Elastic Cloud Storage (ECS)

Version 3.0

Data Access Guide

302-003-221

04

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

CONTENTS

FIGURES

TABLES

PART 1

S3

CHAPTER 1

Introduction to Amazon S3 Support in ECS

Amazon S3 API support in ECS

This part describes ECS support for the Amazon S3 API.

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

Protocol	Ports
HTTP	9020
HTTPS	9021

The following topics describe the support for the S3 API, the extension provided by ECS, and describe how to authenticate with the service and how to use SDKs to develop clients to access the service:

- S3 API Supported and Unsupported Features on page 18
- S3 Extensions on page 22
- Use Metadata Search on page 30
- Create and manage secret keys on page 42
- Authenticating with the S3 service on page 48
- Use SDKs to access the S3 service on page 52

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 following topic:

- Administrators Guide: Set the Base URL

CHAPTER 2

S3 Supported Features

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 18
- Unsupported S3 APIs on page 19

Supported S3 APIs

The following table lists the supported S3 API methods.

Table 1 Supported S3 APIs

Feature	Notes

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

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 filesystem-enabled buckets.
GET Bucket (List Objects)	
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 filesystem-enabled buckets.
GET Bucket Object versions	
GET Bucket versioning	
HEAD Bucket	
List Multipart Uploads	
PUT Bucket	
PUT Bucket cors	
PUT Bucket acl	

Table 1 Supported S3 APIs (continued)

Feature	Notes
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 filesystem-enabled buckets.
PUT Bucket versioning	
DELETE Object	
Delete Multiple Objects	
GET Object	
GET Object ACL	
HEAD Object	
PUT Object	Supports chunked PUT
PUT Object acl	
PUT Object - Copy	
OPTIONS object	
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	

Table 2 Additional features

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

Unsupported S3 APIs

The following table lists the unsupported S3 API methods.

Table 3 Unsupported S3 APIs

Feature	Notes
DELETE Bucket policy	
DELETE Bucket tagging	
DELETE Bucket website	
GET Bucket policy	
GET Bucket location	ECS is only aware of a single virtual data center.
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 has its own model for payments.
GET Bucket website	
PUT Bucket policy	
PUT Bucket logging	
PUT Bucket notification	Notification is only defined for reduced redundancy feature in S3. ECS does not currently support notifications.
PUT Bucket tagging	
PUT Bucket requestPayment	ECS has 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.

CHAPTER 3

S3 Extensions

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
- Retention extension
- File system enabled extension
- Metadata search extension

Byte range extensions

The following byte range extensions are provided:

- Updating a byte range within an object on page 22
- Overwriting part of an object on page 23
- Appending data to an object on page 24
- Reading multiple byte ranges within an object on page 25

Updating a byte range within an object

An example of using the ECS API extensions to update a byte range of an object is provided below.

First do a GET request on the object named `object1` located in `bucket1` to review the object. `object1` has the value `The quick brown fox jumps over the lazy dog.`

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 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, 17 Jun 2013 20:04:40 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 20:04:28 GMT
ETag: 6
Content-Type: application/json
Content-Length: 43

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

Now you want to update a specific byte range within this object. To do this, 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, 17 Jun 2013 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, 17 Jun 2013 20:15:16 GMT
```

When reading the object again, the new value is now `The quick green fox jumps over the lazy dog`. (The word `brown` has been replaced with `green`.) You have updated a specific byte range within this object.

```
GET /bucket1/object1 HTTP/1.1
Cookie: JSESSIONID=wdit99359t8rnvipinz4tbtu
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 17 Jun 2013 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, 17 Jun 2013 20:16:00 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 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

An example of using the ECS API extensions to overwrite part of an object is provided below.

You can overwrite part of an object by providing only the starting offset in the data request. The data in the request will be written starting at the provided offset. The format is: `Range: <startingOffset>-`

For example, to write the data `brown cat` starting at offset 10, you would 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, 17 Jun 2013 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, 17 Jun 2013 20:51:41 GMT
```

When retrieving the object, you can see the final value `The quick brown cat jumps over the lazy dog and cat.` (`green fox` has been replaced with `brown cat`). You have overwritten part of the data in this object at the provided starting offset.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 17 Jun 2013 20:51:55 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 20:51:41 GMT
ETag: 25
Content-Type: application/json
Content-Length: 51

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

Appending data to an object

An example of using the ECS API extensions to append data to an object is provided below.

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-` 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-`. 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, 17 Jun 2013 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, 17 Jun 2013 20:46:01 GMT
```

When retrieving the object again, you can see the full value `The quick green fox jumps over the lazy dog and cat`. You have appended data to this object.

```
GET /bucket1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 17 Jun 2013 20:46:56 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nG1gMDTg=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 17 Jun 2013 20:46:56 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 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

An example of using the ECS API extensions to read multiple byte ranges within an object is provided below.

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

Note

The Amazon S3 API only supports one range when using the HTTP header `Range` for reading; ECS supports multiple byte ranges.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 20:51:55 -0000
```

```
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 206 Partial Content
Date: Mon, 17 Jun 2013 20:51:55 GMT
Content-Type: multipart/byteranges;boundary=bound04acf7f0ae3ccc
Last-Modified: Mon, 17 Jun 2013 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 associate 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.

Retention period on bucket

A retention period stored against a bucket can be used to set a retention for all objects, with the object level retention period or policy used to provide and 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 using the and object retention periods or the object retention policy, is used to determine whether the operation can be performed.

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 buckets that have versioning enabled and for buckets that do not have versioning enabled.

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 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 file system 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 (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 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 following:

- Configure HDFS
- Administrators Guide: Configure NFS file access

CHAPTER 4

S3 Metadata Search Extension

Use Metadata Search

The ECS S3-compatible API provides a metadata search extension to the API 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.

In addition to system metadata, objects also have attributes which can be returned as part of metadata search results.

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

- Assign metadata index values to a bucket on page 30
- Assign metadata to objects using the S3 protocol on page 33
- Use metadata search queries on page 34

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.

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 39.

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. Refer to Administrators Guide: Create and manage buckets for details.

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 indexing on new buckets.
PUT /{bucket} -H x-emc-metadata-search: {name[;datatype],...}	Creates a bucket with the search metadata key indicated in the header.

API Path	Description
	Note A datatype must be associated with a user metadata key, but is not necessary for a system metadata key.
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.

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
`GET /{bucket}/?query={expression}&attributes={fieldname,…}&sorted={selector}&include_older_version={true|false}` (…also standard pagination parameters apply)	`<BucketQueryResult xmlns:ns2="http://s3.amazonaws.com/doc/2006-03-01/"> <Name>mybucket</Name> <Marker/> <NextMarker>NO MORE PAGES</NextMarker> <MaxKeys>0</MaxKeys> <IsTruncated>false</IsTruncated> <ObjectMatches> <object> <objectName>file4</objectName> <objectId>09998027b1b7fbb21f50e13fabb481a237ba2f60f352d437c8da3c7c1c8d7589</objectId> <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>`

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 > 2015-09-02T11:22:00Z
```

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

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-num

The maximum number of objects that match the query that should be returned. If there are more objects than the max-num, 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:2015-11-30T12:00:00Z
```

User and System query expression format:

```
?query=CreateTime>2015-01-01:00:00:00Z and x-amz-meta-
Foo==2015-11-30T12:00:00Z
```

Query results fragment - system metadata

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

Query results fragment - user metadata

```
<key>x-amz-meta-Foo</key> <value>2015-11-30T12:00:00Z</value>
```

Using markers and max-num 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>2015-01-01:00:00:00Z and x-amz-meta-
Foo==2015-11-30T12:00:00Z&max-num=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>2015-01-01:00:00:00Z and x-amz-meta-
Foo==2015-11-30T12:00:00Z&max-num=3&marker=rO0ABXNyAD...
```

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>
  <IsTruncated>false</IsTruncated>
  <ObjectMatches>
    <object>
      <objectName>file4</objectName>

<objectId>09998027b1b7fbb21f50e13fabb481a237ba2f60f352d437c8da3c7c1c
8d7589</objectId>
      <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>1ad87d86ef558ca0620a26855662da1030f7d9ff1d4bbc7c2ffdfe2994
3b9150</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>Sring5</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.

Name (Alias)	Type	Description
		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

CHAPTER 5

Create and Manage Secret Keys

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:

- Administrator creates a keys and distributes to the object user (Create a key for an object user on page 42).
- Object user who is a domain user creates a new key using the self-service API provided by the ECS Management REST API (Create an S3 secret key: self-service on page 43).

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 42
- Create an S3 secret key using the ECS Management REST API on page 42

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 Admin or Namespace Admin

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

Procedure

1. At the ECS Portal, select the **Manage** > **Users** page.
2. In the Object Users table, select **Edit** for the user to which you want to assign a secret key.
3. For S3, select **Generate & Add Password**.
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}	API to allow secret keys to be assigned to object users and enable secret keys to be managed. Namespace Admin can create keys for users in their namespace. System Admin can assign keys to users in any namespace.

You can find out more information about the API call in the ECS Management REST 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 Management REST 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	API to allow S3 client users to create a new secret key that enables them to access objects and buckets within their namespace. This is also referred to as a self-service API.

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

There are a number of operations that you might want to perform with self-service secret keys using the ECS Management REST API Reference.

The examples provided use the `curl` tool to demonstrate the following activities.

- Log in as a domain user on page 44
- Generate first key on page 44
- Generate second key on page 45
- Check keys on page 45
- Delete all secret keys on page 45

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, 27 Apr 2015 17:29:38 GMT
Content-Type: application/xml
Content-Length: 107
Connection: keep-alive
X-SDS-AUTH-TOKEN:
BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwNzQ4ODA1NTQD
AC
51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A
8=

<?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:
BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAw
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRj
OTdlNGQ0AgAC0A8="
-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>2015-04-27 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:
BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwN
zQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjO
TdlNGQ0AgAC0A8="
-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>2015-04-27 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
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRj
OTdlNGQ0AgAC0A8="
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>2015-04-27 17:50:12.369</
key_expiry_timestamp_1>
  <key_expiry_timestamp_2/>
  <key_timestamp_1>2015-04-27 17:39:13.813</key_timestamp_1>
  <key_timestamp_2>2015-04-27 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
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRj
OTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{}" https://
10.241.48.31:4443/object/secret-keys/deactivate
```

CHAPTER 6

Authenticating with the S3 service

Authenticating with the S3 service

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 will 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=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc96317663032
6f1024
```

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=e75a150daa28a2b2f7ca24f6fd0e161cb58648a25121d310
8f0af5c9451b09ce
x-emc-namespace: ns1
x-emc-rest-client: TRUE
x-amz-content-sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
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
```

CHAPTER 7

Java Client Access

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 EMC 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 EMC ECS Java SDK.

- Using the Java Amazon SDK on page 52
- Java SDK client for ECS on page 54

Note

If you want to make use of the ECS API Extensions (see S3 Extensions on page 22), support for these extensions is provided in the EMC 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.vipr.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) {

        }

    }
}
```

PART 2

OpenStack Swift

CHAPTER 8

Introduction to OpenStack Swift support in ECS

OpenStack Swift API support in ECS

ECS includes support for the OpenStack Swift API. This part describes the supported operations and describes the mechanisms for authorization and authentication.

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

Protocol	Ports
HTTP	9024
HTTPS	9025

The following topics describe supported methods, the ECS extensions, and the mechanism for authentication:

- OpenStack Swift supported operations on page 62
- Swift API Extensions on page 66
- OpenStack Swift Authentication on page 72
- Authorization on Container on page 80

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

- OpenStack API Examples

CHAPTER 9

Swift Supported Features

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 4 OpenStack Swift supported calls

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

Table 4 OpenStack Swift supported calls (continued)

Method	Path	Description
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.

Unsupported OpenStack Swift calls

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

Table 5 OpenStack Swift unsupported calls

Method	Path	Description
HEAD	v1/{account}	Retrieve the account metadata, for example, the number of containers, the total bytes stored in OpenStackObject Storage for the account/tenant.
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-*.
COPY	v1/{account}/ {container}/{object}	Copy is supported using PUT v1/{account}/ {container}/{object} with X-Copy-From header.

CHAPTER 10

Swift Extensions

Swift API Extensions

A number of extensions to the object APIs are supported.

Updating a byte range within an object

An example of using the ECS API extensions to update a byte range of an object is provided below.

First do a GET request on the object named `object1` located in `bucket1` to review the object. `object1` has the value `The quick brown fox jumps over the lazy dog.`

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 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, 17 Jun 2013 20:04:40 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 20:04:28 GMT
ETag: 6
Content-Type: application/json
Content-Length: 43

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

Now you want to update a specific byte range within this object. To do this, 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, 17 Jun 2013 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, 17 Jun 2013 20:15:16 GMT
```

When reading the object again, the new value is now `The quick green fox jumps over the lazy dog`. (The word `brown` has been replaced with `green`.) You have updated a specific byte range within this object.

```
GET /bucket1/object1 HTTP/1.1
Cookie: JSESSIONID=wdit99359t8rnvipinz4tbtu
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 17 Jun 2013 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, 17 Jun 2013 20:16:00 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 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

An example of using the ECS API extensions to overwrite part of an object is provided below.

You can overwrite part of an object by providing only the starting offset in the data request. The data in the request will be written starting at the provided offset. The format is: `Range: <startingOffset>-`

For example, to write the data `brown cat` starting at offset 10, you would 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, 17 Jun 2013 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, 17 Jun 2013 20:51:41 GMT
```

When retrieving the object, you can see the final value `The quick brown cat jumps over the lazy dog and cat.` (`green fox` has been replaced with `brown cat`). You have overwritten part of the data in this object at the provided starting offset.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 17 Jun 2013 20:51:55 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 20:51:41 GMT
ETag: 25
Content-Type: application/json
Content-Length: 51

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

Appending data to an object

An example of using the ECS API extensions to append data to an object is provided below.

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-` 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-`. 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, 17 Jun 2013 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, 17 Jun 2013 20:46:01 GMT
```

When retrieving the object again, you can see the full value `The quick green fox jumps over the lazy dog and cat`. You have appended data to this object.

```
GET /bucket1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/
octet-stream
Date: Mon, 17 Jun 2013 20:46:56 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nG1gMDTg=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 200 OK
Date: Mon, 17 Jun 2013 20:46:56 GMT
Content-Type: application/octet-stream
Last-Modified: Mon, 17 Jun 2013 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

An example of using the ECS API extensions to read multiple byte ranges within an object is provided below.

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

Note

The Amazon S3 API only supports one range when using the HTTP header `Range` for reading; ECS supports multiple byte ranges.

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 17 Jun 2013 20:51:55 -0000
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

HTTP/1.1 206 Partial Content
Date: Mon, 17 Jun 2013 20:51:55 GMT
Content-Type: multipart/byteranges;boundary=bound04acf7f0ae3ccc
Last-Modified: Mon, 17 Jun 2013 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--
```

CHAPTER 11

Authentication

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 73.

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 75

v3

ECS validates Keystone V3 users that present tokens scoped to a Keystone project. See Authentication using ECS Keystone V3 integration on page 77.

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 at the ECS Portal on page 72.

Create Swift users at the ECS Portal

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

Before you begin

You will need to be an ECS System Admin.

You can find more information on adding ECS object users in Administrators Guide: Manage users and roles .

Procedure

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

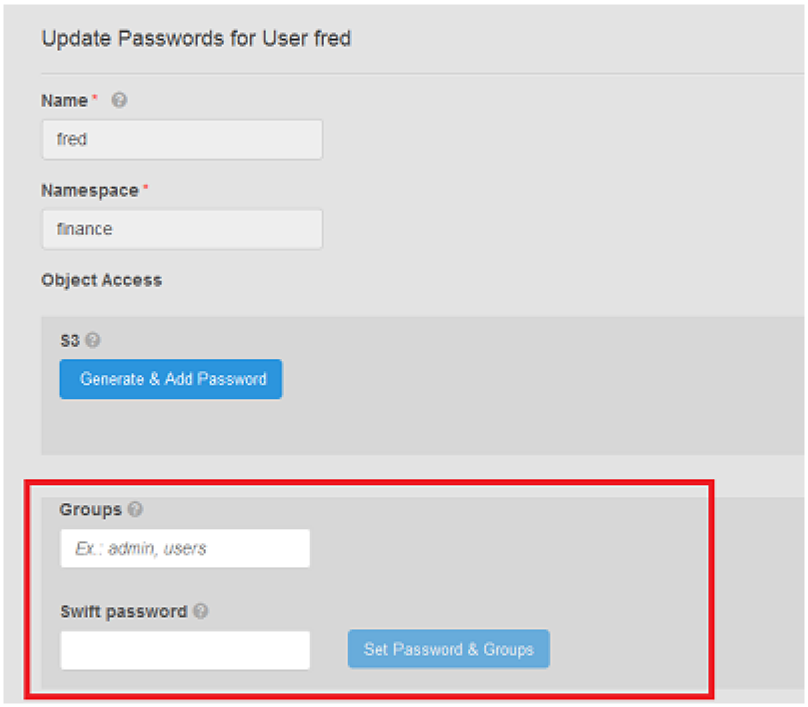
   The User Management page is displayed.

2. At the User Management page, either select New Object User or add a Swift password to an existing user by selecting the **Edit** action for a user listed in the users table.

   If you are creating a new user, you will need to enter a name for the user and select the namespace to which the users belongs.

3. In the Swift area, Groups field, enter a group that the user will belong to.

   The Swift area is highlighted in the figure below.


Update Passwords for User fred

Name *: fred

Namespace *: finance

Object Access

S3

Groups: Ex.: admin, users

Swift password

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 80 for more information on container authorization.

4. Enter a password for the Swift user.
5. Select **Set Password & Groups**.

OpenStack Version 1 authentication

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

Procedure

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

   You can do this from the ECS Portal (see Create Swift users at the ECS Portal on page 72) 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: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
      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: 1fO9X3xyrVhfcokqy3U1UyTY029gha5T+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 tenant/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 retrieve 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 and preceded by the "ecs_" prefix.

```
{"access": {"token":
{"id":"ecs_d668b72a011c4edf960324ab2e87438b","expires":"137663
3127950"l},"user":
                 {"name": "sysadmin", "roles":[ ],
"role_links":[ ] },"serviceCatalog":[ ] }} , }
```

2. Retrieve tenant info 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 scoped token along with 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 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" role/group in order 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 ECS to authenticate keystone users on page 78.

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 associate with the object to determine if the operation is allowed.
3. If the operation is a container operation, ECS evaluates the requested operation against the user's roles for the project, as follows:
    a. If the operation is a list containers operation and user has admin role, then allow
    b. If the operation is a create containers operation and user has admin role, then allow
    c. If the operation is an update container metadata operation and user has admin role, then allow
    d. If the operation is a read container metadata operation and user has admin role, then allow

e. If the operation is a delete containers operation and user has admin role, then allow

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 ECS to authenticate keystone users

To authenticate Keystone users, you must add an authentication provider to ECS and create a namespace that the Swift users belong to.

Before you begin

The following pre-requisites apply:

- You will need credentials for an ECS System Admin account.
- You will need to obtain the identity of the Keystone project to which Swift users that will access ECS belong.

Procedure

1. Log into ECS as a System Admin.
2. 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 Administrators Guide: Configure an Authentication Provider .
3. Create an ECS namespace that has the same ID as the Keystone project/ account that the users that wish to authenticate belong to.

   You will need to obtain the Keystone project ID.

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.

CHAPTER 12

Authorization

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 per 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 belongs to target group(s) can perform below 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 belongs to target group(s) can perform following operations:

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

Additional information on authorization can be found in: Container Operations

PART 3

EMC Atmos

CHAPTER 13

Introduction to EMC Atmos support in ECS

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 EMC 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 EMC Supportzone.

- 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.

CHAPTER 14

Atmos 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 the object and namespace interfaces are shown.

Table 6 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 /rest/namespace/<path>?metadata/user	Get user metadata for an object
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>	Get all object metadata

Table 6 Supported Atmos REST API calls (continued)

Method	Path	Description
	/rest/namespace/<path>	
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.
- Expiration and retention of objects is not supported.
- Read, Write, and Delete ACLs work in ECS the same as Atmos.

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.

In ECS Atmos, 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.

Unsupported EMC Atmos REST API Calls

The following Atmos REST API calls are not supported.

Table 7 Unsupported Atmos REST API calls

Method	Path	Description
Object Versioning		

Table 7 Unsupported Atmos REST API calls (continued)

Method	Path	Description
POST	/rest/objects/<objectID>?versions	Create a version of an object
DELETE	/rest/objects/<objectID>?versions	Delete an object version
GET	/rest/objects/<objectID>?versions	List versions of an object
PUT	/rest/objects/<objectID>?versions	Restore object version
Anonymous Access		
POST	/rest/accesstokens	Create an access token
GET	/rest/accesstokens/<token_id>?info	Get access token detail
DELETE	/rest/accesstokens/<token_id>	Delete access token
GET	/rest/accesstokens	List access tokens
GET	/rest/accesstokens/<token_id>	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: /rest/subtenant Required headers: x-emc-uid (for example, x-emc-uid=wuser1@SANITY.LOCAL) x-emc-signature. 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@SANITY.LOCAL) x-emc-signature

Note

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

CHAPTER 15

Atmos API Extensions

API Extensions

A number of extensions to the object APIs are supported.

The extensions and the APIs that support them are listed in the following table.

Table 8 Object API Extensions

Feature	Notes
PUT Object (range-append)	Uses Range header to specify object range appended. Appending data to an object on page 94

Appending data to an object

An example of using the ECS API extensions to append data to an object is provided below.

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-` 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-`. 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 retrieving the object again, you can see the full value `The quick green fox jumps over the lazy dog and cat.` You have appended data to this object.

PART 4

CAS

CHAPTER 16

Setting up CAS support in ECS

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 Administrators Guide: Configure storage pools, VDCs, and replication groups .

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:

- Administrators Guide: Configure a namespace
- Administrators Guide: Manage users and roles
- Administrators Guide: Create and manage buckets

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 9 Requirements for regular and cold archives compared

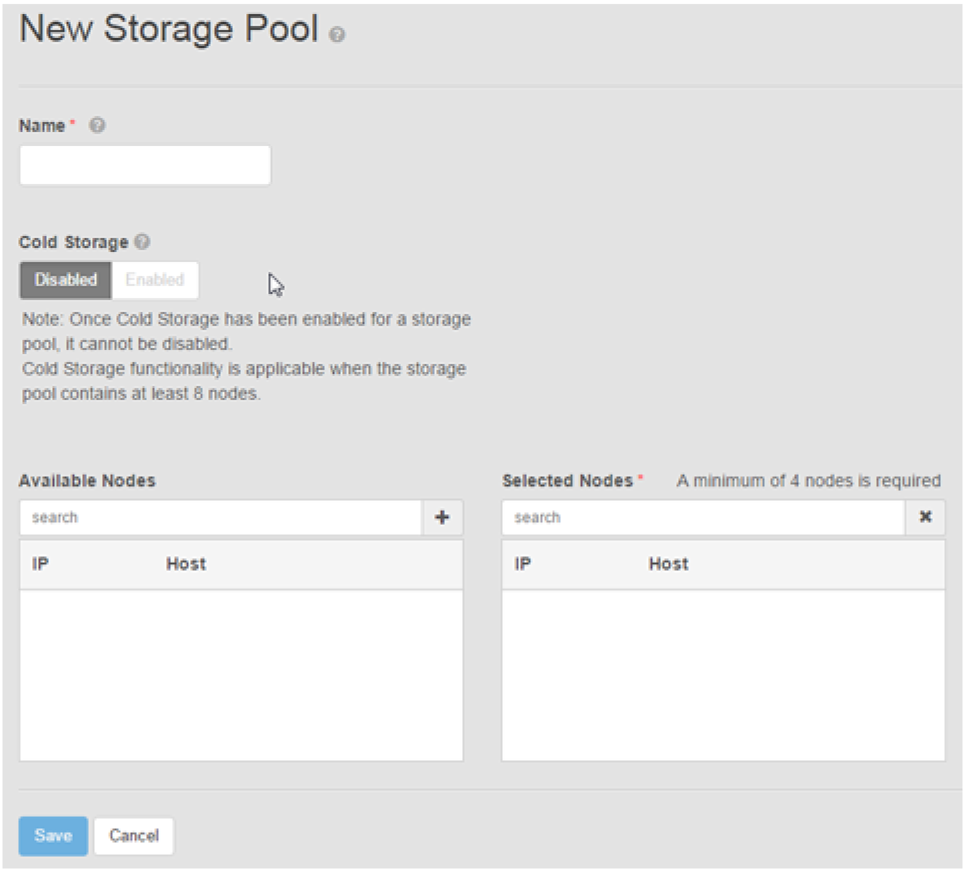
Use case	How enabled	Minimum required nodes	Minimum required disks	Recommended disks	EC efficiency	EC scheme
Regular archive	Default	4	16*	32	1.33x	12/16
Cold archive	Configured by System Administrator	8	12*	24	1.2x	10/12

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

[✓ Disabled] Enabled

Note: Once Cold Storage has been enabled for a storage pool, it cannot be disabled.
Cold Storage functionality is applicable when the storage pool contains at least 8 nodes.

Available Nodes

Selected Nodes * A minimum of 4 nodes is required

IP	Host

IP	Host

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 is certified on ECS Appliances with ECS version 2.2.1 software and later. Installations of ECS Software Only version 3.0 and later running on ECS-certified third-party hardware are also certified.

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

Describes the ECS security features supporting Compliance standards.

ECS platform security features:

- Root access to nodes is disabled; that is, no logins to the root account are permitted.
- ECS customers can access nodes through the `admin` account which is established at install time.
- Authorized `admin` account users run commands on nodes using `sudo`.
- There is full audit logging for `sudo` commands.
- ESRS provides the ability to shutdown all remote access to nodes. In ESRS Policy Manager, set the `Start Remote Terminal` action to **`Never Allow`**.
- All unnecessary ports, like `ftpd` , `sshd`, and so on are closed.
- The ECS Lock Admin (login: `emcsecurity`) can lock nodes in a cluster. This means that remote access over the network by SSH is disabled. The Lock Admin can then unlock a node to allow for remote maintenance activities or other authorized access. (Node locking does not affect authorized ECS Portal or ECS Management API users.) See the *ECS Administrator's Guide* for information on locking and unlocking nodes.

Compliance and retention policy

Describes enhanced rules for record retention on a compliance-enabled ECS system.

ECS has object retention features enabled or defined at the object-, bucket-, and namespace-level. Compliance strengthens these features by limiting changes that can be made to retention settings on objects under retention. Rules include:

- Compliance is enabled 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 enabled.
- Compliance can only be enabled on a namespace when the namespace is created. (Compliance cannot be added to an existing namespace.)
- Compliance cannot be disabled once enabled.
- All buckets in a namespace must have a retention period greater than zero.

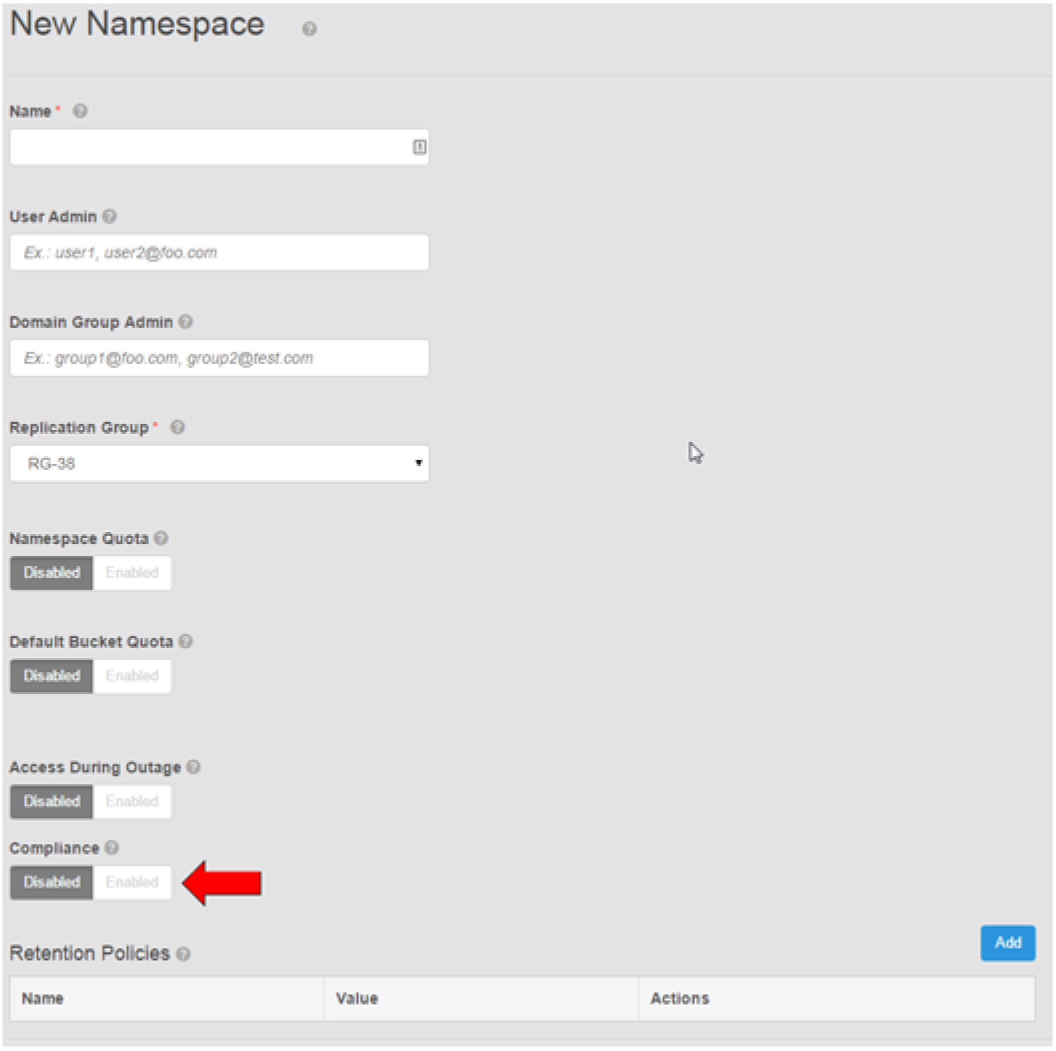
  Note

  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 will lead to application errors.

- A bucket with data in it cannot be deleted regardless of its retention value.
- Using the Infinite option on a bucket mean objects in the bucket in a Compliance-enabled namespace can never be deleted.
- The retention period for an object cannot be deleted or shortened. Therefore, the retention period for a bucket cannot be deleted or shortened.

- Object and bucket retention periods can be increased.
- No user can delete an object under retention. This includes users with the CAS privileged-delete permission.

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


New Namespace

Name *

User Admin
Ex.: user1, user2@foo.com

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

Replication Group *
RG-38

Namespace Quota
[✓ Disabled] Enabled

Default Bucket Quota
[✓ Disabled] Enabled

Access During Outage
[✓ Disabled] Enabled

Compliance
[✓ Disabled] Enabled

Retention Policies

Name	Value	Actions

Compliance agent

Describes the operation of the Compliance agent.

Compliance features are all enabled by default, except for Compliance monitoring. If monitoring is enabled, the agent periodically logs a message to a thread.

Note

Make arrangements with EMC Customer Support to enable 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.

Behavior change for 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.

This is a change from ECS 2.2.1 behavior which brings ECS in line with Centera behavior, where default pool retention in CE+ Compliance mode is always infinite (-1).

In ECS 2.2.1, 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 retention period assigned to the C-Clips will be zero (0). Here, the effective retention period for the C-Clips will be the bucket retention value (6 months). The C-Clips can be deleted in 6 months.

After upgrading from ECS 2.2.1 to ECS 3.0 or any later version, applications that rely on the ECS 2.2.1 behavior will be writing C-Clips that can never be deleted.

Halt and reconfigure your applications to assign appropriate object-level retention before they interact with ECS 3.0 or later versions.

In the example above, the application should assign 6 month object-level retention to the C-Clips.

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 polices for namespaces in the ECS Portal or with the ECS Management API. See Set up namespace retention policies.

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 10 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 here: ECS Management REST API. The online reference is here: ECS Management REST 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.

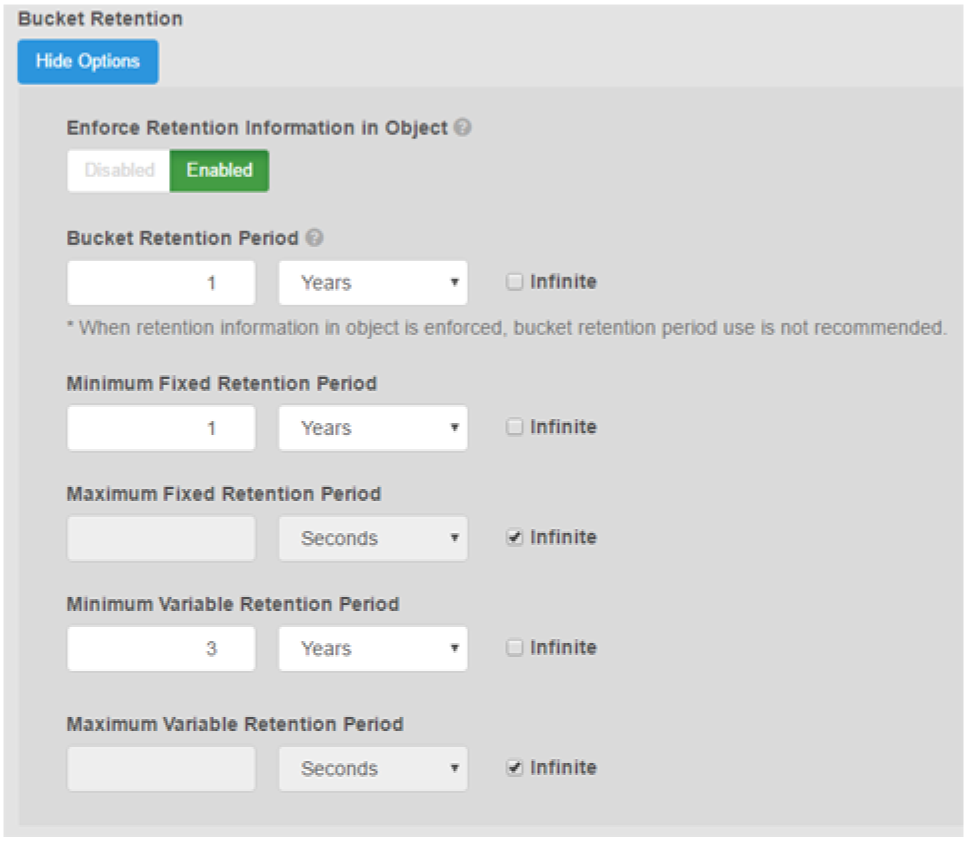
Note

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 controls shown on the screen below are CAS-only features except for the **Bucket Retention Period** control. **Bucket Retention Period** is the standard ECS bucket retention feature supported on all ECS bucket types.

Figure 2 Retention options for CAS buckets



Bucket Retention

Enforce Retention Information in Object: Disabled [✓ Enabled]

Bucket Retention Period: 1 Years ☐ Infinite

* When retention information in object is enforced, bucket retention period use is not recommended.

Minimum Fixed Retention Period: 1 Years ☐ Infinite

Maximum Fixed Retention Period: Seconds ☑ Infinite

Minimum Variable Retention Period: 3 Years ☐ Infinite

Maximum Variable Retention Period: Seconds ☑ Infinite

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

Feature	Description
Enforce Retention Information in Object	If this control is enabled, 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 enabled, it is possible not to configure **Bucket Retention Period** even in compliance-enabled environment. **Note** When a CE+ mode Centera is migrated to ECS, **Enforce Retention Information in Object** is enabled by default on the bucket.
Bucket Retention Period	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. 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.
Minimum Fixed Retention Period Maximum Fixed Retention Period	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. When using retention policies, the min/max settings are not enforced.

Feature	Description
	Selecting **Infinite** for **Minimum Fixed Retention Period** means all retention values must be infinite. Selecting if for **Mamimum Fixed Retention Period** means there is no maximum limit. 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.
Minimum Variable Retention Period	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. When using retention policies, the min/max settings are not enforced. Selecting **Infinite** for **Minimum Variable Retention Period** means all retention values must be infinite. Selecting if for **Mamimum Variable Retention Period** means there is no maximum limit. 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.
Maximum Variable Retention Period	

Note

If the System Adm 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:

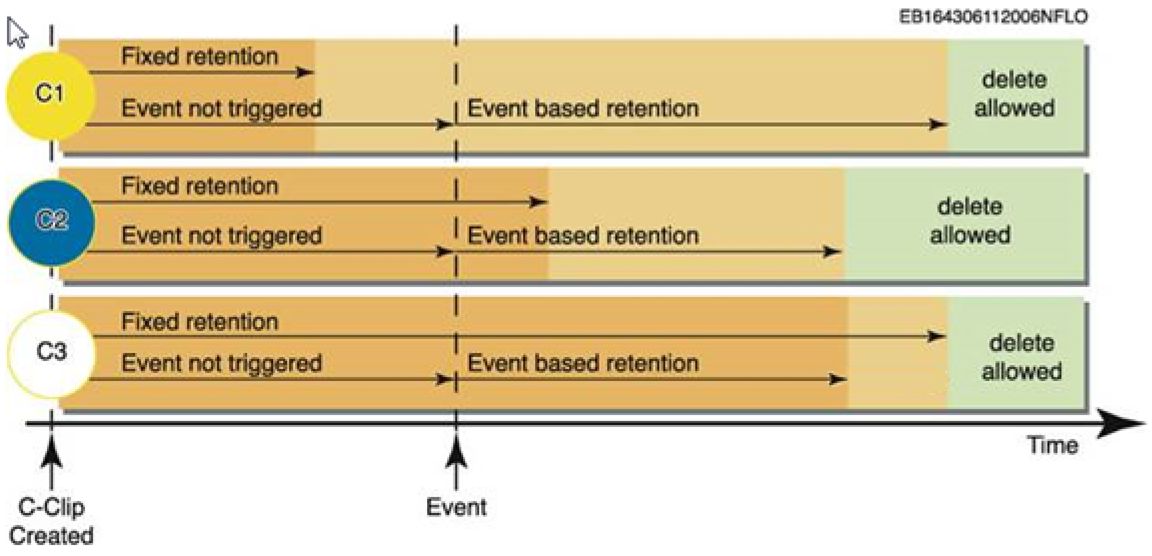
- ·**Create**: the application creates a new C-Clip and marks it as being under EBR. The application can provide a fixed retention period which acts as a minimum retention and it must provide an event based retention period or policy.
- ·**Trigger Event**: The application triggers the event, which is 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 11 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).

Table 11 CAS API functions for event-based retention (continued)

Function	Description
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.
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:

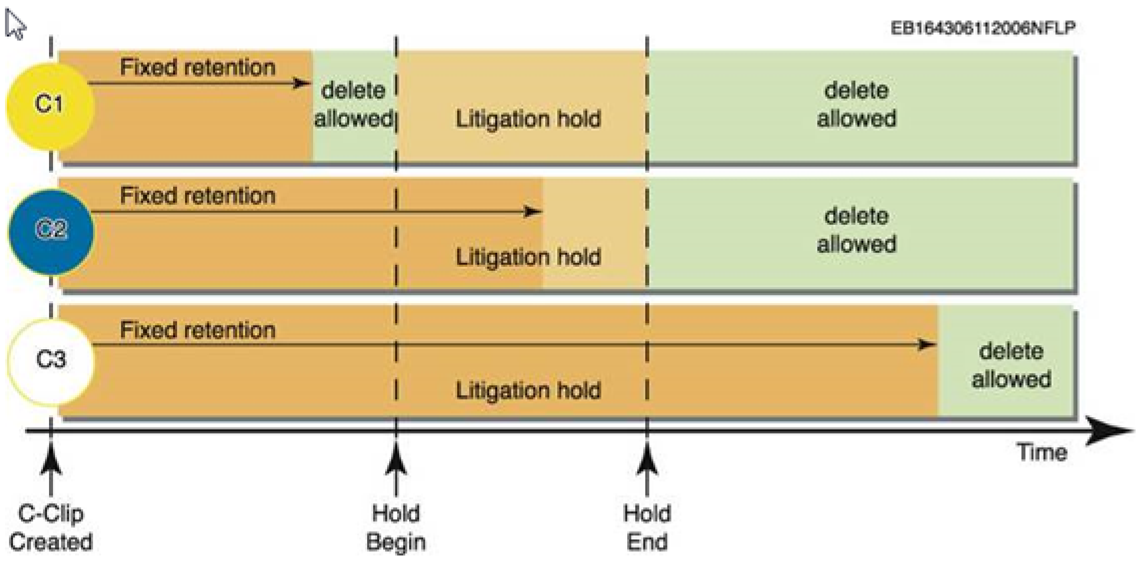
- Create: The application creates a new C-Clip and provides a fixed and/or event-based retention period.
- Set litigation hold: An application puts the C-Clip on hold. This application can be different from the application that wrote the C-Clip.
- Release litigation hold: An application releases the 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 12 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.

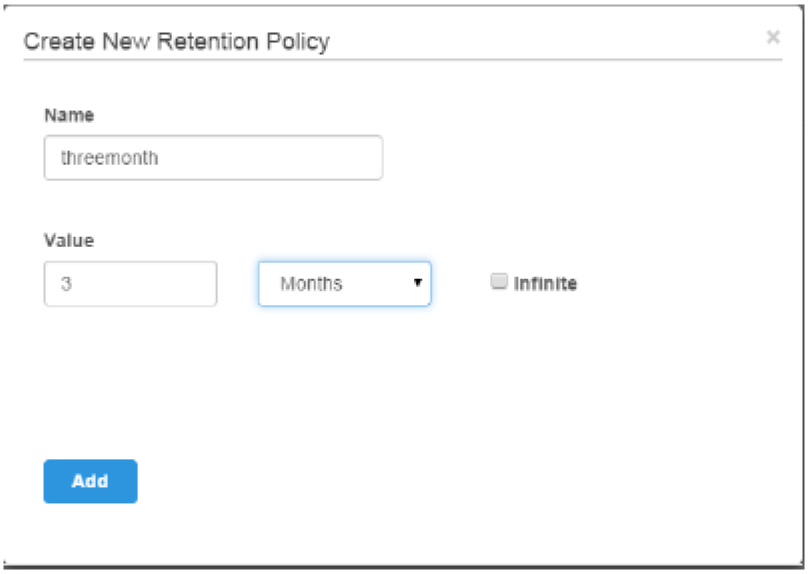
A namespace can have many retention polices, where each policy defines a retention period. By applying a retention policy to a number of C-Clips (with the API), a change to the retention policy changes the retention period for all objects associated with the policy. For CAS, retention classes are applied 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



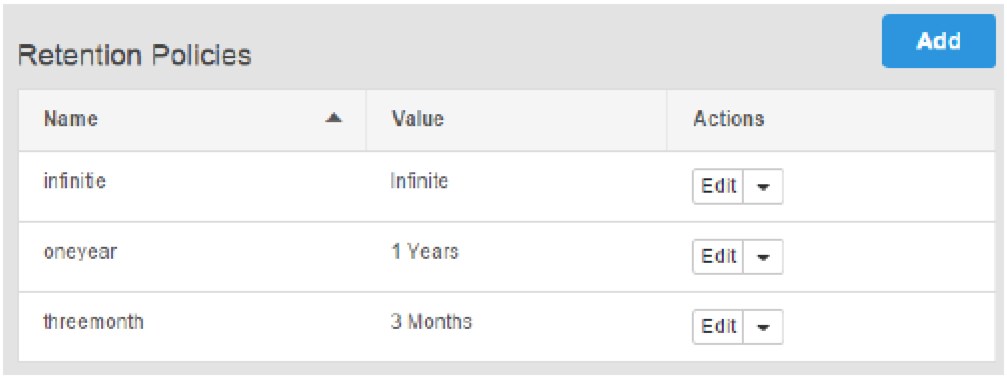
Create New Retention Policy

Name: threemonth

Value: 3 Months

4. Select **Save**.

Figure 6 Retention policies for a namespace



Retention Policies

Add

Name	Value	Actions
infinite	Infinite	Edit
oneyear	1 Years	Edit
threemonth	3 Months	Edit

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	Enabled

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

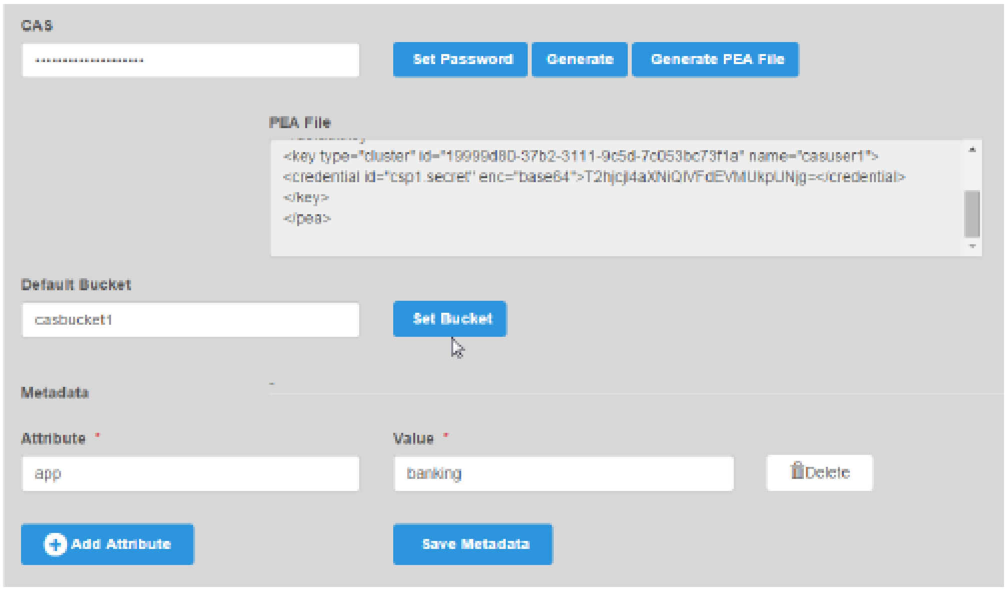
Set up an object user to use CAS.

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: ••••••••••••••••••••

PEA File:

```
<key type="cluster" id="19999d80-37b2-3111-9c5d-7c053bc73f1a" name="casuser1">
<credential id="csp1.secret" enc="base64">T2hjcjI4aXNIQlVFdEVMUkpUNjg=</credential>
</key>
</pea>
```

Default Bucket: casbucket1

Metadata

Attribute *	Value *
app	banking

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

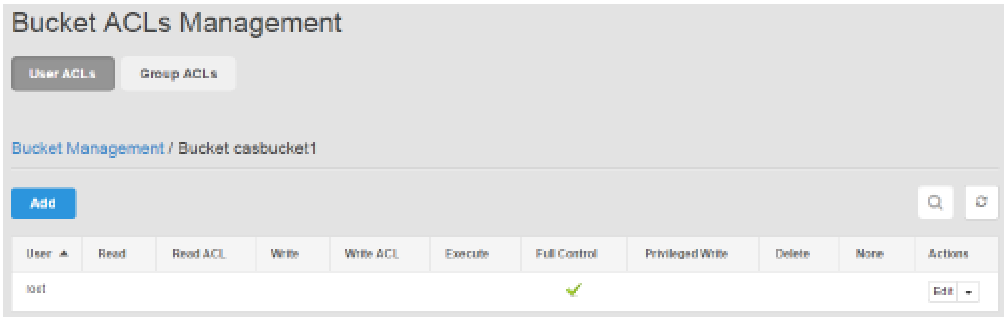
Edit a bucket's access control list to limit a user's access.

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

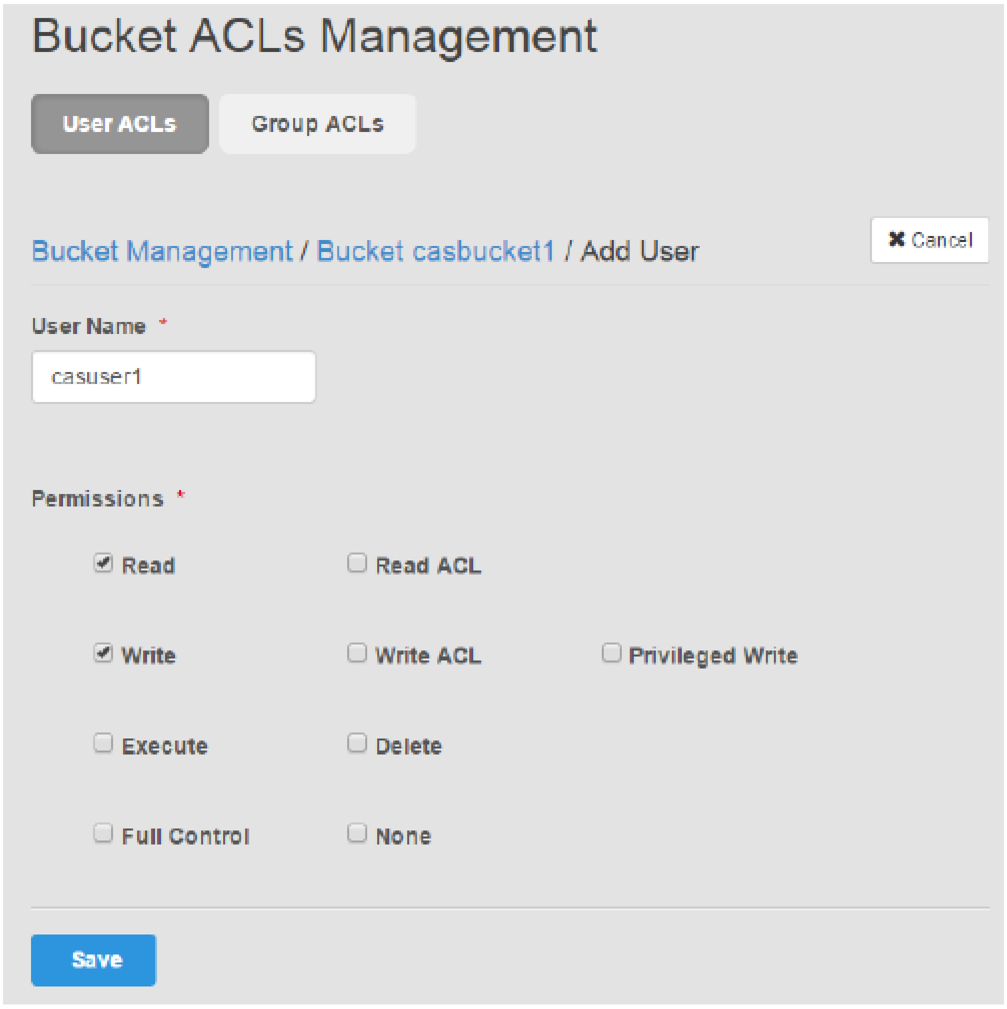


Bucket Management / Bucket casbucket1

User ▲	Read	Read ACL	Write	Write ACL	Execute	Full Control	Privileged Write	Delete	None	Actions
root						✓				Edit

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

Figure 9 Bucket ACLs Management


Bucket ACLs Management

Bucket Management / Bucket casbucket1 / Add User

User Name *

casuser1

Permissions *

[✓ Read] Read ACL

[✓ Write] Write ACL Privileged Write

Execute Delete

Full Control None

4. Modify the permissions.

Table 13 Bucket ACLs

ECS ACL	ACL definition
READ	Read, Query, and Exist capabilities
WRITE	Write and Litigation Hold capabilities
FULL_CONTROL	Read, Delete, Query, Exist, Clip Copy, Write, Litigation Hold
PRIVILEDGED_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 14 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 ECS Management REST API Reference 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 Configure support for CAS SDK applications with the ECS Portal.

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

PART 5

ECS Management API

CHAPTER 17

Introduction to the ECS Management REST API

ECS Management REST API

This part describes how to access the ECS Management REST API, it describes how to authenticate with it, and provides a summary of the API paths. The ECS Management REST API enables the object store to be configured and managed. Once the object store is configured, subsequent object create, read, update, and delete operations are performed using the ECS-supported object and file protocols.

You can refer to the following topic to get an understanding of how to access the REST API and how to authenticate:

- Authenticate with the ECS Management REST API on page 124

and the API paths are summarized in:

- ECS Management REST API summary on page 130

In addition, an API Reference is provided in:

- ECS Management REST API Reference

The ECS Management REST API Reference is auto-generated from the source code and provides a reference for the methods available in the API.

CHAPTER 18

Authentication with the ECS Management Service

Authenticate with the ECS Management REST API

ECS uses a token-based authentication system for all its REST API calls. Examples are provided for authentication with the ECS REST API, with cookies and without cookies.

Once a user is authenticated against ECS, an authentication token is returned and can be used to authenticate the user in subsequent calls.

- An HTTP 401 code is returned if the client is automatically following redirects, indicating that you need to login and authenticate to obtain a new token.
- An HTTP 302 code is returned if the client is not automatically following redirects. The 302 code directs the client to where to 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 along in subsequent requests.
- Reading the X-SDS-AUTH-TOKEN HTTP header from a successful authentication request and copying that header into any subsequent request.

The REST API is available on port :4443 and clients access ECS by issuing a login request in the form:

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

Authenticate with AUTH-TOKEN

This 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 here are written in curl and formatted for readability.

This command executes a GET on the /login resource. The -u option indicates the user of basic authentication header. The user designation must be included in the request. Upon successful authentication, a HTTP 200 code is returned as well as the X-SDS-AUTH-TOKEN header containing the encoded token.

The default management 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 2 hour; after a 2 hour idle time, the token will expire. If a user uses an expired token, they will be redirected to the "/login" URL and subsequent use of the expired token will cause HTTP status error code 401 to be returned.

```
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:
BAAcQ0xOd3g0MjRCUG4zT3NJdnNuMlAvQTFYblNrPQMAUAQADTEzODU0OTQ4NzYzNTIC
AAEABQA5dXJu

OnN0b3JhZ2VvczpUb2tlbjo2MjIxOTcyZS01NGUyLTRmNWQtYWZjOC1kMGE3ZDJmZDU3
MmU6AgAC0A8=
<
<?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):
```

The token can then be passed back in the next API call. You can copy the X-SDS-AUTH-TOKEN contents and pass it to the next request through curl's -H switch.

```
curl https://10.247.100.247:4443/object/namespaces
     -k
     -H "X-SDS-AUTH-TOKEN:
BAAcOHZLaGF4MTl3eFhpY0czZ0tWUGhJV2xreUE4PQMAUAQADTEzODU0OTQ4NzYzNTIC
AAEABQA5dXJu

OnN0b3JhZ2VvczpUb2tlbjpkYzc3ODU3Mi04NWRmLTQ2YjMtYjgwZi05YTdlNDFkY2Qw
ZDg6AgAC0A8="
```

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

Authenticate with cookies

This example shows how to use authentication tokens by saving the cookie from a successful authentication request, then passing the cookie in a subsequent request. The examples here are written in curl and formatted for readability.

In this example, you specify the ?using-cookies=true parameter to indicate that you want to receive cookies in addition to the normal HTTP header. This 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 next command passes the cookie with the authentication token through the -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</name>
   </namespace>
</namespaces>
```

Logout

The logout API ends a session.

A given user is allowed a maximum of 100 concurrent authentication tokens. Past this limit, the system refuses any new connection for this user until tokens free up. They can free up by expiring naturally, or by explicitly calling this URI:

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

If you have multiple sessions running simultaneously, this URI forces the termination of all tokens related to the current user.

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

An example logout request follows.

Request

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

X-SDS-AUTH-TOKEN:{Auth_Token}
```

Pass in the header or cookie with the authentication token to logout.

Response

```
HTTP 200
```

Whoami

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
```

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>
```

This example 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.

CHAPTER 19

ECS Management REST API Summary

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 15 ECS Management REST API- methods summary

API Area	Description
Configuration	

Certificate	`/object-cert`
	API for managing certificates.
	`/object-cert/keystore`
	API to enable the certificate chain used by EMC to be specified and for the certificate to be rotated.

Configuration Properties	`/config/object/properties` API to enable the user scope to be set as GLOBAL or NAMESPACE. 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. If the user scope is NAMESPACE, a user is associated with a namespace, so there might be more than user with the same name, each associated with a different namespace. In NAMESPACE mode a namespace must be provided for every operation. Must be set before the first user is created. The default is GLOBAL.
Licensing	`/license` API to enable a license to be added and license details to be retrieved.
Feature	`/feature/` API for retrieving the details for a feature.
Syslog	`/vdc/syslog/config` API for managing Syslog configuration and sending alerts to Syslog server for troubleshooting and debugging purposes.
SNMP	`/vdc/snmp/config` API for managing SNMP configuration and sending alerts to SNMP server for troubleshooting and debugging purposes.
CAS	
CAS User Profile	`/object/user-cas` API to enable secret keys to be assigned to CAS users and enables Pool Entry Authorization (PEA) file to be generated.
File System Access	

Table 15 ECS Management REST API- methods summary (continued)

API Area	Description
NFS	`/object/nfs` API to enable the creation of an NFS export based on an ECS bucket and to enable acxess to the export by Unix users and groups.
Metering	
Billing	`/object/billing` API to enable the metering of object store usage at the tenant and bucket level. See Administrators Guide: Manage tenants for more information.
Migration	
Transformation	`/object/transformation` API to enable data transformation.
Monitoring	
Capacity	`/object/capacity` API for retrieving the current managed capacity.
Dashboard	
Alerts	`/vdc/alerts` API for retrieving audit alerts.
Events	`/vdc/events` API to return the audit events for a specified namespace.
Multi-tenancy	
Namespace	`/object/namespaces` API to enable a namespace to be created and managed. Also enables the retention period for the namespace to be set. See Administrators Guide: Manage tenants for more information.
Geo-replication	
Replication Group	`/data/data-service/vpools` API to enable the creation and administration of replication groups.
Temporary Failed Zone	`/tempfailedzone/` API to enable all temporary failed zones or the temporary failed zones for a specified replication group to be retrieved.
Provisioning	
Base URL	`/object/baseurl` API to enable the creation of a Base URL that allows existing applications to work with the ECS object store. More information on Base URL can be found in: Administrators Guide: Set the Base URL .

Table 15 ECS Management REST API- methods summary (continued)

API Area	Description
Bucket	`/object/bucket` API for provisioning and managing buckets. `/object/bucket/{bucketName}/lock` API to enable bucket access to be locked. See Administrators Guide: Manage tenants for more information.
Data Store	`/vdc/data-stores` API to enable the creation of data stores on file systems (`/vdc/data-stores/filesystems`) or on commodity nodes (`/vdc/data-stores/commodity`).
Node	`/vdc/nodes` API for retrieving the nodes that are currently configured for the cluster.
Storage Pool	`/vdc/data-services/varrays` API to allow the creation and management of storage pools.
Virtual Data Center	`/object/vdcs` Enables a VDC to be added and its inter VDC endpoints and secret key to be specified for replication of data between ECS sites.
VDC Keystore	`/vdc/keystore` API for managing the certificates for a VDC.
Support	
Call Home	`/vdc/callhome/` API to enable ConnectEMC to be configured and create alert events for ConnectEMC.
CLI Package	`/cli` API to download the ECS package.
User Management	
Authentication Provider	`/vdc/admin/authproviders` API to enable authentication providers to be added and managed.
Password Group (Swift)	`/object/user-password` API to enable a password to be generated for use with OpenStack Swift authentication.
Secret Key	`/object/user-secret-keys` API to allow secret keys to be assigned to object users and to enable secret keys to be managed.
Secret Key Self-Service	`/object/secret-keys`

Table 15 ECS Management REST API- methods summary (continued)

API Area	Description
	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.
User (Object)	`/object/users` API for creating and managing 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. `/object/users/lock`. Enables user access to be locked.
User (Management)	`/vdc/users` API for creating and managing management users. Management users can be assigned to the System Admin role or to the Namespace Admin role. Local management user password can be changed.

PART 6

HDFS

CHAPTER 20

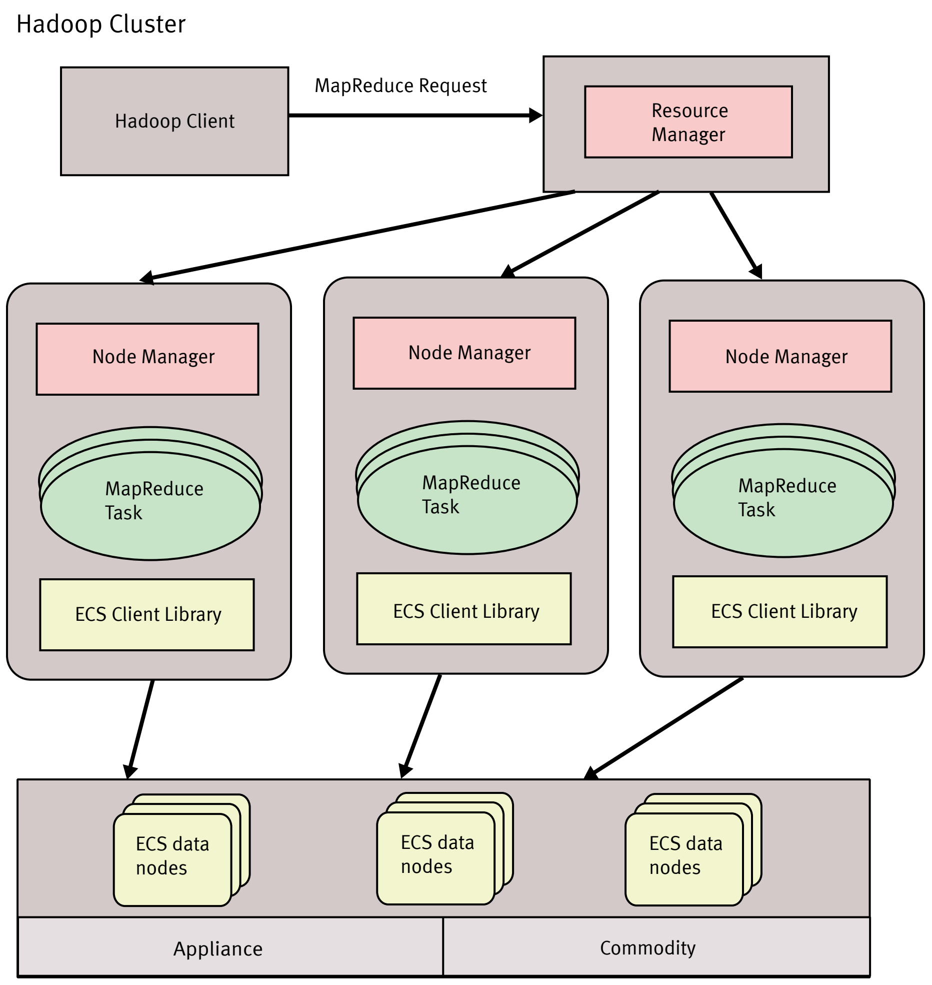
What is ECS HDFS?

What is ECS HDFS?

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

You can configure your Hadoop distribution to run against the built-in Hadoop file system, against ECS HDFS, or any combination of HDFS, ECS HDFS, or other Hadoop Compatible File Systems available in your environment. 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 configured to use ECS HDFS, each of the ECS HDFS data nodes functions as a traditional Hadoop NameNode which means that all of the ECS HDFS data nodes are capable of accepting HDFS requests and servicing them.

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 HDFS 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 already installed and configured. The following distributions are supported:
  - Hortonworks 2.0, 2.1, 2.2, 2.3

  Note

  HDFS on ECS has not been officially certified by Cloudera or Pivotal HD. This is part of the future ECS Hadoop roadmap.

  When using the Hortonworks distribution, you can use Hortonworks Ambari. Hortonworks 2.3 (Hadoop 2.7) comes complete with an ECS stack that can be enabled to simplify integration with ECS HDFS. The instruction for using this distribution are provided in: Deploying a Hortonworks cluster with Ambari on page 156.
- Hadoop installed and configured to support ECS HDFS, which requires:
  - One or more filesystem enabled buckets used for HDFS access.
  - The ECS Client Library deployed to the cluster.
- For a Hadoop cluster that uses Kerberos or Kerberos with Active Directory.
  - Kerberos service principals deployed to the ECS nodes
  - Secure metadata deployed to the bucket.

Configuring Hadoop to use ECS HDFS

Hadoop stores system configuration information in a number of files, such as `core-site.xml`, `hdfs-site.xml`, `hive-site.xml`, etc. Editing `core-site.xml` is a required part of the ECS HDFS configuration.

There are several types of properties to add or modify in `core-site.xml` including:

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

`core-site.xml` 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, it is always recommended to use the management interface rather than hand-editing files. In addition, changes made using the management interface are persisted across the cluster.

ECS HDFS URI for file system access

After you configure Hadoop to use the ECS file system, you can access it by specifying the ECS HDFS URI with `viprfs://` as the scheme and a combination of ECS bucket, tenant namespace, and user-defined installation name for the authority.

The ECS HDFS URI looks like this:

```
viprfs://bucket_name.namespace.installation/path
```

The *bucket_name* corresponds to a HDFS-enabled bucket. It contains the data you want to analyze with Hadoop. The *namespace* corresponds to a tenant namespace, and the *installation_name* is a name you assign to a specific set of ECS nodes or a load balancer. ECS HDFS resolves the *installation_name* to a set of ECS nodes or to a load balancer by using the fs.vipr.installation.[*installation_name*].hosts property, which includes the IP addresses of the ECS nodes or load balancer.

If the installation_name maps to a set of ECS ECS nodes, you can specify how often to query ECS for the list of active nodes by setting the fs.vipr.installation.[*installation_name*].resolution to dynamic, and the fs.vipr.installation.[*installation_name*].resolution.dynamic.time_to_live_ms to specify how often to query ECS for the list of active nodes.

You can specify the ECS HDFS URI as the default file system in `core-site.xml`, for both simple and Kerberos environments, by setting it as the value of the `fs.defaultFS` property, but this is not a requirement. Where ECS HDFS is not the default file system, you must use the full URI including the path each time you access ECS data. If you have existing applications that already use a different default file system, you need to update those applications.

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 Kerberized operation, the identity of a client process is determined by its Kerberos credentials. For example, in a Kerberized environment, a user may use the kinit utility to obtain a Kerberos ticket-granting-ticket (TGT) and use klist to determine their current principal. When mapping a Kerberos principal to an HDFS username, using auth_to_local Hadoop property, all components except for the primary are dropped. For example, a principal `todd/foobar@CORP.COMPANY.COM` will act 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 is secured using uses Kerberos, ECS can be configured to grant access to users with Kerberos principals in the form `user@REALM.COM`, alternatively, where ECS uses AD to authenticate users, a one-way trust can be configured between the Kerberos environment and AD so that users can authenticate using their AD credentials, in the form `user@DOMAIN.COM`.

Files and directories permissions are determined 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 tell ECS to make it available as a file system.

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

The bucket owner is the owner of the root filesystem and the permissions assigned to that owner on the bucket translate to permissions on the root filesystem. In addition, bucket ACLs need to be assigned so that every user that is required to access the HDFS filesystem has permission on the bucket. This can be done by explicitly adding a user ACLs for every user on the bucket, or by specifying custom group ACLs. See Bucket Custom Group ACLs and Default Group on page 141. The owner of the bucket must be an ECS object user, other users do not and can be Unix usernames from the Hadoop cluster.

Once users have access to the file system, either because they own it, or have been added as a user to the bucket, or because they are a member of the group that the filesystem belongs to, the files and directories that they create will have permissions determined by the umask property in `core-site.xml`.

ECS also supports the superuser and supergroup mechanism for enabling access to HDFS.

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 that will access applications on the Hadoop cluster) and you can create corresponding groups at the ECS Portal and assign permissions to them.

It is also possible to assign a *Default Group* for the bucket. The Default Group will be the group assigned to the root '/' filesystem. For example, if your bucket owner is hdfs and the Default Group is set to hadoop, '/' will be set to hdfs:hadoop as user and group, respectively. A Default Group is also a custom group and will display in the Custom Group ACL list.

If a Default Group is not defined, the root of the filesystem has no group as shown below.

```
drwx---rwx+ - hdfs 0 2015-12-09 12:30 /
```

If a Default Group of "hadoop" is defined, the ownership and permissions will be as show below.

```
drwxrwxrwx+ - hdfs hadoop 0 2015-12-09 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 hdfs dfs -chgrp and hdfs dfs -chmod.

Hadoop superuser and supergroup

The superuser in a Hadoop environment is the user that starts the namenode, usually hdfs or hdfs@REALM.COM. As far as the ECS HDFS is concerned, the superuser is the owner of the bucket. Hence, 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.supergroup property in `core-site.xml`. 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 will have full (read and write) access to the bucket. Users without superuser privileges will normally have read access, but that will depend 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 needs to match a Unix local, Kerberos, or AD user (depending on 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 own by that group. This default Unix group is assigned to objects when they are created from S3, so that when accessed by HDFS they will not only have an owner, but can also have group membership and permissions that enable access from the Hadoop cluster.

Files created using HDFS and accessed using S3 protocol will not be affected by the default permissions, they are only applied when they are created using the S3 protocol.

Hadoop Kerberos authentication mode

When Kerberos and the ECS Active Directory 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 Active Directory realm used to authenticate your ECS users.

The following identity translation properties in `core-site.xml` 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 `core-site.xml` to define a service principal:

- viprfs.security.principal

Proxy user

ECS HDFS supports the use of the Hadoop proxy user.

A proxy user allows a Hadoop superuser 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.

SymLink support

In standard HDFS, a symbolic link that does not specify the full URI to a file points to a path in the same HDFS instance.

The same rule is true in ECS HDFS. When you do not specify the full URI in a symlink, ECS HDFS uses the current namespace and bucket as the root. To provide a symlink to a file outside of the current namespace and bucket, you must provide the full URI that includes both the scheme and the authority.

Note

Hadoop 2.2 does not support SymLinks.

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 an inbuilt 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 chown them 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 will need to 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 will 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 will be the same when accessing them from a simple or Kerberized cluster. When accessing from a simple evironment, group membership is determined from the Unix box, when accessing from a Kerberized cluster you can configure group membership by assigning the mapping.

When using Active Directory credentials, the mapping between AD principals and Unix principals is achieved by removing the domain suffix, so user `hdfs@domain.com` becomes hdfs. This in 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 Active Directory, an Authentication Provider must have been configured in ECS so that membership of the group can be checked.

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 and unsupported Hadoop applications

ECS HDFS supports the majority of applications in the Hadoop ecosystem.

Supported applications
The following applications in the Hadoop ecosystem are supported:

- HDFS
- MapRedeuce
- Yarn
- Pig
- Hive
- HBase

Unsupported applications
The following applications in the Hadoop ecosystem are not supported:

- HttpFS
- Hue
- Cloudera Impala

CHAPTER 21

Create a bucket for the HDFS filesystem

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

- You must be a Namespace Admin or a System Admin to create a bucket at the ECS portal.
- If you are a Namespace Admin you can create buckets in your namespace.
- If you are System Admin you can create a bucket belonging to any namespace.

You will need to ensure that Hadoop users and services have access to the HDFS file system (the bucket) and that files and directories that they create 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 will automatically be assigned 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.

You can read more about users and permissions in Accessing the bucket as a file system on page 141 and typical bucket user permissions are shown in Example Hadoop and ECS bucket permissions on page 152.

Procedure

1. At the ECS Portal, select **Manage** > **Buckets** > **New Bucket**.
2. Enter a name for the bucket.

   Note

   You should not use underscores in bucket names as they are not supported by the URI Java class. For example, `viprfs://my_bucket.ns.site/` will not work as this is an invalid URI and is thus not understood by Hadoop.

   ---

3. Specify the namespace that the bucket will belong to.
4. Select a Replication Group or leave blank to use the default replication group for the namespace.
5. Enter 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 will require superuser privileges on the HDFS and this can be achieved by making hdfs the owner of the bucket. Other Hadoop users may also require superuser privileges and these privileges can be granted by assigning users to a group and making that group a superuser group.

6. Do not enable CAS.

   Note

   A bucket that is intended for use as HDFS cannot be used for CAS. The CAS control is disabled when File System is enabled.

7. Enable any other bucket features that you require.

   You can enable any of the following features on a HDFS bucket:

   - Quota
   - Server-side Encryption
   - Metadata Search
   - Access During Outage
   - Compliance (see note)
   - Bucket Retention

   Refer to Administrators Guide: Create and manage buckets for information on each of these settings and how to configure them.

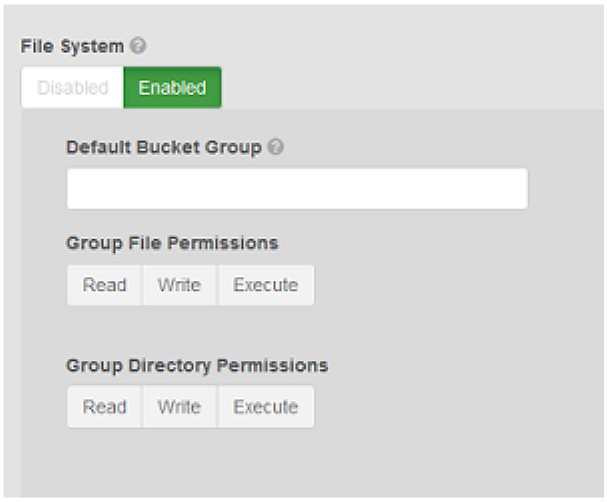
   Note

   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. Select Enabled for the File System.

   Once enabled, controls for setting a Default Group for the filesystem/bucket and for assigning group permissions for files and directories created in the bucket are available.

9. At the File System panel, shown below, enter a name for the Default Bucket Group.



This group will be the group associated with the HDFS root filesystem and enables Hadoop users who are members of the group to access the HDFS.

It 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 Active Directory group. When the Hadoop users want to access the S3 data, they will be able to do so because the files were uploaded and assigned to that group

This group must be specified at bucket creation. If it is not, the group would have to be assigned later from Hadoop by the filesystem owner.

10. Set the default permissions for files and directories created in the bucket using the object protocol.

    These setting are used to apply Unix group permissions to objects created using object protocols. These permissions will apply to the HDFS group (the Default Bucket Group) when the object or directory is listed from Hadoop. You can refer to Multi-protocol (cross-head) access on page 142 for more information on setting the Default Group and permissions for the file system.

    a. Set the Group File Permissions by clicking the appropriate permission buttons.

       You will normally set Read and Execute permissions.

    b. Set the Group Directory Permissions by clicking the appropriate permission buttons.

       You will normally set Read and Execute permissions.

11. Click **Save** to create the bucket.

Set custom group bucket ACLs

The ECS Portal enables the group ACL for a bucket to be set. Bucket ACLs can be granted for a group of users (Custom Group ACL) or 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

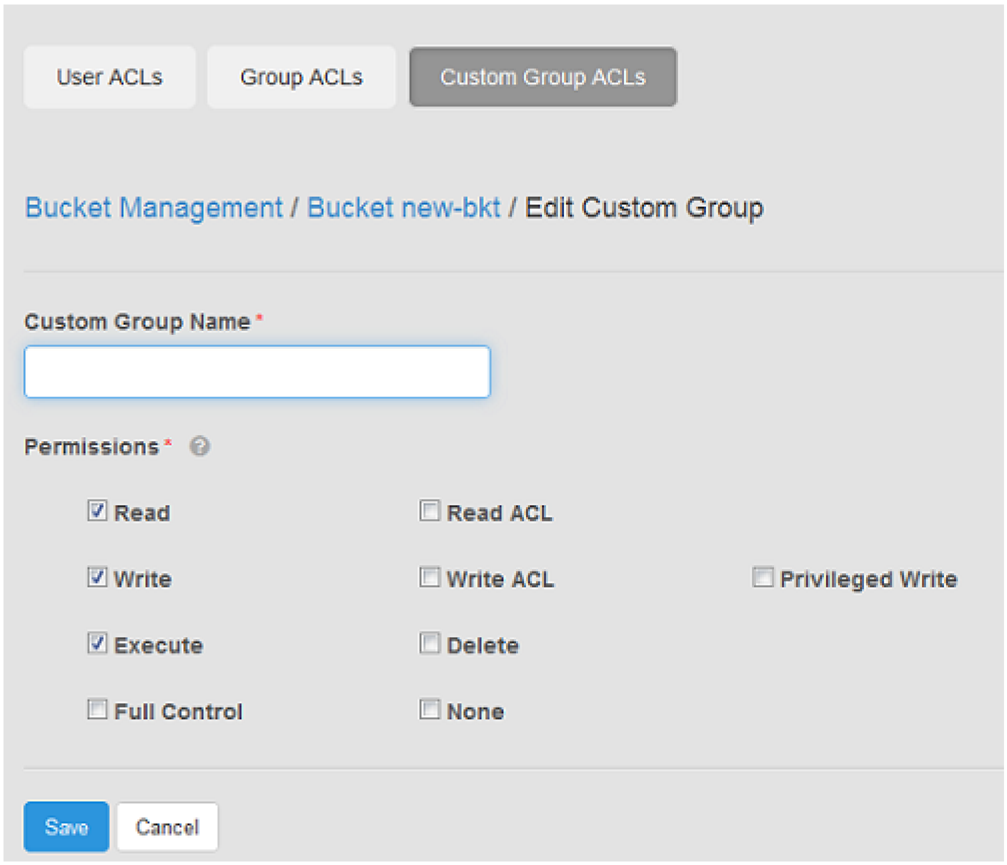
- You must be a Namespace Admin or a System Admin to edit the group ACL for a bucket.
- If you are a Namespace Admin you can edit the group ACL settings for buckets belonging to your namespace.
- If you are System Admin you can edit the group ACL settings for a bucket belonging to any namespace.

When the bucket is accessed using HDFS, using ECS multi-protocol access, members of the Unix group will be able to access the bucket.

Procedure

1. At the ECS Portal, select **Manage** > **Buckets**.

2. In the Buckets table, select the **Edit ACL** action for the bucket for which you want to change the settings.
3. To set the ACL for a custom group, select **Custom Group User ACLs**.
4. At the **Custom Group User ACLs** page, select **Add**.



Bucket Management / Bucket new-bkt / Edit Custom Group

Custom Group Name *

Permissions *

✓ Read	☐ Read ACL	
✓ Write	☐ Write ACL	☐ Privileged Write
✓ Execute	☐ Delete	
☐ Full Control	☐ None	

5. Enter the name for the group.

   This name can be a Unix/Linux group, or an Active Directory group.
6. Set the permissions for the group.

   At a minimum you will want to assign Read, Write, Execute and Read ACL.
7. Select **Save**.

Set the bucket ACL permissions for a user

The ECS Portal enables the ACL for a bucket to be set for a user. The bucket owner is assigned permissions automatically. Other Hadoop users can be assigned User ACLs to enable access to the bucket/filesystem, 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 Admin or a System Admin to edit the ACL for a bucket.
- If you are a Namespace Admin you can edit the ACL settings for buckets belonging to your namespace.

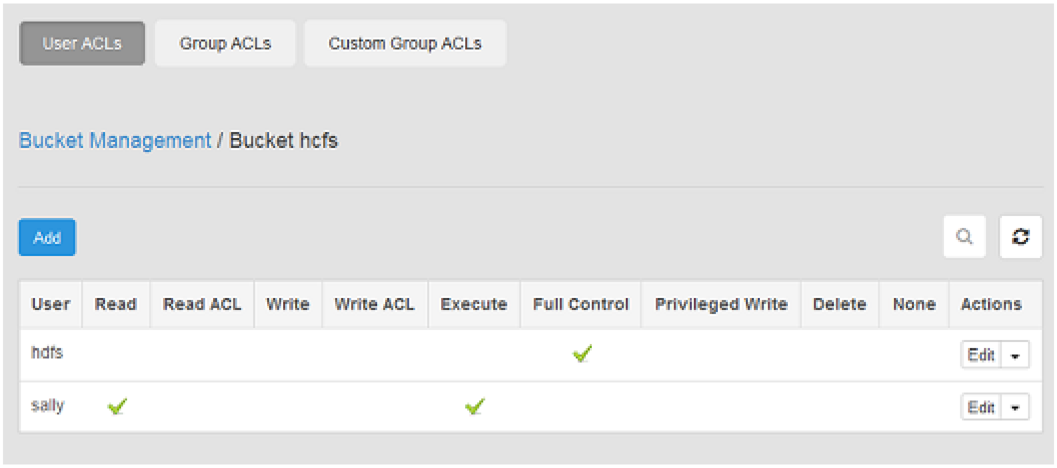
- If you are System Admin you can edit the ACL settings for a bucket belonging to any namespace.

Procedure

1. At the ECS Portal, select **Manage** > **Buckets**.
2. In the Buckets table, select the **Edit ACL** action for the bucket for which you want to change the settings.
3. To set the ACL permissions for a user, select the **User ACLs** button.
4. You can edit the permissions for a user that already has permissions assigned, 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 to which you want to assign permissions, select **Add**.

   The user that you have set as the bucket owner will have already have default permissions assigned.

   The bucket shown below is owned by the "hdfs" users and, as the owner, has been given full control. Full control translates to read-write-execute permissions in a Hadoop/Unix environment. User "sally" has been give read-execute permissions to enable that user to access the bucket.



User	Read	Read ACL	Write	Write ACL	Execute	Full Control	Privileged Write	Delete	None	Actions
hdfs						✓				Edit
sally	✓				✓					Edit

   More information on ACL privileges is provided in Administrators Guide: Create and manage buckets .

5. If you have added an ACL, enter the username of the user that the permissions will apply to.
6. Specify the permissions that you want to apply to the user.
7. Select **Save**.

Example Hadoop and ECS bucket permissions

Examples of the relationship between Hadoop users and groups and the users and groups assigned permission to access the bucket through ECS User ACLs and Custom Group ACLs are provided here.

On bucket creation, the bucket owner and the Default Group determine the owner and the group assignment for the bucket when accessed using HDFS, and ACLs are automatically assigned to them. A bucket must always have an owner, however, a Default Group does not have to be assigned. Other users and groups (called Custom Groups), apart from the bucket owner and the Default Group, can be assigned ACLs on the bucket and ACLs assigned in this way translate to permissions for Hadoop users.

Table 16 Example bucket permissions for filesystem access in simple mode

Hadoop Users and Groups	Bucket Permissions	Bucket/Filesystem 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 **User ACL** hdfs (owner)	Custom Group ACLs set on the bucket to enable the hadoop and users group to have permissions on the bucket/root filesystem This example assumes that hdfs 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 **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 should be defined (hadoop) so that Hadoop users and services are granted permissions on the files due to group membership. Default Group automatically has Custom Group ACLs on the bucket/filesystem. As is Default Group set, the root filesystem will have 777: `drwxrwxrwx+ - s3user hadoop 0 2015-12-09 12:28 /` Users can be given access either by adding User ACLs or by adding Custom Group ACLs for the group to which the users belong.

Table 17 Example bucket permissions for filesystem access in Kerberos mode

Hadoop user	Bucket Permissions	Bucket/Filesystem Access
Users (service) hdfs@REALM.COM. mapred@REALM.COM, yarn@REALM.COM, hive@REALM.COM, pig@REALM.COM **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	**Bucket owner** hdfs@REALM.COM **Default Group** hadoop **Custom Group ACL** hadoop (default), users **User ACL** hdfs@REAL.COM (owner)	Custom Group ACLs set on the bucket to enable the hadoop and users group to have permissions on the bucket/root filesystem User information from the Hadoop cluster must be available to ECS so that it can provide secure access to the bucket. Information about this metadata is provided in: Example Hadoop and ECS bucket permissions on page 152 and an example metadata file is provided here.

CHAPTER 22

Use Hortonworks Ambari to set up Hadoop with ECS HDFS

Deploying a Hortonworks cluster with Ambari

Ambari makes it easy to deploy a Hortonworks Hadoop cluster and uses the concept of a stack to bring together the services that are required for a specific release of Hadoop. Hortonworks 2.3 (Hadoop 2.7) provides a custom Hadoop stack for ECS that simplifies the integration of Hadoop with ECS. For other Hortonworks releases you can use Ambari in its normal mode.

The Hortonworks Hadoop ECS (HDP ECS) stack makes it easy to integrate ECS HDFS with Hadoop by deploying the ECS Client Library to all Hadoop nodes and simplifying the configuration of the cluster to use the ECS HDFS.

To deploy and configure the Hortonworks HDP ECS stack, perform the following steps:

1. Download Ambari
2. Download the ECS HDFS Client Library
3. Set up a local repository from which to deploy the ECS Client Library
4. Install the Ambari server
5. Enable the Ambari ECS stack
6. Install the Ambari Agent manually
7. Install the Hadoop cluster

Download Ambari

Download Ambari 2.2.

Ambari can be used to install and manage a Hadoop (HDP) distribution. Ambari 2.2 provides the ability to install the Hortonworks Hadoop ECS stack

You can download the Ambari repository from the following link:

```
http://hortonworks.com/hdp/downloads/
```

You should download the Ambari repository to all nodes in your cluster to enable the Ambari Server to be installed on the server node and Ambari Agent to be installed on all nodes.

Download the ECS HDFS Client Library

Download the ECS HDFS Client Library RPM from the ECS Support Zone.

```
http://support.emc.com
```

Set up a local repository from which to deploy the ECS Client Library

Set up a local repository from which Ambari can deploy the ECS Client Library with the Hadoop stack.

Before you begin

Setting up a repository will normally involve using a package manager to create a set of metadata about the packages contained in a repository directory, and providing a mechanism for accessing the repository directory, such as over HTTP, or over a network.

There are a number of tools that can be used to create a repository. Information on using `yum` to create a package repository is provided here.

Procedure

1. Create the local repository.
2. Add the ECS Client Library RPM to the local repository.

Install the Ambari server

Install the Ambari server.

The basic commands for installing and setting up the Ambari sever as provided in this procedure. More complete information can be found in the Hortonworks documentation, here.

Procedure

1. Install the Ambari server using the following command.

```
yum install ambari-server -y
```

2. Set up the Ambari sever, as follows.

```
ambari-server setup -s
```

3. Start the Ambari server, as follows

```
ambari-server start
```

Enable the Ambari Hadoop ECS stack

The Hortonworks Hadoop (HDP) ECS stack is disabled by default and must be enabled before it can be selected for deployment. The stack can be enabled from the command line.

Procedure

1. From the Ambari server machine, open a command prompt.
2. Run the following command:

   ```
   ambari-server enable-stack --stack HDP --version 2.3.ECS
   ```

   If you want to enable more than one stack, you can include the `--version` option for each additional stack. For example:

   ```
   ambari-server enable-stack --stack HDP --version 2.3.ECS --
   version <stack_name>
   ```

Install the Ambari Agent Manually

The steps for installing the Ambari Agent manually on each node are provided in this procedure.

Before you begin

- The Ambari repository must be present on each cluster node in order to install the agent on the node.

Procedure

1. On one of the Ambari slave hosts, open a command prompt.
2. Install the Ambari Agent using the following command.

   ```
   yum install ambari-agent -y
   ```

3. Get the hostname of the Ambari server node.

   You can do this using the `hostname` command.
4. Edit the `ambari-agent.ini` file to specify the hostname of the Ambari server.

   For example, using `vi`, you would type:

   ```
   vi /etc/ambari-agent/conf/ambari-agent.ini
   ```

   The server hostname is in the [server] parameters, shown below, and you should replace "localhost" with the hostname of the Ambari server.

   ```
   [server]
   hostname=localhost
   ```

```
url_port=8440
secure_url_port=8441
```

5. Start the Ambari Agent.

```
ambari-agent start
```

6. Repeat these steps for each Ambari slave node. If you want to use the Ambari server node as a data node in the Hadoop cluster, you should also install the agent on that machine.

Install Hadoop

Install the Hadoop cluster.

Procedure

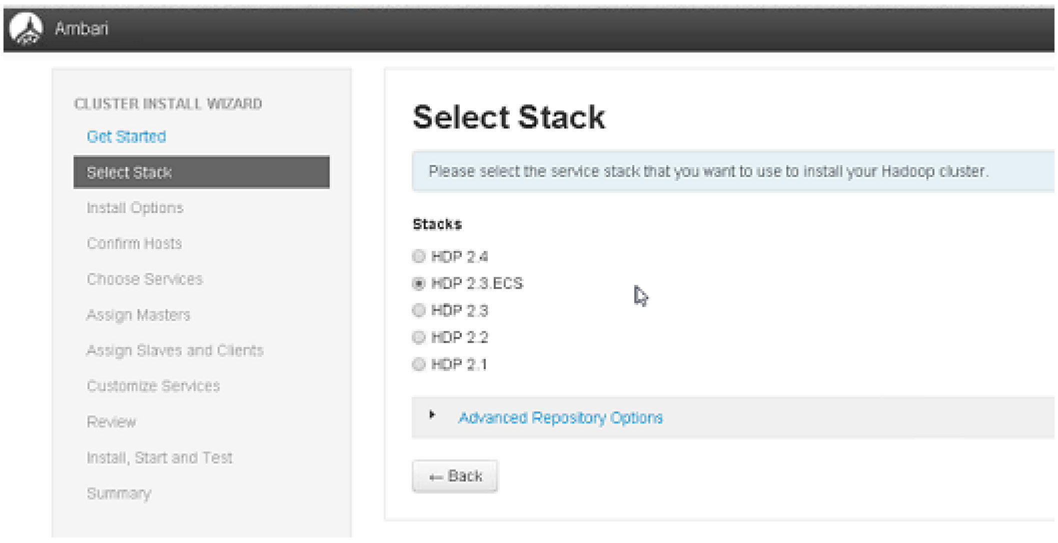
1. Follow the Ambari Wizard for installing a Hadoop Cluster.

   The steps that follow indentify the main ECS integration features.

   Refer to the Hortonworks documentation where you need clarification of the requirements for a step.

2. When prompted for the stack to deploy, select the HDP ECS stack.

   For example, HDP 2.3.ECS as shown below.



Select Stack

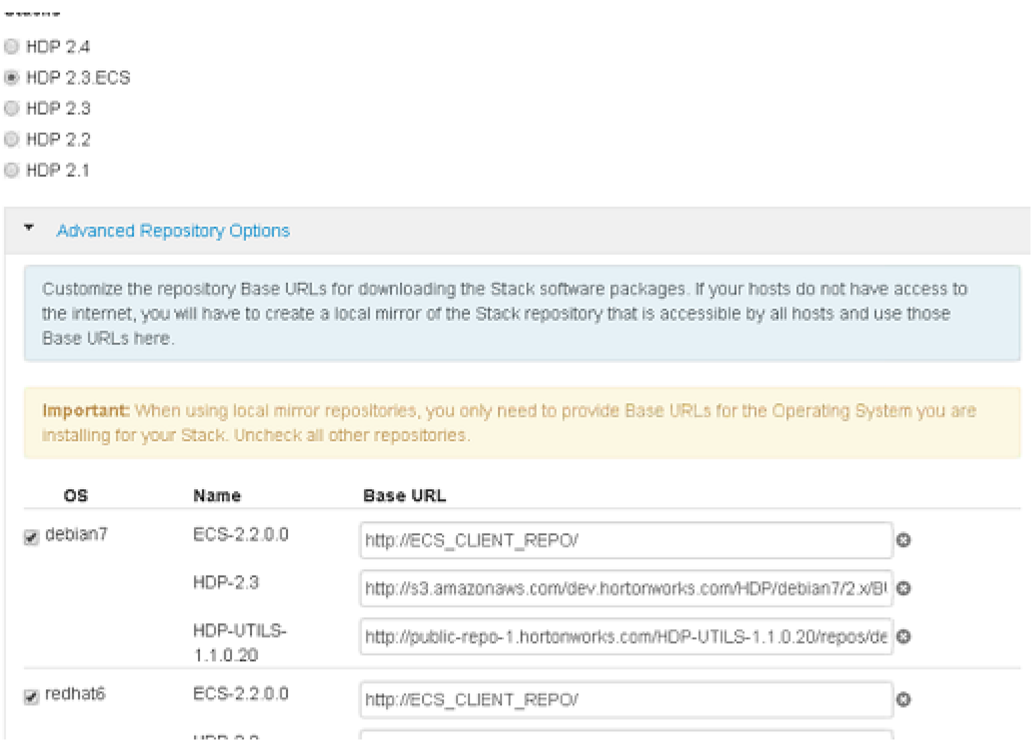
Please select the service stack that you want to use to install your Hadoop cluster.

Stacks

○ HDP 2.4
● HDP 2.3.ECS
○ HDP 2.3
○ HDP 2.2
○ HDP 2.1

3. Specify the repository in which the ECS Client Library RPM is located.

   For your operating system, replace `http://ECS_CLIENT_REPO/` with the location of the repository in which the ECS Client Library RPM is located.



○ HDP 2.4
● HDP 2.3.ECS
○ HDP 2.3
○ HDP 2.2
○ HDP 2.1

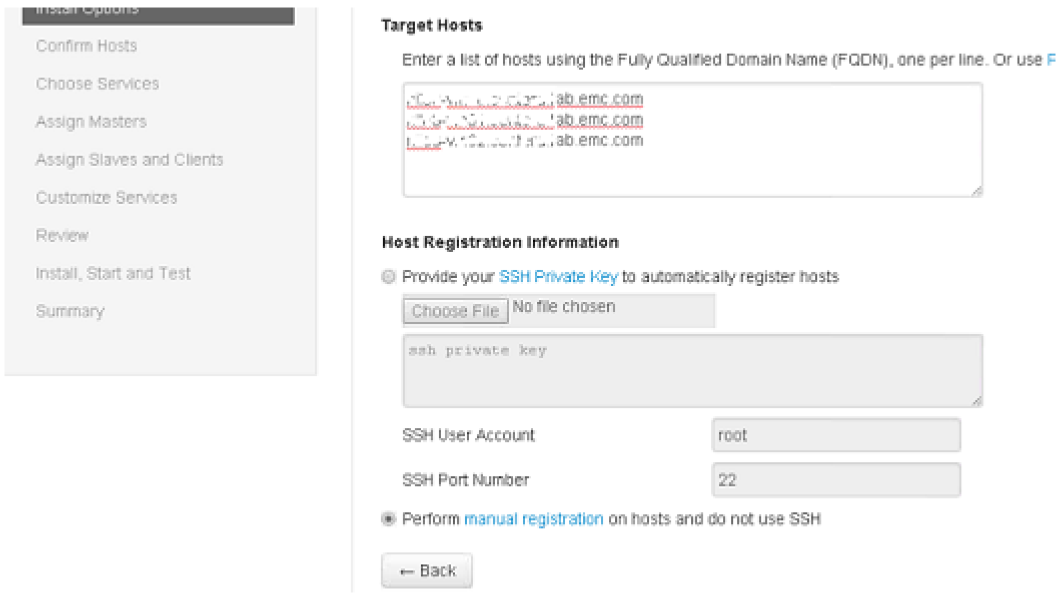
Advanced Repository Options

Customize the repository Base URLs for downloading the Stack software packages. If your hosts do not have access to the internet, you will have to create a local mirror of the Stack repository that is accessible by all hosts and use those Base URLs here.

Important: When using local mirror repositories, you only need to provide Base URLs for the Operating System you are installing for your Stack. Uncheck all other repositories.

OS	Name	Base URL
✓ debian7	ECS-2.2.0.0	http://ECS_CLIENT_REPO/
	HDP-2.3	http://s3.amazonaws.com/dev.hortonworks.com/HDP/debian7/2.x/BI
	HDP-UTILS-1.1.0.20	http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/de
✓ redhat6	ECS-2.2.0.0	http://ECS_CLIENT_REPO/

4. Enter the list of hosts for the Haddop cluster and, under Host Registration Information, select Perform Manual Registration.



Target Hosts

Enter a list of hosts using the Fully Qualified Domain Name (FQDN), one per line. Or use F

```
[redacted]ab.emc.com
[redacted]ab.emc.com
[redacted]ab.emc.com
```

Host Registration Information

○ Provide your SSH Private Key to automatically register hosts

Choose File No file chosen

ssh private key

SSH User Account	root
SSH Port Number	22

● Perform manual registration on hosts and do not use SSH

← Back

Note

If you create an SSH key and deploy it to all nodes, you can use this mechanism to deploy the Ambari agent to all nodes. As we have deployed it manually, this is not necessary.

---

5. You will need to provide properties to customize the operations of the Hadoop Hive and Hbase services.

a. For Hive, you will need to provide a metastore database.

   You can tell Ambari to create a new database (New MySQL Database). In which case, you will simply need to supply a password for the new database. Alternatively, you can tell Ambari to use an exiting database.

b. For Hbase, the `hbase.rootdir` must be pointed at an ECS bucket (HDFS filesystem).

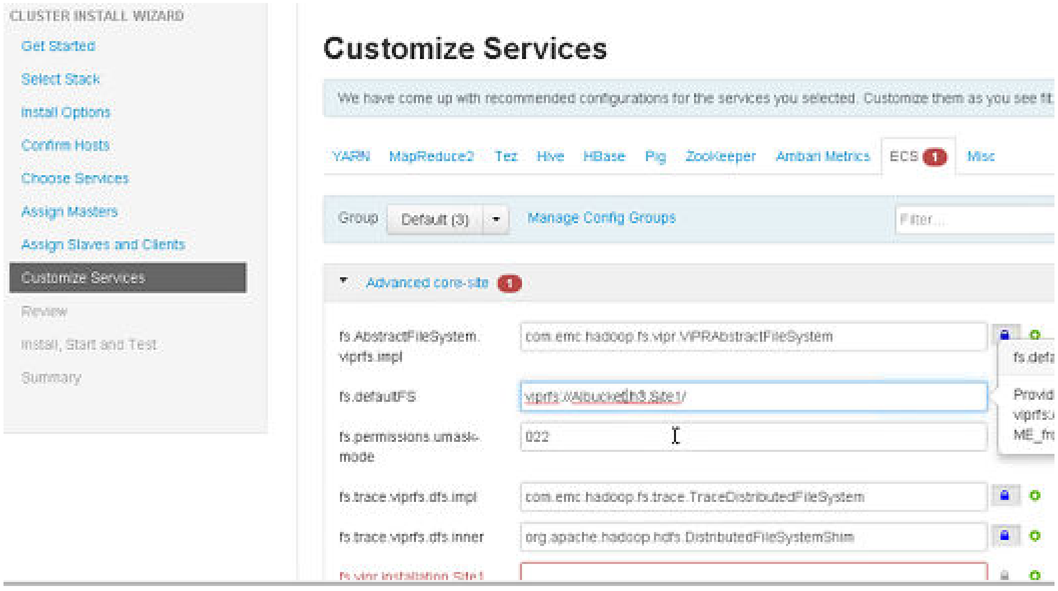
   For example:

```
viprfs://mybucket.mynamespace.Site1/apps/hbase/data
```

6. You will need to provide properties to customize the operations of the Hadoop HDFS/ECS service.

   As part of the Hortonworks Hadoop ECS stack customization, ECS-specific core-site properties have already been added. Some of these properties are fixed, and never need to be modified. Some are set to default values that are appropriate for a Hadoop simple mode environment and will only need to be changed when setting up the environment to use Kerberos security.

   Some parameters are specific to your cluster and need to be provided.



We have come up with recommended configurations for the services you selected. Customize them as you see fit

fs.AbstractFileSystem.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem
fs.defaultFS	viprfs://Abucket.h3.Site1/
fs.permissions.umask-mode	022
fs.trace.viprfs.dfs.impl	com.emc.hadoop.fs.trace.TraceDistributedFileSystem
fs.trace.viprfs.dfs.inner	org.apache.hadoop.hdfs.DistributedFileSystemShim
fs.vipr.installation.Site1	

   Enter the ECS/HDFS core-site configuration properties for the missing properties.

   a. Enter a value for the default filesystem property: fs.defaultFS

      This is

```
viprfs://<bucket>.<namespace>.<installation>
```

      For example:

```
viprfs://mybucket.mynamespace.Site1
```

b. Enter the ECS node addresses in the fs.vipr.installation.<site>.hosts (by default: fs.vipr.installation.Site1.hosts) property.

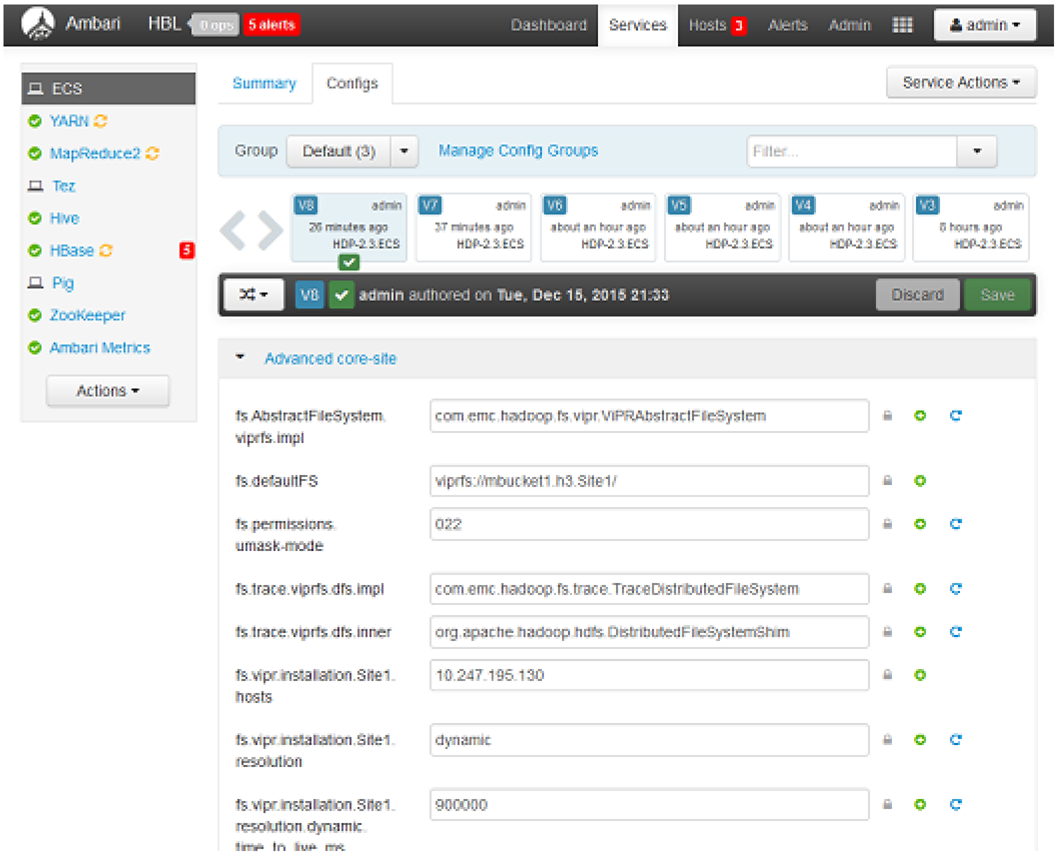
   This can be a comma separated list of IP addresses, for example:

   ```
   203.0.113.10,203.0.113.11,203.0.113.12
   ```

You can read more about the properties in Edit Hadoop core-site.xml file on page 167, which describes the property settings for a simple Hadoop cluster.

Results

Once installed, the Ambari interface for the Hadoop ECS cluster is customized so that the HDFS service displays as ECS. In addition, ECS-specific `core-site.xml` properties that would normally need to be added are already present and default values have been set for the most of the parameters.
The custom interface is shown below:



V8 admin authored on Tue, Dec 15, 2015 21:33

fs.AbstractFileSystem.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem
fs.defaultFS	viprfs://mbucket1.h3.Site1/
fs.permissions.umask-mode	022
fs.trace.viprfs.dfs.impl	com.emc.hadoop.fs.trace.TraceDistributedFileSystem
fs.trace.viprfs.dfs.inner	org.apache.hadoop.hdfs.DistributedFileSystemShim
fs.vipr.installation.Site1.hosts	10.247.195.130
fs.vipr.installation.Site1.resolution	dynamic
fs.vipr.installation.Site1.resolution.dynamic.time_to_live_ms	900000

CHAPTER 23

Configure ECS HDFS integration with a simple Hadoop cluster

Configure ECS HDFS Integration with a simple Hadoop cluster

This procedure describes how to set up your existing Hadoop distribution to use the ECS storage infrastructure with ECS HDFS.

If you are using the Hortonworks Ambari distribution, you can use the procedure described in Deploying a Hortonworks cluster with Ambari on page 156 to install and configure Hadoop.

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 need to be performed:

1. Plan the ECS HDFS and Hadoop integration on page 164
2. Obtain the ECS HDFS installation and support package on page 165
3. Deploy the ECS HDFS Client Library on page 165 (Not required if you have used Ambari Hortoworks for ECS)
4. Edit Hadoop core-site.xml file on page 167
5. Edit HBASE hbase-site.xml on page 170
6. Restart and verify access on page 171

Plan the ECS HDFS and Hadoop integration

Use this list to verify that you have the information necessary to ensure a successful integration.

Table 18 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 a bucket enabled for HDFS to be created within an ECS replication group and the bucket is accessed as a filesystem 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 EMC Support site for ECS as described in Obtain the ECS HDFS installation and support package on page 165.

The HDFS Client Library uses the following naming convention`viprfs-client-<ECS version>-hadoop-<Hadoop version>.jar` and the JARs for use with each release are listed in the table below.

Table 19 ECS HDFS Client Library

Hadoop distribution	Version	ECS HDFS JAR
Hortonworks	HWX 2.0	viprfs-client-<ECS version>-hadoop-2.2.jar
	HWX 2.1	viprfs-client-<ECS version>-hadoop-2.3.jar (Hadoop 2.5 - No 2.4 client)
	HWX 2.2	viprfs-client-<ECS version>-hadoop-2.6.jar
	HWX 2.3	viprfs-client-<ECS version>-hadoop-2.7.jar

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.
- For Hortonworks 2.3 (Hadoop 2.7), you can use Ambari to install a HDP release that is pre-configured with the ECS Client Library.

Procedure

1. Log in to a ECS client node.
2. Run the classpath command to get the list of directories in the classpath:

   ```
   # hadoop classpath
   ```
3. Copy ECS HDFS Client Library JAR to one of folders listed by the classpath command that occurs after the `/conf` folder.

   For example, the classpath command output normally looks like this:

   ```
   /usr/hdp/2.2.0.0-1256/hadoop/conf:/usr/hdp/2.2.0.0-1256/
   hadoop/lib/*:/usr/hdp/2.2.0.0-1256/hadoop/.//*:/usr/hdp/
   2.2.0.0-1256/hadoop-hdfs/./:/usr/hdp/2.2.0.0-1256/hadoop-
   hdfs/lib/*:/
   ```

   With the `/conf` folder listed first. And it is suggested that you add the Client Library JAR to the first `/lib` folder, which is usually as listed in the table below.

ECS distribution	Classpath location (suggested)
Hortonworks	`/usr/hdp/<version>/hadoop/lib`

4. Repeat this procedure on each ECS client node.
5. Update the classpath configuration setting for MapReduce, yarn and also explicitly specify path to the JAR for Tez.

   An example of these configuration settings is provided below:

   ```
   mapreduce.application.classpath
   $PWD/mr-framework/hadoop/share/hadoop/mapreduce/*:$PWD/mr-
   framework/hadoop/share/hadoop/mapreduce/lib/*:$PWD/mr-
   framework/hadoop/share/hadoop/common/*:$PWD/mr-framework/
   hadoop/share/hadoop/common/lib/*:$PWD/mr-framework/hadoop/
   share/hadoop/yarn/*:$PWD/mr-framework/hadoop/share/hadoop/
   yarn/lib/*:$PWD/mr-framework/hadoop/share/hadoop/hdfs/*:
   $PWD/mr-framework/hadoop/share/hadoop/hdfs/lib/*:$PWD/mr-
   framework/hadoop/share/hadoop/tools/lib/*:/usr/hdp/$
   {hdp.version}/hadoop/lib/hadoop-lzo-0.6.0.$
   {hdp.version}.jar:/etc/hadoop/conf/secure:/usr/hdp/
   2.3.2.0-2950/hadoop/lib/*


   yarn.application.classpath
   $HADOOP_CONF_DIR,/usr/hdp/current/hadoop-client/*,/usr/hdp/
   current/hadoop-client/lib/*,/usr/hdp/current/hadoop-hdfs-
   client/*,/usr/hdp/current/hadoop-hdfs-client/lib/*,/usr/hdp/
   current/hadoop-yarn-client/*,/usr/hdp/current/hadoop-yarn-
   client/lib/*,/usr/hdp/2.3.2.0-2950/hadoop/lib/*


   tez.cluster.additional.classpath.prefix
   ```

```
/usr/hdp/${hdp.version}/hadoop/lib/hadoop-lzo-0.6.0.$
{hdp.version}.jar:/etc/hadoop/conf/secure:/usr/hdp/
2.3.2.0-2950/hadoop/lib/viprfs-client-2.2.0.0-hadoop-2.7.jar
```

Edit Hadoop core-site.xml file

Use this procedure to update `core-site.xml` with the properties needed to integrate ECS HDFS with a Hadoop cluster that uses simple authentication mode.

Before you begin

- It is always preferable to add/manage these properties using a Hadoop management UI to reduce the chance of errors and to ensure these changes are persistent across the cluster. Manually editing files on multiple Hadoop nodes is cumbersome and error prone. You must have a set of user credentials that enable you to log in to the management UI for your distribution.
- If you do modify `core-site.xml` directly, you must have a set of user credentials that enable you to log in to Hadoop nodes and modify `core-site.xml`.

Some properties are specific to ECS and usually need to be added to `core-site.xml`. If you are using the Hortonworks Ambari Hadoop ECS stack, the ECS-specific parameters are already present.

If you intend to edit `core-site.xml` directly, the location of `core-site.xml` depends on the distribution you are using, as shown in the following table.

Table 20 core-site.xml locations

Hadoop Distribution	core-site.xml location	Nodes to update
Hortonworks	`/etc/hadoop/conf`	All nodes

`core-site.xml` resides on each node in the Hadoop cluster. You must modify the same properties in each instance. You can make the change in one node, and then use secure copy command (scp) to copy the file to the other nodes in the cluster.

See core_site.xml property reference for more information about each property you need to set.

Procedure

1. If you are using a management interface, such as Ambari. Log in as an administrator and go to the HDFS configuration page.
2. If you intend to make the changes by manually editing `core-site.xml`, follow these steps
   a. Log in to one of the HDFS nodes where `core-site.xml` is located.
   b. Make a backup copy of `core-site.xml`.
   ```
   cp core-site.xml core-site.backup
   ```
   c. Using the text editor of your choice, open `core-site.xml` for editing.

3. Add the following properties and values to define the Java classes that implement the ECS HDFS file system:

   ```
   <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>
   ```

4. Add the fs.vipr.installations property. In the following example, the value is set to Site1.

   ```
   <property>
     <name>fs.vipr.installations</name>
     <value>Site1</value>
   </property>
   ```

5. Add the fs.vipr.installation.[*installation_name*].hosts property as a comma-separated list of ECS data nodes or load balancer IP addresses. In the following example, the installation_name is set to Site1.

   ```
   <property>
     <name>fs.vipr.installation.Site1.hosts</name>
     <value>203.0.113.10,203.0.113.11,203.0.113.12</value>
   </property>
   ```

6. Add the fs.vipr.installation.[*installation_name*].resolution property, and set it to one of the following values:

Option	Description
dynamic	Use when accessing ECS data nodes directly without a load balancer.
fixed	Use when accessing ECS data nodes through a load balancer.

   In the following example, installation_name is set to Site1.

   ```
   <property>
     <name>fs.vipr.installation.Site1.resolution</name>
     <value>dynamic</value>
   </property>
   ```

   a. If you set fs.vipr.installation.[*installation_name*].resolution to dynamic, add the fs.vipr.installation.[installation_name].resolution.dynamic.time_to_live_ms property to specify how often to query ECS for the list of active nodes.

      In the following example, installation_name is set to Site1.

      ```
      <property>
      <name>fs.vipr.installation.Site1.resolution.dynamic.time_to_
      live_ms</name>
      ```

```
<value>900000</value>
</property>
```

7. Locate the fs.defaultFS property and modify the value to specify the ECS file system URI. .

   This setting is optional and you can specify the full file system URL to connect to the ECS ViPRFS.

   Use the following format: `viprfs://` *`<bucket_name.namespace.installation_name`*, where

   - *bucket_name*: The name of the bucket that contains the data you want to use when you run Hadoop jobs. If running in simple authentication mode, the owner of the bucket must grant permission to Everybody. In the following example, the bucket_name is set to mybucket.
   - *namespace*: The tenant namespace where *bucket_name* resides. In the following example, the namespace is set to mynamespace.
   - *installation_name*: The value specified by the fs.vipr.installations property. In the following example, installation_name is set to Site1.

   ```
   <property>
     <name>fs.defaultFS</name>
     <value>viprfs://mybucket.mynamespace.Site1/</value>
   </property>
   ```

8. Locate fs.permissions.umask-mode, and set the value to 022.

   In some configurations, this property might not already exist. If it does not, then add it.

   ```
   <property>
     <name>fs.permissions.umask-mode</name>
     <value>022</value>
   </property>
   ```

9. Add the fs.viprfs.auth.anonymous_translation property; used to specify the owner and group of a file or directory created using HDFS.

   Note

   Prior to ECS 2.2, this parameter was used to assign an owner to files and directories that were created without an owner (anonymously owned files) so that the current user had permission to modify them. Files and directories are no longer created anonymously and have an owner assigned according to the setting of this parameter.

Option	Description
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 Kerberized Hadoop cluster. Assigns the Kerberos principal (user@REALM.COM) as the file or directory owner, and uses the group that has been assigned as the default for the bucket.

Option	Description
NONE (default)	(Deprecated) Previously indicated that no mapping from the anonymously owned objects to the current user should be performed.

```
<property>
  <name>fs.viprfs.auth.anonymous_translation</name>
  <value>LOCAL_USER</value>
</property>
```

10. Add the fs.viprfs.auth.identity_translation property. It provides a way to assign users to a realm when Kerberos is not present.

Option	Description
CURRENT_USER_REALM	Use this setting for a Kerberized Hadoop cluster. When specified, the realm is automatically detected.
NONE (default)	Use this setting with a Hadoop cluster that uses simple security. With this setting ECS HDFS does not perform realm translation.
FIXED_REALM	(Deprecated) Provides the ability to hard-code the user's realm using the fs.viprfs.auth.realm property.

```
<property>
  <name>fs.viprfs.auth.identity_translation</name>
  <value>NONE</value>
</property>
```

11. Save `core-site.xml`.
12. Update the `core-site.xml` on the required nodes in your Hadoop cluster.
13. Use Hortonworks Ambari to update the `core-site.xml` with the same set of properties and values.

Edit HBASE hbase-site.xml

When you use HBASE with ECS HDFS, you must set the hbase.rootdir in `hbase-site.xml` to the same value as the `core-site.xml` fs.defaultFS property.

`hbase-site.xml` is located in one of the following locations:

Table 21 hbase-site.xml locations

Hadoop Distribution	hbase-site.xml location
Hortonworks	`/etc/hbase/conf/`

Procedure

1. Open `hbase-site.xml`.

2. Set the hbase.rootdir property to the same value as fs.defaultFS adding `/hbase` as the suffix.
3. Save your changes.
4. Restart the services for your distribution.

Hadoop Distribution	Description
Hortonworks	`# bin/start-hbase.sh`

Example 1 hbase.rootdir entry

```
<property>
  <name>hbase.rootdir</name>
  <value>viprfs://testbucket.s3.testsite/hbase</value>
</property>
```

Restart and verify access

Once you have performed the configuration steps, you can restart the Hadoop services and check that you have access to the HDFS.

When the system is configured to use ECS HDFS, the HDFS NameNode can fail to start. When ECS HDFS is configured as the HDFS, ECS HDFS performs all NameNode functions and does not require the NameNode to be up.

Procedure

1. Restart the Hadoop services.

   This will normally include HDFS, MapReduce, Yarn, and HBase.

   If you are restarting the services manually, you can refer to the table below.

Hadoop Distribution	Commands

Hortonworks	`# stop-all.sh`
	`# start-all.sh`

2. Test the configuration by running the following command to get a directory listing:

```
# hdfs dfs -ls viprfs://mybucket.mynamespace.Site1/
```

```
13/12/13 22:20:37 INFO vipr.ViPRFileSystem: Initialized
ViPRFS for viprfs://mybucket.mynamespace.Site1/
```

   If you have set fs.defaultFS, you can use:
   `# hdfs dfs -ls /`

CHAPTER 24

Configure ECS HDFS integration with a secure (Kerberized) Hadoop cluster

Integrate secure Hadoop cluster with ECS HDFS

This procedure describes how to integrate your existing Hadoop distribution, that is secured using Kerberos, with ECS HDFS.

If you have configured ECS HDFS to work with a Hadoop cluster configured for simple authentication, and have migrated the Hadoop cluster to use Kerberos authentication, you can also use this procedure.

Before performing the integration steps:

- Verify that a Kerberos 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. Help with setting up the Kerberos KDC and configuring trust is provided in Guidance on Kerberos configuration on page 190.
- Ensure that you have created a bucket for the HDFS filesystem (see Create a bucket for HDFS using the ECS Portal on page 148)
- Ensure that you have read the guidelines for planning the integration (see Plan the ECS HDFS and Hadoop integration on page 164).
- Ensure that you have downloaded the installation and support package (see Obtain the ECS HDFS installation and support package on page 165).

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 174
2. Map group names on page 175
3. Configure ECS nodes with the ECS Service Principal on page 175
4. Secure the ECS bucket using metadata on page 179
5. Restart and verify access on page 186

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 read the section on migration: Migration from a simple to a Kerberos Hadoop cluster on page 144.

In general, the ECS migration feature will enable files and directories to be accessible by Kerberos users seamlessly. However the following notes apply:

- If you migrate your Hadoop cluster to Kerberos and then restart the cluster, processes such as MapReduce will not be able to access directories that it previously created. You should wait until you have configured ECS HDFS using this procedure before restarting Hadoop.
- For users and processes to be able to access the bucket, they need to be members of the group that has access to the bucket, or you will need to change the bucket ACLs so that Kerberos users have access.

Map group names

ECS needs to be able to map group details for Hadoop service principals like hdfs, hive, etc. If you are using Active Directory, group information can be found from two different sources: the bucket metadata or Active Directory. ECS determines which source to use from a configuration parameter setting in `/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) in preference to Active Directory, define the parameter as follows:

```
[hdfs.fs.request]
prefer_secure_metadata_bucket_for_groups = true
```

If you want ECS to determine group information from Active Directory in preference to 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 will determine group details for a Kerberos principal from the bucket metadata. If you make a change, you must restart dataheadsvc.

Configure ECS nodes with the ECS Service Principal

The ECS service principal and its corresponding keytab file must reside on each ECS data node. 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 165.
- 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 will be 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` 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 `inventory.txt` in the `playbooks/samples` directory to refer to the ECS data nodes and 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

   This step should be performed only if you are using strong encryption type.

   Kerberos may be configured to use a strong encryption type, such as AES-256. In that situation, the JRE within the ECS nodes must be reconfigured to use the 'unlimited' 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.txz
      ```

   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                    1.5.0.0-403.cb6738e
186bd8577a7a        2 weeks ago         738.5 MB
```

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

    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 exist and the `krb5.conf` file is in the working directory `/ansible/samples` directory.

```
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 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`.

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 needs to have Hadoop access to a bucket needs to be uploaded to ECS. The following data is required:

- Principal name
- Principal shortname (mapped name)
- Principal groups

If there are 10 Kerberos principals on your Hadoop node, you must create 30 name value pairs in the JSON input file. Every name must be unique, so you will need to uniquely assign a name for every Principal name, Principal shortname, and Principal groups. 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. An example is show below.

```
{
    "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 is 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 yarn principal), you must use the 'mapped' principal in the shortname value, so the entry for 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 will need to tell the ECS which Linux group of users on the Hadoop nodes will get superuser privileges based on their group. Only one entry in the JSON input file will be expected for the supergroup designation. It should be like this:

```
{
    "name": "dfs.permissions.supergroup",
    "value": "hdfs"
}
```

Proxy settings
For proxy support, you need to identify all proxy settings that are allowed for each Hadoop application' where application means one of the Hadoop-supported applications, for example, hive, hbase, etc.

In the example below, proxy support for the hive application is granted to users who are members of the group s3users (AD or Linux group), and can run hive on any of the hosts in the Hadoop cluster. So the JSON entry for this would be 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 below.

```
{
    "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 212.

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 will be rejected. If metadata has not been loaded, the bucket is not secure and a request from a secure Hadoop node will be rejected.

The following error will be 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

The metadata values required to secure an ECS bucket for use with a secure Hadoop cluster can be supplied by running ECS Management REST API commands.

Before you begin

You must have ECS System Admin credentials.

If the Hadoop administrator is NOT the ECS administrator, the Hadoop administrator will need to work in conjunction with the ECS System Admin to load the secure metadata to the bucket.

The Hadoop administrator can make the JSON metadata file available to the ECS System Admin, who can then use this procedure to load the metadata. If the two roles are assumed by the same user, then that user would be responsible for creating the JSON metadata file and loading it to the ECS bucket.

Procedure

1. Create the JSON file that contains the metadata described in: Secure the ECS bucket using metadata on page 179.
2. Log in to ECS using your System Admin credentials in order to obtain an authentication token that can be used when running ECS management commands.

   You can run this command using curl. In the example below, you will need to replace the <username>:<password> with ECS System Admin 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 ECS Management REST API command to deploy the metadata.

   The API is: PUT object/bucket/<bucketname>/metadata. An example of running this command using curl is shown below.

   ```
    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 will need to replace:

   - <username> with an ECS system administrator username.
   - <password> with the password for the specified ECS System Admin 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 will be available to all ECS nodes.

Edit core-site.xml

Use this procedure to update `core-site.xml` with the properties that are required when using ECS HDFS with a ECS cluster that uses Kerberos authentication mode.

Before you begin

- It is always preferable to add/manage these properties using a Hadoop management UI to reduce the chance of errors and to ensure these changes are persistent across the cluster. Manually editing files on multiple Hadoop nodes is cumbersome and error prone. You must have a set of user credentials that enable you to log in to the management UI for your distribution.
- If you do modify `core-site.xml` directly, you must have a set of user credentials that enable you to log in to Hadoop nodes and modify `core-site.xml`.

Some properties are specific to ECS and usually need to be added to `core-site.xml`. If you are using the Hortonworks Ambari Hadoop ECS stack, the ECS-specific parameters are already present.

If you intend to edit `core-site.xml` directly, the location of `core-site.xml` depends on the distribution you are using, as shown in the following table.

Table 22 Location of `core-site.xml` files

ECS Distribution	core-site.xml location	Nodes to update
Hortonworks	`/etc/hadoop/conf`	All nodes

`core-site.xml` resides on each node in the Hadoop cluster, and you must modify the same properties in each instance. You can make the change in one node, and then use secure copy command (scp) to copy the file to the other nodes in the cluster. As a best practice, back up `core-site.xml` before you start the configuration procedure.

See core_site.xml property reference for more information about each property you need to set.

Procedure

1. If you are using a management interface, such as Hortonworks Ambari, log in as an administrator and go to the HDFS configuration page.
2. If you intend to make the changes by manually editing `core-site.xml`, follow the steps below.

   a. Log in to one of the HDFS nodes where `core-site.xml` is located.

   b. Make a backup copy of `core-site.xml`.

   ```
   cp core-site.xml core-site.backup
   ```

   c. Using the text editor of your choice, open `core-site.xml` for editing.
3. Add the following properties and values to define the Java classes that implement the ECS HDFS file system:

```
<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>
```

4. Add the fs.vipr.installations property. In the following example, the value is set to Site1.

   ```
   <property>
     <name>fs.vipr.installations</name>
     <value>Site1</value>
   </property>
   ```

5. Add the fs.vipr.installation.[*installation_name*].hosts property as a comma-separated list of ECS data nodes or load balancer IP addresses. In the following example, the installation_name is set to Site1.

   ```
   <property>
     <name>fs.vipr.installation.Site1.hosts</name>
     <value>203.0.113.10,203.0.113.11,203.0.113.12</value>
   </property>
   ```

6. Add the fs.vipr.installation.[*installation_name*].resolution property, and set it to one of the following values:

Option	Description
dynamic	Use when accessing ECS data nodes directly without a load balancer.
fixed	Use when accessing ECS data nodes through a load balancer.

   In the following example, installation_name is set to Site1.

   ```
   <property>
     <name>fs.vipr.installation.Site1.resolution</name>
     <value>dynamic</value>
   </property>
   ```

   a. If you set fs.vipr.installation.[*installation_name*].resolution to dynamic, add the fs.vipr.installation.[installation_name].resolution.dynamic.time_to_live_ms property to specify how often to query ECS for the list of active nodes.

      In the following example, installation_name is set to Site1.

      ```
      <property>
      <name>fs.vipr.installation.Site1.resolution.dynamic.time_to_
      live_ms</name>
      <value>900000</value>
      </property>
      ```

7. Locate the fs.defaultFS property and modify the value to specify the ECS file system URI. .

This setting is optional and you can specify the full file system URL to connect to the ECS ViPRFS.

Use the following format: `viprfs://` *`<bucket_name.namespace.installation_name`*, where

- *bucket_name*: The name of the bucket that contains the data you want to use when you run Hadoop jobs. If running in simple authentication mode, the owner of the bucket must grant permission to Everybody. In the following example, the bucket_name is set to mybucket.
- *namespace*: The tenant namespace where *bucket_name* resides. In the following example, the namespace is set to mynamespace.
- *installation_name*: The value specified by the fs.vipr.installations property. In the following example, installation_name is set to Site1.

```
<property>
  <name>fs.defaultFS</name>
  <value>viprfs://mybucket.mynamespace.Site1/</value>
</property>
```

8. Locate fs.permissions.umask-mode, and set the value to 022.

   In some configurations, this property might not already exist. If it does not, then add it.

```
<property>
        <name>fs.permissions.umask-mode</name>
        <value>022</value>
</property>
```

9. Add the fs.viprfs.auth.anonymous_translation property; used to specify the owner and group of a file or directory created using HDFS.

Note

Prior to ECS 2.2, this parameter was used to assign an owner to files and directories that were created without an owner (anonymously owned files) so that the current user had permission to modify them. Files and directories are no longer created anonymously and have an owner assigned according to the setting of this parameter.

Option	Description
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 Kerberized Hadoop cluster. Assigns the Kerberos principal (user@REALM.COM) as the file or directory owner, and uses the group that has been assigned as the default for the bucket.
NONE (default)	(Deprecated) Previously indicated that no mapping from the anonymously owned objects to the current user should be performed.

```
<property>
  <name>fs.viprfs.auth.anonymous_translation</name>
  <value>LOCAL_USER</value>
</property>
```

10. Add the fs.viprfs.auth.identity_translation property, and set it to CURRENT_USER_REALM, which maps to the realm of the user signed in via `kinit`.

```
<property>
        <name>fs.viprfs.auth.identity_translation</name>
        <value>CURRENT_USER_REALM</value>
</property>
```

11. Add the viprfs.security.principal property. This property tells the KDC who the ECS user is.

    The principal name can include "_HOST" which is automatically replaced by the actual data node FQDN at run time.

```
<property>
        <name>viprfs.security.principal</name>
        <value>vipr/_HOST@example.com</value>
</property>
```

12. Add the cached location of Kerberos tickets.

    For example:

```
<property>
<name>hadoop.security.kerberos.ticket.cache.path</name>
    <value>/tmp/<krbcc_1000</value>
</property>
```

    The value can be obtained from the output of the `klist` command.

13. Use Hortonworks Ambari to update the `core-site.xml` with the same set of properties and values.

Restart and verify access

Once you have performed the configuration steps, you can restart the Hadoop services and check that you have access to the HDFS.

When the system is configured to use ECS HDFS, the HDFS NameNode can fail to start. When ECS HDFS is configured as the HDFS, ECS HDFS performs all NameNode functions and does not require the NameNode to be up.

Procedure

1. Restart the Hadoop services.

   This will normally include HDFS, MapReduce, Yarn, and HBase.

   If you are restarting the services manually, you can refer to the table below.

Hadoop Distribution	Commands

Hortonworks	`# stop-all.sh`
	`# start-all.sh`

Hadoop Distribution	Commands

2. Test the configuration by running the following command to get a directory listing:

```
# kinit <service principal>
```

```
# hdfs dfs -ls viprfs://mybucket.mynamespace.Site1/
```

```
13/12/13 22:20:37 INFO vipr.ViPRFileSystem: Initialized
ViPRFS for viprfs://mybucket.mynamespace.Site1/
```

CHAPTER 25

Guidance on Kerberos configuration

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 /passwordt 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$/$1/g
   RULE:[2:$1@$0](^.*@CAMBRIDGE\.ACME\.COM$)s/^(.*)@CAMBRIDGE\.ACME
   \.COM$/$1/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
```

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.
- 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 `inventory.txt` in the `playbooks/samples` directory to add the added ECS nodes.

   The default entries are shown below.

   ```
   [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.

      For example:

      ```
      sudo docker load -i /opt/emc/caspian/checker/docker/images/
      utilities.txz
      ```

   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                  1.5.0.0-403.cb6738e
186bd8577a7a        2 weeks ago         738.5 MB
```

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.

4. Change to the working directory in the container.

   For 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`.

CHAPTER 26

Troubleshooting

Troubleshooting

This area provides workarounds for issue that may be encountered when configuring ECS HDFS.

Verify AD/LDAP is correctly configured with 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 "kinit" against 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 `/etc/krb5.conf` on the KDC server for correctness and syntax. Realms can be case sensitive in the config files as well as when used with the `kinit` command.
- Check that `/etc/krb5.conf` 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. Refer to appropriate documentation on how to do this.
- Make sure that the encryption type on the AD/LDAP server matches that on the KDC server.
- Check that `/var/kerberos/krb5kdc/kadm5.acl` and `/var/kerberos/krb5kdc/kdc.conf` 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.

Restart services after hbase configuration

After editing the hbase.rootdir property in hbase-site.xml, the hbase service does not restart correctly.

Workaround
The following steps should be performed when this issue arises on Hortonworks to get hbase-master running.

1. Connect to the zookeeper cli.

   ```
   hbase zkcli
   ```

2. Remove the hbase directory.

   ```
   rmr /hbase
   ```

3. Restart the hbase service.

Pig test fails: unable to obtain Kerberos principal

Pig test fails with the following 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 firstten".

This issue is caused due to the fact that Pig (<0.13) doesn't generate a delegation token for ViPRFS as a secondary storage.

Workaround
Append the `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

When running an application as an AD user, a "Permission denied" error is raised.

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 it when running a hadoop fs command, or you may see it in application log, such as the log for mapreduce or hive.

INSUFFICIENT_PERMISSIONS errors

In the example below, 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. The cause of this error should be obvious to most users.

```
root@lrmk042:/etc/security/keytabs# hadoop fs -mkdir /tmp
15/11/08 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
15/11/08 21:03:09 ERROR vipr.ViPRFileSystemClientBase: Request
message sent:
MkDirRequestMessage[kind=MKDIR_REQUEST,namespace=s3,bucket=hdfsBucke
t3,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 2015-11-08 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 that below in the `dataheadsvc-error.log`:

```
2015-11-08 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 groups for the principal user making the request. For a Kerberos user, this will return this error.

The error will tell you the user name making the request. Make a note of it.

Bucket Access Error

If a user making are request to access a bucket does not have ACL permissions, you could see this error stack in `dataheadsvc-error.log`.

```
2015-11-08 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(ObjectAccessException.java:286)
at
com.emc.storageos.data.object.ipc.protocol.impl.ObjectAccessExceptio
nParser.parseFrom(ObjectAccessExceptionParser.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 stack below is an example of an object access error:

```
2015-11-08 22:36:21,995 [pool-68-thread-6] ERROR
FileSystemAccessHelper.java (line 1364) nfsProcessOperation failed
to process path: mr-history/done
2015-11-08 22:36:21,995 [pool-68-thread-6] ERROR
ObjectControllerExceptionHelper.java (line 186) Method nfsGetSMD
failed due to exception
com.emc.storageos.data.object.exception.ObjectControllerException:
directory server returns error ERROR_ACCESS_DENIED
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccess
Helper.nfsProcessOperation(FileSystemAccessHelper.java:1368)
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccess
Helper.getSystemMetadata(FileSystemAccessHelper.java:466)
at
com.emc.storageos.data.object.FileSystemAccessLayer.FileSystemAccess
Layer.getSystemMetadata(FileSystemAccessLayer.java:532)
at
com.emc.storageos.data.object.blob.client.BlobAPI.getStat(BlobAPI.ja
va: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 202
- Hadoop Client Debugging on page 202

Blobsvc Log Debugging

A failed permission request will have an error in blobsvc like this:

```
2015-11-08 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
```

Just 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 doesn't 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 our 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 2015-11-08 16:58 /mr-history/done
```

This 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 2015-11-08 16:58 /mr-history
```

Likewise, this directory has no access issues

```
root@lrmk042:/var/log/hadoop-mapreduce/mapred# hadoop fs -ls -d /
drwxr-x--- - hdfs hdfs 0 2015-11-08 16:58 /
```

The problem here, is that the root directory is owned by hdfs, 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 to list the /mr-history/done directory. Performing a chmod on the root directory will enable 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 2015-11-08 16:58 /
```

Failed to process request

When listing a bucket an error: Failed to Process Request is generated.

When performing list bucket, 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. Your 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 an 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
Verbose logging is enabled using an environment variable that applies only to your current SSH session.

```
 export HADOOP_OPTS="-Dsun.security.krb5.debug=true"
```

Enable Hadoop client-side debugging
To troubleshoot Hadoop activity between the Hadoop node and the ECS, you can enable Hadoop verbose logging as follows:

```
export HADOOP_ROOT_LOGGER="Debug,console"
```

Debug Kerberos on the KDC

Tail the KDC's `/var/log/krb5kdc.log` file when you do an HDFS operation to make it easier to debug.

```
tail -f /var/log/krb5kdc.log
```

Eliminate clock skew

It is important to ensure that time is synchronized between client and server as Kerberos relies on time being accurate.

If your AD has a clock skew with your data nodes/KDC, you will have 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

CHAPTER 27

Hadoop core-site.xml properties for ECS HDFS

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 23 Hadoop core-site.xml properties

Property	**Description**
File system implementation properties	

fs.viprfs.impl	`<property>`
	`<name>fs.viprfs.impl</name>`
	`<value>com.emc.hadoop.fs.vipr.ViPRFileSystem</value>`
	`</property>`

fs.AbstractFileSystem.viprfs.impl	`<property>`
	`  <name>fs.AbstractFileSystem.viprfs.impl</name>`
	`  <value>com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem</value>`
	` </property>`

Properties that define the authority section of the ECS HDFS file system URI	

fs.vipr.installations	A comma-separated list of names. The names are further defined by the fs.vipr.installation.[installation_name].hosts property to uniquely identify sets of ECS data nodes. The names are used as a component of the authority section of the ECS HFDS file system URI. For example:
	`<property>`
	`    <name>fs.vipr.installations</name>`
	`    <value><site1>,<abc>,<testsite></value>`
	` </property>`

fs.vipr.installation.[installation_name].hosts	The IP addresses of the ECS cluster's data nodes or the load balancers for each name listed in the fs.vipr.installations property. Specify the value in the form of a comma-separated list of IP addresses. For example:
	`<property>`
	`  <name>fs.vipr.installation.<site1>.hosts</name>`
	`  <value>203.0.113.10,203.0.113.11,203.0.113.12</value>`
	` </property>`
	
	`<property>`
	`  <name>fs.vipr.installation.<abc>.hosts</name>`
	`  <value>198.51.100.0,198.51.100.1,198.51.100.2</value>`
	` </property>`
	
	`<property>`
	`  <name>fs.vipr.installation.<testsite>.hosts</name>`

Table 23 Hadoop core-site.xml properties (continued)

Property	Description
	```
	    <value>198.51.100.10,198.51.100.11,198.51.100.12</value>
	  </property>
	```

fs.vipr.installation.[installation_name].resolution	Specifies how the ECS HDFS software knows how to access the ECS data nodes. Values are:
	• 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.
	```
	<property>
	    <name>fs.vipr.installation.testsite.resolution</name>
	    <value>dynamic</value>
	  </property>
	```

fs.vipr.installation.[installation_name].resolution.dynamic.time_to_live_ms	When the fs.vipr.installation.[installation_name].resolution property is set to dynamic, this property specifies how often to query ECS for the list of active nodes. Values are in milliseconds. The default is 10 minutes.
	```
	<property>
	    <name>fs.vipr.installation.<*testsite*>.resolution.dynamic.time_to_live_ms</
	name>
	    <value>600000</value>
	  </property>
	```

ECS file system URI	

fs.defaultFS	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:
	`viprfs://[*bucket_name*].[*namespace*].[*installation_name*]`• *bucket_name*: The name of the HDFS-enabled bucket that contains the data you want to use when you run Hadoop jobs.
	• *namespace*: The tenant namespace associated with the HDFS-enabled bucket.
	• *installation_name*: 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 fs.vipr.installations property.
	For example:
	```
	<property>
	      <name>fs.defaultFS</name>
	      <value>viprfs://testbucket.s3.testsite</value>
	  </property>
	```

Table 23 Hadoop core-site.xml properties (continued)

Property	**Description**
	HBase requires that a default file system be defined.
UMASK property	

fs.permissions.umask-mode	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:
	`<property>`
	`<name>fs.permissions.umask-mode</name>`
	`<value>022</value>`
	`</property>`

Identity translation properties	

fs.viprfs.auth.identity_translation	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 username@REALM, while Hadoop stores file owners as just the username.
	The possible values are:
	• 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 EMC.COM because sally is in the EMC.COM realm. The file ownership is changed john@EMC.COM.
	`# kinit sally@EMC.COM`
	`# hdfs dfs -chown john /path/to/file`
	Realms provided at the command line takes precedence over the property settings.
	`<property>`
	`  <name>fs.viprfs.auth.identity_translation`
	`           </name>`
	`  <value>CURRENT_USER_REALM</value>`
	` </property>`
	Note
	FIXED_REALM is now deprecated.

fs.viprfs.auth.realm	The realm assigned to users when the fs.viprfs.auth.identity_translation property is set to FIXED_REALM. This is now deprecated.
fs.viprfs.auth.anonymous_translation	This property is used to determine how user and group are assigned to newly created files

Table 23 Hadoop core-site.xml properties (continued)

Property	Description
	Note This property was previously used to determine what happened to files that had no owner. These files were said to be owned by "anonymous". Files and directories are no longer anonymously owned and . The values are: • 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 Kerberized Hadoop cluster. Assigns the Kerberos principal (user@REALM.COM) 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. `<property>` `<name>fs.viprfs.auth.anonymous_translation</name>` `<value>CURRENT_USER</value>` `</property>`
Kerberos realm and service principal properties	
viprfs.security.principal	This property specifies the ECS service principal. This property tells the KDC about the ECS service. This value is specific to your configuration. The principal name can include "_HOST" which is automatically replaced by the actual data node FQDN at run time. For example: `<property>` `<name>viprfs.security.principal</name>` `<value>vipr/_HOST@example.com</value>` `</property>`

Sample core-site.xml for simple authentication mode

This `core-site.xml` is an example of ECS HDFS properties for simple authentication mode.

Example 2 core-site.xml

```
<property>
  <name>fs.viprfs.impl</name>
  <value>com.emc.hadoop.fs.vipr.ViPRFileSystem</value>
</property>
```

Example 2 core-site.xml (continued)

```
<property>
  <name>fs.AbstractFileSystem.viprfs.impl</name>
  <value>com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem</value>
</property>

<property>
  <name>fs.vipr.installations</name>
  <value>Site1</value>
</property>

<property>
  <name>fs.vipr.installation.Site1.hosts</name>
  <value>203.0.113.10,203.0.113.11,203.0.113.12</value>
</property>

<property>
  <name>fs.vipr.installation.Site1.resolution</name>
  <value>dynamic</value>
</property>

<property>

<name>fs.vipr.installation.Site1.resolution.dynamic.time_to_live_ms<
/name>
  <value>900000</value>
</property>

<property>
  <name>fs.defaultFS</name>
  <value>viprfs://mybucket.mynamespace.Site1/</value>
</property>

<property>
  <name>fs.viprfs.auth.anonymous_translation</name>
  <value>CURRENT_USER</value>
</property>

<property>
  <name>fs.viprfs.auth.identity_translation</name>
  <value>FIXED_REALM</value>
</property>

<property>
  <name>fs.viprfs.auth.realm</name>
  <value>MY.TEST.REALM</value>
</property>
```

CHAPTER 28

Secure bucket metadata example

Secure bucket metadata

The following listing

```
{
   "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.amshbase.name",
           "value":    "amshbase@EXAMPLE_HDFS.EMC.COM"
       },
       {
           "name":    "internal.kerberos.user.amshbase.shortname",
           "value":    "ams"
       },
       {
           "name":    "internal.kerberos.user.amshbase.groups",
           "value":    "hadoop"
       },
       {
           "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.hbase.name",
           "value":    "hbase@EXAMPLE_HDFS.EMC.COM"
       },
       {
           "name":    "internal.kerberos.user.hbase.shortname",
           "value":    "hbase"
       },
       {
            "name":    "internal.kerberos.user.hbase.groups",
            "value":    "hadoop"
        },
        {
            "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.oozie.name",
            "value":    "oozie@EXAMPLE_HDFS.EMC.COM"
        },
        {
            "name":    "internal.kerberos.user.oozie.shortname",
            "value":    "oozie"
        },
        {
            "name":    "internal.kerberos.user.oozie.groups",
            "value":    "hadoop,users"
        },
        {
            "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.hbase.hosts",
            "value":    "*"
        },
        {
            "name":    "hadoop.proxyuser.hbase.users",
            "value":    "cmaurer"
        },
        {
            "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"
        }
    ]
}
```