystone authentication provider](#) on page 77.

4. Create an ECS namespace that has the same ID as the Keystone project/ account that the users that wish to authenticate belong to.

Obtain the Keystone project ID.

a. In the ECS Portal, select **Manage > Namespace > New Namespace**

b. Enter the name of the namespace.

This should be the name of the Swift project.

c. Enter the namespace administrator account as the **User Admin**.

This should be the name of a management user that has previously been created.

- d. Configure any other namespace settings that you require.

For more information on Namespace settings and about creating users in ECS, refer to the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Results

Once the namespace is created, users belonging to the corresponding Keystone project, and who have a token that is scoped to that project, can authenticate with ECS (through ECS communicating with the Keystone authentication provider) and use the Swift API to access the ECS object store.

Add a Keystone authentication provider

You can add a Keystone authentication provider to authenticate OpenStack Swift users.

Before you begin

- This operation requires the System Administrator role in ECS.
- You can add only one Keystone authentication provider.
- Obtain the authentication provider information listed in [Keystone authentication provider settings](#) on page 78.

Procedure

1. In the ECS Portal, select **Manage > Authentication**.
2. On the **Authentication Provider Management** page, click **New Authentication Provider**.
3. On the **New Authentication Provider** page, in the **Type** field, select **Keystone V3**.

The required fields are displayed.

New Authentication Provider ?

Name * ?

Description * ?

Type * ?

Keystone V3 ▼

Server URL *

Ex.: ldap://10.1.1.1

Keystone Administrator *

Admin Password *

Save **Cancel**

4. Type values in the **Name**, **Description**, **Server URL**, **Keystone Administrator**, and **Admin Password** fields. For more information about these fields, see [Keystone authentication provider settings](#) on page 78.
5. Click **Save**.

Keystone authentication provider settings

You must provide authentication provider information when you add or edit a Keystone authentication provider.

Table 13 Keystone authentication provider settings

Field	Description
Name	The name of the Keystone authentication provider. This name is used to identify the provider in ECS.
Description	Free text description of the authentication provider.
Type	Keystone V3.
Server URL	URI of the Keystone system that ECS connects to in order to validate Swift users.

Table 13 Keystone authentication provider settings (continued)

Field	Description
Keystone Administrator	Username for an administrator of the Keystone system. ECS connects to the Keystone system using this username.
Admin Password	Password of the specified Keystone administrator.

Authorization on Container

OpenStack Swift authorization targets only containers.

Swift currently supports two types of authorization:

- Referral style authorization
- Group style authorization.

ECS supports only group-based authorization.

Admin users can perform all operations within the account. Non-admin users can only perform operations for each container based on the container's X-Container-Read and X-Container-Write Access Control Lists. The following operations can be granted to non-admin users:

Admin assigns read access to the container

The "admin" user can assign read permissions to a group using:

```
curl -X PUT -v -H 'X-Container-Read: {GROUP LIST}'
-H 'X-Auth-Token: {TOKEN}'
http://127.0.0.1:8080/v1/{account}/{container1}"
```

This command allows users belonging to the GROUP LIST to have read access rights to container1. For example, to assign read permissions to the group "Member":

```
curl -X PUT -v -H 'X-Container-Read: Member' -H 'X-Auth-Token:
{ADMIN TOKEN}'
http://127.0.0.1:8080/v1/{account}/{container1}
```

After read permission is granted, users who belong to target group(s) can perform the following operations:

- HEAD container - Retrieve container metadata. Only allowed if user is assigned to group that has Tenant Administrator privileges.
- GET container - List objects within a container.
- GET objects with container - Read contents of the object within the container.

Admin assigns write access to the container

The "admin" user can assign read permissions to a group using:

```
curl -XPUT -v -H 'X-Container-Write: {GROUP LIST}'
-H 'X-Auth-Token: {TOKEN}'
http://127.0.0.1:8080/v1/{account}/{container1}"
```

This command allows users belonging to the GROUP LIST to have write access rights to container1. For example, to assign write permissions to the group "Member":

```
curl -X PUT -v -H 'X-Container-Write: Member' -H 'X-Auth-Token:
{ADMIN_TOKEN}'
http://127.0.0.1:8080/v1/{account}/{container1}
```

The users in the group GROUP LIST are granted write permission. Once write permission is granted, users who belong to the target group(s) can perform the following operations:

- POST container - Set metadata. Start with prefix "X-Container-Meta".
- PUT objects within container - Write/override objects within container.

Additional information on authorization can be found in: [Container Operations](#).

ECS Swift error codes

The error codes that can be generated by the OpenStack Swift head are listed in the following table. All errors are of type: `ObjectAccessException`.

Error Code	HTTP Status Code	HTTP Status	Description
ERROR_NAMESPACE_NOT_FOUND	400	BAD_REQUEST	Namespace not found.
ERROR_KEYPOOL_NOT_FOUND	404	NOT_FOUND	Keypool not found.
ERROR_KEYPOOL_NOT_EMPTY	409	CONFLICT	Keypool not empty.
ERROR_OBJECT_NOT_FOUND	404	NOT_FOUND	Object not found.
ERROR_VERSION_NOT_FOUND	404	NOT_FOUND	Version not found.
ERROR_ACCESS_DENIED	403	FORBIDDEN	null
ERROR_SERVICE_BUSY	503	SERVICE_UNAVAILABLE	null
ERROR_PRECONDITION_FAILED	412	PRECONDITION_FAILED	null
ERROR_INVALID_ARGUMENT	400	BAD_REQUEST	Invalid argument.
ERROR_BAD_ETAG	422	SC_UNPROCESSABLE_ENTITY	Bad etag.
ERROR_PROJECT_NOT_FOUND	404	NOT_FOUND	SwiftException. NO_PROJECT_FOUND.
ERROR_NO_DEVICE	404	NOT_FOUND	SwiftException. NO_DATA_STORE_FOUND. //add 416- Requested Range Not Satisfiable to errorMap.
ERROR_INVALID_RANGE	422	SC_REQUESTED_RANGE_NOT_SATISFIABLE	Requested range cannot be satisfied.
ERROR_INSUFFICIENT_STORAGE	507	SC_INSUFFICIENT_STORAGE	The server cannot process the request because there is not enough space on disk.

Error Code	HTTP Status Code	HTTP Status	Description
ERROR_RETENTION_INCORRECT	404	SC_NOT_FOUND	The specified retention does not exist.
ERROR_OBJECT_UNDER_RETENTION	409	SC_CONFLICT	The object is under retention and cannot be deleted or modified.
ERROR_METHOD_NOT_ALLOWED	403	SC_FORBIDDEN	Quota may have been exceeded.
ERROR_BUCKET_NOT_FOUND	404	NOT_FOUND	Bucket not found.
ERROR_KEYPOOL_OPERATION_NOT_SUPPORTED	400	BAD_REQUEST	VersionEnabled and FileSystemEnabled functionality is not supported.
ERROR_REP_GROUP_NOT_FOUND	400	BAD_REQUEST	Specified Replication Group is Invalid.
ERROR_OBJECT_METADATA_REACH_MAXIMUM	400	BAD_REQUEST	Metadata exceeds max allowed length.
ERROR_KEYPOOL_LOCKED	409	CONFLICT	Bucket may be locked.
ERROR_INVALID_PART	409	CONFLICT	Segment eTag differs from that of the manifest.
ERROR_DELETE_DIRECTORY_NOT_EMPTY	409	CONFLICT	Directory is not empty.
ERROR_API_INVALID	400	BAD_REQUEST	Cross head access is not supported.

PART 3

EMC Atmos

Chapter 3, "EMC Atmos"

CHAPTER 3

EMC Atmos

- [EMC Atmos API support in ECS.....](#)86
- [Supported EMC Atmos REST API Calls.....](#) 86
- [Unsupported EMC Atmos REST API Calls.....](#)88
- [Subtenant Support in EMC Atmos REST API Calls.....](#) 88
- [API Extensions.....](#) 89
- [ECS Atmos error codes.....](#)94

EMC Atmos API support in ECS

ECS supports a subset of the EMC Atmos API. This part details the supported operations and the ECS extensions.

The Atmos Object Service is made available on the following ports.

Protocol	Ports
HTTP	9022
HTTPS	9023

More information on the supported operations can be found in the *Atmos Programmer's Guide* which is available from the ECS [Support Site](#).

- [Atmos Programmer's Guide](#)

Wire format compatibility is provided for all supported operations. Therefore, the operations described in the *Atmos Programmer's Guide* apply to the API operations exposed by ECS.

The *Atmos Programmer's Guide* also provides information on authenticating with the Atmos API and provides comprehensive examples for many of the supported features.

Supported EMC Atmos REST API Calls

ECS supports a subset of the EMC Atmos API.

The following Atmos REST API calls are supported. Calls for both the object and namespace interfaces are shown.

Table 14 Supported Atmos REST API calls

Method	Path	Description
Service Operations		
GET	/rest/service	Get information about the system
Object Operations		
POST	/rest/objects /rest/namespace/<path>	Create an object (See notes below)
DELETE	/rest/objects/<ObjectID> /rest/namespace/<path>	Delete object
PUT	/rest/objects/<ObjectID> /rest/namespace/<path>	Update object (See notes below)
GET	/rest/objects/<ObjectID> /rest/namespace/<path>	Read object (or directory list)
POST	/rest/namespace/<path>?rename	Rename an object
MetaData Operations		
GET	/rest/objects/<ObjectID>?metadata/user	Get user metadata for an object

Table 14 Supported Atmos REST API calls (continued)

Method	Path	Description
	/rest/namespace/<path>?metadata/user	
POST	/rest/objects/<ObjectID>?metadata/user /rest/namespace/<path>?metadata/user	Set user metadata
DELETE	/rest/objects/<objectID>?metadata/user /rest/namespace/<path>?metadata/user	Delete user metadata
GET	/rest/objects/<ObjectID>?metadata/system /rest/namespace/<path>?metadata/system	Get system metadata for an object
GET	/rest/objects/<ObjectID>?acl /rest/namespace/<path>?acl	Get ACL
POST	/rest/objects/<ObjectID>?acl /rest/namespace/<path>?acl	Set ACL
GET	/rest/objects/<ObjectID>?metadata/tags /rest/namespace/<path>?metadata/tags	Get metadata tags for an object
GET	/rest/objects/<ObjectID>?info /rest/namespace/<path>?info	Get object info
Head	/rest/objects/<ObjectID> /rest/namespace/<path>	Get all object metadata
Object-space Operations		
GET	/rest/objects	List objects
GET	/rest/objects?listabletags	Get listable tags
Anonymous Access		
GET	/rest/objects/<ObjectID>?uid=<uid>&expires=<exp>&signature=<sig> /rest/namespace/<path>?uid=<uid>&expires=<exp>&signature=<sig>	Shareable URL

Note

- The x-emc-wschecksum header is supported in ECS.
- HTML form upload is not supported.
- GET /rest/objects does not support different response types with x-emc-accept. For example, text/plain is not supported.
- Read, Write, and Delete ACLs work in ECS the same as Atmos.
- POST /rest/objects supports the x-emc-object-id header to enable legacy (44 character) object IDs.

Atmos listable tags

Listable tags are special user-defined tags used to list or filter objects. For example, an application could allow an end user to tag a group of pictures (objects) with a tag like "Vacation2016". Later the application can respond to a query of "Vacation2016" by listing only the objects tagged with this listable tag.

Using the Atmos protocol with ECS, a user cannot delete or modify another user's listable tags. Under some conditions, this ability is allowed in native Atmos.

Listable tags are indexed in ECS, increasing the performance and scalability of the retrieval of tagged objects.

In ECS, the `EMC_TAGS` metadata tag is used to persist listable tags. This tag name should not be used in user-defined metadata tags.

Object ID length

Support for the Atmos API in ECS expands the length of the object Id from 44 characters to 101 characters. Hence, when moving applications from Atmos to ECS you will need to be aware that the object Id length will be different.

To create an object with the legacy 44 character Id length, you can use the `x-emc-object-id` header. This enables objects to be migrated to Atmos.

Unsupported EMC Atmos REST API Calls

The following Atmos REST API calls are not supported.

Table 15 Unsupported Atmos REST API calls

Method	Path	Description
Object Versioning		
POST	<code>/rest/objects/<objectID>?versions</code>	Create a version of an object
DELETE	<code>/rest/objects/<objectID>?versions</code>	Delete an object version
GET	<code>/rest/objects/<objectID>?versions</code>	List versions of an object
PUT	<code>/rest/objects/<objectID>?versions</code>	Restore object version
Anonymous Access		
POST	<code>/rest/accesstokens</code>	Create an access token
GET	<code>/rest/accesstokens/<token_id>?info</code>	Get access token detail
DELETE	<code>/rest/accesstokens/<token_id></code>	Delete access token
GET	<code>/rest/accesstokens</code>	List access tokens
GET	<code>/rest/accesstokens/<token_id></code>	Download content anonymously

Subtenant Support in EMC Atmos REST API Calls

ECS includes two native REST API calls that are specifically to add ECS subtenant support to Atmos applications.

These calls are as follows:

API Call	Example
Subtenant create	PUT Http url: <code>/rest/subtenant</code> Required headers: <code>x-emc-uid</code> (for example, <code>x-emc-uid=wuser1@example.com</code>) and <code>x-emc-signature</code> .

API Call	Example
	The subtenantID is set in the header "subtenantID" of the response.
Subtenant delete	DELETE Http url: /rest/subtenants/{subtenantID} Required headers: x-emc-uid (for example, x-emc-uid=wuser1@example.com) and x-emc-signature.

Note

Subtenant IDs are preserved in ECS after migration: The header is `x-emc-subtenant-id: {original_subt_id}`.

API Extensions

ECS supports a number of extensions to the Atmos API.

The extensions and the APIs that support them are listed below:

- [Appending data to an object](#) on page 89
- [ECS support for retention and retention expiration periods for Atmos objects](#) on page 90

Appending data to an object

You can use ECS extensions to the EMC Atmos protocol to append data to an object.

There may be cases where you need to append to an object, but determining the exact byte offset is not efficient or useful. For this scenario, ECS provides the ability to atomically append data to the object without specifying an offset (the correct offset is returned to you in the response).

A Range header with the special value `bytes=-1-` is used to append data to an object. In this way, the object can be extended without knowing the existing object size. The format is: `Range: bytes=-1-`

A sample request showing appending to an existing object using a Range value of `bytes=-1-` is shown in the following example. Here the value `and cat` is sent in the request.

```
PUT /rest/namespace/myObject HTTP/1.1
Content-Length: 8
Range: bytes=-1-
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 17 Jun 2013 20:46:01 -0000
x-emc-date: Mon, 17 Jun 2013 20:46:01 -0000
x-emc-namespace: emc
x-emc-uid: fa4e31a68d3e4929bec2f964d4cac3de/wuser1@sanity.local
x-emc-signature: ZpW9KjRb5+YFdSzZjwufZUqkExc=
Content-Type: application/octet-stream
Accept-Encoding: gzip, deflate, compress

and cat

HTTP/1.1 200 OK
x-emc-mtime: 1431626712933
Date: Mon, 17 Jun 2013 20:46:01 GMT
x-emc-policy: default
```

```
x-emc-utf8: true
x-emc-request-id: 0af9ed8d:14cc314a9bc:112f6:9
x-emc-delta: 8
x-emc-append-offset: 24
Content-Length: 0
Server: Jetty(7.6.4.v20120524)
```

The offset position at which the data was appended is returned in the x-emc-append-offset header.

When the object is retrieved, and cat has been appended, and you can see the full value: The quick green fox jumps over the lazy dog and cat.

ECS support for retention and retention expiration periods for Atmos objects

ECS supports setting retention periods, and retention expiration periods on Atmos objects.

Retention periods

Retention periods define how long an object will be retained by ECS before it can be edited or deleted. During the retention period the object cannot be edited or deleted from the system until the retention period has expired.

While creating an Atmos object in ECS, the object retention can be:

- Defined directly on the object
- Inherited from the retention period set on the ECS bucket in which the object is created.

When a retention policy is set on the ECS namespace, you must also set the retention period directly on the object. The retention policy in the namespace is not inherited by the object.

Table 16 Atmos retention periods

Retention set on the	Using the	Notes
Object	<p>Atmos API through the</p> <ul style="list-style-type: none"> • Header retention period in seconds: 'x-emc-retention-period: 60' • User meta data (UMD), end date: 'x-emc-meta:user.maui.retentionEnabled=true,user.maui.retentionEnd=2016-10-21:10:00Z' • Both header, and UMD: 'x-emc-meta:user.maui.retentionEnabled=true,user.maui.retentionEnd=2016-10-21T18:14:30Z' -header 'x-emc-retention-period:60' 	<ul style="list-style-type: none"> • Retention can be set on the object while creating, or updating the object settings. • Header retention period is defined in seconds. • UMD retention is defined by an end date. • If retention period is set from both the header and the UMD, the UMD attribute is checked first and takes precedence over the setting in the header. • You cannot modify the retention period after it has been set on the object until the period has expired. • When using the x-emc header to set retention <ul style="list-style-type: none"> ▪ -1 will set an infinite retention period, as well as disable the expiration period if one is defined. ▪ -2 will disable the retention period set on the object.

Table 16 Atmos retention periods (continued)

Retention set on the	Using the	Notes
ECS namespace	ECS Portal from the New Namespace or Edit Namespace page.	<ul style="list-style-type: none"> If you want to set a retention period for an object, and a retention policy has been defined on the object user's namespace, you must still define a retention period directly on the object as described above. If a retention policy is set on the ECS namespace, and/or a retention period is set on a bucket within the namespace, and an object is created within the bucket, ECS retains the namespace, bucket, and object for the longest retention periods set for either the namespace, or bucket.
	ECS REST API POST /object/namespaces/ namespace/{namespace}/ retention	
ECS bucket	ECS Portal from the New Bucket , or Edit Bucket page.	<ul style="list-style-type: none"> If a retention period has been set on the object itself through the object header, ECS retains the object for the longest period of time set on the namespace, bucket, or object. If a retention end date was defined on an object through the Atmos API, ECS will use the Atmos API retention end date set on the object, and ignore the namespace retention policy, and bucket retention periods when creating the object. When applying a retention policy on a subtenant (bucket) containing Atmos objects, the retention policy will be applied to both objects created in the subtenant after the retention policy was set, and objects that were created in the subtenant before the retention policy was set.
	ECS REST API PUT /object/bucket/ {bucketName}/retention	

Note

For further details about Namespace Retention Policies and Bucket Retention Periods refer to the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

Example: Request and response to create an object with retention set:

```
POST /rest/namespace/file1 HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Thu, 16 Feb 2017 19:28:13 GMT
x-emc-meta:user.maui.retentionEnable=true,user.maui.retentionEnd=2017-06-30T06%3A38%3A44Z
x-emc-uid:f082110e13f249649340e172fb7b4956/u1
x-emc-utf8:true
Content-Type:plain/text
x-emc-signature:2Gz51WT+jQdMjlobDV0mz7obsXM=
Content-Length: 774
```

Response

```
HTTP/1.1 201 Created
Date: Thu, 16 Feb 2017 19:28:17 GMT
x-emc-policy: default
x-emc-utf8: true
x-emc-request-id: 0af7b3e4:15a4849d95e:37c:0
```

```
x-emc-delta: 774
Location: /rest/objects/
0a40bd045f7373d367639f095d1db0d15acadb82d5d2cd108e2142f4be04635c-59bdb9b6-20c0-4f55-
bc91-9db728a58854
x-emc-mtime: 1487273295379
Content-Length: 0
Server: ViPR/1.0
```

Example: Request and response to get object meta-data:

```
curl --head -H "x-emc-date:Mon, 30 Jan 2017 16:56:35 GMT"
-H "x-emc-uid:7a2593be81374744adbf8e3983e7bd84/u1"
-H "x-emc-signature:CQgfoiIQ/DCif7TafcIskWyVpME="
http://10.247.179.228:9022/rest/objects/
dlbced53f2ebbc51af1d84747bd198d123d3b8585293a5bf0d32bb73c6cf4b-365f4482-c24a-4eca-
b24a-070efe29bf63
```

Response

```
HTTP/1.1 200 OK
Date: Mon, 30 Jan 2017 16:56:35 GMT
x-emc-mtime: 1485795387838
x-emc-retention-period: 21798212
x-emc-meta: user.maui.retentionEnd=2017-10-10T00:00:00Z,user.maui.retentionEnable=true, allow-
inline-update=false, atime=2017-01-30T16:45:48Z, ctime=2017-01-30T16:56:27Z, ctype=plain/
text, data-range=CAAQgFA=, dek=kq/WlRg/
7qbmaCcLF8pFvqlDJ8+suPTdVddBBZFwZA86muG3P0Pb7w==, dekAlgo=AESKeyWrapRFC5649, etag=0-, fs-
mtime-
millisec=1485795387838, itime=2017-01-30T16:45:48Z, kekId=s3.7a2593be81374744adbf8e3983e7bd84
3cdda755061bac6c12c06eb02800a7fee4b11ac2e03f62bb01eee02995068e56, keypoolid=s3.7a2593be81374
744adbf8e3983e7bd84, keypoolname=7a2593be81374744adbf8e3983e7bd84, keyversion=0, mtime=2017-01
-30T16:56:27Z, namespace=s3, nlink=1, object-
name=, objectid=dlbced53f2ebbc51af1d84747bd198d123d3b8585293a5bf0d32bb73c6cf4b-365f4482-
c24a-4eca-
b24a-070efe29bf63, objname=file, parentOid=53ae036bfcfb46f5580b912222f3026835e3ef972c7e3e532b
a4a5de30b1946e, parentZone=urn:storageos:VirtualDataCenterData:365f4482-c24a-4eca-
b24a-070efe29bf63, policynname=default, retention=CgYIoKOZmlE=, size=0, type=regular, uid=u1, pare
nt=apache, gid=apache
x-emc-useracl: u1=FULL_CONTROL
x-emc-groupacl: other=READ
x-emc-policy: default
x-emc-request-id: 0af7b3e4:159f0185cf7:957:4
Content-Type: plain/text
Content-Length: 0
Server: ViPR/1.0
```

Example: Update an object with retention values

```
POST /rest/namespace/file2?metadata/user HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Thu, 16 Feb 2017 19:37:15 GMT
x-emc-meta:user.maui.retentionEnable=true,user.maui.retentionEnd=2017-07-30T06%3A38%3A44Z
x-emc-uid:f082110e13f249649340e172fb7b4956/u1
x-emc-utf8:true
Content-Type:plain/text
x-emc-signature:5UPpZcCfO0vtxMTW62fa2/2SmLg=
```

Response

```
HTTP/1.1 200 OK
```

```
Date: Thu, 16 Feb 2017 19:37:16 GMT
x-emc-policy: _int
x-emc-utf8: true
x-emc-request-id: 0af7b3e4:15a4849d95e:582:0
Content-Length: 0
Server: ViPR/1.0
```

Expiration period

When a retention period end date is defined for an Atmos object, and an the expiration period is also set on the object, ECS automatically deletes the object at the date defined in the expiration period. The expiration period:

- Can be set on objects using the Atmos API, or the `x-emc` header.
- The expiration period must be later than the retention end date.
- The expiration period is disabled by default.
- When using the `x-emc` header to set retention and expiration, a -1 value will disable the expiration period.

Example: Set the expiration period using the `x-emc` header:

```
POST /rest/namespace/file2 HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Tue, 31 Jan 2017 19:38:00 GMT
x-emc-expiration-period:300
x-emc-uid:a2b85977fd08488b80e646ea875e990b/u1
Content-Type:plain/text
x-emc-signature:krhYBfKSIm3mFOT6FtRB+2/xZnw=
Content-Length: 10240
Expect: 100-continue
```

Example: Request and response using the Atmos API:

```
POST /rest/namespace/file2 HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Thu, 02 Feb 2017 02:47:32 GMT
x-emc-meta:user.maui.expirationEnable=true,user.maui.expirationEnd=2017-03-30T20:20:00Z
x-emc-uid:239e20dec7a54301a0b02f6090edcace/u1
Content-Type:plain/text
x-emc-signature:5tGEyK/9qUZCPSnQ9OPodktN+Zo=
Content-Length: 10240
Expect: 100-continue
```

Response

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Date: Thu, 02 Feb 2017 02:47:33 GMT
x-emc-policy: default
x-emc-request-id: 0af7b3e4:159fb81ddae:345e:0
x-emc-delta: 10240
Location: /rest/objects/5c3abaf60e0e207abec96baf0618c0461b7cd716898f8a12ee236aed1ec94bea-c86ee0e9-8709-4897-898e-c3d1895eld93
x-emc-mtime: 1486003652813
Content-Length: 0
Server ViPR/1.0 is not blacklisted
Server: ViPR/1.0
```

Example: Request and response for update meta data with Atmos API:

```

POST /rest/namespace/file?metadata/user HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Thu, 02 Feb 2017 02:44:13 GMT
x-emc-meta:user.maui.expirationEnable=true,user.maui.expirationEnd=2017-03-30T20:20:00Z
x-emc-uid:239e20dec7a54301a0b02f6090edcace/u1
Content-Type:plain/text
x-emc-signature:9pzcc/Ce4Lq3k52QKdfWLYlZ1Yc=

```

Response

```

HTTP/1.1 200 OK
Date: Thu, 02 Feb 2017 02:44:14 GMT
x-emc-policy: _int
x-emc-request-id: 0af7b3e4:159fb81ddae:339e:0
Content-Length: 0
Server ViPR/1.0 is not blacklisted
Server: ViPR/1.0

```

ECS Atmos error codes

The error codes that can be generated by the EMC Atmos head are listed in the following table.

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1001	The server encountered an internal error. Please try again.	500	Internal Server Error
1002	One or more arguments in the request were invalid.	400	Bad Request
1003	The requested object was not found.	404	Not Found
1004	The specified range cannot be satisfied.	416	Requested Range Not Satisfiable
1005	One or more metadata tags were not found for the requested object.	400	Bad Request
1006	Operation aborted because of a conflicting operation in process against the resource. Note: This error code may indicate that the system is temporarily too busy to process the request. This is a non-fatal error; you can re-try the request later.	409	Conflict
1007	The server encountered an internal error. Please try again.	500	Internal Server Error
1008	The requested resource was not found on the server.	400	Bad Request
1009	The method specified in the Request is not allowed for the resource identified.	405	Method Not Allowed
1010	The requested object size exceeds the maximum allowed upload/download size.	400	Bad Request
1011	The specified object length does not match the actual length of the attached object.	400	Bad Request

1012	There was a mismatch between the attached object size and the specified extent size.	400	Bad Request
1013	The server encountered an internal error. Please try again.	500	Internal Server Error
1014	The maximum allowed metadata entries per object has been exceeded.	400	Bad Request
1015	The request could not be finished due to insufficient access privileges.	401	Unauthorized
1016	The resource you are trying to create already exists.	400	Bad Request
1019	The server encountered an I/O error. Please try again.	500	Internal Server Error
1020	The requested resource is missing or could not be found.	500	Internal Server Error
1021	The requested resource is not a directory.	400	Bad Request
1022	The requested resource is a directory.	400	Bad Request
1023	The directory you are attempting to delete is not empty.	400	Bad Request
1024	The server encountered an internal error. Please try again.	500	Internal Server Error
1025	The server encountered an internal error. Please try again.	500	Internal Server Error
1026	The server encountered an internal error. Please try again.	500	Internal Server Error
1027	The server encountered an internal error. Please try again.	500	Internal Server Error
1028	The server encountered an internal error. Please try again.	500	Internal Server Error
1029	The server encountered an internal error. Please try again.	500	Internal Server Error
1031	The request timestamp was outside the valid time window.	403	Forbidden
1032	There was a mismatch between the signature in the request and the signature as computed by the server.	403	Forbidden
1033	Unable to retrieve the secret key for the specified user.	403	Forbidden
1034	Unable to read the contents of the HTTP body.	400	Bad Request
1037	The specified token is invalid.	400	Bad Request
1040	The server is busy. Please try again	500	Internal Server Error
1041	The requested filename length exceeds the maximum length allowed.	400	Bad Request
1042	The requested operation is not supported.	400	Bad Request
1043	The object has the maximum number of links	400	Bad Request

1044	The specified parent does not exist.	400	Bad Request
1045	The specified parent is not a directory.	400	Bad Request
1046	The specified object is not in the namespace.	400	Bad Request
1047	Source and target are the same file.	400	Bad Request
1048	The target directory is not empty and may not be overwritten	400	Bad Request
1049	The checksum sent with the request did not match the checksum as computed by the server	400	Bad Request
1050	The requested checksum algorithm is different than the one previously used for this object.	400	Bad Request
1051	Checksum verification may only be used with append update requests	400	Bad Request
1052	The specified checksum algorithm is not implemented.	400	Bad Request
1053	Checksum cannot be computed for an object on update for which one wasn't computed at create time.	400	Bad Request
1054	The checksum input parameter was missing from the request.	400	Bad Request
1056	The requested operation is not supported for symlinks.	400	Bad Request
1057	If-Match precondition failed.	412	Precondition failed
1058	If-None-Match precondition failed.	412	Precondition failed
1059	The key you are trying to create already exists.	400	Bad Request
1060	The requested key was not found.	404	Not found
1061	The requested pool already exists.	400	Bad Request
1062	The requested pool was not found.	404	Not found
1063	The maximum number of pools has been reached.	400	Bad request
1064	The request could not be completed because the subtenant is over quota	403	Forbidden
1065	The request could not be completed because the UID is over quota	403	Forbidden
1070	Did not receive the expected amount of data.	400	Bad request
1071	Client closed connection before reading all data.	499	Client Closed Request
1072	Could not write all bytes to the client.	499	Client Closed Request
1073	Timeout writing data to the client.	499	Client Closed Request

PART 4

CAS

Chapter 4, "CAS"

CHAPTER 4

CAS

• Setting up CAS support in ECS	100
• Cold Storage	100
• Compliance	101
• CAS retention in ECS	104
• Advanced retention for CAS applications: event-based retention, litigation hold, and the min/max governor	105
• Set up namespace retention policies	110
• Create and set up a bucket for a CAS user	111
• Set up a CAS object user	112
• Set up bucket ACLs for CAS	113
• ECS Management APIs that support CAS users	115
• Content Addressable Storage (CAS) SDK API support	116
• ECS CAS error codes	117

Setting up CAS support in ECS

Introduces CAS (content addressable storage) support in ECS.

ECS CAS allows CAS SDK-based client applications to store, retrieve, and delete fixed content objects from ECS storage.

The underlying ECS storage must be provisioned before you can configure your ECS set up. Provisioning is usually completed when a new ECS rack is installed. This includes setting up a storage pool, VDC, and replication group. In ECS CAS, you can use the standard documentation if you need to create or edit these objects to support CAS. See *ECS Administration Guide* available from the [ECS Product Documentation page](#).

For your storage pools, you might consider setting up a cold archive. See [Cold Storage](#) on page 100.

Next, set up your namespaces, users, and buckets using the standard documentation. See *ECS Administration Guide* available from the [ECS Product Documentation page](#).

This chapter describes how to modify your basic configuration to support CAS.

Cold Storage

Describes cold storage archives.

Cold archives store objects that do not change frequently and do not require the robust default EC scheme. The EC scheme used for a cold archive is 10 data fragments plus 2 coding fragments (10/12). The efficiency is 1.2x.

You can specify a cold archive (Cold Storage) when creating a new storage pool. After the storage pool is created, the EC scheme cannot be changed. This scheme can support the loss of a single node. It also supports loss of one drive out of six or two drives out of 12 on two separate nodes.

EC requirements

Table 17 Requirements for regular and cold archives compared

Use case	How enabled	Minimum required nodes	Minimum required disks	Recommended disks	EC efficiency	EC scheme
Regular archive	Default	4	16*	32	1.33x	12/16
Cold archive	Configured by System Administrator	8	12*	24	1.2x	10/12

Note

*Since the minimum deployable configuration for the C-Series appliance is two appliances with 12 disks each, 24 disks is the effective minimum.

Storage pool configuration

To establish a cold archive from the portal, Select **Cold Storage** when you create a new storage pool. Once a storage pool has been created, this setting cannot be changed.

New Storage Pool ⓘ

Name * ⓘ

Cold Storage ⓘ

☐ Off ☐ On

Available Nodes

search →

IP	Host

Selected Nodes * A minimum of 4 nodes is required

search →

IP	Host

Available Capacity Alerting

Critical ⓘ

No Alert ▼

Error ⓘ

No Alert ▼

Warning ⓘ

No Alert ▼

Compliance

Describes ECS features that support government and industry standards for the storage of electronic records.

ECS meets the storage requirements of the following standards, as certified by Cohasset Associates Inc:

- Securities and Exchange Commission (SEC) in regulation 17 C.F.R. § 240.17a-4(f)
- Commodity Futures Trading Commission (CFTC) in regulation 17 C.F.R. § 1.31(b)-(c)

Compliance has three components:

- Platform hardening: addressing common security vulnerabilities.
- Policy-based record retention: limiting the ability to change retention policies for records under retention.
- Compliance reporting: periodic reporting by a system agent records the system's compliance status.

Platform hardening and Compliance

The following ECS security features support Compliance standards.

ECS platform security features:

- User root access to nodes is disabled (no user root logins permitted).
- ECS customers can access nodes through the admin user set up during first-time installations.
- The admin user runs commands on nodes using `sudo`.
- There is full audit logging for `sudo` commands.
- ESRS provides the ability to shut down all remote access to nodes. In **ESRS Policy Manager**, set the **Start Remote Terminal** action to **Never Allow**.
- All unnecessary ports (ftpd, sshd) are closed.
- The `emcsecurity` user with the Lock Administrator role can lock nodes in a cluster. This means that remote access over the network by SSH is disabled. The Lock Administrator can then unlock a node to allow for remote maintenance activities or other authorized access.

Note

Node locking does not affect authorized ECS Portal or ECS Management API users.

Compliance and retention policy

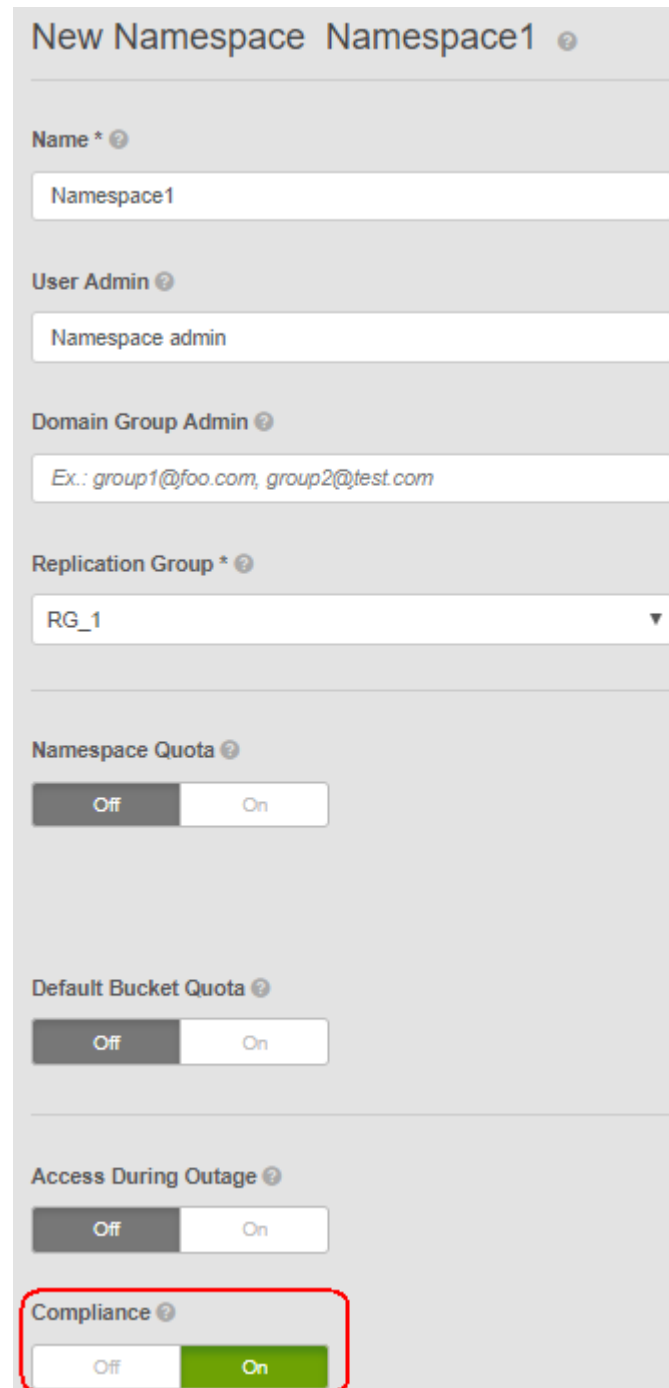
Describes enhanced rules for record retention on a Compliance-enabled ECS system. ECS sets object retention features to **On** at the object, bucket, and namespace levels. Compliance strengthens these features by limiting changes that can be made to retention settings on objects under retention. Rules include:

- Compliance is set at the namespace level. This means that all buckets in the namespace must have a retention period greater than zero. For CAS, buckets with zero retention can be created, provided that the **Enforce Retention Information in Object** setting is turned **On**.
- You can only turn Compliance on when you create a namespace. (You cannot add Compliance to an existing namespace.)
- You cannot turn Compliance off once it is turned on.
- All buckets in a namespace must have a retention period greater than zero.

Note

If you have an application that assigns object-level retention periods, do not use ECS to assign a retention period greater than the application retention period. This action causes application errors.

- A bucket with data in it cannot be deleted regardless of its retention value.
- Applying the **Infinite** option to a bucket means that objects in the bucket in a Compliance-enabled namespace cannot be deleted permanently.
- The retention period for an object cannot be deleted or shortened. Therefore, the retention period for a bucket cannot be deleted or shortened.
- You can increase object and bucket retention periods.
- No user can delete an object under retention. This includes users with the CAS privileged-delete permission.

Figure 1 Set Compliance on a new namespace in the ECS Portal

The screenshot shows the 'New Namespace' configuration page in the ECS Portal. The namespace is named 'Namespace1'. The 'User Admin' is 'Namespace admin' and the 'Domain Group Admin' is 'Ex.: group1@foo.com, group2@test.com'. The 'Replication Group' is set to 'RG_1'. The 'Namespace Quota', 'Default Bucket Quota', and 'Access During Outage' are all set to 'Off'. The 'Compliance' toggle is set to 'On' and is highlighted with a red rectangle.

New Namespace Namespace1 ?

Name * ?
Namespace1

User Admin ?
Namespace admin

Domain Group Admin ?
Ex.: group1@foo.com, group2@test.com

Replication Group * ?
RG_1

Namespace Quota ?
Off On

Default Bucket Quota ?
Off On

Access During Outage ?
Off On

Compliance ?
Off On

Compliance agent

Describes the operation of the Compliance agent.

Compliance features are turned on by default, except for Compliance monitoring. If monitoring is turned on, the agent periodically logs a message.

Note

Contact your customer support representative to turn on Compliance monitoring. Monitoring messages are available by command from the node. They do not appear in the ECS Portal.

CAS retention in ECS

A CAS C-Clip can have a retention period that governs the length of time the associated object is retained in ECS storage before an application can delete it.

Retention periods

Retention periods are assigned in the C-Clip for the object by the CAS application.

For example, if a financial document must be retained for three years from its creation date, then a three-year retention period is specified in the C-Clip associated with the financial document. It is also possible to specify that the document is retained indefinitely.

Retention policies (retention classes)

Note

The Centera concept of *retention classes* maps to *retention policies* in ECS. This documentation uses *retention policies*.

Retention policies enable retention use cases to be captured and applied to C-Clips. For example, different types of documents could have different retention periods. You could require the following retention periods:

- Financial: 3 years
- Legal: 5 years
- Email: 6 months

When a retention policy is applied to a number of C-Clips, by changing the policy, the retention period changes for all objects to which the policy applies.

Retention policies are associated with namespaces in ECS and are recognized by the CAS application as retention classes.

ECS bucket-level retention and CAS

Bucket-level retention is not the default pool retention in Centera. In ECS, CAS default retention is constantly zero.

Default retention period in objects written without object-level retention in Compliance namespaces

Starting with ECS 3.0, when an application writes C-Clips with no object retention to an ECS CAS bucket in a Compliance namespace, and the bucket has a retention value (6 months, for example), the default retention period of infinite (-1) will be assigned to the C-Clips. The C-Clips can never be deleted because their effective retention period is the longest one between the two: the bucket-level retention period and the default object-level retention.

CAS precedence

When multiple retention periods are applied to a CAS object in ECS, the retention period with the higher value has precedence no matter how the retention was applied.

How to apply CAS retention

You can define retention policies for namespaces in the ECS Portal or with the ECS Management API. See [Set up namespace retention policies](#) on page 110.

Your external CAS application can assign a fixed retention period or a retention policy to the C-Clip during its creation.

When applying retention periods through APIs, specify the period in seconds.

Note that ECS CAS takes the creation time of the C-Clip for all retention related calculations and not the migration time.

How to create retention policies with the ECS Management API.

You can create retention periods and policies using the ECS Management REST API, a summary of which is provided below.

Table 18 ECS Management API resources for retention

Method	Description
PUT /object/bucket/{bucketName}/retention	The retention value for a bucket defines a mandatory retention period which is applied to every object within a bucket. If you set a retention period of 1 year, an object from the bucket cannot be deleted for one year.
GET /object/bucket/{bucketName}/retention	Returns the retention period that is currently set for a specified bucket.
POST /object/namespaces/namespace/{namespace}/retention	For namespaces, the retention setting acts like a policy, where each policy is a <Name>:<Retention period> pair. You can define a number of retention policies for a namespace and you can assign a policy, by name, to an object within the namespace. This allows you to change the retention period of a set of objects that have the same policy assigned by changing the corresponding policy.
PUT /object/namespaces/namespace/{namespace}/retention/{class}	Updates the period for a retention period that is associated with a namespace.
GET /object/namespaces/namespace/{namespace}/retention	Returns the retention policy defined for a namespace.

You can find more information about the ECS Management API in [ECS Management REST API introduction](#) on page 126. The online reference is here: [ECS API Reference](#).

Advanced retention for CAS applications: event-based retention, litigation hold, and the min/max governor

Describes advanced retention features available in the CAS API that are supported by ECS.

Customer applications use the CAS API to enable retention strategies. When CAS workloads are migrated to ECS, ECS awareness of CAS API features allow the customer applications to continue working with the migrated data. In ECS, the following advanced retention management (ARM) features are available without a separate license:

- Event-based retention: the ability to configure an object through its C-Clip to apply (trigger) a retention period or retention policy when the CAS application receives a specified event.
- Litigation hold: the ability to prevent deletion of an object if the CAS application has applied a litigation hold to the object through its C-Clip. The CAS application can apply up to 100 litigation holds to an object by creating and applying unique litigation hold IDs.
- Min/Max governor: The ability for an administrator to set bucket-level limits for fixed retention period or variable retention period. A variable retention period is one that is set to support event-based retention. In ECS, System or Namespace Admins can set the values with the ECS Portal. Programmers can use the ECS Management API to set the values.

Note

ARM is supported for legacy CAS data written with any naming scheme that is migrated to ECS.

Min/max governor for CAS bucket-level retention

From the ECS Portal, locate a CAS bucket and select **Edit**. All the features shown on the screen below are CAS-only features except for the **Bucket Retention Period** feature. **Bucket Retention Period** is the standard ECS bucket retention feature supported on all ECS bucket types.

Figure 2 Retention options for CAS buckets

The CAS bucket retention features are explained in the following table.

Feature	Description
Enforce Retention	<p>If this feature is turned on, no CAS object can be created without retention information (period or policy). An attempt to save such an object will return an error. If it is turned on, it is possible not to configure Bucket Retention Period even in a compliance-enabled environment.</p> <p>Note</p> <p>When a CE+ mode Centera is migrated to ECS, Enforce Retention is turned on by default on the bucket.</p>
Bucket Retention Period	<p>If a bucket retention period is specified, then the longer period will be enforced if there is both a bucket-level and an object-level retention period.</p> <p>In a Compliance-enabled environment Bucket Retention Period is mandatory unless retention information in the object is enforced. However, once configured the Bucket Retention Period cannot be reset even when retention information in the object is enforced.</p>
Minimum Fixed Retention Period	<p>This feature governs the retention periods specified in objects. If an object's retention period is outside of the bounds specified here, then an attempt to write the object fails.</p>

Feature	Description
Maximum Fixed Retention Period	<p>When using retention policies, the min/max settings are not enforced.</p> <p>Selecting Infinite for Minimum Fixed Retention Period means all retention values must be infinite. Selecting if for Maximum Fixed Retention Period means there is no maximum limit.</p> <p>Min/max retention constrains are applied to any C-Clip written to a bucket. If a clip is migrated by any SDK-based third-party tool the retention should be within bounds, otherwise an error is thrown.</p>
Minimum Variable Retention Period	<p>This feature governs variable retention periods specified in objects using event-based retention (EBR). In EBR, a base retention period is set and the programmed trigger function has the ability to increase the retention period when the trigger fires. If an object's new retention period is outside of the bounds specified here, then an attempt to write the object in response to the trigger fails.</p> <p>When using retention policies, the min/max settings are not enforced.</p> <p>Selecting Infinite for Minimum Variable Retention Period means all retention values must be infinite. Selecting if for Maximum Variable Retention Period means there is no maximum limit.</p> <p>Min/max retention constrains are applied to any C-Clip written to a bucket. If a clip is migrated by any SDK-based third-party tool the retention should be within bounds, otherwise an error is thrown.</p>
Maximum Variable Retention Period	

Note

If the System Administrator or programmer has not set any values for the fixed and variable retention periods, the ECS Management API `get` function will not return values for the min/max settings. The **Enforce Retention Information in C-Clip** will return a default value of `false`.

Event-based retention

Event-based retention (EBR) is an instruction specifying that a record cannot be deleted before an event and during a specified period after the event. In CAS, EBR is a C-Clip with a specified base retention period or retention policy and an application-defined trigger that can set a longer retention period when the trigger fires. The retention period only begins when the trigger fires. When a C-Clip is marked for EBR, it cannot be deleted prior to the event unless a privileged delete is used.

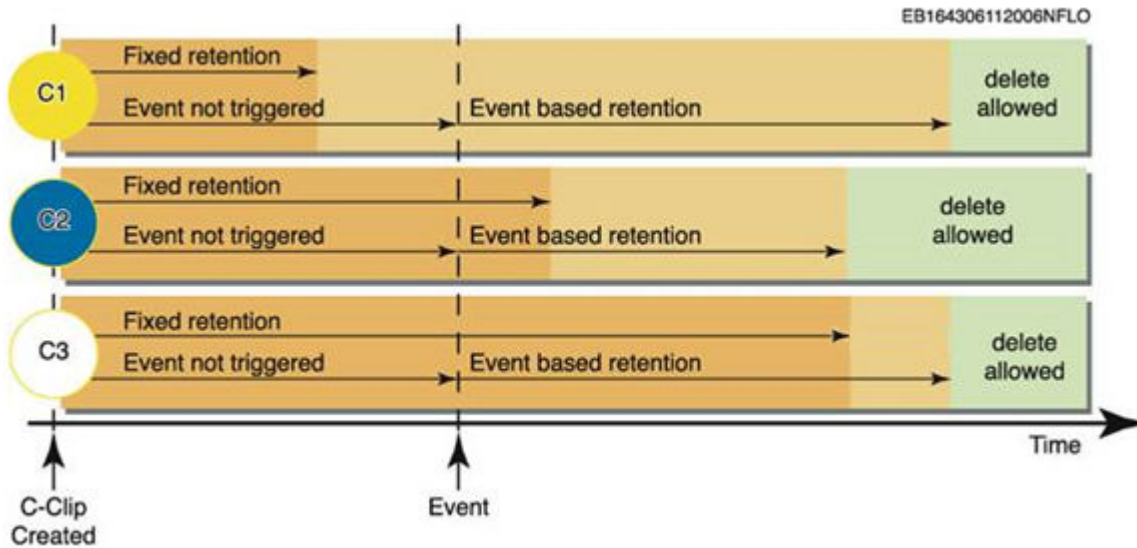
When using EBR, the C-Clip life-cycle is as follows:

- **Create:** the application creates a new C-Clip and marks it as being under EBR. The application can provide a fixed retention period which acts as a minimum retention and it must provide an event based retention period or policy.
- **Trigger Event:** The application triggers the event, which is the starting point of the event-based retention period or retention policy. At this point the application can assign a new event-based retention period, provided that it is longer than the one assigned at the time of the C-Clip creation.
- **Delete:** When the application tries to delete the C-Clip, the following conditions must be met:
 - Policy (Namespace) retention has expired
 - Bucket retention has expired
 - Fixed retention has expired
 - The event has been triggered
 - Both the EBR set at the time of creation and any subsequent changes (extensions) at the time of the event have expired

The following figure shows the three possible scenarios for a C-Clip under EBR:

- C1 has a fixed or minimal retention which already expired before the event was triggered.
- C2 has a fixed or minimal retention which will expire before the EBR expires.
- C3 has a fixed or minimal retention which will expire after the EBR expires.

Figure 3 EBR scenarios



For non-compliant namespaces, privileged delete commands can override fixed and variable retention for EBR.

When applying EBR retention, it must comply with the Min/Max Governor settings for the variable retention period.

Table 19 CAS API functions for event-based retention

Function	Description
FPClip_EnableEBRWithClass	This function sets a C-Clip to be eligible to receive a future event and enables an event-based retention (EBR) class to be assigned to the C-Clip during C-Clip creation time.
FPClip_EnableEBRWithPeriod	This function sets a C-Clip to be eligible to receive a future event and enables an event-based retention (EBR) period to be assigned to the C-Clip during C-Clip creation time.
FPClip_IsEBREnabled	This function returns a Boolean value to indicate whether or not a C-Clip is enabled for event-based retention (EBR).
FPClip_GetEBRClassName	This function retrieves the name of the event-based retention (EBR) policy assigned to the C-Clip.
FPClip_GetEBREventTime	This function returns the event time set on a C-Clip when the event-based retention (EBR) event for that C-Clip was triggered.
FPClip_GetEBRPeriod	This function returns the value (in seconds) of the event-based retention (EBR) period associated with a C-Clip.
FPClip_TriggerEBREvent	This function triggers the event of a C-Clip for which event-based retention (EBR) was enabled.

Table 19 CAS API functions for event-based retention (continued)

Function	Description
FPClip_TriggerEBREventWithClass	This function triggers the event of a C-Clip for which event-based retention (EBR) was enabled and assigns a new EBR policy to the C-Clip.
FPClip_TriggerEBREventWithPeriod	This function triggers the event of a C-Clip for which event-based retention (EBR) was enabled and assigns a new EBR period to the C-Clip.

Litigation hold

Litigation hold allows CAS applications to temporarily prevent deletion of a C-Clip. Litigation hold is useful for data that is subject to an official investigation, subpoena, or inquiry and that may not be deleted until the investigation is over. Once there is no need to hold the data, the litigation hold can be released by the application and normal retention behavior resumes. The CAS application places and removes a litigation hold at the C-Clip level.

Note

Even a privileged delete cannot delete a C-Clip under litigation hold.

One C-Clip can be under several litigation holds. The application must generate unique litigation hold IDs and be able to track the specific litigation holds associated with a C-Clip. The application cannot query a C-Clip for this information. There is only a function that determines the litigation hold state of the C-Clip. If there is one or several litigation holds on the C-Clip, this function returns true, otherwise, it is false.

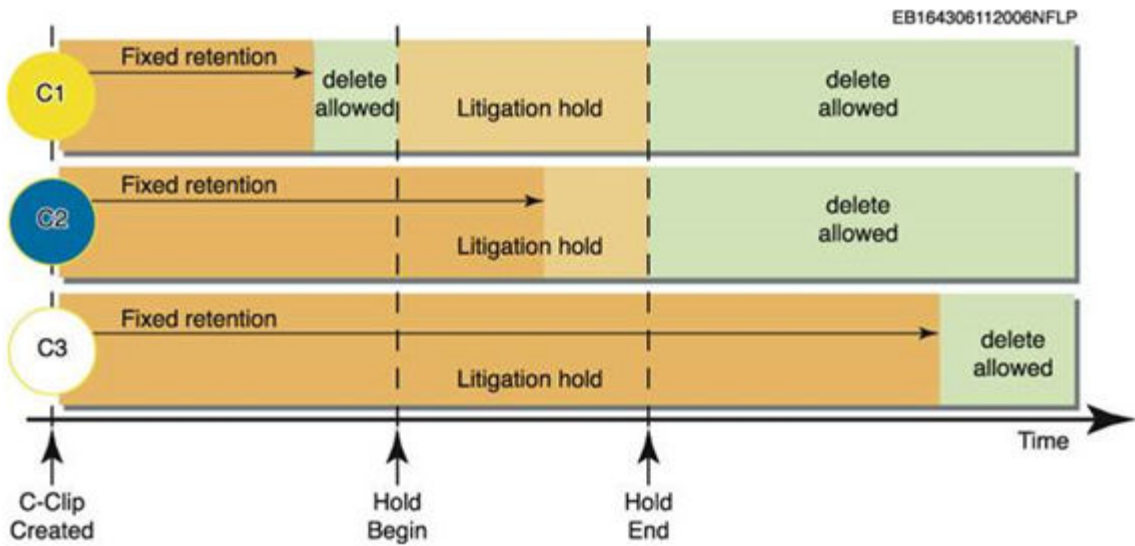
When using litigation hold, the C-Clip life-cycle is as follows:

- **Create:** The application creates a new C-Clip and provides a fixed and/or event-based retention period.
- **Set litigation hold:** An application puts the C-Clip on hold. This application can be different from the application that wrote the C-Clip.
- **Release litigation hold:** An application releases the C-Clip. This application can be different from the application that sets the litigation hold or wrote the C-Clip.
- **Delete:** When the application tries to delete the C-Clip, the following conditions must be satisfied:
 - There are no other litigation holds outstanding on the C-Clip.
 - Policy retention has expired.
 - Standard bucket retention has expired. (Standard bucket retention is available to all ECS object types, but is not recommended for CAS.)
 - Fixed retention period has expired (CAS-only feature).
 - Event-based retention has expired (CAS-only feature).

The following figure shows the three possible scenarios for a C-Clip put under litigation hold:

- C1 has a fixed retention that already expired when put under hold.
- C2 has a fixed retention that expires during the hold.
- C3 has a fixed retention that will expire after release of the hold.

Figure 4 Litigation hold scenarios



A C-Clip can have multiple litigation holds assigned to it. If this is the case, each litigation hold requires a separate API call with a unique identifier for the litigation hold.

Note

The maximum size of litigation hold ID is 64 characters. The maximum litigation hold IDs per C-Clip is 100. These limitations are enforced by the CAS API.

Table 20 CAS API functions for litigation hold

Function	Description
FPClip_GetRetentionHold	This function determines the hold state of the C-Clip and returns true or false.
FPClip_SetRetentionHold	This function sets or resets a retention hold on a C-Clip. For multiple litigation holds, provide a unique litigation hold ID for each hold. For multiple holds, make one call per ID.

Set up namespace retention policies

Provides CAS-specific set up instructions for namespace retention policies.

The Retention Policy feature for namespace provides a way to define and manage CAS retention classes for all C-Clips created in the namespace.

A namespace can have many retention policies, where each policy defines a retention period. By applying a retention policy to a number of C-Clips (with the API), a change to the retention policy changes the retention period for all objects associated with the policy. For CAS, retention classes are applied to an object's C-Clip by the application. If an object is under a retention period, requests to modify the object are not allowed.

Procedure

1. At the ECS portal, select **Manage > Namespace**.

2. To edit the configuration of an existing namespace, choose the **Edit** action associated with the existing namespace.
3. Add and Configure Retention Policies.
 - a. In the Retention Policies area, select **Add** to add a new policy.
 - b. Enter a name for the policy.
 - c. Specify the period for the Retention Policy.

Select the **Infinite** checkbox to ensure that objects with this policy are never deleted.

Figure 5 New Retention Policy

The screenshot shows the 'Retention Policies' interface. At the top, there's a header 'Retention Policies' with a help icon. Below it, there are input fields for 'Name' (containing 'threemonth'), 'Value' (containing '3'), and a dropdown menu for the unit (set to 'Months'). An 'Add' button is to the right. Below these fields is a table with columns 'Name', 'Value', and 'Actions'. The table currently shows 'No data.'.

4. Select **Save**.

Figure 6 Retention policies for a namespace

The screenshot shows the 'Retention Policies' interface after several policies have been added. The 'Add' button is still present. The table below now contains three rows of policies:

Name	Value	Actions
Infinite	Infinite	Delete
oneyear	1 Years	Delete
threemonth	3 Months	Delete

Create and set up a bucket for a CAS user

Configure a bucket to support a CAS user.

In ECS, management users create buckets and become the bucket owners. For CAS, object users need to be set up as bucket owners. Follow this procedure to properly set up a CAS bucket and make the CAS user the bucket owner. In this example, `newcasadmin1` is a management user, `newcasuser1` is a CAS object user, and `newcasns1` is the namespace. The procedure assumes the two users and namespace have been set up.

Procedure

1. Login to the ECS Portal as `newcasadmin1`.
2. At the ECS portal, select **Manage > Bucket**.
3. Choose **New Bucket**.

4. Fill in the fields as shown below:

Field	Value
Replication Group	Your replication group
Set current user as Bucket Owner	Check
CAS	On

5. Choose **Save**.
6. Select **Manage > User**.
7. Make sure the **Object User** tab is active, search for `newcasuser1` and choose **Edit**.
8. In **Default Bucket**, type `newcasbucket1` and choose **Set Bucket**.
9. Choose **Close**.
10. Select **Manage > Bucket**.
11. Search for `newcasbucket1` and choose **Edit bucket**.
12. In **Bucket Owner**, type `newcasuser1`.
13. Choose **Save**.

Set up a CAS object user

Set up an object user to use CAS.

When you set up an object user, you can assign CAS features to the profile that make up the elements of a CAS profile. You will be able to view the resulting PEA file for use in your CAS applications.

Procedure

1. At the ECS portal, select **Manage > Users**.
2. To edit the configuration of an existing object user, choose the **Edit** action associated with the user.

Figure 7 CAS settings for object users

CAS

.....

Set Password Generate Generate PEA File

PEA File

```
<pea version="1.0.0">
  <defaultkey name="casuser1">
    <credential id="csp1.secret" enc="base64">RnBRN0NGd25uWnZmNGbxdUJkQnE=
  </credential>
  </defaultkey>
  <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
    <value>
      <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
        <value>
          <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
            <value>
              <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                <value>
                  <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                    <value>
                      <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                        <value>
                          <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                            <value>
                              <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                <value>
                                  <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                    <value>
                                      <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                        <value>
                                          <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                            <value>
                                              <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                <value>
                                                  <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                    <value>
                                                      <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                        <value>
                                                          <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                            <value>
                                                              <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                                <value>
                                                                  <key type="cluster" id="e9f1a3c9-5cb1-3a76-a589-f639de9ad2e4" name="casuser1">
                                                                  </value>
                                                                </value>
                                                              </value>
                                                            </value>
                                                          </value>
                                                        </value>
                                                      </value>
                                                    </value>
                                                  </value>
                                                </value>
                                              </value>
                                            </value>
                                          </value>
                                        </value>
                                      </value>
                                    </value>
                                  </value>
                                </value>
                              </value>
                            </value>
                          </value>
                        </value>
                      </value>
                    </value>
                  </value>
                </value>
              </value>
            </value>
          </value>
        </value>
      </value>
    </value>
  </key>
</pea>
```

Default Bucket

CASbucket1

Set Bucket

Metadata

Attribute	Value	
app	banking	Delete

+ Add Attribute Save Metadata

3. In the CAS area, type a password (secret) or choose **Generate** to have the portal create one for you.
4. Choose **Set Password**.
5. Choose **Generate PEA File** to generate the PEA file your application will need to authenticate to the CAS storage on ECS.
6. By setting a default bucket, every action the user takes that does not specify a bucket will use the specified default bucket. Type the name of the default bucket and choose **Set Bucket**.
7. Choose **Add Attribute** to add a metadata tag to the user.
8. Add the metadata tag name and value.
See the CAS SDK documentation for more info on metadata tags.
9. Choose **Save Metadata**.

Set up bucket ACLs for CAS

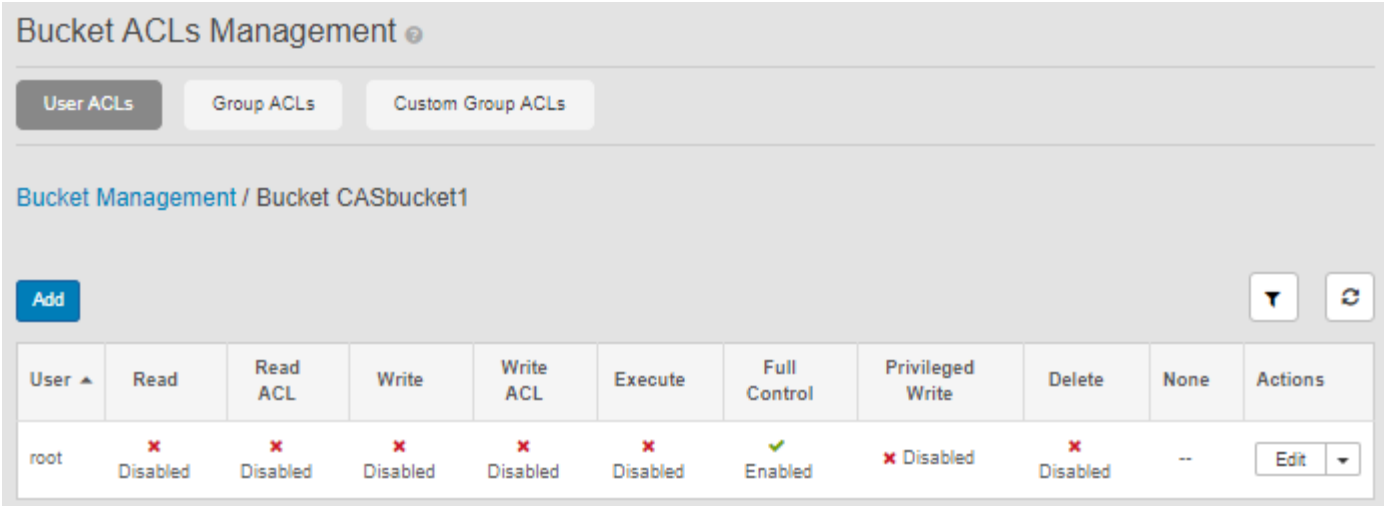
Edit a bucket's access control list to limit a user's access.

Some ECS bucket ACLs map to CAS permissions and some have no meaning for CAS data.

Procedure

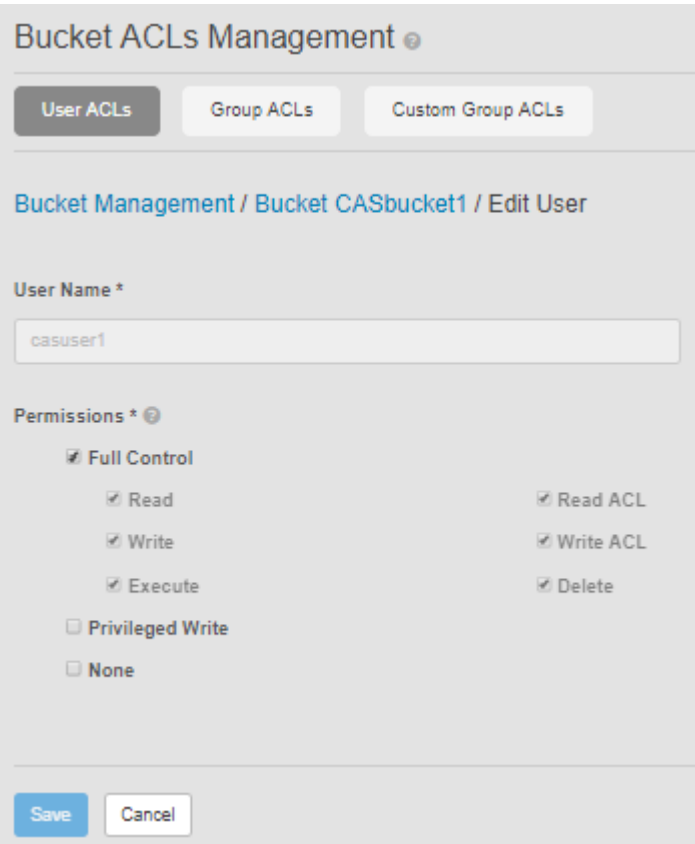
1. At the ECS portal, select **Manage > Bucket**.
2. To edit the ACLs of an existing bucket, choose the **Edit ACL** action associated with the existing bucket.

Figure 8 Edit bucket ACL



3. Choose the **Edit** associated with the user.

Figure 9 Bucket ACLs Management



4. Modify the permissions.

Table 21 Bucket ACLs

ECS ACL	ACL definition
READ	Read, Query, and Exist capabilities

Table 21 Bucket ACLs (continued)

ECS ACL	ACL definition
WRITE	Write and Litigation Hold capabilities
FULL_CONTROL	Read, Delete, Query, Exist, Clip Copy, Write, Litigation Hold
PRIVILEGED_WRITE	Privileged Delete
DELETE	Delete

Note

Other ECS ACLs have no meaning to CAS.

5. Select **Save**.
6. You can also edit the ACLs at the group level. Groups are predefined and membership in the group is automatic based on user criteria. Choose **Group ACLs**.
7. Choose **Add**.
8. Select the group you want to edit from the **Group Name** list.

Table 22 Bucket ACL groups

Bucket ACL group	Description
public	All users authenticated or not.
all users	All authenticated users.
other	Authenticated users but not the bucket owner.
log delivery	Not supported.

9. Edit the ACLs and select **Save**.

ECS Management APIs that support CAS users

Describes ECS Management API resources that you can use to manage CAS user and profile settings.

ECS Management API resource descriptions:

- `GET /object/user-cas/secret/{uid}` : Gets the CAS secret for the specified user.
- `GET /object/user-cas/secret/{namespace}/{uid}`: Gets the CAS secret for the specified namespace and user.
- `POST /object/user-cas/secret/{uid}`: Creates or updates the CAS secret for a specified user.
- `GET /object/user-cas/secret/{namespace}/{uid}/pea`: Generates a PEA file for the specified user.
- `POST /object/user-cas/secret/{uid}/deactivate`: Deletes the CAS secret for a specified user.

- `GET /object/user-cas/bucket/{namespace}/{uid}`: Gets the default bucket for the specified namespace and user.
- `GET /object/user-cas/bucket/{uid}`: Gets the default bucket for a specified user.
- `POST /object/user-cas/bucket/{namespace}/{uid}`: Updates the default bucket for the specified namespace and user.
- `GET /object/user-cas/applications/{namespace}`: Gets the CAS registered applications for a specified namespace.
- `POST /object/user-cas/metadata/{namespace}/{uid}`: Updates the CAS registered applications for a specified namespace and user.
- `GET /object/user-cas/metadata/{namespace}/{uid}`: Gets the CAS user metadata for the specified namespace and user.

See the for more information.

Content Addressable Storage (CAS) SDK API support

Supported versions

ECS supports the CAS build 3.1.544 or higher. Additionally you should verify that your ISV's application supports ECS.

More information on ECS CAS support is provided in [Setting up CAS support in ECS](#) on page 100.

CAS Query support

CAS Query is supported beginning with ECS 2.2.

Note

In ECS, CAS Query operations return results based on the creation time of the existing C-Clip and the deletion time of the deleted C-Clip (reflection). In EMC Centera, query operations return results based on the write-time of the object.

Unsupported APIs in ECS versions before ECS 3.0

CAS SDK API calls not supported in versions of ECS prior to ECS 3.0:

- `FPCLip_EnableEBRWithClass`
- `FPCLip_EnableEBRWithPeriod`
- `FPCLip_SetRetentionHold`
- `FPCLip_TriggerEBREvent`
- `FPCLip_TriggerEBREventWithClass`
- `FPCLip_TriggerEBREventWithPeriod`
- `FPCLip_GetEBRClassName`
- `FPCLip_GetEBREventTime`
- `FPCLip_GetEBRPeriod`
- `FPCLip_GetRetentionHold`
- `FPCLip_IsEBREnabled`

ECS CAS error codes

The error codes that can be generated by the CAS head are listed in the following table.

Value	Error name	Description
10001	FP_INVALID_NAME	The name that you have used is not XML compliant.
10002	FP_UNKNOWN_OPTION	You have used an unknown option name with <code>FPPool_SetIntOption()</code> or <code>FPPool_GetIntOption()</code> .
10003	FP_NOT_SEND_REQUEST_ERR	An error occurred when you sent a request to the server. This internal error was generated because the server could not accept the request packet. Verify all LAN connections and try again.
10004	FP_NOT_RECEIVE_REPLY_ERR	No reply was received from the server. This internal error was generated because the server did not send a reply to the request packet. Verify all LAN connections and try again.
10005	FP_SERVER_ERR	The server reports an error. An internal error on the server occurred. Try again.
10006	FP_PARAM_ERR	You have used an incorrect or unknown parameter. Example: Is a string-variable too long, null, or empty when it should not be? Does a parameter have a limited set of values? Check each parameter in your code.
10007	FP_PATH_NOT_FOUND_ERR	This path does not correspond to a file or directory on the client system. The path in one of your parameters does not point to an existing file or directory. Verify the path in your code.
10008	FP_CONTROLFIELD_ERR	The server reports that the operation generated a "Controlfield missing" error. This internal error was generated because the required control field was not found. Try again. (Obsolete from v2.0.)
10009	FP_SEGDATA_ERR	The server reports that the operation generated a "Segdatafield missing" error. This internal error was generated because the required field containing the blob data was not found in the packet. Try again. (Obsolete from v2.0.)
10010	FP_DUPLICATE_FILE_ERR	A duplicate CA already exists on the server. If you did not enable duplicate file detection, verify that you have not already stored this data and try again.
10011	FP_OFFSET_FIELD_ERR	The server reports that the operation generated an "Offsetfield missing" error. This internal error was generated because the offset field was not found in the packet. Try again. (Obsolete from v2.0.)
10012	FP_OPERATION_NOT_SUPPORTED	This operation is not supported. If <code>FPClip_Write()</code> , <code>FPTag_GetSibling()</code> , <code>FPTag_GetPrevSibling()</code> , <code>FPTag_GetFirstChild()</code> or <code>FPTag_Delete()</code> returned this error, then this operation is not supported for C-Clips opened in 'flat' mode. If <code>FPStream</code> returned this error, then

		you are trying to perform an operation that is not supported by that stream.
10013	FP_ACK_NOT_RCV_ERR	A write acknowledgement was not received. Verify your LAN connections and try again.
10014	FP_FILE_NOT_STORED_ERR	Could not write the blob to the server OR could not find the blob on the server. This internal error was generated because the store operation of the blob was not successful. Verify that the original data was correctly stored, verify your LAN connections and try again.
10015	FP_NUMLOC_FIELD_ERR	The server reports that the operation generated a "Numlockfield missing" error. This internal error was generated because the numlock field was not found in the packet. Try again. (Obsolete from v2.0.)
10016	FP_SECTION_NOT_FOUND_ERR	The GetSection request could not retrieve the defined section tag. This internal error was generated because a required section is missing in the CDF. Verify the content of your code and try again. (Obsolete from v2.0.)
10017	FP_TAG_NOT_FOUND_ERR	The referenced tag could not be found in the CDF. This internal error was generated because information is missing from the description section in the CDF. Verify the content of your code and try again.
10018	FP_ATTR_NOT_FOUND_ERR	Could not find an attribute with that name. If FPTag_GetXXXAttribute() returned this error, then the attribute was not found in the tag. If FPTag_GetIndexAttribute() returned this error, then the index parameter is larger than the number of attributes in the tag.
10019	FP_WRONG_REFERENCE_ERR	The reference that you have used is invalid. The reference was not opened, already closed, or not of the correct type.
10020	FP_NO_POOL_ERR	It was not possible to establish a connection with a cluster. The server could not be located. This means that none of the IP addresses could be used to open a connection to the server or that no cluster could be found that has the required capability. Verify your LAN connections, server settings, and try again.
10021	FP_CLIP_NOT_FOUND_ERR	Could not find the referenced C-Clip in the cluster. Returned by FPClip_Open(), it means the CDF could not be found on the server. Verify that the original data was correctly stored and try again.
10022	FP_TAGTREE_ERR	An error exists in the tag tree. Verify the content of your code and try again.
10023	FP_ISNOT_DIRECTORY_ERR	A path to a file has been given but a path to a directory is expected. Verify the path to the data and try again.
10024	FP_UNEXPECTEDTAG_ERR	Either a "file" or "folder" tag was expected but not given. An unexpected tag was found when retrieving the CDF. The CDF is probably corrupt.
10025	FP_TAG_READONLY_ERR	The tag cannot be changed or deleted (it is probably a top tag). Verify your program logic.

10026	FP_OUT_OF_BOUNDS_ERR	The options parameter is out of bounds. One of the function parameters exceeds its preset limits. Verify each parameter in your code.
10027	FP_FILESYS_ERR	A file system error occurred, for example an incorrect path was given, or you are trying to open an unknown file or a file in the wrong mode. Verify the path and try again.
10029	FP_STACK_DEPTH_ERR	You have exceeded the nested tag limit. Review the structure of your content description and try again. Deprecated.
10030	FP_TAG_HAS_NO_DATA_ERR	You are trying to access blob data of a tag that does not contain blob data.
10031	FP_VERSION_ERR	The C-Clip has been created using a more recent version of the client software than you are using. Upgrade to the latest version.
10032	FP_MULTI_BLOB_ERR	The tag already has data associated with it. You need to create a new tag to store the new data or delete this tag and recreate it and try again.
10033	FP_PROTOCOL_ERR	You have used an unknown protocol option (Only HPP is supported). Verify the parameters in your code. It is also possible that an internal communication error occurred between the server and client. If you have verified your code and the problem persists then you need to upgrade to the latest client and server versions.
10034	FP_NO_SOCKET_AVAIL_ERR	No new network socket is available for the transaction. Reduce the number of open transactions between the client and the server or use the function <code>FPPool_SetGlobalOption()</code> to increase the number of available sockets with <code>FP_OPTION_MAXCONNECTIONS</code> .
10035	FP_BLOBIDFIELD_ERR	A BlobID field (the Content Address) was expected but not given. Upgrade to the latest client and server versions. (Obsolete from v2.0.)
10036	FP_BLOBIDMISMATCH_ERR	The blob is corrupt: a BlobID mismatch occurred between the client and server. The Content Address calculation on the client and the server has returned different results. The blob is corrupt. If <code>FPClip_Open()</code> returns this error, it means the blob data or metadata of the C-Clip is corrupt and cannot be decoded.
10037	FP_PROBEPACKET_ERR	The probe packet does not contain valid server addresses. Upgrade to the latest client and server versions. (Obsolete from v2.0.)
10038	FP_CLIPCLOSED_ERR	(Javaonly.) You tried to perform an operation on a closed C-Clip. This operation requires access to an open C-Clip. Verify your code and try again.
10039	FP_POOLCLOSED_ERR	(Javaonly.) You tried to perform an operation on a closed pool. This operation requires access to an open pool. Verify your code and LAN connections and try again.

10040	FP_BLOBBUSY_ERR	The blob on the cluster is busy and cannot be read from or written to. You tried to read from or write to a blob that is currently busy with another read/write operation. Try again.
10041	FP_SERVER_NOTREADY_ERR	The server is not ready yet. This error can occur when a client tries to connect to the server to execute an operation and the nodes with the access role are running but the nodes with the storage role have not been initialized yet. This error can also occur when not enough mirror groups are found on the server. Allow the SDK to perform the automatic number of configured retries.
10042	FP_SERVER_NO_CAPACITY_ERR	The server has no capacity to store data. Enlarge the server's capacity and try again.
10043	FP_DUPLICATE_ID_ERR	The application passed in a sequence ID that was previously used.
10044	FP_STREAM_VALIDATION_ERR	A generic stream validation error occurred.
10045	FP_STREAM_BYTECOUNT_MISMATCH_ERR	A generic stream byte count mismatch was detected.
10101	FP_SOCKET_ERR	An error on the network socket occurred. Verify the network.
10102	FP_PACKETDATA_ERR	The data packet contains wrong data. Verify the network, the version of the server or try again later.
10103	FP_ACCESSNODE_ERR	No node with the access role can be found. Verify the IP addresses provided with FPPool_Open().
10151	FP_OPCODE_FIELD_ERR	The Query Opcode field is missing from the packet.
10152	FP_PACKET_FIELD_MISSING_ERR	The packet field is missing.
10153	FP_AUTHENTICATION_FAILED_ERR	Authentication to get access to the server failed. Check the profile name and secret.
10154	FP_UNKNOWN_AUTH_SCHEME_ERR	An unknown authentication scheme has been used.
10155	FP_UNKNOWN_AUTH_PROTOCOL_ERR	An unknown authentication protocol has been used.
10156	FP_TRANSACTION_FAILED_ERR	Transaction on the server failed.
10157	FP_PROFILECLIPID_NOTFOUND_ERR	No profile clip was found.
10158	FP_ADVANCED_RETENTION_DISABLED_ERR	The Advanced Retention Management feature is not licensed or enabled for event-based retention (EBR) and retention hold.
10159	FP_NON_EBR_CLIP_ERR	An attempt was made to trigger an EBRevent on a C-Clip that is not eligible to receive an event.
10160	FP_EBR_OVERRIDE_ERR	An attempt was made to trigger or enable the event-based retention period/class of a C-Clip a second time. You can set EBR information only once.
10161	FP_NO_EBR_EVENT_ERR	The C-Clip is under event-based retention protection and cannot be deleted.
10162	FP_RETENTION_OUT_OF_BOUNDS_ERR	The event-based retention period being set does not meet the minimum/maximum rule.

10163	FP_RETENTION_HOLD_COUNT_ERR	The number of retention holds exceeds the limit of 100.
10164	FP_METADATA_MISMATCH_ERR	Mutable metadata mismatch found.
10201	FP_OPERATION_REQUIRES_MARK	The application requires marker support but the stream does not provide that.
10202	FP_QUERYCLOSED_ERR	The FP Query for this object is already closed. (Java only).
10203	FP_WRONG_STREAM_ERR	The function expects an input stream and gets an output stream or vice-versa.
10204	FP_OPERATION_NOT_ALLOWED	The use of this operation is restricted or this operation is not allowed because the server capability is false.
10205	FP_SDK_INTERNAL_ERR	An SDK internal programming error has been detected.
10206	FP_OUT_OF_MEMORY_ERR	The system ran out of memory. Check the system's capacity.
10207	FP_OBJECTINUSE_ERR	Cannot close the object because it is in use. Check your code.
10208	FP_NOTYET_OPEN_ERR	The object is not yet opened. Check your code.
10209	FP_STREAM_ERR	An error occurred in the generic stream. Check your code.
10210	FP_TAG_CLOSED_ERR	The FP Tag for this object is already closed. (Java only.)
10211	FP_THREAD_ERR	An error occurred while creating a background thread.
10212	FP_PROBE_TIME_EXPIRED_ERR	The probe limit time was reached.
10213	FP_PROFILECLIPID_WRITE_ERR	There was an error while storing the profile clip ID.
10214	FP_INVALID_XML_ERR	The specified string is not valid XML.
10215	FP_UNABLE_TO_GET_LAST_ERROR	The call to FPPool_GetLastError() or FPPool_GetLastErrorInfo() failed. The error status of the previous function call is unknown; the previous call may have succeeded.
10216	FP_LOGGING_CALLBACK_ERR	An error occurred in the application-defined FP Logging callback.

PART 5

ECS Management REST API

[Chapter 5, "ECS Management REST API"](#)

CHAPTER 5

ECS Management REST API

- [ECS Management REST API introduction](#)..... 126
- [Authenticate with the ECS Management REST API](#)..... 126

ECS Management REST API introduction

This section describes how to access and authenticate with the ECS Management REST API and provides a summary of the API paths. You can configure and manage the object store using the ECS Management REST API. Once the object store is configured, you can perform object create, read, update, and delete operations using the ECS-supported object and file protocols.

For more information on the ECS Management REST API, see these topics:

- [Authenticate with the ECS Management REST API](#) on page 126
- [ECS Management REST API summary](#) on page 129

In addition, you can refer to the [ECS API Reference](#) which is auto-generated from the source code and provides a reference for the methods available in the API.

Authenticate with the ECS Management REST API

ECS uses a token-based authentication system for REST API calls. This section provides examples of authenticating with the ECS API, with and without cookies.

When you are authenticated by ECS, the ECS API returns an authentication token. You can use this token for authentication in subsequent calls.

- The ECS API returns an HTTP 401 code if the client automatically follows redirects. You must then log in and authenticate to obtain a new token.
- The ECS API returns an HTTP 302 code if the client does not automatically follow redirects. The 302 code directs the client to where it must get re-authenticated.

You can retrieve and use authentication tokens by:

- Saving the X-SDS-AUTH-TOKEN cookie from a successful authentication request and sending that cookie with subsequent requests.
- Reading the X-SDS-AUTH-TOKEN HTTP header from a successful authentication request and copying that header into any subsequent request.

The ECS API is available on port :4443. Clients access ECS by issuing a login request in the form:

```
https://<ECS_IP>:4443/login
```

Authenticate without cookies

The following example shows how to use authentication tokens by reading the X-SDS-AUTH-TOKEN HTTP header from a successful authentication request and copying that header into a subsequent request. This example does not use cookies. The examples are written using the `curl` command line tool and formatted for readability.

The following ECS API call executes a GET on the `/login` resource. The `-u` option specifies the user of the basic authentication header. You must specify the user in the request. Upon successful authentication, the ECS API returns a HTTP 200 code and the X-SDS-AUTH-TOKEN header containing the encoded token.

The default ECS API token lifetime is 8 hours, which means that after 8 hours the token is no longer valid. The default idle time for a token is two hours; after a two hour idle time, the token expires. If you use an expired token, you are redirected to the /

login URL. You will receive an HTTP status error code 401 upon any subsequent use of the expired token.

```
curl -L --location-trusted -k https://10.247.100.247:4443/login -u "root:ChangeMe" -v

> GET /login HTTP/1.1
> Authorization: Basic cm9vdDpDaGFuZ2VNZQ==
> User-Agent: curl/7.24.0 (i386-pc-win32) libcurl/7.24.0 OpenSSL/0.9.8t zlib/1.2.5
> Host: 10.247.100.247:4443
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 26 Nov 2013 22:18:25 GMT
< Content-Type: application/xml
< Content-Length: 93
< Connection: keep-alive
< X-SDS-AUTH-TOKEN:
BAAcQ0xOd3g0MjRCUG4zT3NJdnNuMlAvQTFYblNrPQMAUAQADTEzODU0OTQ4NzYzNTICAAEABQA5dXJu
OnN0b3JhZ2VvczpwU2t1bjo2MjIxOTcyZS01NGUyLFRmNWQtYWZjOC1kMGE3ZDJmZDU3MmU6AgAC0A8=
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loggedIn>
  <user>root</user>
</loggedIn>
* Connection #0 to host 10.247.100.247 left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
```

You can copy the X-SDS-AUTH-TOKEN contents and pass it into the next API call through the curl tool `-H` switch, as shown in the following example.

```
curl https://10.247.100.247:4443/object/namespaces
-k
-H "X-SDS-AUTH-TOKEN:
BAAcOHZLaGF4MTl3eFhpY0czZ0tWUGhJV2xreUE4PQMAUAQADTEzODU0OTQ4NzYzNTICAAEABQA5dXJu
OnN0b3JhZ2VvczpwU2t1bjo2MjIxOTcyZS01NGUyLFRmNWQtYWZjOC1kMGE3ZDJmZDU3MmU6AgAC0A8="
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<namespaces>
  <namespace>
    <id>ns1</id>
    <link rel="self" href="/object/namespaces/namespace/ns1"/>
    <names>ns1</names>
  </namespace>
</namespaces>
```

Authenticate with cookies

This example shows how to use authentication tokens by saving the cookie from a successful authentication request and passing the cookie into a subsequent request.

The following example uses the `?using-cookies=true` parameter to indicate that you want to receive cookies in addition to the normal HTTP header. The Curl command saves the authentication token to a file named `cookiefile` in the current directory.

```
curl -L --location-trusted -k https://<ECS_IP>:4443/login?using-cookies=true
-u "root:Password"
-c cookiefile
```

```
-v
```

The following command passes the cookie with the authentication token using the Curl command `-b` switch, and returns the user's tenant information.

```
curl -k https://10.247.100.247:4443/object/namespaces -b cookiefile -v

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<namespaces>
  <namespace>
    <id>ns1</id>
    <link rel="self" href="/object/namespaces/namespace/ns1"/>
    <names>ns1</names>
  </namespace>
</namespaces>
```

Logout

The logout API ends a session.

Each user is allowed a maximum of 100 concurrent authentication tokens. Beyond this limit, the system refuses any new connection for a user until tokens free up. Tokens can free up by expiring naturally, or by issuing the following ECS API call:

```
GET https://<ECS_IP>:4443/logout
```

If you have multiple sessions running simultaneously, the following API call forces the termination of all tokens related to the current user.

```
GET https://<ECS_IP>:4443/logout?force=true
```

The following example shows a logout request. You pass in the authentication token from header or cookie to log out.

```
GET https://<ECS_IP>:4443/logout
X-SDS-AUTH-TOKEN: {Auth_Token}
```

The response should be HTTP 200.

ECS Management REST API whoami command

An ECS user can view their own user name, tenant association, and roles using the `whoami` API call.

Request

```
GET https://<ECS_IP>:4443/user/whoami
```

The following responses shows the `whoami` output for the root user and for a user who has been assigned to the `NAMESPACE_ADMIN` role for the `ns1` namespace.

Response

```

HTTP 200

GET /user/whoami
<user>
  <common_name>root</common_name>
  <distinguished_name/>
  <namespace/>
  <roles>
    <role>SYSTEM_ADMIN</role>
  </roles>
</user>

```

```

HTTP 200

GET /user/whoami
<user>
  <common_name>fred@corp.sean.com</common_name>
  <distinguished_name/>
  <namespace>ns1</namespace>
  <roles>
    <role>NAMESPACE_ADMIN</role>
  </roles>
</user>

```

ECS Management REST API summary

The ECS Management REST API enables the ECS object store to be configured and managed.

The following table summarizes the ECS Management REST API.

Table 23 ECS Management REST API- methods summary

API Area	Description
Configuration	
Certificate	<p>/object-cert</p> <p>API to manage certificates.</p> <p>/object-cert/keystore</p> <p>API to specify and rotate the certificate chain used by ECS.</p>
Configuration Properties	<p>/config/object/properties</p> <p>API to set the user scope as GLOBAL or NAMESPACE. This must be set before the first user is created. The default is GLOBAL.</p> <p>In GLOBAL scope, users are global and are can be shared across namespaces. In this case, the default namespace associated with a user determines the namespace for object operations and there is no need to supply a namespace for an operation. In NAMESPACE scope, a user is associated with a namespace. In this case, there might be more than one user with the same name, each associated with a different namespace, and a namespace must be provided for every operation.</p>
Licensing	<p>/license</p> <p>API to add a license and retrieve license details.</p>

Table 23 ECS Management REST API- methods summary (continued)

API Area	Description
Feature	<code>/feature/ServerSideEncryption</code> API to retrieve the details of the ServerSideEncryption feature.
Syslog	<code>/vdc/syslog/config</code> API to manage Syslog configuration and send alerts to the Syslog server for troubleshooting and debugging purposes.
SNMP	<code>/vdc/snmp/config</code> API to manage SNMP configuration and send alerts to SNMP server for troubleshooting and debugging purposes.
CAS	
CAS user profile	<code>/object/user-cas/secret</code> API to assign secret keys to CAS users and generate the Pool Entry Authorization (PEA) file. <code>/object/user-cas/bucket</code> API to retrieve or update the default bucket of a specified CAS user. <code>/object/user-cas/applications/{namespace}</code> API to retrieve the CAS registered applications for a specified namespace. <code>/object/user-cas/metadata/{namespace}/{uid}</code> API to retrieve or update the CAS user metadata for the specified namespace and CAS user.
File system access	
NFS	<code>/object/nfs</code> API to create an NFS export based on an ECS bucket and enable access to the export by UNIX users and groups. <code>/object/nfs/users</code> API to manage mapping between ECS user/group and corresponding UNIX user ID. <code>/object/nfs/exports</code> API to create and manage NFS exports.
Metering	
Billing	<code>/object/billing</code> API to meter object store usage at the namespace and bucket level.
Migration	
Transformation	<code>/object/transformation</code> API to enable data transformation from a Centera cluster.
Monitoring	
Capacity	<code>/object/capacity</code> API to retrieve the current managed capacity.

Table 23 ECS Management REST API- methods summary (continued)

API Area	Description
Dashboard	<p><code>/dashboard/zones/localzone</code></p> <p>API to retrieve the local VDC details, including details on replication groups, storage pools, nodes, and disks.</p> <p><code>/dashboard/zones/hostedzone</code></p> <p>API to retrieve the hosted VDC details, including details on replication groups.</p> <p><code>/dashboard/replicationgroups/{id}</code></p> <p>API to retrieve the replication group instance details.</p> <p><code>/dashboard/storagepools/{id}</code></p> <p>API to retrieve the storage pool details, including details on the storage pool nodes.</p> <p><code>/dashboard/nodes/{id}</code></p> <p>API to retrieve the node instance details, including node instance disk and process details.</p> <p><code>/dashboard/disks/{id}</code></p> <p>API to retrieve the disk instance details.</p> <p><code>/dashboard/processes/{id}</code></p> <p>API to retrieve the process instance details.</p> <p><code>/dashboard/rglinks/{id}</code></p> <p>API to retrieve the replication group link instance details.</p> <p><code>/dashboard/datatables/{id}</code></p> <p>API to retrieve the replication group datatables instance details.</p>
Events	<p><code>/vdc/events</code></p> <p>API to retrieve audit events for a specified namespace.</p>
Alerts	<p><code>/vdc/alerts</code></p> <p>API to retrieve audit alerts.</p>
Multi-tenancy	
Namespace	<p><code>/object/namespaces</code></p> <p>API to create and manage a namespace.</p> <p>This API also sets the retention period and quota for the namespace. For more information on retention periods and quotas, see the <i>ECS Administration Guide</i> available from the ECS Product Documentation page.</p>
Geo-replication	
Replication Group	<p><code>/data/data-service/vpools</code></p> <p>API to create and manage replication groups.</p>
Temporary Failed Zone	<p><code>/tempfailedzone/</code></p> <p>API to retrieve all temporary failed zones, or the temporary failed zones for a specified replication group.</p>

Table 23 ECS Management REST API- methods summary (continued)

API Area	Description
Provisioning	
Base URL	<p>/object/baseurl</p> <p>API to create a Base URL that allows existing applications to work with the ECS object store. For more information on Base URL, see the <i>ECS Administration Guide</i> available from the ECS Product Documentation page.</p>
Bucket	<p>/object/bucket</p> <p>API to provision and manage buckets.</p> <p>/object/bucket/{bucketName}/lock</p> <p>API to lock bucket access.</p> <p>/object/bucket/{bucketName}/tags</p> <p>API to add tags to a specified bucket.</p> <p>/object/bucket/{bucketName}/retention</p> <p>API to set the retention period for a specified bucket.</p> <p>/object/bucket/{bucketName}/quota</p> <p>API to set the quota for a specified bucket.</p> <p>/object/bucket/{bucketName}/policy</p> <p>API to add a policy for a specified bucket.</p> <p>/object/bucket/{bucketName}/metadata</p> <p>API to add metadata for a specified bucket.</p>
Data store	<p>/vdc/data-stores</p> <p>API to create data stores on file systems (/vdc/data-stores/filesystems) or on commodity nodes (/vdc/data-stores/commodity).</p>
Node	<p>/vdc/nodes</p> <p>API to retrieve the nodes that are currently configured for the cluster.</p> <p>/vdc/nodes/{nodename}/lockdown</p> <p>API to set the locked or unlocked status for a specified node.</p> <p>/vdc/lockdown</p> <p>API to retrieve the locked or unlocked status for a VDC.</p>
Storage pool	<p>/vdc/data-services/varrays</p> <p>API to create and manage storage pools.</p>
Virtual data center	<p>/object/vdcs</p> <p>API to add a VDC and specify the inter-VDC endpoints and secret key for replication of data between ECS sites.</p>
VDC keystore	<p>/vdc/keystore</p> <p>API to manage certificates for a VDC.</p>
Support	

Table 23 ECS Management REST API- methods summary (continued)

API Area	Description
Call home	<p><code>/vdc/callhome/</code></p> <p>API for managing ESRS configuration and sending alerts to ConnectEMC for troubleshooting and debugging purposes.</p>
User Management	
Authentication provider	<p><code>/vdc/admin/authnproviders</code></p> <p>API to add and manage authentication providers.</p>
Password group (Swift)	<p><code>/object/user-password</code></p> <p>API to generate a password for use with OpenStack Swift authentication.</p>
Secret key	<p><code>/object/user-secret-keys</code></p> <p>API to assign secret keys to object users and to manage secret keys.</p>
Secret key self-service	<p><code>/object/secret-keys</code></p> <p>API to allow S3 users to create a new secret key that enables them to access objects and buckets within their namespace in the object store.</p>
User (Object)	<p><code>/object/users</code></p> <p>API to create and manage object users. Object users are always associated with a namespace. The API returns a secret key that can be used for S3 access. An object user assigned an S3 secret key can change it using the REST API.</p> <p><code>/object/users/lock.</code></p> <p>API to lock user access.</p> <p><code>/object/users/{userName}/tags.</code></p> <p>API to associate tags with a user ID. Tags are in the form of name=value pairs.</p>
User (management)	<p><code>/vdc/users</code></p> <p>API to create and manage users. Management users can be assigned to the System Administrator role or to the Namespace Administrator role. You can use this API the change the local management user password.</p>

PART 6

HDFS

[Chapter 6, "ECS HDFS Introduction"](#)

[Chapter 7, "Configure a simple Hadoop cluster with ECS HDFS"](#)

[Chapter 8, "Configure a Kerberized Hadoop cluster with ECS HDFS"](#)

[Chapter 9, "Troubleshooting"](#)

[Chapter 10, "Appendix: Guidance on Kerberos Configuration"](#)

[Chapter 11, "Appendix: Hadoop core-site.xml properties for ECS HDFS"](#)

[Chapter 12, "Appendix: Secure bucket metadata example"](#)

CHAPTER 6

ECS HDFS Introduction

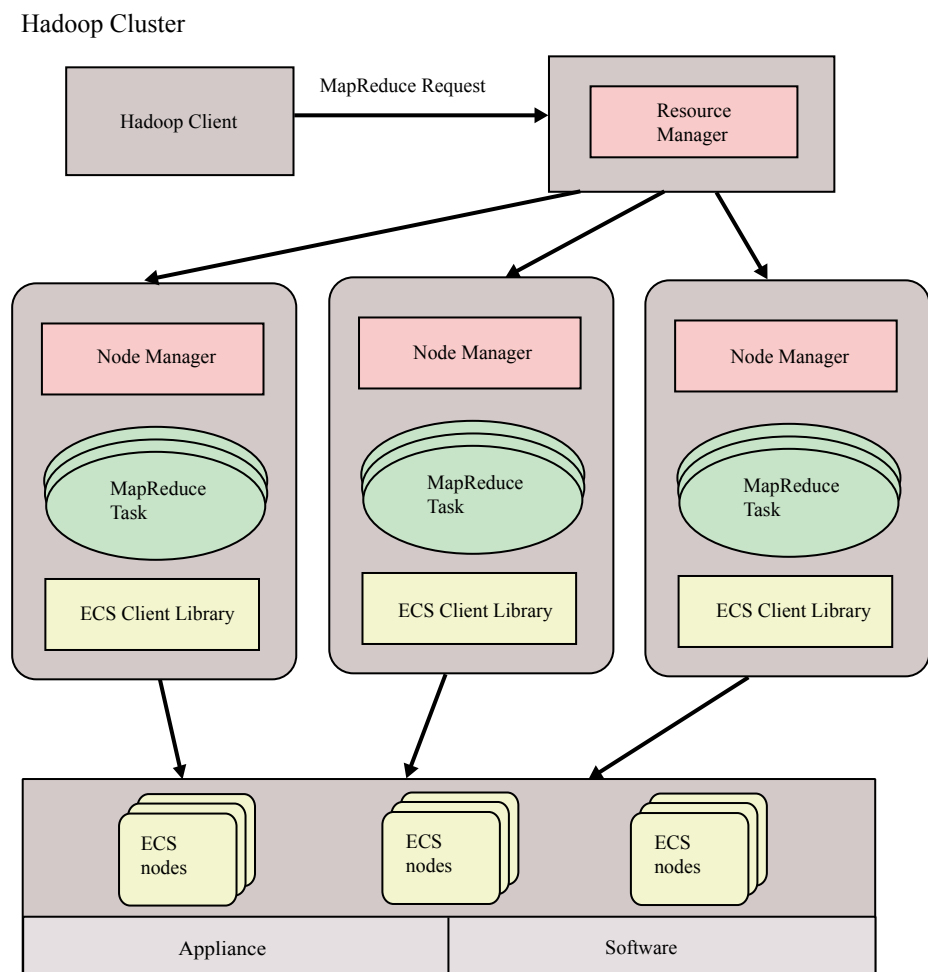
• ECS HDFS Introduction.....	138
• Configuring Hadoop to use ECS HDFS	139
• Hadoop authentication modes.....	139
• Migration from a simple to a Kerberos Hadoop cluster.....	143
• File system interaction.....	144
• Supported Hadoop applications.....	144

ECS HDFS Introduction

ECS HDFS is a Hadoop Compatible File System (HCFS) that enables you to run Hadoop 2.x applications on top of the ECS storage infrastructure.

When using ECS HDFS, the Hadoop distribution is configured to run against the ECS HDFS instead of the built-in Hadoop file system. The following figure illustrates how ECS HDFS integrates with an existing Hadoop cluster.

Figure 10 ECS HDFS integration in a Hadoop cluster



In a Hadoop environment that is configured to use ECS HDFS, each of the ECS nodes functions as a traditional Hadoop NameNode and DataNode, so that all of the ECS nodes can accept and service HDFS requests.

When you set up the Hadoop client to use ECS HDFS instead of traditional HDFS, the configuration points to ECS HDFS to do all the HDFS activity. On each ECS HDFS client node, any traditional Hadoop component would use the ECS Client Library (the ViPRFS JAR file) to perform the HDFS activity.

To integrate ECS HDFS with an existing Hadoop environment, you must have the following:

- A Hadoop cluster that is already installed and configured. The following distributions are supported:
 - Hortonworks HDP 2.6.2
- A Hadoop cluster that is installed and configured to support ECS HDFS, which requires:
 - A file system-enabled bucket for HDFS access.

Note

Only one bucket is supported per Hadoop cluster and the ECS HDFS must be the default file system.

- The ECS Client Library that is deployed to the cluster.
- For a Hadoop cluster that uses Kerberos or Kerberos with Active Directory.
 - Kerberos configuration files and service principal keytab files that are deployed to the ECS cluster.
 - Secure metadata that is deployed to the bucket.

Configuring Hadoop to use ECS HDFS

Hadoop stores system configuration information in a number of files, including `core-site.xml`, `hdfs-site.xml` and `hive-site.xml`. The ECS HDFS configuration requires you to edit `core-site.xml`.

You must add or modify several types of properties in the `core-site.xml` file, including:

- ECS HDFS Java classes: This set of properties defines the ECS HDFS implementation classes that are contained in the ECS HDFS Client Library.
- File system location properties: These properties define the file system URI (scheme and authority) to use when running Hadoop jobs, and the IP addresses or FQDNs of the ECS data nodes for a specific ECS file system.
- Kerberos realm and service principal properties: These properties are required only in a Hadoop environment where Kerberos is present. These properties map Hadoop and ECS HDFS users.

The `core-site.xml` file resides on each node in the Hadoop cluster. You must add the same properties to each instance of `core-site.xml`.

Note

When modifying configuration files, you should use the management interface (Ambari) rather than manually editing files. Changes you make using the Ambari management interface are persisted across the cluster.

Hadoop authentication modes

Hadoop supports two different modes of operation for determining the identity of a user, simple and Kerberos.

Simple

In simple mode, the identity of a client process is determined by the host operating system. On Unix-like systems, the user name is the equivalent of `whoami`.

Kerberos

In a Hadoop environment with Kerberos, the identity of a client process is determined by its Kerberos credentials. For example, you can use the `kinit` utility to obtain a Kerberos ticket-granting-ticket (TGT) and use `klist` to determine your current principal. When mapping a Kerberos principal to an HDFS username, using the `auth_to_local` Hadoop property, all components except for the primary are dropped. For example, a principal `todd/foobar@CORP.COMPANY.COM` acts as the simple username “todd” on HDFS.

ECS HDFS integrates with Hadoop clusters configured to use either simple or Kerberos authentication modes.

When the Hadoop cluster uses Kerberos, you can configure ECS to grant access to users with Kerberos principals in the form `user@REALM.COM`. Alternatively, where ECS uses AD to authenticate users, you can configure a one-way trust between the Kerberos environment and AD so that users can authenticate using their AD credentials, in the form `user@DOMAIN.COM`.

The permissions of newly created files and directories are restricted by the `umask` (`fs.permissions.umask-mode`). The recommended `umask` is 022.

Accessing the bucket as a file system

The HDFS file system storage is provided by an ECS bucket. When you create a bucket, you must configure it in ECS so that it is available as a file system.

ECS (through the ECS Client Library) uses the permissions configured against the bucket and the settings in the Hadoop `core-site.xml` file to determine access to the root file system (bucket). You must ensure that you have configured sufficient access to enable Hadoop users and services to create files and directories in the bucket.

In general, all file and directory operations must be permitted by the bucket ACLs. Additionally, each individual file and directory object within the bucket has its own object ACL and all object operations must also be permitted by the object ACL. If the object operation does not satisfy the bucket ACL, the operation is denied. If the object operation does not satisfy the object ACL, the operation is denied.

An exception to this is that the bucket owner and the Hadoop superuser and members of the Hadoop supergroup, defined in `hdfs-site.xml`, are always permitted to perform any file system operation regardless of bucket and object ACLs.

You can set bucket ACLs by explicitly adding user ACLs on the bucket for every user, or by specifying custom group ACLs. For more information, see [Bucket Custom Group ACLs and Default Group](#) on page 141. The bucket owner must be an ECS object user. Other users do not need to be ECS object users and can be Unix usernames from the Hadoop cluster.

A further exception is that, unlike normal ECS buckets, a file system-enabled ECS bucket has a special object that represents the root directory and a special object for each directory. The root directory object does not exist in a new file system-enabled bucket does not have a root directory object, but is created when the first file system operation is performed on the bucket. When such a root directory object exists, some ECS HDFS API calls do not perform bucket ACL checks.

To ensure consistent permissions regardless of the API call, you should ensure that the root directory object ACL duplicates the bucket ACL.

Once users have access to the file system, the files and directories that they create have permissions determined by the `umask` property in the `core-site.xml` file.

Bucket Custom Group ACLs and Default Group

You can enable access to the bucket based on user ACLs or by assigning Custom Group ACLs. Custom groups are names of user groups as defined on the Hadoop cluster and enable Hadoop users to access the bucket using HDFS.

Typical groups defined on the Hadoop cluster are `hdfs` (with user `hdfs`), `hadoop` (typically includes all service users), and `users` (includes other non-service users who access applications on the Hadoop cluster). You can create corresponding groups in the ECS Portal and assign permissions to them.

It is also possible to assign a *Default Group* for the bucket. The Default Group is the group assigned to the root (/) file system. For example, if the bucket owner is `hdfs` and the Default Group is set to `hadoop`, / is set to `hdfs:hadoop` as user and group, respectively. A Default Group is also a custom group and displays in the Custom Group ACL list.

If a Default Group is not defined, the root of the file system has no group as shown in the following example.

```
drwx---rwx+ - hdfs 0 2018-03-09 12:30 /
```

If a Default Group of `hadoop` is defined, the ownership and permissions display as shown in the following example.

```
drwxrwxrwx+ - hdfs hadoop 0 2018-03-09 12:28 /
```

These permissions are not inherited by directories created in the root.

If a Default Group is not assigned, the bucket owner (the owner of the root file system) can assign a group when accessing the HDFS from Hadoop, using the `hdfs dfs -chgrp` and `hdfs dfs -chmod` commands.

Hadoop superuser and supergroup

The superuser in a Hadoop environment is the user that starts the namenode, usually `hdfs` or `hdfs@REALM.COM`. In an ECS HDFS configuration, the superuser is the bucket owner. Therefore, if you want the Hadoop superuser to have superuser access to the ECS bucket, you should ensure that the bucket is owned by `hdfs`, or `hdfs@REALM.COM`, or `hdfs@DOMAIN.COM` if you are using Active Directory to authenticate users in the Hadoop environment.

To ensure that the Hadoop client has superuser access, you can also configure a superuser group using the `dfs.permissions.superusergroup` property in the `core-site.xml` file. In simple mode, the check to determine if a user is a member of the supergroup is made on the client by checking the value of the `dfs.permissions.supergroup` Hadoop property. In Kerberos mode, the check to determine if a user is a member of the supergroup is made on the ECS server.

In general, when buckets are configured for access by the Hadoop superuser or by a Hadoop superuser group, the superuser has full (read and write) access to the bucket. Users without superuser privileges normally have read access, but that depends on how the bucket was created. A user does not have to be an ECS object user to be granted access to the bucket. The name must match a Unix local, Kerberos, or AD user (depending on the authentication mode being used).

It is a best practice to ensure that the `hdfs` user or `hdfs` principal either be the bucket owner (superuser), or a member of a superuser group.

Multi-protocol (cross-head) access

ECS supports the ability to write data to a bucket using the S3 protocol and to make that data available as files through HDFS.

Multi-protocol access (also referred to as cross-head access) means objects written to the bucket using the S3 protocol can become the subject of Hadoop jobs, such as MapReduce. Similarly, directories and files written by HDFS can be read and modified using S3 clients.

In order that data written using S3 can be accessed as files, the bucket administrator can set a Default Group on the bucket and can set default permissions for files and directories owned by that group. This default Unix group is assigned to objects when they are created from S3 so that they have an owner and have group membership and group permissions that enable HDFS access from the Hadoop cluster.

Files created using HDFS and accessed using the S3 protocol are not affected by the default permissions as they are only applied to objects created using the S3 protocol.

Proxy user

ECS HDFS supports the use of the Hadoop proxy user.

A proxy user allows a Hadoop user to submit jobs or access HDFS on behalf of another user. The proxy user functionality can be compared to the UNIX/Linux *effective user* capabilities where running a command as one user assumes the identity of a different user as identified by the permission settings on the executable.

You configure proxy users for secure impersonation on a per-namespace (or per-bucket) basis. Proxy users are supported in simple and Kerberos mode. In either mode, the administrator can restrict proxy impersonations using the `hadoop.proxyuser.*.*` properties.

Equivalence user

ECS converts three part principals to two part principals.

A Kerberos principal is generally in the form `primary/instance@realm`, although the instance is not required, so `primary@realm` principal applies to all hosts in the realm. If the instance is specified, it may be used to designate a specific host, such as `joe/host1.company.com@COMPANY.COM` or `joe/host2.company.com@COMPANY.COM`. These two principals are for the same primary user (joe), but are targeted to only be granted authentication on the hosts (host1 or host2).

This type of user principal is recommended to provide an enhanced level of security. From an ECS perspective, each principal would have to be added to ECS. This becomes quite cumbersome, so the equivalence user feature allows ECS authorization to be performed by using a two-part principal (`primary@realm`), even if three-part principals are being used.

Migration from a simple to a Kerberos Hadoop cluster

ECS provides support for migrating from a simple Hadoop environment to a Hadoop environment secured by Kerberos.

When ECS HDFS is integrated with a Hadoop environment that uses simple security, files and directories created by Hadoop users, and processes, will be owned by non-secure users. If you subsequently migrate the Hadoop cluster to use Kerberos security, the files and directories written to ECS HDFS will no longer be accessible to those users.

ECS provides a built-in migration feature that enables you to provide ECS with a mapping between shortnames and Kerberos principals, so that files owned by non-secure shortnames will be accessible as the mapped Kerberos principal.

Where you only have a small number of files that have been written by shortname users, you might want to change them (using `chown`) to be owned by the Kerberos principal. However, where you have a large number of files, the migration feature means you do not have to change their ownership.

This feature is not implemented for buckets and you must change the bucket ACLs to allow access by the Kerberos principals if you are relying on access by users. However, if you use group membership as the primary means for enabling access, you do not have to change the bucket ACLs.

ECS allows the use of groups to simplify access to buckets, files, and directories. Groups always use UNIX simple names, so the group name associated with a bucket, file or directory is the same when accessing them from a simple or Kerberized cluster. When accessing from a simple environment, group membership is determined from the UNIX machine. When accessing from a Kerberized cluster you can configure group membership by assigning the mapping. Refer to [Map group names](#) on page 165 for information on mapping group names.

When using AD credentials, the mapping between AD principals and UNIX principals is achieved by removing the domain suffix, so user `hdfs@domain.com` becomes `hdfs`. This is not quite as flexible as when using Kerberos principal mapping which allow mappings such as `hdfs-xx@realm.com` to `hdfs`.

When using groups with AD, an authentication provider must have been configured in ECS so that membership of the group can be checked.

Hadoop Kerberos authentication mode

When Kerberos and the ECS AD server are integrated, the Kerberos realm provides a single namespace of users so that the Hadoop users authenticated with `kinit` are recognized as credentialed ECS users.

In a Hadoop cluster running in Kerberos mode, there must be a one-way cross-realm trust from the Kerberos realm to the AD realm used to authenticate ECS users.

The following identity translation properties in the `core-site.xml` file are used to ensure the proper Hadoop-to-ECS user translation:

- `fs.permissions.umask-mode`: Set the value to 022.
- `fs.viprfs.auth.anonymous_translation`: Set the value to `CURRENT_USER`.
- `fs.viprfs.auth.identity_translation`: Set the value to `CURRENT_USER_REALM` so the realm of users is auto-detected.

In addition, you must set the following properties in the `core-site.xml` file to define a service principal:

- `viprfs.security.principal: vipr/_HOST@REALM.COM` where `REALM.COM` is replaced by your Kerberos realm name.

File system interaction

When you are interacting directly with ECS HDFS, you might notice the following differences from interaction with the standard HDFS file system:

- Applications that expect the file system to be an instance of `DistributedFileSystem` do not work. Applications hardcoded to work against the built-in HDFS implementation require changes to use ECS HDFS.
- ECS HDFS does not support checksums of the data.
- When you use the `listCorruptFileBlocks` function, all blocks are reported as OK because ECS HDFS has no notion of corrupted blocks.
- The replication factor is always reported as a constant N, where $N=1$. The data is protected by the ECS SLA, not by Hadoop replication.

Supported Hadoop applications

ECS HDFS supports the majority of applications in the Hadoop ecosystem.

The following applications in the Hadoop ecosystem are supported:

- YARN
- MapReduce
- Pig
- Hive
- Spark
- Zookeeper
- Ambari
- Sqoop
- Flume

CHAPTER 7

Configure a simple Hadoop cluster with ECS HDFS

• Integrate a simple Hadoop cluster with ECS HDFS	146
• Install Hortonworks HDP using Ambari	146
• Create a bucket for HDFS using the ECS Portal	147
• Plan the ECS HDFS and Hadoop integration	154
• Obtain the ECS HDFS installation and support package	155
• Deploy the ECS HDFS Client Library	155
• Configure ECS client properties	156
• Set up Hive	157
• Verify Hadoop access to ECS	159
• Secure the bucket	159
• Relocate the default file system from HDFS to an ECS bucket	160

Integrate a simple Hadoop cluster with ECS HDFS

You can configure a Hadoop distribution to use the ECS storage infrastructure with ECS HDFS.

To perform this integration procedure, you must have:

- A working knowledge of your Hadoop distribution and its associated tools.
- The Hadoop credentials that allow you to log in to Hadoop nodes, to modify Hadoop system files, and to start and stop Hadoop services.

The following steps must be performed:

1. [Install Hortonworks HDP using Ambari](#) on page 146
2. [Create a bucket for HDFS using the ECS Portal](#) on page 147
3. [Plan the ECS HDFS and Hadoop integration](#) on page 154
4. [Obtain the ECS HDFS installation and support package](#) on page 155
5. [Deploy the ECS HDFS Client Library](#) on page 155 (Not required if you have used Ambari Hortonworks for ECS)
6. [Configure ECS client properties](#) on page 156
7. [Verify Hadoop access to ECS](#) on page 159
8. [Relocate the default file system from HDFS to an ECS bucket](#) on page 160

Once the configuration is complete, files in the default file system of the Hadoop cluster map to files in ECS buckets. For example, `/foo/bar` on the default file system maps to `viprfs://`

`<bucket_name>.<namespace>.<federation_name>/foo/bar`.

Install Hortonworks HDP using Ambari

Install the Ambari Server and use it install Hortonworks HDP.

The basic commands for installing and setting up the Ambari sever are provided in this procedure. For more information on how to install the Ambari server, see the [Hortonworks documentation](#).

Procedure

1. Download the Ambari repository.

```
wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.5.2.0/
ambari.repo -O /etc/yum.repos.d/ambari.repo
```

2. Install the Ambari server.

```
yum install -y ambari-server
```

3. Set up the Ambari server.

```
ambari-server setup -s
```

4. Start the Ambari server.

```
ambari-server start
```

5. Browse to <http://ambari.example.com:8080/>
6. On the **Select Stack** page, select the Hadoop version, HDP 2.6.2, and select the OS version.
7. Select the Hadoop services that you want to enable, as shown in the following example:

<input type="checkbox"/> Service	Version	Description
<input checked="" type="checkbox"/> HDFS	2.7.3	Apache Hadoop Distributed File System
<input checked="" type="checkbox"/> YARN + MapReduce2	2.7.3	Apache Hadoop NextGen MapReduce (YARN)
<input checked="" type="checkbox"/> Tez	0.7.0	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
<input checked="" type="checkbox"/> Hive	1.2.1000	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
<input checked="" type="checkbox"/> HBase	1.1.2	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.

8. Complete the installation wizard.

Create a bucket for HDFS using the ECS Portal

Use the ECS Portal to create a bucket configured for use with HDFS.

Before you begin

Ensure that you are assigned to the Namespace Administrator or a System Administrator role. If you are a Namespace Administrator you can create buckets in your namespace. If you are System Administrator you can create a bucket belonging to any namespace.

You must ensure that Hadoop users and services have access to the HDFS file system (the bucket), and that files and directories are accessible to the appropriate users and groups. You can do this in the following ways:

- Make the owner of the bucket the same as the Hadoop superuser, usually `hdfs` or `hdfs@REALM.COM`.
- Enable access to the bucket by group membership:
 - Assign a Default Group to the bucket. This automatically assigns Custom Group ACLs.
 - After bucket creation, add Custom Group ACLs for any other groups that need access.
- Enable access for individuals by adding User ACLs to the bucket.
- Ensure that the Hadoop users that need superuser access to the HDFS are part of the Hadoop supergroup.

If you want object data written to the bucket using object protocols to be accessible from HDFS, you should ensure that a default group is assigned to the bucket and that default file and directory permissions are set for the group.

For more information about users and permissions, see [Accessing the bucket as a file system](#) on page 140 and [Example Hadoop and ECS bucket permissions](#) on page 152.

Procedure

1. In the ECS Portal, select **Manage > Buckets > New Bucket**.
2. On the **New Bucket** page, at the **Name** field, enter a name for the bucket.

Note

Do not use underscores in bucket names as they are not supported by the URI Java class. For example, `viprfs://my_bucket.ns.site/` does not work as it is an invalid URI and is thus not understood by Hadoop.

3. In the **Namespace** field, select the namespace that the bucket will belong to.
4. In the **Replication Group** field, select a replication group or leave this field blank to use the default replication group for the namespace.
5. In the **Bucket Owner** field, type the name of the bucket owner.

For a HDFS bucket, the bucket owner will usually be `hdfs`, or `hdfs@REALM.COM` for Kerberos buckets. The Hadoop `hdfs` user requires superuser privileges on the HDFS; this can be achieved by making `hdfs` the owner of the bucket. Other Hadoop users may also require superuser privileges and these privileges are granted by assigning users to a group and making that group a superuser group.

6. Do not turn on **CAS**.
-

Note

A bucket that is intended for use as HDFS cannot be used for CAS. The **CAS** field is turned off when **File System** is turned on.

7. Turn on any other bucket features that you require.

You can turn on the following features for a HDFS bucket:

- Quota
- Server-side Encryption
- Metadata Search
- Access During Outage
- Compliance (see note)
- Bucket Retention

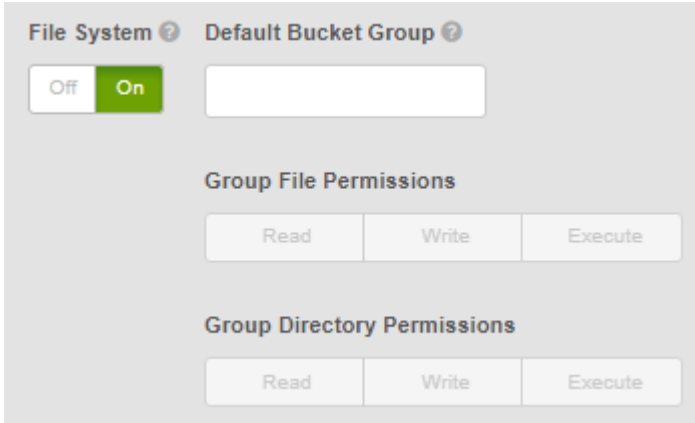
Refer to the *ECS Administration Guide* available from the [ECS Product Documentation page](#), for information on each of these settings and how to configure them.

Note

A bucket that is compliance-enabled cannot be written to using the HDFS protocol. However, data written using object protocols can be read from HDFS.

8. In the **File System** field, click **On**.

Once this feature is turned on, the fields for setting a default group for the file system/bucket and for assigning group permissions for files and directories created in the bucket are available. These fields are shown in the following example.



The screenshot shows a configuration interface for HDFS. At the top, there are two tabs: 'File System' and 'Default Bucket Group'. Under 'File System', there are 'Off' and 'On' buttons, with 'On' being selected. To the right of the 'On' button is a text input field. Below this, there are two sections: 'Group File Permissions' and 'Group Directory Permissions'. Each section contains three buttons: 'Read', 'Write', and 'Execute'.

9. In the **Default Bucket Group** field, type a name for the default bucket group.

This group is the group associated with the HDFS root file system and enables Hadoop users who are members of the group to access the HDFS.

The default group could be a group, such as `hdfs` or `hadoop` to which the services that you need to access the data on the HDFS belong, but it can be whatever group name makes sense for your Hadoop configuration. For example, the administrator might want all S3 files uploaded to the bucket to be assigned to group `S3DataUsers`. All S3 files will then have this group assigned to them. On the Hadoop node, the Hadoop administrator will have users that are members of the group `S3DataUsers`. `S3DataUsers` can be a Linux group, or an AD group. When the Hadoop users want to access the S3 data, they can do so because the files were uploaded and assigned to that group.

You must specify the default group at bucket creation. If you do not, the group must be assigned later from Hadoop by the file system owner.

10. In the **Group File Permissions** and **Group Directory Permissions** fields, set the default permissions for files and directories created in the bucket using the object protocols.

You use these settings to apply UNIX group permissions to objects created using object protocols. These permissions apply to the HDFS group (the Default Bucket Group) when the object or directory is listed from Hadoop. For more information on setting the Default Group and permissions for the file system, see [Multi-protocol \(cross-head\) access](#) on page 142.

- a. In the **Group File Permissions** field, select the appropriate permission buttons. You normally set **Read** and **Execute** permissions.

- b. In the **Group Directory Permissions** field, select the appropriate permission buttons. You normally set **Read** and **Execute** permissions.

11. Click **Save** to create the bucket.

Set custom group bucket ACLs

You can set a group ACL for a bucket in the ECS Portal and you can set bucket ACLs for a group of users (Custom Group ACL), for individual users, or a combination of both. For example, you can grant full bucket access to a group of users, but you can also restrict (or even deny) bucket access to individual users in that group.

Before you begin

- This operation requires the System Administrator or Namespace Administrator role in ECS.
- A System Administrator can edit the group ACL settings for a bucket in any namespace.
- A Namespace Administrator can edit the group ACL settings for a bucket in the namespace in which they are the administrator.

Custom group ACLs enable groups to be defined and for permissions to be assigned to the group. The main use case for assigning groups to a bucket is to support access to the bucket as a file system, for example, when making the bucket available for NFS or HDFS.

Members of the UNIX group can access the bucket when it is accessed as a file system (using NFS or HDFS).

Procedure

1. In the ECS Portal, select **Manage > Buckets**.
2. On the **Bucket Management** page, locate the bucket you want to edit in the table and select the **Edit ACL** action.
3. Click the **Custom Group User ACLs** tab to set the ACL for a custom group.
4. Click **Add**.

The **Edit Custom Group** page displays, as shown in the following screenshot.

5. On the **Edit Custom Group** page, in the **Custom Group Name** field, type the name for the group.
This name can be a Unix/Linux group, or an Active Directory group.
6. Select the permissions for the group.
At a minimum you should assign **Read**, **Write**, **Execute** and **Read ACL**.
7. Click **Save**.

Set user bucket ACLs

You can set a user ACL for a bucket in the ECS Portal. ECS assigns permissions automatically to the bucket owner. You can assign user ACLs to other Hadoop users to enable access to the bucket/file system or alternatively they can gain access to the bucket by being a member of group that has been assigned Custom Group ACLs.

Before you begin

- You must be an ECS Namespace Administrator or a System Administrator edit the ACL for a bucket.
- If you are a Namespace Administrator you can edit the ACL settings for buckets belonging to your namespace.
- If you are System Admin you can edit the ACL settings for a bucket belonging to any namespace.

Procedure

1. In the ECS Portal, select **Manage > Buckets**.
2. On the **Bucket Management** page, locate the bucket you want to edit in the table and select the **Edit ACL** action.

3. On the **Bucket ACLs Management** page, ensure the **User ACLs** tab is selected; this is the default.
4. On the **User ACLs** tab, you can edit the permissions for a user that already has assigned permissions, or you can add a user that you want to assign permissions for.
 - To set (or remove) the ACL permissions for a user that already has permissions, select **Edit** (or Remove) from the Action column in the ACL table.
 - To add a user for whom you want to assign permissions, click **Add** and type the username of the user that the permissions will apply to. Specify the permissions that you want to apply to the user.

The user who you have set as the bucket owner already has default permissions assigned.

The bucket in the following example is owned by the `hdfs` user and `hdfs`, as the owner, has been given full control. Full control translates to read, write, and execute permissions in a Hadoop/UNIX environment. The user `sally` has been give read and execute permissions to access the bucket.

User	Read	Read ACL	Write	Write ACL	Execute	Full Control	Privileged Write	Delete	None	Actions
hdfs						✓				Edit
sally	✓				✓					Edit

For more information on ACL privileges, see the *ECS Administration Guide* available from the [ECS Product Documentation page](#).

5. Click **Save**.

Example Hadoop and ECS bucket permissions

Examples are provided in this topic to demonstrate the relationship between Hadoop users/groups and the users/groups that are assigned permission to access the bucket using ECS User ACLs and Custom Group ACLs.

When a bucket is created, ECS automatically assigns ACLs to the bucket owner and to the default group, which is the group assignment for the bucket when accessed using HDFS. A bucket must always have an owner, however, a bucket does not require an assigned default group. Users and groups other than the bucket owner, that is, Custom Groups, can be assigned ACLs on the bucket. ACLs assigned in this way translate to permissions for Hadoop users.

Table 24 Example bucket permissions for file system access in a simple Hadoop cluster

Hadoop users and groups	Bucket permissions	Bucket/file system access
Bucket access using Group ACL		
Users (service) hdfs. mapred, yarn, hive, pig Users (applications) sally, fred Groups hdfs (hdfs) hadoop (hdfs, mapred, yarn, hive, pig) users (sally, fred) Supergroup hdfs	Bucket owner hdfs Default Group Custom Group ACL hadoop, users, hive, spark (Full Control) User ACL hdfs (owner)	Custom Group ACLs must be set on the bucket in the ECS Portal assign Full Control on the bucket/root file system to the <code>hadoop,users,hive, and spark</code> groups. This example assumes that <code>hdfs</code> is the superuser - the user that started the namenode.
Bucket created by s3 user - crosshead access		
Users (service) hdfs. mapred, yarn, hive, pig Users (applications) sally, fred Groups hdfs (hdfs) hadoop (hdfs, mapred, yarn, hive, pig) users (sally, fred) Supergroup hdfs	Bucket owner s3user Default Group hadoop (Group File Permissions: Read, Write Group Directory Permissions: Read, Write, Execute) Custom Group ACL hadoop (default) User ACL s3user (owner), sally, fred	Where you want objects written by an S3 user to be accessible as files from HDFS, a default group must be defined (<code>hadoop</code>) so that Hadoop users and services have permissions on the files due to group membership. The default group automatically has Custom Group ACLs on the bucket/file system. The following example shows that <code>hadoop</code> has been set default group and the root file system permissions are 777: <pre>drwxrwxrwx+ - s3user hadoop 0 2018-03-09 12:28 /</pre> You can give users access either by adding User ACLs or by adding Custom Group ACLs for the group to which the users belong.

Table 25 Example bucket permissions for file system access in a Kerberized Hadoop cluster

Hadoop user	Bucket permissions	Bucket/file system access
Users (service) hdfs@REALM.COM. mapred@REALM.COM, yarn@REALM.COM, hive@REALM.COM, pig@REALM.COM	Bucket owner hdfs@REALM.COM Default Group hadoop	Custom Group ACLs set on the bucket in the Portal enable the <code>hadoop</code> and <code>users</code> group to have permissions on the bucket/root file system. User information from the Hadoop cluster must be available to ECS so that it can

Table 25 Example bucket permissions for file system access in a Kerberized Hadoop cluster

Hadoop user	Bucket permissions	Bucket/file system access
Users (applications) sally@REALM.COM, fred@REALM.COM, ambari- qa@REALM.COM Groups hdfs (hdfs@REALM.COM) hadoop (hdfs@REALM.COM, mapred@REALM.COM, yarn@REALM.COM, hive@REALM.COM, pig@REALM.COM) users (sally@REALM.COM, fred@REALM.COM) Supergroup hdfs	Custom Group ACL hadoop (default), users User ACL hdfs@REAL.COM (owner)	provide secure access to the bucket. This information is provided using bucket metadata and an example metadata file is provided in Secure bucket metadata on page 200.

Plan the ECS HDFS and Hadoop integration

Use the following table to verify that you have the information necessary to ensure a successful integration.

Table 26 ECS HDFS configuration prerequisites

Element	What to do
Hadoop cluster	Verify the cluster is installed and operational. Record the admin credentials for use later in this procedure.
ECS cluster:ECS nodes	Record the ECS node IP addresses for use later in this procedure.
ECS cluster: bucket	HDFS requires that a bucket enabled for HDFS is created within an ECS replication group. The bucket is accessed as a file system using the namespace and bucket name. Record the name of the bucket.
ECS cluster: tenant namespace	Verify a tenant namespace is configured. Record the name.

Obtain the ECS HDFS installation and support package

The ECS HDFS Client Library, and HDFS support tools are provided in a HDFS Client ZIP file, `hdfsclient-<ECS version>-<version>.zip`, that you can download from the ECS support pages on support.emc.com.

The ZIP file contains `/playbooks` and `/client` directories. Before you unzip the file, create a directory to hold the zip contents (your unzip tool might do this for you), then extract the contents to that directory. After you extract the files, the directories will contain the following:

- `/playbooks`: Contains Ansible playbooks for configuring a secure Hadoop environment to talk to ECS HDFS.
- `/client`: Contains the following files:
 - ECS Client Library (ViPPRFS) JAR files (`viprfs-client-<ECS version>-hadoop-<Hadoop version>.jar`): Used to configure different Hadoop distributions.

Deploy the ECS HDFS Client Library

Use this procedure to put the ECS HDFS Client Library JAR on the classpath of each client node in the Hadoop cluster.

Before you begin

Obtain the ECS HDFS Client Library for your Hadoop distribution from the ECS support page as described in [Obtain the ECS HDFS installation and support package](#) on page 155.

The HDFS Client Library uses the following naming convention `viprfs-client-<ECS version>-hadoop-<Hadoop version>.jar` and the JAR file for use with this release is listed in the following table.

Table 27 ECS HDFS Client Library

Hadoop distribution	Version	ECS HDFS JAR
Hortonworks	HDP 2.6.2	<code>viprfs-client-<ECS version>-hadoop-2.7.jar</code>

Note

- When you upgrade to a later version of ECS, you must deploy the ECS HDFS Client Library for the release to which you have upgraded.

Procedure

1. Log in to a node that has password-less SSH access to all Hadoop nodes.
2. Run the classpath command to get the list of directories in the classpath:


```
# hadoop classpath
```
3. Deploy the client JAR file to all Hadoop nodes by performing the following steps:

- a. Create a text file named `masters` that contains a list of IP addresses or FQDNs for all Hadoop master nodes, one per line.
- b. Create a text file named `workers` that contains a list of IP addresses or FQDNs for all Hadoop worker nodes, one per line.
- c. Create the directory `/usr/lib/hadoop/lib` on all nodes. Use the following command:

```
# cat masters workers | xargs -i -n 1 ssh root@{} mkdir -p /usr/lib/hadoop/lib
```

- d. Copy the ECS client jar to all nodes using the following command:

```
cat masters workers | xargs -i -n 1 scp viprfs-client-3.2.0.0-hadoop-2.7.jar root@{}:/usr/lib/hadoop/lib/
```

Configure ECS client properties

You can use Ambari to set the following configuration properties that are required by the ECS client.

For more information on the `core-site.xml` parameters, see [Hadoop core-site.xml properties for ECS HDFS](#) on page 194.

Table 28 Hadoop configuration to enable ECS access

Hadoop location	Property	Value
core-site	fs.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRFileSystem
	fs.AbstractFileSystem.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem
	fs.viprfs.auth.identity_translation	NONE
	fs.viprfs.auth.anonymous_translation	LOCAL_USER
	fs.vipr.installations	Can be any name, such as <i>federation</i> and will be referred to as \$FEDERATION. If you have multiple independent ECS federations, enter multiple values separated by commas.
	fs.vipr.installation.\$FEDERATION.hosts	Comma-separated list of FQDN or IP address of each ECS host in the local site
	fs.vipr.installation.\$FEDERATION.hosts.resolution	dynamic
	fs.vipr.installation.\$FEDERATION.resolution.dynamic.time_to_live_ms	900000
hdfs-site	fs.permissions.umask-mode	022

Table 28 Hadoop configuration to enable ECS access (continued)

Hadoop location	Property	Value
yarn-site	yarn.application.classpath	Append the following: <code>/usr/lib/hadoop/lib/*</code>
mapred-site	mapreduce.application.classpath	Append the following: <code>/usr/lib/hadoop/lib/*</code>
tez-site	tez.cluster.additional.classpath.prefix	Append the following: <code>/usr/lib/hadoop/lib/*</code>
HDFS	hadoop-env template	Append the following: <code>export HADOOP_CLASSPATH=\${HADOOP_CLASSPATH}:/usr/lib/hadoop/lib/*</code>
Spark	spark-env template	Append the following: <code>export SPARK_DIST_CLASSPATH="\${SPARK_DIST_CLASSPATH}:/usr/lib/hadoop/lib/*:/usr/hdp/current/hadoop-client/client/guava.jar"</code>

Set up Hive

The additional steps provided in this procedure are required to configure Hive.

Before you begin

When using Hive, you should also ensure that the Hive metastore warehouse is being directed to the ViPRFS location. Assuming that mysql is being used to identify the Hive metastore location, start mysql, go to the Hive databases, and show the contents of the DBS table and set it as below.

Procedure

1. If Hive is using templeton, you should modify the following properties, and these properties are already defined.

Table 29 Hive templeton configuration

Hadoop location	Property	Value (example)
Advanced webhcat-site	templeton.hive.archive	viprfs://hdfsBucket2.s3.site1/hdp/apps/\${hdp.version}/hive/hive.tar.gz
	templeton.pig.archive	viprfs://hdfsBucket2.s3.site1/hdp/apps/\${hdp.version}/pig/pig.tar.gz
	templeton.sqoop.archive	viprfs://hdfsBucket2.s3.site1/hdp/apps/\${hdp.version}/sqoop/sqoop.tar.gz
	templeton.streaming.jar	viprfs://hdfsBucket2.s3.site1/hdp/apps/\${hdp.version}/mapreduce/hadoop-streaming.jar

2. Start mysql.

```
[root@hdfs-pansy2 lib]# mysql -u hive -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

3. Go to the Hive database.

```
mysql> use hive;
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with -A
```

4. Show the contents of the database.

```
select * from DBS;
+-----+-----+-----+-----+-----+-----+
| DB_ID | DESC          | DB_LOCATION_URI          | NAME   | OWNER | OWNER |
|       |               |                          |       | _NAME | _TYPE |
+-----+-----+-----+-----+-----+-----+
| 1     | Default Hive | hdf://hdfs-pansyl.ecs.lab.emc. | default | public | ROLE  |
|       | database     | com:8020/apps/hive/warehouse |        |        |       |
| 6     | NULL         | viprfs://hdfsbucket.ns.Site1/ | retail | hdfs   | USER  |
|       |               | apps/hive/warehouse/retail_demo.db | _demo |        |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

5. Change the database.

```
mysql> update DBS set DB_LOCATION_URI='viprfs://hdfsbucket3.ns.Site1/apps/hive/
warehouse' where DB_ID=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Verify Hadoop access to ECS

You must verify access to the ECS bucket.

Once all Hadoop client services have started, ensure that you can access the ECS bucket using the Hadoop CLI. The URI is of the form `viprfs://bucket.namespace.federation/`.

For a bucket with URI `viprfs://hive-warehouse-1.ns1.federation1/`, you can attempt a directory listing using:

```
[root@mycluster1-master-0 ~]# hdfs dfs -ls viprfs://hive-warehouse-1.ns1.federation1/
```

A new bucket will be empty and nothing will be returned.

For the same bucket, the following commands create an empty file and then perform a directory listing that shows the file.

```
[root@mycluster1-master-0 ~]# hdfs dfs -touchz viprfs://hive-warehouse-1.ns1.federation1/hive-warehouse-1
[root@mycluster1-master-0 ~]# hdfs dfs -ls viprfs://hive-warehouse-1.ns1.federation1/
```

Secure the bucket

In addition to configuring a bucket ACL, the root directory entry should be created and secured immediately after bucket creation.

Before you begin

This procedure should be performed as the bucket owner, which is `hdfs` in this example.

Procedure

1. Set the mode bits in the root directory object ACL so that only the bucket owner and the default group have access to the bucket. The `other` group, which includes all ECS HDFS client users, is not allowed access the root directory, and therefore it is not allowed access to any files in the bucket.

```
[hdfs@hadoop-0 ~]$
fs=viprfs://bucket.ns.fed
hadoop fs -chmod 750 $fs/
hadoop fs -chown hdfs:hdfs $fs/
```

2. Specific groups and users should be added to the root directory object ACL using the `setfacl` command.

Note that these permissions duplicate the bucket's Custom Group ACLs to ensure that all HDFS APIs have the same effective permissions.

```
hadoop fs -setfacl -m group:hadoop:r-x $fs/
hadoop fs -setfacl -m group:users:r-x $fs/
hadoop fs -setfacl -m group:hive:r-x $fs/
hadoop fs -setfacl -m group:spark:r-x $fs/
```

3. Confirm the permissions.

```
hadoop fs -ls -d $fs/
drwxr-x---+ - hdfs hdfs 0 2017-08-22 20:44 viprfs://
bucket.ns.fed/
```

```
hadoop fs -getfacl $fs/
# file: viprfs://bucket.ns.fed/
# owner: hdfs
# group: hdfs
user::rwx
group::r-x
group:hadoop:r-x
group:hive:r-x
group:spark:r-x
group:users:r-x
mask::r-x
other:----
```

Relocate the default file system from HDFS to an ECS bucket

Although the system is now usable and may appear to work well, a configuration with HDFS as the default file system is not supported. You must therefore relocate the default file system from HDFS to the root ECS bucket. This procedure copies all files from the HDFS file system to an ECS bucket and then sets the ECS bucket as the default file system.

Procedure

- 1. Use Ambari to stop all services except HDFS, YARN, and Zookeeper.
- 2. Copy all existing files on the DAS HDFS file system to the ECS bucket. Even for a new installation of Hadoop, there are critical directories that must exist in the default Hadoop file system. Use `DistCp` to perform the file copy.

```
[hdfs@mycluster1-master-0~]$ hadoop distcp -skipcrccheck -update -pugp -i / viprfs://
mycluster1-root.ns1.federation/
```

3. Use Ambari to configure the following settings.

Table 30 Hadoop configuration to enable Hive concurrency and ACID transactions

Hadoop location	Property	Value (example)
HDFS Advanced core-site	fs.defaultFS	<code>viprfs://<bucket_name>.<namespace>.<federation_name></code> For example: <code>viprfs://mycluster1-root.ns1.federation1</code>

Table 30 Hadoop configuration to enable Hive concurrency and ACID transactions (continued)

Hadoop location	Property	Value (example)
Spark Advanced spark-defaults	spark.eventLog.dir	viprfs:// <bucket_name>.<namespace>.<federation>/<spark-history> For example: viprfs://mycluster1- root.ns1.federation1/spark- history
Spark Advanced spark-defaults	spark.history.fs.logDirectory	viprfs:// <bucket_name>.<namespace>.<federation>/<spark-history> For example: viprfs://mycluster1- root.ns1.federation1/spark- history

4. Use Ambari to stop and start all services.
5. Ensure proper directory permissions. If `DistCp` encounters any errors, the necessary permissions may not have been applied to critical directories. The following commands set the correct permissions.

```
[hdfs@mycluster1-master-0~]$
hadoop fs -chmod 777 /apps/hive/warehouse
hadoop fs -chown hive:hdfs /apps/hive/warehouse
hadoop fs -chmod -R 770 /user/ambari-qa
hadoop fs -chown -R ambari-qa:hdfs /user/ambari-qa
```

Configure a simple Hadoop cluster with ECS HDFS

CHAPTER 8

Configure a Kerberized Hadoop cluster with ECS HDFS

• Integrate a secure Hadoop cluster with ECS HDFS	164
• Plan migration from a simple to a Kerberos cluster	164
• Map group names	165
• Configure ECS nodes with the ECS service principal	165
• Enable Kerberos using Ambari	169
• Secure the ECS bucket using metadata	170
• Reconfigure ECS client properties	173
• Start Hadoop services and verify Hadoop access to ECS	174

Integrate a secure Hadoop cluster with ECS HDFS

You can integrate your existing Hadoop distribution, that is secured using Kerberos, with ECS HDFS.

You must perform a non-secure installation of Hadoop and ECS completely before optionally enabling Kerberos.

Before performing the integration steps, you must do the following:

- Verify that a Kerberos Key Distribution Center (KDC) is installed and configured to handle authentication of the Hadoop service principals. If you are using Active Directory to authenticate ECS users, you must set up a cross-realm trust between the Kerberos realm and the ECS user realm. For information on setting up the Kerberos KDC and configuring trust, see [Guidance on Kerberos configuration](#) on page 188.
- Ensure that you have created a bucket for the HDFS file system (see [Create a bucket for HDFS using the ECS Portal](#) on page 147).
- Ensure that you have read the guidelines for planning the integration (see [Plan the ECS HDFS and Hadoop integration](#) on page 154).
- Ensure that you have downloaded the installation and support package (see [Obtain the ECS HDFS installation and support package](#) on page 155).

To integrate ECS HDFS with your secure Hadoop cluster, complete the following tasks:

1. [Plan migration from a simple to a Kerberos cluster](#) on page 164
2. [Map group names](#) on page 165
3. [Configure ECS nodes with the ECS service principal](#) on page 165
4. [Secure the ECS bucket using metadata](#) on page 170
5. [Reconfigure ECS client properties](#) on page 173
6. [Start Hadoop services and verify Hadoop access to ECS](#) on page 174

Plan migration from a simple to a Kerberos cluster

ECS supports migration from a Hadoop cluster that uses simple security to a Hadoop cluster secured by Kerberos.

If you are migrating from a simple to a secure environment, you should refer to [Migration from a simple to a Kerberos Hadoop cluster](#) on page 143.

In general, the ECS migration feature enables files and directories to be accessible seamlessly by Kerberos users. However, the following notes apply:

- The Ambari wizard to secure a Hadoop cluster reports errors in the final step when it starts the Hadoop services. This is expected behavior. Once the ECS bucket is reconfigured to be secure, you can start the Hadoop services.
- For users and processes to be able to access the bucket, they must be members of the group that has access to the bucket. Otherwise, you must change the bucket ACLs so that Kerberos users have access.

Map group names

ECS must be able to map group details for Hadoop service principals like `hdfs`, `hive`, and so on. If you are using Active Directory (AD), group information can be found from two different sources: the bucket metadata or AD. ECS determines which source to use from a configuration parameter setting in the `/opt/storageos/conf/hdfssvc.conf` configuration file in the `[hdfs.fs.request]` section.

If you want ECS to use bucket metadata for group information (if available) instead of AD, define the parameter as follows:

```
[hdfs.fs.request]
prefer_secure_metadata_bucket_for_groups = true
```

If you want ECS to determine group information from AD instead of bucket metadata, define the parameter as follows:

```
[hdfs.fs.request]
prefer_secure_metadata_bucket_for_groups = false
```

The default value is `true`, so if this value is not defined, ECS determines group details for a Kerberos principal from the bucket metadata. You must apply any change to all ECS nodes and you must restart `dataheadsvc` on all nodes.

Configure ECS nodes with the ECS service principal

The ECS service principal and its corresponding keytab file must reside on each ECS data node. You must use the Ansible playbooks provided to automate these steps.

Before you begin

You must have the following items before you can complete this procedure:

- Access to the Ansible playbooks. Obtain the Ansible playbooks from the ECS HDFS software package as described in [Obtain the ECS HDFS installation and support package](#) on page 155.
- The list of ECS node IP addresses.
- IP address of the KDC.
- The DNS resolution where you run this script should be the same as the DNS resolution for the Hadoop host, otherwise the `vipr/_HOST@REALM` will not work.

ECS provides reusable Ansible content called 'roles', which consist of Python scripts, YAML-based task lists, and template files.

- `vipr_kerberos_config`: Configures an ECS node for Kerberos.
- `vipr_jce_config`: Configures an ECS data node for unlimited-strength encryption by installing JCE policy files.
- `vipr_kerberos_principal`: Acquires a service principal for an ECS node.

In this procedure, Ansible is run using the utility Docker container that is installed with ECS.

Procedure

1. Log in to ECS Node 1 and copy the `hdfsclient-<ECS version>-<version>.zip` file to that node.

For example: `/home/admin/ansible` . You can use `wget` to obtain the package directly from `support.emc.com` or you can use `scp` if you have downloaded it to another machine.

2. Unzip the `hdfsclient-<ECS version>-<version>.zip` file.

The steps in this procedure use the playbooks contained in the `viprfs-client-<ECS version>-<version>/playbooks/samples` directory and the steps are also contained in `viprfs-client-<ECS version>-<version>/playbooks/samples/README.md`.

3. Edit the `inventory.txt` file in the `playbooks/samples` directory to refer to the ECS data nodes and the KDC server.

The default entries are shown below.

```
[data_nodes]
192.168.2.[100:200]

[kdc]
192.168.2.10
```

4. Download the *unlimited JCE* policy archive from `oracle.com`, and extract it to an `UnlimitedJCEPolicy` directory in `viprfs-client-<ECS version>-<version>/playbooks/samples`.

Note

You should only perform this step if you are using strong encryption type.

You can configure Kerberos to use a strong encryption type, such as AES-256. In that case, you must reconfigure the JRE within the ECS nodes to use the policy.

5. Start the utility container on ECS Node 1 and make the Ansible playbooks available to the container.
 - a. Load the utility container image.

For example:

```
sudo docker load -i /opt/emc/caspian/checker/docker/images/utilities.tgz
```

- b. Get the identity of the docker image.

For example:

```
admin@provo-lilac:~> sudo docker images
```

The output will give you the image identity:

REPOSITORY	TAG	IMAGE ID
CREATED	VIRTUAL SIZE	

utilities ago	738.5 MB	1.5.0.0-403.cb6738e	186bd8577a7a	2 weeks
------------------	----------	---------------------	--------------	---------

c. Start and enter utilities image.

For example:

```
sudo docker run -v /opt/emc/caspian/fabric/agent/services/object/main/log:/opt/
storageos/logs
-v /home/admin/ansible/viprfs-client-3.0.0.0.85325.a05145b/playbooks:/ansible --
name=ecs-tools -i -t --privileged --net=host 186bd8577a7a /bin/bash
```

In the example, the location to which the Ansible playbooks were unzipped /
home/admin/ansible/viprfs-client-3.0.0.0.85325.a05145b/
playbooks is mapped to the /ansible directory in the utility container.

6. Change to the working directory in the container.

For example:

```
cd /ansible
```

7. Copy the krb5.conf file from the KDC to the working directory.

8. Install the supplied Ansible roles.

```
ansible-galaxy install -r requirements.txt -f
```

9. Edit the generate-vipr-keytabs.yml as necessary and set the domain name.

For example:

```
[root@nile3-vm22 samples]# cat generate-vipr-keytabs.yml
---
###
# Generates keytabs for ViPR/ECS data nodes.
###

- hosts: data_nodes
  serial: 1

  roles:
    - role: vipr_kerberos_principal
      kdc: "{{ groups.kdc | first }}"
      principals:
        - name: vipr/_HOST@MA.EMC.COM
          keytab: keytabs/_HOST@MA.EMC.COM.keytab
```

In this example, the default value (vipr/_HOST@EXAMPLE.COM) has been replaced with (vipr/_HOST@MA.EMC.COM) and the domain is MA.EMC.COM.

10. Run the following command.

```
export ANSIBLE_HOST_KEY_CHECKING=False
```

11. Run the Ansible playbook to generate keytabs.

```
ansible-playbook -v -k -i inventory.txt --user admin -b --become-user=root generate-vipr-keytabs.yml
```

12. Edit the `setup-vipr-kerberos.yml` file as necessary.

The default file contents are shown below.

```
# cat setup-vipr-kerberos.yml

---
###
# Configures ViPR/ECS for Kerberos authentication.
# - Configures krb5 client
# - Installs keytabs
# - Installs JCE policy
###

- hosts: data_nodes

  roles:
    - role: vipr_kerberos_config
      krb5:
        config_file: krb5.conf
        service_principal:
          name: vipr/_HOST@EXAMPLE.COM
          keytab: keytabs/_HOST@EXAMPLE.COM.keytab

    - role: vipr_jce_config
      jce_policy:
        name: unlimited
        src: UnlimitedJCEPolicy/
```

In this example, the default value (`vipr/_HOST@EXAMPLE.COM`) has been replaced with (`vipr/_HOST@MA.EMC.COM`) and the domain is `MA.EMC.COM`.

Note

You must remove the `vipr_jce_config` role if you are not using strong encryption type.

13. Run the Ansible playbook to configure the data nodes with the ECS service principal.

Make sure the `/ansible/samples/keytab` directory exists and the `krb5.conf` file is in the working directory `/ansible/samples`.

```
ansible-playbook -v -k -i inventory.txt --user admin -b --become-user=root setup-vipr-kerberos.yml
```


Verify that the correct ECS service principal, one per data node, has been created (from the KDC):

```
# kadmin.local -q "list_principals" | grep vipr
vipr/nile3-vm42.centera.lab.emc.com@MA.EMC.COM
vipr/nile3-vm43.centera.lab.emc.com@MA.EMC.COM
```

Verify that the correct keytab is generated and stored in the location: `/data/hdfs/krb5.keytab` on all ECS data nodes. You can use the `strings` command on the keytab to extract the human readable text, and verify that it contains the correct principal. For example:

```
dataservice-10-247-199-69:~ # strings /data/hdfs/krb5.keytab
MA.EMC.COM
vipr
nile3-vm42.centera.lab.emc.com
```

In this case the principal is `vipr/nile3-vm42.centera.lab.emc.com`.

Enable Kerberos using Ambari

You must enable Kerberos using Ambari.

This procedure provides the basic steps you must perform to enable Kerberos. For more information on the Ambari Kerberos Wizard, see [here](#).

Procedure

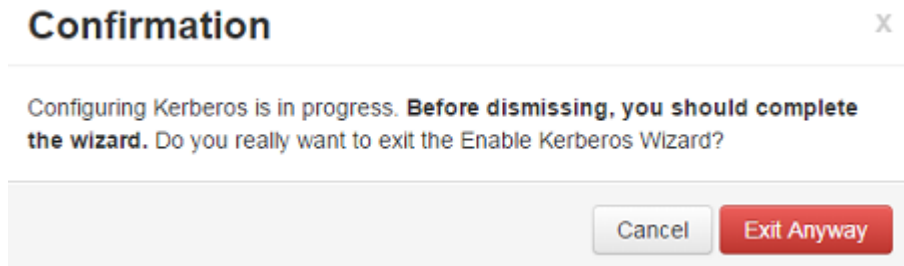
1. In the Ambari interface, select **Ambari > Admin > Kerberos > Enable Kerberos** to get to the **Enable Kerberos Wizard**.
2. Follow the steps in the wizard **Kerberize Cluster** panel. Click the X to abort the Kerberos wizard.

Enable Kerberos Wizard



This Kerberize Cluster step can be skipped now since the Hadoop services cannot be started at this point.

3. Confirm that you want to exit the wizard by clicking **Exit Anyway** in the **Confirmation** dialog.



Secure the ECS bucket using metadata

To ensure that the ECS bucket can work with a secure Hadoop cluster, the bucket must have access to information about the cluster.

In a secure Hadoop cluster, the Kerberos principal must be mapped to a HDFS username. In addition, the user must be mapped to a UNIX group. Within the Hadoop cluster, the NameNode gathers this information from the Hadoop nodes themselves and from the configuration files (`core-site.xml` and `hdfs.xml`).

To enable the ECS nodes to determine this information and to validate client requests, the following data must be made available to the ECS nodes:

- Kerberos user to UNIX user and group mapping
- Superuser group
- Proxy user settings

The data is made available to the ECS nodes as a set of name-value pairs held as metadata.

Kerberos users

Information about every Kerberos user (not AD users) that requires Hadoop access to a bucket must be uploaded to ECS. The following data is required:

- Principal name
- Principal shortname (mapped name)
- Principal groups

If there are 10 Kerberos principals on a Hadoop node, you must create 30 name value pairs in the JSON input file. Every name must be unique, so you will must uniquely assign a name for every principal name, principal shortname, and principal group. ECS expects a constant prefix and suffix for the JSON entry names.

The required prefix for every Kerberos user entry is `internal.kerberos.user`, and the three possible suffixes are `name`, `shortname` and `groups`. As shown in the following example.

```
{
  {
    "name": "internal.kerberos.user.hdfs.name",
    "value": "hdfs-cluster999@EXAMPLE_HDFS.EMC.COM"
  },
  {
    "name": "internal.kerberos.user.hdfs.shortname",
    "value": "hdfs"
  },
  {
    "name": "internal.kerberos.user.hdfs.groups",
    "value": "hadoop,hdfs"
  },
}
```

The value between the prefix and suffix can be anything, as long as it uniquely identifies the entry. For example, you could use:

```
"name": "internal.kerberos.user.1.name",
"name": "internal.kerberos.user.1.shortname",
"name": "internal.kerberos.user.1.groups",
```

Principals can map to a different users. For example, the `rm` principal user is usually mapped to the `yarn` users using `auth_to_local` setting for the Hadoop cluster, like this.

```
RULE: [2:$1@$0] (rm@EXAMPLE_HDFS.EMC.COM) s/./yarn/
```

So for any principal that maps to a different principal (for example, the `rm` principal maps to the `yarn` principal), you must use the mapped principal in the `shortname` value, so the entry for the `rm` principal would be:

```
{
  "name": "internal.kerberos.user.rm.name",
  "value": "rm@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.yarn.shortname",
  "value": "yarn@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.yarn.groups",
  "value": "hadoop"
},
```

Supergroup

You must tell ECS which Linux group of users on the Hadoop nodes get superuser privileges based on their group. Only one entry in the JSON input file is expected for the supergroup designation. It must be like the following:

```
{
  "name": "dfs.permissions.supergroup",
  "value": "hdfs"
}
```

Proxy settings

For proxy support, you must identify all proxy settings that are allowed for each Hadoop application, where application means one of the Hadoop-supported applications, for example, hive, and so on.

In the following example, proxy support for the hive application is granted to users who are members of the `s3users` group (AD or Linux group), and can run hive on any of the hosts in the Hadoop cluster. So the JSON entry for this is two name/value pairs, one for the hosts setting, and one for the groups setting.

```
{
  "name": "hadoop.proxyuser.hive.hosts",
  "value": "*"
},
{
```

```

    "name": "hadoop.proxyuser.hive.groups",
    "value": "s3users"
  }

```

The complete file

The three types of metadata must be combined into a single JSON file. The JSON file format is as shown in the following example.

```

{
  "head_type": "hdfs",
  "metadata": [
    {
      "name": "METADATANAME_1",
      "value": "METADATAVALUE_1"
    },
    {
      "name": "METADATANAME_2",
      "value": "METADATAVALUE_2"
    },
    :
    {
      "name": "METADATANAME_N",
      "value": "METADATAVALUE_N"
    }
  ]
}

```

Note

The last name/value pair does not have a trailing “,” character.

An example of a JSON file is shown in: [Secure bucket metadata](#) on page 200.

Secure and non-secure buckets

Once metadata is loaded into a bucket, it is referred to as a *secure bucket* and you must have Kerberos principals to access it. A request from a non-secure Hadoop node is rejected. If metadata is not loaded, the bucket is not secure and a request from a secure Hadoop node is rejected.

The following error is seen if you try and access a secure bucket from a non-secure cluster. A similar message is seen if you try and access a non-secure bucket from a secure cluster.

```

[hdfs@sandbox ~]$ hadoop fs -ls -R viprfs://hdfsBucket3.s3.site1/
ls: ViPRFS internal error (ERROR_FAILED_TO_PROCESS_REQUEST).

```

Load metadata values to ECS using the Management REST API

You can supply the metadata values required to secure an ECS bucket for use with a secure Hadoop cluster by running ECS Management REST API commands.

Before you begin

You must have ECS System Administrator credentials.

If the Hadoop administrator is not the ECS System Administrator, the Hadoop administrator must work in conjunction with the ECS System Administrator to load the secure metadata to the bucket.

The Hadoop administrator can make the JSON metadata file available to the ECS System Administrator, who can then use this procedure to load the metadata. If the two roles are assumed by the same user, then that user is responsible for creating the JSON metadata file and loading it to the ECS bucket.

Procedure

1. Create the JSON file that contains the metadata, as described in: [Secure the ECS bucket using metadata](#) on page 170.
2. Log in to ECS using your System Administrator credentials in order to obtain an authentication token that can be used when running ECS management commands.

You can run the login command using curl. In the following example, you must replace the `<username>:<password>` with ECS System Administrator credentials and supply the IP address or hostname of an ECS node.

```
TOKEN=$(curl -s -k -u <username>:<password> -D - -o /dev/null https://<ECS node IP or hostname>:4443/login | grep X-SDS-AUTH-TOKEN | tr -cd '\40-\176')
```

3. Run the PUT `object/bucket/<bucketname>/metadata` ECS Management REST API command to deploy the metadata, as shown in the following example

```
curl -s -k -X PUT -H "$TOKEN" -H "Accept: application/json" -H "Content-Type: application/json" -T <bucketDetails>.json https://<hostname>:4443/object/bucket/<bucketname>/metadata?namespace=<namespace>
```

You must replace:

- `<username>` with an ECS System Administrator username.
- `<password>` with the password for the specified ECS System Administrator username.
- `<bucketname>` with the name of the bucket you are using for HDFS data.
- `<hostname>` with the IP address or hostname of an ECS node.
- `<bucketdetails>` with the filename of the JSON file containing name-value pairs.
- `<namespace>` with the name of the namespace the bucket resides in.

Once deployed, the metadata is available to all ECS nodes.

Reconfigure ECS client properties

You can use Ambari to set the configuration properties that are required by the ECS client.

The following table lists the required properties.

Table 31 Hadoop configuration to enable ECS access

Hadoop location	Property	Value
core-site	fs.viprfs.auth.identity_translation	CURRENT_USER_REALM
	fs.viprfs.auth.anonymous_translation	CURRENT_USER
	viprfs.security.principal	vipr/_HOST@REALM.COM where REALM.COM is replaced by your Kerberos realm name.

For more information on each `core_site.xml` parameter, see [Hadoop core-site.xml properties for ECS HDFS](#) on page 194.

Start Hadoop services and verify Hadoop access to ECS

Start the Hadoop services.

Procedure

1. Start the Hadoop services using Ambari.
2. Once all Hadoop client services have started, ensure that the ECS bucket can be accessed using the Hadoop CLI, using the following command.

```
hdfs dfs -ls /
```

CHAPTER 9

Troubleshooting

• Introduction.....	176
• Verify that AD/LDAP is correctly configured with a secure Hadoop cluster.....	176
• Pig test fails: unable to obtain Kerberos principal.....	177
• Permission denied for AD user.....	177
• Permissions errors.....	177
• Failed to process request.....	180
• Enable Kerberos client-side logging and debugging.....	181
• Debug Kerberos on the KDC.....	181
• Eliminate clock skew.....	181
• Configure one or more new ECS nodes with the ECS service principal.....	182
• Workaround for Yarn directory does not exist error.....	184

Introduction

This section provides workarounds for issues that you may encounter when configuring ECS HDFS.

Verify that AD/LDAP is correctly configured with a secure Hadoop cluster

You should verify that AD or LDAP is correctly set up with Kerberos (KDC) and the Hadoop cluster.

When your configuration is correct, you should be able to use the `kinit` for an AD/LDAP user. In addition, if the Hadoop cluster is configured for local HDFS, you should check that you can list the local HDFS directory before ECS gets added to the cluster.

Workaround

If you cannot successfully authenticate as an AD/LDAP user with the KDC on the Hadoop cluster, you should address this before proceeding to ECS Hadoop configuration.

An example of a successful login is shown below:

```
[kcluser@lvipri054 root]$ kinit kcluser@QE.COM
Password for kcluser@QE.COM:

[kcluser@lvipri054 root]$ klist
Ticket cache: FILE:/tmp/krb5cc_1025
Default principal: kcluser@QE.COM

Valid starting    Expires          Service principal
04/28/15 06:20:57 04/28/15 16:21:08 krbtgt/QE.COM@QE.COM
                renew until 05/05/15 06:20:57
```

If the above is not successful, you can investigate using the following checklist:

- Check the `/etc/krb5.conf` file on the KDC server for correctness and syntax. Realms can be case sensitive in the configuration files as well as when used with the `kinit` command.
- Check that the `/etc/krb5.conf` file from the KDC server is copied to all the Hadoop nodes.
- Check that one-way trust between AD/LDAP and the KDC server was successfully made.
- Make sure that the encryption type on the AD/LDAP server matches that on the KDC server.
- Check that the `/var/kerberos/krb5kdc/kadm5.acl` and `/var/kerberos/krb5kdc/kdc.conf` files are correct.
- Try logging in as a service principal on the KDC server to indicate that the KDC server itself is working correctly.

- Try logging in as the same AD/LDAP user on the KDC server directly. If that does not work, the issue is likely to be on the KDC server directly.

Pig test fails: unable to obtain Kerberos principal

Pig test fails with the error: `Info:Error: java.io.IOException: Unable to obtain the Kerberos principal even after kinit as AD user, or with Unable to open iterator for alias firsttten.`

This issue is caused due to the fact that Pig (release 0.13 and lower) does not generate a delegation token for ViPRFS as a secondary storage.

Workaround

Append `viprfs://bucket.ns.installation/` to the `mapreduce.job.hdfs-servers` configuration setting. For example:

```
set mapreduce.job.hdfs-servers viprfs://KcdhbuckTM2.s3.site1
```

Permission denied for AD user

The `Permission denied` error is displayed when you run an application as an AD user.

Workaround

Set the permissions for the `/user` directory as:

```
hdfs dfs -chmod 1777 /user
```

Permissions errors

Insufficient permissions errors can occur for a number of reasons. You may receive this type of error when running a `hadoop fs` command, or you may see it in an application log, such as the log for mapreduce or hive.

INSUFFICIENT_PERMISSIONS errors

In the following example, the `jhs` principal tried to create a directory (`/tmp`) and received an `INSUFFICIENT_PERMISSIONS` error. In this case, the permissions of the root directory did not allow this user to create a directory.

```
root@lrmk042:/etc/security/keytabs# hadoop fs -mkdir /tmp
18/02/26 21:03:09 ERROR vipr.ViPRFileSystemClientBase: Permissions failure for request:
User: jhs/lrmk042.lss.emc.com@HOP171_HDFS.EMC.COM (auth:KERBEROS), host:
hdfsBucket3.s3.site1, namespace: s3, bucket: hdfsBucket3
18/02/26 21:03:09 ERROR vipr.ViPRFileSystemClientBase: Request message sent:
MkdirRequestMessage[kind=MKDIR_REQUEST,namespace=s3,bucket=hdfsBucket3,path=/
tmp,hdfsTrustedStatus=HDFS_USER_NOT_TRUSTED,permissions=rwxr-xr-x,createParent=true]
mkdir: java.security.AccessControlException: ERROR_INSUFFICIENT_PERMISSIONS

root@lrmk042:/etc/security/keytabs# hadoop fs -ls -d /
drwxr-xr-x - hdfs hdfs 0 2018-02-26 16:58 /
root@lrmk042:/etc/security/keytabs#
```

When the case of an insufficient permissions error is not obvious on the client, you may have to look at the server logs. Start with `dataheadsvc-error.log` to find the error. Open a terminal window to each ECS node, and edit the `dataheadsvc-error.log` file. Find the error that corresponds to the time you saw the error on the client.

Failed to get credentials

Where you see an error like the following in the `dataheadsvc-error.log`:

```
2018-02-26 22:36:21,985 [pool-68-thread-6] ERROR RequestProcessor.java (line 1482) Unable
to get group credentials for principal 'jhs@HOP171_HDFS.EMC.COM'. This principal will
default to use local user groups. Error message: java.io.IOException: Failed to get group
credentials for 'jhs@HOP171_HDFS.EMC.COM', status=ERROR
```

This is not an error. The message means that the server tried to look up the principal's name to see if there are any cached Active Directory(AD) groups for the principal user making the request. This error is returned for a Kerberos user.

The error indicates the user name making the request. Make a note of it.

Bucket Access Error

If a user making a request to access a bucket does not have ACL permissions, you may see this error in `dataheadsvc-error.log`.

```
2018-02-26 21:35:26,652 [pool-68-thread-1] ERROR BucketAPIImpl.java (line 220) Getting
bucket failed with
com.emc.storageos.objcontrol.object.exception.ObjectAccessException: you don't have
GET_KEYPOOL_ACL permission to this keypool
at
com.emc.storageos.objcontrol.object.exception.ObjectAccessException.createExceptionForAPI(O
bjectAccessException.java:286)
at
com.emc.storageos.data.object.ipc.protocol.impl.ObjectAccessExceptionHandler.parseFrom(Objec
tAccessExceptionHandler.java:61)
```

In this case, you should either add an explicit user ACL for the bucket, or add a custom group ACL for one of the groups that the user is a member of.

Object Access Error

Another type of permission error is an object access error. Access to objects (files and directories) should not be confused with access to a bucket. A user may have full control (read/write/delete) to a bucket, but may receive an **INSUFFICIENT_PERMISSIONS** error because they do not have access to one or more objects in the path they are trying to access. The following provides an example of an object access error.

```
2018-02-26 22:36:21,995 [pool-68-thread-6] ERROR FileSystemAccessHelper.java (line 1364)
nfsProcessOperation failed to process path: mr-history/done
2018-02-26 22:36:21,995 [pool-68-thread-6] ERROR ObjectControllerExceptionHandler.java
(line 186) Method nfsGetSMD failed due to exception
com.emc.storageos.data.object.exception.ObjectControllerExceptionHandler: directory server
returns error ERROR_ACCESS_DENIED
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccessHelper.nfsProcessOperat
```

```

ion(FileSystemAccessHelper.java:1368)
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccessHelper.getSystemMetadat
a(FileSystemAccessHelper.java:466)
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccessLayer.getSystemMetadat
a(FileSystemAccessLayer.java:532)
at com.emc.storageos.data.object.blob.client.BlobAPI.getStat(BlobAPI.java:1294)
at com.emc.vipr.engine.real.RealBlobEngine.stat(RealBlobEngine.java:1976)
at com.emc.vipr.engine.real.RealBlobEngine.stat(RealBlobEngine.java:802)
at com.emc.vipr.hdfs.fs.RequestProcessor.accept(RequestProcessor.java:499)
at com.emc.vipr.hdfs.net.ConnectionManager$RequestThread.run(ConnectionManager.java:136)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)

```

The two important items to note here are the requested action (`stat`) and the path of the object (`mr-history/done`). Note that the leading slash character is not displayed, so the real path is `/mr-history/done`. Now you have three pieces of information that are important for debugging:

- user principal (`jhs@HOP171_HDFS.EMC.COM`)
- action (`stat is hadoop fs -ls`)
- path (`/mr-history/done`)

There are two approaches for additional debugging:

- [Blobsvc log debugging](#) on page 179
- [Hadoop client debugging](#) on page 179

Blobsvc log debugging

A failed permission request will have an error in `blobsvc` like this:

```

2018-02-26 22:36:21,994
[TaskScheduler-BlobService-COMMUNICATOR-ParallelExecutor-5892]
ERROR ObjectAclChecker.java (line 101) not permit, cred
jhs@HOP171_HDFS.EMC.COM[hadoop]false1 with
action GET_OBJECT_ACL on object with acl/owner/group
user={hdfs@hop171_hdfs.emc.com=[FULL CONTROL]},
groups={hdfs=[READ_ACL, EXECUTE, READ]}, other=[], owner=hdfs@hop171_hdfs.emc.com,
group=hdfs

```

Look for `not permit`. This tells us the user making the request (`jhs`), the object's owner (`hdfs`), object group (`hdfs`) and the permissions for owner, group, and others. What it does not tell us is the actual object that failed the permission check. On the Hadoop node, become the `hdfs` principal, and start with the path, and work up the tree, which leads to the other method of debugging, looking at the Hadoop file system from the client.

Hadoop client debugging

When a permission error is received, you should know the user principal making the request, what action the request is, and what items are being requested. In the example, the `jhs` user received an error listing the `/mr-history/done` directory. You can do some analysis to determine the root cause. If you have access to the superuser account, perform these steps as that account.

```

root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -ls -d /mr-history/done
drwxrwxrwt - mapred hadoop 0 2018-02-26 16:58 /mr-history/done

```

The following example shows that the `jhs` principal should have had access to list this directory.

```
root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -ls -d /mr-history
drwxr-xr-x - hdfs hdfs 0 2018-02-26 16:58 /mr-history
```

Likewise, the following output shows that the directory has no access issues.

```
root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -ls -d /
drwxr-x--- - hdfs hdfs 0 2018-02-26 16:58 /
```

The problem here, is that the root directory is owned by `hdfs`, the group name is `hdfs`, but the `others` setting is `- (0)`. The user making the request is `jhs@REALM`, and this user is a member of `hadoop`, but not `hdfs`, so this user has no object ACL permissions to list the `/mr-history/done` directory. Performing the `chmod` command on the root directory enables this user to perform their task.

```
root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -chmod 755 /
root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -ls -d /
drwxr-xr-x - hdfs hdfs 0 2018-02-26 16:58 /
```

Failed to process request

The Failed to Process Request is displayed when listing a bucket.

When performing the list bucket command, for example:

```
# hadoop fs -ls viprfs://hdfsBucket2.s3.site1/
```

The following ViPRFS internal error occurs:

```
ERROR_FAILED_TO_PROCESS_REQUEST
```

Workaround

Possible reasons for this error are:

1. The `viprfs-client` JAR file on the Hadoop node is not in sync with the ECS software.
2. You are attempting to access a secure (Kerberos) bucket from a non-secure (non-Kerberos) Hadoop node.
3. You are attempting to access a non-secure (non-Kerberos) bucket from a secure (Kerberos) Hadoop node.

Enable Kerberos client-side logging and debugging

To troubleshoot authentication issues, you can enable verbose logging and debugging on the Hadoop cluster node that you are using.

Enable client-side verbose logging

You can enable verbose logging using an environment variable that applies only to your current SSH session, as shown in the following example.

```
export HADOOP_OPTS="-Dsun.security.krb5.debug=true"
```

Enable Hadoop client-side debugging

To troubleshoot Hadoop activity between the Hadoop node and ECS, you can enable Hadoop verbose logging as follows:

```
export HADOOP_ROOT_LOGGER="Debug,console"
```

Debug Kerberos on the KDC

You can debug Kerberos on the KDC by using the `tail` command on the KDC `/var/log/krb5kdc.log` file to make it easier to debug when you perform an HDFS operation.

```
tail -f /var/log/krb5kdc.log
```

Eliminate clock skew

It is important to ensure that time is synchronized between the client and server as Kerberos relies on accurate time.

If your Active Directory (AD) has a clock skew with your data nodes/KDC, you will must configure its NTP server. You can do this as follows:

1. Use Remote Desktop to connect to your AD server.
2. Run the following commands:
 - a. `w32tm /config /syncfromflags:manual /manualpeerlist:<ntp-server1>,<ntp-server2>`
 - b. `net stop w32time`
 - c. `net start w32time`

Configure one or more new ECS nodes with the ECS service principal

Where you are adding one or more new nodes to an ECS configuration, the ECS service principal and corresponding keytab must be deployed to the new nodes.

Before you begin

- This procedure assumes that you have previously performed the steps [here](#) and have the Ansible playbooks installed and accessible.

You must have the following items before you can complete this procedure:

- The list of ECS node IP addresses.
- The IP address of the KDC.
- The DNS resolution where you run this script should be the same as the DNS resolution for the Hadoop host, otherwise the `vipr/_HOST@REALM` will not work.

Procedure

1. Log in to Node 1 and check that the tools have previously been installed and the playbooks are available.

The example used previously was:

```
/home/admin/ansible/viprfs-client-<ECS version>--<version>/playbooks
```

2. Edit the `inventory.txt` file in the `playbooks/samples` directory to add the ECS nodes.

The default entries are shown in the following extract.

```
[data_nodes]
192.168.2.[100:200]

[kdc]
192.168.2.10
```

3. Start the utility container on ECS Node 1 and make the Ansible playbooks available to the container.

- a. Load the utility container image.

Example:

```
sudo docker load -i /opt/emc/caspian/checker/docker/images/utilities.tgz
```

- b. Get the identity of the docker image.

Example:

```
admin@provo-lilac:~> sudo docker images
```

The output will give you the image identity:

REPOSITORY SIZE	TAG	IMAGE ID	CREATED	VIRTUAL
utilities	1.5.0.0-403.cb6738e	186bd8577a7a	2 weeks ago	738.5 MB

c. Start and enter utilities image.

Example:

```
sudo docker run -v /opt/emc/caspian/fabric/agent/services/object/main/log:/opt/
storageos/logs
-v /home/admin/ansible/viprfs-client-3.0.0.0.85325.a05145b/playbooks:/ansible
--name=ecs-tools -i -t --privileged --net=host 186bd8577a7a /bin/bash
```

In the example, the location to which the Ansible playbooks were unzipped /
home/admin/ansible/viprfs-client-3.0.0.0.85325.a05145b/
playbooks is mapped to the /ansible directory in the utility container.

4. Change to the working directory in the container.

Example:

```
cd /ansible
```

5. Run the Ansible playbook to generate keytabs.

```
ansible-playbook -v -k -i inventory.txt generate-vipr-keytabs.yml
```

6. Run the Ansible playbook to configure the data nodes with the ECS service principal.

Make sure the /ansible/samples/keytab directory exists and the
krb5.conf file is in the working directory /ansible/samples directory.

```
ansible-playbook -v -k -i inventory.txt setup-vipr-kerberos.yml
```

Verify that the correct ECS service principal, one per data node, has been
created (from the KDC):

```
# kadmin.local -q "list_principals" | grep vipr
vipr/nile3-vm42.centera.lab.emc.com@MA.EMC.COM
vipr/nile3-vm43.centera.lab.emc.com@MA.EMC.COM
```

Verify that correct keytab is generated and stored in location: /data/hdfs/
krb5.keytab on all ECS data nodes. You can use the strings command on
the keytab to extract the human readable text, and verify that it contains the
correct principal. For example:

```
dataservice-10-247-199-69:~ # strings /data/hdfs/krb5.keytab
MA.EMC.COM
vipr
nile3-vm42.centera.lab.emc.com
```

In this case the principal is `vipr/nile3-vm42.centera.lab.emc.com`.

Workaround for Yarn directory does not exist error

When you configure ECS as the default file system with a Kerberized HDP cluster, you can get an error like `/ats/done does not exist`. In addition to this, Resource Manager does not start.

This procedure provides a workaround for these issues.

Procedure

1. Check if your Hadoop nodes can resolve ECS nodes.

- a. Install the `nslookup` tool in the Hadoop nodes.

```
yum install -y bind-utils
```

- b. Check if it can resolve the ECS node.

```
nslookup <address of ECS node>
```

- c. If it does not resolve to the correct hostname, add the ECS DNS to the `/etc/resolv.conf` on the Hadoop nodes.

You can check the DNS entries are there by running:

```
cat /etc/resolv.conf
```

- d. Now the Hadoop node will resolve the ECS nodes.

Run `nslookup` again to check it resolves.

```
nslookup <address of ECS node>
```

2. Check the system time in the Hadoop node, ECS nodes, and KDC.

Use:

```
# date
```

If the time of the systems are not consolidated, they should be synced to the same NTP server.

Information on enabling NTP on the cluster and on the browser host is described [here](#).

3. If the previous steps do not work, you can try to manually create folder `done` or `active` under `/ats`.

```
# sudo -u hdfs hdfs dfs -mkdir /ats/done
```

```
# sudo -u hdfs hdfs dfs -mkdir /ats/active
```

and check that the directories exist.

```
$ hdfs dfs -ls /ats
```

```
Found 2 items
drwxrwxrwt - yarn hadoop 0 2016-07-12 09:00 /ats/active
drwx----- - yarn hadoop 0 2016-07-12 09:00 /ats/done
```


CHAPTER 10

Appendix: Guidance on Kerberos Configuration

- [Guidance on Kerberos configuration](#)..... 188

Guidance on Kerberos configuration

Provides guidance on configuring Kerberos in the Hadoop cluster.

Set up the Kerberos KDC

Set up the Kerberos KDC by following these steps.

Procedure

1. Install krb5-workstation.

Use the command:

```
yum install -y krb5-libs krb5-server krb5-workstation
```

2. Modify `/etc/krb5.conf` and change the realm name and extensions.
3. Modify `/var/kerberos/krb5kdc/kdc.conf` and change the realm name to match your own.
4. If your KDC is a VM, recreate `/dev/random` (otherwise your next step of creating the KDC database will take a very long time).

- a. Remove using:

```
# rm -rf /dev/random
```

- b. Recreate using:

```
# mknod /dev/random c 1 9
```

5. Create the KDC database.

```
# kdb5_util create -s
```

Note

If you made a mistake with the initial principals. For example, you ran "`kdb5_util create -s`" incorrectly, you might need to delete these principals explicitly in the `/var/kerberos/krb5kdc/` directory.

6. Modify `kadm5.acl` to specify users that have admin permission.

```
*/admin@DET.EMC.COM *
```

7. Modify `/var/kerberos/krb5kdc/kdc.conf` and take out any encryption type except `des-cbc-crc:normal`. Also modify the realm name.

8. Ensure iptables and selinux are off on all nodes (KDC server as well as Hadoop nodes).
9. Start KDC services and create a local admin principal.

```
kadmin.local

# service krb5kdc start

# service kadmin start

# /usr/kerberos/sbin/kadmin.local-q "addprinc root/admin"

# kinit root/admin
```

10. Copy the `krb5.conf` file to all Hadoop nodes.

Any time you make a modification to any of the configuration files restart the below services and copy the `krb5.conf` file over to relevant Hadoop host and ECS nodes.

11. Restart the services.

```
service krb5kdc restart

service kadmin restart
```

12. You can visit following link to setup a Kerberos KDC based on steps at <http://www.centos.org/docs/4/html/rhel-rg-en-4/s1-kerberos-server.html>.

Configure AD user authentication for Kerberos

Where you have a Hadoop environment configured with Kerberos security, you can configure it to authenticate against the ECS AD domain.

Make sure you have an AD user for your ADREALM. The user "detscr" for ADREALM CAMBRIDGE.ACME.COM is used in the example below. Create a one-way trust between the KDCREALM and the ADREALM as shown in the example. Do not try to validate this realm using "netdom trust".

On Active Directory

You must set up a one-way cross-realm trust from the KDC realm to the AD realm. To do so, run the following commands at a command prompt.

```
ksetup /addkdc KDC-REALM <KDC hostname>
netdom trust KDC-REALM /Domain:AD-REALM /add /realm /
passwordt:<TrustPassword>
ksetup /SetEncTypeAttr KDC-REALM <enc_type>
```

For example:

```
ksetup /addkdc LSS.EMC.COM lcigb101.lss.emc.com
netdom trust LSS.ACME.COM /Domain:CAMBRIDGE.ACME.COM /add /realm /
passwordt:ChangeMe
ksetup /SetEncTypeAttr LSS.ACME.COM DES-CBC-CRC
```

For this example, encryption des-cbc-crc was used. However, this is a weak encryption that was only chosen for demonstration purposes. Whatever encryption you choose, the AD, KDC, and clients must support it.

On your KDC (as root)

To set up a one-way trust, you will need to create a "krbtgt" service principal. To do so, the name is krbtgt/KDC-REALM@AD-REALM. Give this the password ChangeMe, or whatever you specified to the /password argument above.

1. On KDC (as root)

```
# kadmin
kadmin: addprinc -e "des-cbc-crc:normal" krbtgt/
LSS.ACME.COM@CAMBRIDGE.ACME.COM
```

Note

When deploying, it is best to limit the encryption types to the one you chose. Once this is working, additional encryption types can be added.

2. Add the following rules to your `core-site.xml` `hadoop.security.auth_to_local` property:

```
RULE: [1:$1@$0] (^.*@CAMBRIDGE\.ACME\.COM$) s/^(.*)@CAMBRIDGE\.ACME
\.COM$/g
RULE: [2:$1@$0] (^.*@CAMBRIDGE\.ACME\.COM$) s/^(.*)@CAMBRIDGE\.ACME
\.COM$/g
```

3. Verify that AD or LDAP is correctly setup with the Kerberos (KDC) server. User should be able to "kinit" against an AD user and list local HDFS directory.

Note

If you are configuring your Hadoop cluster and ECS to authenticate through an AD, create local Linux user accounts on all Hadoop nodes for the AD user you will be kinit'ed as, and also make sure that all Hadoop host are kinit'ed using that AD user. For example, if you kinit as userX@ADREALM, create userX as a local user on all Hadoop hosts, and kinit using: 'kinit userX@ADREALM' on all hosts for that user.

In the example below, we will authenticate as "kinit detscr@CAMBRIDGE.EMC.COM", so will create a user called "detscr" and kinit as this user on the Hadoop host. As shown below:

```
[root@lviprb159 ~]# su detscr
[detscr@lviprb159 root]$ whoami
detscr
[detscr@lviprb159 root]$ kinit detscr@CAMBRIDGE.ACME.COM
Password for detscr@CAMBRIDGE.ACME.COM:
[detscr@lviprb159 root]$ klist
Ticket cache: FILE:/tmp/krb5cc_1010
Default principal: detscr@CAMBRIDGE.ACME.COM
Valid starting Expires Service principal
12/22/14 14:28:27 03/02/15 01:28:30 krbtgt/
CAMBRIDGE.ACME.COM@CAMBRIDGE.ACME.COM
renew until 09/17/17 15:28:27

[detscr@lviprb159 root]$ hdfs dfs -ls /
Found 4 items
drwx---rwx - yarn hadoop 0 2014-12-23 14:11 /app-logs
```

drwx---rwt	-	hdfs	0	2014-12-23	13:48	/apps
drwx---r-x	-	mapred	0	2014-12-23	14:11	/mapred
drwx---r-x	-	hdfs	0	2014-12-23	14:11	/mr-history

CHAPTER 11

Appendix: Hadoop core-site.xml properties for ECS HDFS

- [Hadoop core-site.xml properties for ECS HDFS](#)..... 194

Hadoop core-site.xml properties for ECS HDFS

When configuring the Hadoop `core-site.xml` file, use this table as a reference for the properties and their related values.

Table 32 Hadoop core-site.xml properties

Property	Description
File system implementation properties	
<code>fs.viprfs.impl</code>	<pre><property> <name>fs.viprfs.impl</name> <value>com.emc.hadoop.fs.vipr.ViPRFileSystem</value> </property></pre>
<code>fs.AbstractFileSystem.viprfs.impl</code>	<pre><property> <name>fs.AbstractFileSystem.viprfs.impl</name> <value>com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem</value> </property></pre>
Properties that define the authority section of the ECS HDFS file system URI	
<code>fs.vipr.installations</code>	<p>A comma-separated list of names. The names are further defined by the <code>fs.vipr.installation.[federation].hosts</code> property to uniquely identify sets of ECS data nodes. The names are used as a component of the authority section of the ECS HDFS file system URI. For example:</p> <pre><property> <name>fs.vipr.installations</name> <value><federation>,<site1>,<testsite></value> </property></pre>
<code>fs.vipr.installation.[federation].hosts</code>	<p>The IP addresses of the ECS cluster's data nodes or the load balancers for each name listed in the <code>fs.vipr.installations</code> property. Specify the value in the form of a comma-separated list of IP addresses or FQDNs. For example:</p> <pre><property> <name>fs.vipr.installation.<federation>.hosts</name> <value>203.0.113.10,203.0.113.11,203.0.113.12</value> </property></pre>
<code>fs.vipr.installation.[installation_name].resolution</code>	<p>Specifies how the ECS HDFS software knows how to access the ECS data nodes. Values are:</p> <ul style="list-style-type: none"> dynamic: Use this value when accessing ECS data nodes directly without a load balancer. fixed: Use this value when accessing ECS data nodes through a load balancer. <pre><property> <name>fs.vipr.installation.<federation>.resolution</name> <value>dynamic</value></pre>

Table 32 Hadoop core-site.xml properties (continued)

Property	Description
	<pre></property></pre>
fs.vipr.installation.[installation_name].resolution.dynamic.time_to_live_ms	<p>When the <code>fs.vipr.installation.[installation_name].resolution</code> property is set to <code>dynamic</code>, this property specifies how often to query ECS for the list of active nodes. Values are in milliseconds. The default is 10 minutes.</p> <pre><property> <name>fs.vipr.installation.<federation>.resolution.dynamic.time_to_live_ms</name> <value>600000</value> </property></pre>
ECS file system URI	
fs.defaultFS	<p>A standard Hadoop property that specifies the URI to the default file system. Setting this property to the ECS HDFS file system is optional. If you do not set it to the ECS HDFS file system, you must specify the full URI on each file system operation. The ECS HDFS file system URI has this format:</p> <pre>viprfs://[bucket_name].[namespace].[federation]</pre> <ul style="list-style-type: none"> <i>bucket_name</i>: The name of the HDFS-enabled bucket that contains the data you want to use when you run Hadoop jobs. <i>namespace</i>: The tenant namespace associated with the HDFS-enabled bucket. <i>federation</i>: The name associated with the set of ECS data nodes that Hadoop can use to access ECS data. The value of this property must match one of the values specified in the <code>fs.vipr.installations</code> property. <p>For example:</p> <pre><property> <name>fs.defaultFS</name> <value>viprfs://testbucket.s3.federation1</value> </property></pre>
umask property	
fs.permissions.umask-mode	<p>This standard Hadoop property specifies how ECS HDFS should compute permissions on objects. Permissions are computed by applying a umask on the input permissions. The recommended value for both simple and Kerberos configurations is: 022. For example:</p> <pre><property> <name>fs.permissions.umask-mode</name> <value>022</value> </property></pre>

Table 32 Hadoop core-site.xml properties (continued)

Property	Description
Identity translation properties	
fs.viprfs.auth.identity_translation	<p>This property specifies how the ECS HDFS client determines what Kerberos realm a particular user belongs to if one is not specified. ECS data nodes store file owners as <code>username@REALM</code>, while Hadoop stores file owners as just the username.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> NONE: Default. Users are not mapped to a realm. Use this setting with a Hadoop cluster that uses simple security. With this setting ECS HDFS does not perform realm translation. CURRENT_USER_REALM: Valid when Kerberos is present. The user's realm is auto-detected, and it is the realm of the currently signed in user. In the example below, the realm is <code>EMC.COM</code> because sally is in the <code>EMC.COM</code> realm. The file ownership is changed <code>john@EMC.COM</code>. <pre># kinit sally@EMC.COM # hdfs dfs -chown john /path/to/file</pre> <p>Realms provided at the command line takes precedence over the property settings.</p> <pre><property> <name>fs.viprfs.auth.identity_translation </name> <value>CURRENT_USER_REALM</value> </property></pre> <hr/> <p>Note</p> <p><code>FIXED_REALM</code> is now deprecated.</p>
fs.viprfs.auth.realm	<p>The realm assigned to users when the <code>fs.viprfs.auth.identity_translation</code> property is set to <code>FIXED_REALM</code>.</p> <p>This is now deprecated.</p>
fs.viprfs.auth.anonymous_translation	<p>This property is used to determine how users and groups are assigned to newly created files.</p> <hr/> <p>Note</p> <p>This property was used to determine what happened to files that had no owner. These files were said to be owned by <code>anonymous</code>. Files and directories are no longer anonymously owned.</p> <hr/> <p>The values are:</p> <ul style="list-style-type: none"> LOCAL_USER: Use this setting with a Hadoop cluster that uses simple security. Assigns the Unix user and group of the Hadoop cluster to newly created files and directories. CURRENT_USER: Use this setting for a Hadoop cluster that uses Kerberos. Assigns the Kerberos principal (<code>user@REALM.COM</code>) as the file or directory owner, and uses the group that has been assigned as the default for the bucket. NONE: (Deprecated) Previously indicated that no mapping from the anonymously owned objects to the current user should be performed. <pre><property></pre>

Table 32 Hadoop core-site.xml properties (continued)

Property	Description
	<pre><name>fs.viprfs.auth.anonymous_translation</name> <value>CURRENT_USER</value> </property></pre>
Kerberos realm and service principal properties	
viprfs.security.principal	<p>This property specifies the ECS service principal. This property tells the KDC about the ECS service. This value is specific to your configuration.</p> <p>The principal name can include <code>_HOST</code> which is automatically replaced by the actual data node FQDN at run time.</p> <p>For example:</p> <pre><property> <name>viprfs.security.principal</name> <value>vipr/_HOST@example.com</value> </property></pre>

Sample core-site.xml for simple authentication mode

The following `core-site.xml` file is an example of ECS HDFS properties for simple authentication mode.

Example 1 core-site.xml

```
<property>
  <name>fs.viprfs.impl</name>
  <value>com.emc.hadoop.fs.vipr.ViPRFileSystem</value>
</property>

<property>
  <name>fs.AbstractFileSystem.viprfs.impl</name>
  <value>com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem</value>
</property>

<property>
  <name>fs.vipr.installations</name>
  <value>federation1</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.hosts</name>
  <value>203.0.113.10,203.0.113.11,203.0.113.12</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.resolution</name>
  <value>dynamic</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.resolution.dynamic.time_to_live_ms</name>
```

Example 1 core-site.xml (continued)

```
<value>900000</value>
</property>

<property>
  <name>fs.defaultFS</name>
  <value>viprfs://mybucket.mynamespace.federation1</value>
</property>

<property>
  <name>fs.viprfs.auth.anonymous_translation</name>
  <value>LOCAL_USER</value>
</property>

<property>
  <name>fs.viprfs.auth.identity_translation</name>
  <value>NONE</value>
</property>
```

CHAPTER 12

Appendix: Secure bucket metadata example

- [Secure bucket metadata](#).....200

Secure bucket metadata

The following example shows a list of secure bucket metadata name value pairs.

```
{
  "head_type": "hdfs",
  "metadata": [
    {
      "name": "internal.kerberos.user.ambari-qa.name",
      "value": "ambari-qa@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.ambari-qa.shortname",
      "value": "ambari-qa"
    },
    {
      "name": "internal.kerberos.user.ambari-qa.groups",
      "value": "hadoop,users"
    },
    {
      "name": "internal.kerberos.user.cmaurer.name",
      "value": "cmaurer@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.cmaurer.shortname",
      "value": "cmaurer"
    },
    {
      "name": "internal.kerberos.user.cmaurer.groups",
      "value": "cmaurer,adm,cdrom,sudo,dip,plugdev,users,lpadmin,sambashare"
    },
    {
      "name": "internal.kerberos.user.dn.name",
      "value": "dn@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.dn.shortname",
      "value": "hdfs@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.dn.groups",
      "value": "hadoop,hdfs"
    },
    {
      "name": "internal.kerberos.user.hdfs.name",
      "value": "hdfs@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.hdfs.shortname",
      "value": "hdfs"
    },
    {
      "name": "internal.kerberos.user.hdfs.groups",
      "value": "hadoop,hdfs"
    },
    {
      "name": "internal.kerberos.user.hive.name",
      "value": "hive@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.hive.shortname",
      "value": "hive"
    }
  ]
}
```



```

{
  "name": "internal.kerberos.user.hive.groups",
  "value": "hadoop"
},
{
  "name": "internal.kerberos.user.jhs.name",
  "value": "jhs@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.jhs.shortname",
  "value": "mapred"
},
{
  "name": "internal.kerberos.user.jhs.groups",
  "value": "hadoop"
},
{
  "name": "internal.kerberos.user.nm.name",
  "value": "nm@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.nm.shortname",
  "value": "yarn@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.nm.groups",
  "value": "hadoop"
},
{
  "name": "internal.kerberos.user.nn.name",
  "value": "nn@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.nn.shortname",
  "value": "hdfs@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.nn.groups",
  "value": "hadoop,hdfs"
},
{
  "name": "internal.kerberos.user.rm.name",
  "value": "rm@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.rm.shortname",
  "value": "yarn@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.rm.groups",
  "value": "hadoop"
},
{
  "name": "internal.kerberos.user.spark.name",
  "value": "spark@EXAMPLE_HDFS.EMC.COM"
},
{
  "name": "internal.kerberos.user.spark.shortname",
  "value": "spark"
},
{
  "name": "internal.kerberos.user.spark.groups",
  "value": "hadoop"
},
{
  "name": "internal.kerberos.user.yarn.name",
  "value": "yarn@EXAMPLE_HDFS.EMC.COM"
},

```

```

    {
      "name": "internal.kerberos.user.yarn.shortname",
      "value": "yarn"
    },
    {
      "name": "internal.kerberos.user.yarn.groups",
      "value": "hadoop"
    },
    {
      "name": "internal.kerberos.user.zookeeper.name",
      "value": "zookeeper@EXAMPLE_HDFS.EMC.COM"
    },
    {
      "name": "internal.kerberos.user.zookeeper.shortname",
      "value": "ams"
    },
    {
      "name": "internal.kerberos.user.zookeeper.groups",
      "value": "hadoop"
    },
    {
      "name": "hadoop.proxyuser.hcat.groups",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.hcat.hosts",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.yarn.users",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.yarn.hosts",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.hive.hosts",
      "value": "10.247.179.42"
    },
    {
      "name": "hadoop.proxyuser.hive.users",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.hcat.groups",
      "value": "*"
    },
    {
      "name": "hadoop.proxyuser.hcat.hosts",
      "value": "*"
    },
    {
      "name": "dfs.permissions.supergroup",
      "value": "hdfs"
    }
  ]
}

```